

Кит Грант

CSS

для профи





CSS in Depth

KEITH J. GRANT



MANNING
SHELTER ISLAND

Кит Грант

CSS

ДЛЯ ПРОФИ



Санкт-Петербург · Москва · Екатеринбург · Воронеж
Нижний Новгород · Ростов-на-Дону · Самара · Минск

2019

ББК 32.988-02-018
УДК 004.738.5
Г77

Грант Кит

Г77 CSS для профи. — СПб.: Питер, 2019. — 496 с.: ил. — (Серия «Библиотека программиста»).

ISBN 978-5-4461-0909-8

Как вы понимаете, что зашли на хороший сайт? Это происходит практически мгновенно, с первого взгляда. Такие сайты привлекают внимание картинкой — отлично выглядят, а кроме этого они интерактивны и отзывчивы. Сразу видно, что такую страничку создавал CSS-профи, ведь именно каскадные таблицы стилей (CSS) отвечают за все наполнение и оформление сайта — от расположения элементов до неуловимых штрихов. Дело за малым — стать CSS-профи, а для этого придется разобраться в принципах CSS, научиться воплощать в жизнь идеи дизайнеров, не забывать о таких важных «мелочах», как красиво подобранный шрифт, плавные переходы и сбалансированная графика.

Перед вами прямой путь в высшую лигу веб-разработки. Книга «CSS для профи» подарит вам не только свежие идеи, но и вдохновит на подвиги, а облегчить этот тернистый путь помогут новейшие технические достижения — адаптивный дизайн, библиотеки шаблонов и многое другое.

16+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.988-02-018
УДК 004.738.5

Права на издание получены по соглашению с Apress. Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме без письменного разрешения владельцев авторских прав.

Информация, содержащаяся в данной книге, получена из источников, рассматриваемых издательством как надежные. Тем не менее, имея в виду возможные человеческие или технические ошибки, издательство не может гарантировать абсолютную точность и полноту приводимых сведений и не несет ответственности за возможные ошибки, связанные с использованием книги.

Издательство не несет ответственности за доступность материалов, ссылки на которые вы можете найти в этой книге. На момент подготовки книги к изданию все ссылки на интернет-ресурсы были действующими

ISBN 978-1617293450 англ
ISBN 978-5-4461-0909-8

© 2018 by Manning Publications Co. All rights reserved
© Перевод на русский язык ООО Издательство «Питер», 2019
© Издание на русском языке, оформление ООО Издательство «Питер», 2019
© Серия «Библиотека программиста», 2019

Краткое содержание

Предисловие.....	16
Введение	18
Благодарности	20
Об этой книге	22

Часть I. Обзор основных принципов

Глава 1. Каскадность, специфичность и наследование	28
Глава 2. Работа с относительными единицами	55
Глава 3. Знакомство с блочной моделью	85

Часть II. Разметка

Глава 4. Плавающие элементы.....	118
Глава 5. Flexbox-верстка.....	149
Глава 6. CSS-сетки	181
Глава 7. Контексты позиционирования и наложения	217
Глава 8. Адаптивный дизайн	243

Часть III. Масштабируемый CSS-код

Глава 9. Модульный CSS	278
Глава 10. Библиотеки компонентов.....	300

Часть IV. Темы повышенной сложности

Глава 11. Фоны, тени и режимы смешивания 326

Глава 12. Контраст, цвета и интервалы..... 348

Глава 13. Шрифтовое оформление 378

Глава 14. Переходы 404

Глава 15. Трансформации 422

Глава 16. Анимация 450

Приложения

Приложение А. Селекторы 474

Приложение Б. Препроцессоры 479

Оглавление

Предисловие.....	16
Введение	18
Благодарности.....	20
Об этой книге	22
Для кого предназначена книга	22
Структура издания	23
Условные обозначения и файлы примеров	24
Версии браузеров.....	25
Об авторе.....	25
Иллюстрация на обложке	26

Часть I. Обзор основных принципов

Глава 1. Каскадность, специфичность и наследование	28
1.1. Каскадность.....	29
1.1.1. Источник стилей.....	33
1.1.2. Специфичность селекторов	36
1.1.3. Исходный порядок.....	41
1.1.4. Два правила	44
1.2. Наследование	45
1.3. Специальные значения	48
1.3.1. Ключевое слово inherit	48
1.3.2. Ключевое слово initial.....	49

1.4. Сокращенная запись свойств	50
1.4.1. Остерегайтесь сокращений, скрыто переопределяющих другие стили ...	51
1.4.2. Порядок записи сокращенных значений	52
Итоги главы.....	54
Глава 2. Работа с относительными единицами	55
2.1. Мощность относительных значений.....	55
2.1.1. Борьба за pixel-perfect-дизайн.....	56
2.1.2. Конец эпохи pixel-perfect	56
2.2. Единицы em и rem	58
2.2.1. Единицы em для указания размера шрифта.....	59
2.2.2. Указание размера шрифта в единицах rem	63
2.3. Перестаньте думать в пикселах	65
2.3.1. Установка адекватного размера шрифта по умолчанию.....	66
2.3.2. Делаем панель адаптивной.....	68
2.3.3. Изменение размера отдельного компонента	69
2.4. Единицы измерения, относящиеся к размеру экрана устройства	71
2.4.1. Единицы vw для указания размера шрифта	74
2.4.2. Функция calc() для указания размера шрифта.....	74
2.5. Числа без единиц измерения и свойство line-height	75
2.6. Пользовательские свойства (или CSS-переменные).....	77
2.6.1. Динамическое изменение пользовательских свойств	79
2.6.2. Изменение пользовательских свойств с помощью JavaScript	82
2.6.3. Экспериментирование с пользовательскими свойствами.....	83
Итоги главы.....	84
Глава 3. Знакомство с блочной моделью	85
3.1. Трудности с шириной элемента.....	86
3.1.1. Избегаем волшебных чисел	89
3.1.2. Настройка блочной модели.....	90
3.1.3. Глобальное применение свойства box-sizing: border-box	91
3.1.4. Добавление зазора между колонками.....	93
3.2. Проблемы высоты элементов	94
3.2.1. Управление переполнением.....	95
3.2.2. Применение альтернатив к высотам, указанным в процентах	96
3.2.3. Свойства min-height и max-height.....	101
3.2.4. Центрирование контента по вертикали.....	102

3.3. Отрицательные значения полей.....	104
3.4. Схлопывание полей	105
3.4.1. Схлопывание между текстом	106
3.4.2. Схлопывание многочисленных полей.....	106
3.4.3. Схлопывание вне контейнера	107
3.5. Расстояние между элементами в контейнере	109
3.5.1. Учет изменения контента	111
3.5.2. Универсальное решение: селектор лоботомированной совы.....	112
Итоги главы.....	116

Часть II. Разметка

Глава 4. Плавающие элементы.....	118
4.1. Предназначение плавающих элементов	118
4.2. Схлопывание контейнера и clearfix	125
4.2.1. Что такое схлопывание контейнера	125
4.2.2. Что такое clearfix	128
4.3. Неожиданный «захват» плавающего элемента	131
4.4. Медиаобъекты и блочный контекст форматирования.....	134
4.4.1. Что такое блочный контекст форматирования	135
4.4.2. Использование блочного контекста форматирования для разметки медиаобъектов	136
4.5. CSS-сетки.....	138
4.5.1. Принципы CSS-сетки.....	139
4.5.2. Создание CSS-сетки	139
4.5.3. Добавление зазоров	144
Итоги главы.....	148
Глава 5. Flexbox-верстка.....	149
5.1. Принципы flexbox-верстки.....	150
5.1.1. Создание базовой flex-навигации.....	153
5.1.2. Добавление отступов и промежутков.....	156
5.2. Размеры flex-элементов	157
5.2.1. Свойство flex-basis	160
5.2.2. Свойство flex-grow	160
5.2.3. Свойство flex-shrink	162
5.2.4. Практические примеры.....	163

5.3. Направление flex-элементов	165
5.3.1. Изменение направления flex-элементов	167
5.3.2. Стилизовое форматирование формы авторизации	168
5.4. Выравнивание, промежутки и другие штрихи.....	170
5.4.1. Свойства flex-контейнеров	174
5.4.2. Свойства flex-элементов	175
5.4.3. Выравнивание flex-блоков	176
5.5. Пара вещей, о которых следует знать.....	178
5.5.1. Flex-баги.....	178
5.5.2. Полноформатная разметка	179
Итоги главы.....	179
Глава 6. CSS-сетки	181
6.1. Веб-разметка уже здесь	182
6.1.1. Создание базовой сетки.....	183
6.2. Анатомия сетки.....	185
6.2.1. Нумерация линий сетки	191
6.2.2. Совместная работа с flex-блоками.....	192
6.3. Альтернативный синтаксис	196
6.3.1. Присвоение имен линиям сетки	196
6.3.2. Присвоение имен областям сетки	198
6.4. Явная и неявная сетка	200
6.4.1. Внесем разнообразие.....	204
6.4.2. Подгонка элементов для заполнения полосы сетки.....	207
6.5. Запросы функций.....	210
6.6. Выравнивание.....	213
Итоги главы.....	216
Глава 7. Контексты позиционирования и наложения	217
7.1. Фиксированное позиционирование	218
7.1.1. Создание модального окна с фиксированным позиционированием	218
7.1.2. Управление размером позиционированных элементов	221
7.2. Абсолютное позиционирование.....	222
7.2.1. Абсолютное позиционирование кнопки Закреть	222
7.2.2. Позиционирование псевдоэлементов	223
7.3. Относительное позиционирование	225
7.3.1. Создание раскрывающегося меню.....	226
7.3.2. Создание треугольника CSS	229

7.4. Контексты наложения и z-индекса	231
7.4.1. Процесс рендеринга и порядок наложения	232
7.4.2. Управление наложением с помощью свойства z-index	234
7.4.3. Контексты наложения	235
7.5. Липкое позиционирование	239
Итоги главы	242
Глава 8. Адаптивный дизайн	243
8.1. Подход Mobile First	244
8.1.1. Создание мобильного меню	252
8.1.2. Добавление метатега viewport	257
8.2. Медиазапросы	258
8.2.1. Типы медиазапросов	260
8.2.2. Добавление контрольных точек на страницу	262
8.2.3. Добавление адаптивных колонок	266
8.3. Резиновые макеты	268
8.3.1. Добавление стилей для большой области просмотра	269
8.3.2. Работа с таблицами	272
8.4. Адаптивные изображения	273
8.4.1. Использование нескольких изображений для экранов разных размеров	274
8.4.2. Использование атрибута srcset для передачи нужного изображения	275
Итоги главы	276

Часть III. Масштабируемый CSS-код

Глава 9. Модульный CSS	278
9.1. Базовые стили: закладываем основы	279
9.2. Простой модуль	281
9.2.1. Вариации модуля	282
9.2.2. Модули с множеством элементов	287
9.3. Составление крупных структур из модулей	290
9.3.1. Разделение ответственности между модулями	290
9.3.2. Именованние модулей	295
9.4. Вспомогательные классы	296
9.5. Методологии CSS	297
Итоги главы	299

Глава 10. Библиотеки компонентов.....	300
10.1. Введение в KSS	301
10.1.1. Установка KSS.....	302
10.1.2. Написание KSS-документации	304
10.1.3. Документирование вариаций модуля	308
10.1.4. Создание начальной страницы.....	311
10.1.5. Документирование модулей, которым требуется JavaScript	311
10.1.6. Упорядочение контента библиотеки компонентов по разделам	314
10.2. Инновационный способ верстки CSS	316
10.2.1. Метод CSS First	317
10.2.2. Библиотека компонентов в качестве API.....	318
Итоги главы.....	324

Часть IV. Темы повышенной сложности

Глава 11. Фоны, тени и режимы смешивания	326
11.1. Градиенты.....	327
11.1.1. Использование нескольких цветовых узлов	329
11.1.2. Использование радиального градиента.....	332
11.2. Тени.....	334
11.2.1. Создание объема с помощью градиентов и теней	335
11.2.2. Элементы с плоским дизайном.....	336
11.2.3. Создание кнопок с более современным дизайном	337
11.3. Режимы смешивания.....	338
11.3.1. Изменение оттенка изображения	341
11.3.2. Виды режимов смешивания	342
11.3.3. Добавление текстуры изображению.....	343
11.3.4. Микширование режимов смешивания	345
Итоги главы.....	347
Глава 12. Контраст, цвета и интервалы.....	348
12.1. Царство контраста	349
12.1.1. Создание шаблона	351
12.1.2. Реализация дизайна.....	352

12.2. Цвета	353
12.2.1. Разбираемся с нотациями цветов	360
12.2.2. Добавление цветов в палитру	364
12.2.3. Применение контраста при выборе цвета текста	367
12.3. Интервалы	368
12.3.1. Единицы em или пиксели?	369
12.3.2. Вычисление высоты строки	371
12.3.3. Интервалы между строчными элементами	374
Итоги главы	377
Глава 13. Шрифтовое оформление	378
13.1. Веб-шрифты	380
13.2. Сервис Google Fonts	381
13.3. Как работает свойство @font-face	386
13.3.1. Форматы шрифтов и замена при необходимости	387
13.3.2. Варианты начертания в одной гарнитуре	388
13.4. Управление интервалами в целях повышения читаемости	390
13.4.1. Интервалы основного текста сайта	391
13.4.2. Заголовки, мелкие элементы и интервалы	392
13.5. Вспышки нестилизованного и невидимого текста	397
13.5.1. Библиотека Font Face Observer	398
13.5.2. Откат к системным шрифтам	400
13.5.3. И наконец, свойство font-display	402
Итоги главы	403
Глава 14. Переходы	404
14.1. Отсюда сюда	404
14.2. Функции времени	407
14.2.1. Изучение кривых Безье	409
14.2.2. Шаги	412
14.3. Неанимируемые свойства	413
14.3.1. Свойства, которые нельзя анимировать	416
14.3.2. Появление и исчезновение	417
14.4. Переход к автоматическому выравниванию высоты	419
Итоги главы	421

Глава 15. Трансформации	422
15.1. Вращение, масштабирование, смещение и наклон	422
15.1.1. Изменение точки трансформации	426
15.1.2. Применение нескольких трансформаций.....	426
15.2. Анимированные трансформации	427
15.2.1. Масштабирование значков.....	433
15.2.2. Создание «вылетающих» меток	436
15.2.3. Поэтапные переходы	438
15.3. Производительность анимации	439
15.3.1. Рендеринг страницы	440
15.4. Трехмерные (3D) трансформации.....	442
15.4.1. Контроль перспективы.....	443
15.4.2. Профессиональные приемы 3D-трансформации	446
Итоги главы.....	449
Глава 16. Анимация	450
16.1. Ключевые кадры	451
16.2. Анимация 3D-трансформаций.....	454
16.2.1. Создание макета без анимации	454
16.2.2. Добавление анимации в макет	459
16.3. Задержка запуска анимации и режим заполнения	461
16.4. Передача смысла с помощью анимации	464
16.4.1. Реакция на действие пользователя	464
16.4.2. Привлечение внимания пользователя	468
16.5. Совет напоследок	471
Итоги главы.....	472

Приложения

Приложение А. Селекторы	474
А.1. Базовые селекторы	474
А.2. Комбинаторы	474
А.3. Селекторы псевдоклассов	475
А.4. Селекторы псевдоэлементов	477
А.5. Селекторы атрибутов	478

Приложение Б. Препроцессоры	479
Б.1. Препроцессор Sass	480
Б.1.1. Установка препроцессора Sass	480
Б.1.2. Запуск препроцессора Sass.....	481
Б.1.3. Важные функции препроцессора Sass.....	482
Б.2. PostCSS.....	491
Б.2.1. Использование инструмента Autoprefixer	491
Б.2.2. Применение cssnext.....	492
Б.2.3. Использование cssnano.....	492
Б.2.4. Использование PreCSS.....	493

Предисловие

«Минута, чтобы научиться... Целая жизнь, чтобы овладеть». Эта фраза в наши дни может показаться немного банальной, но она отображает суть CSS. Фраза обрела популярность, став слоганом настольной игры «Реверси», в которой игроки по очереди перемещают белые и черные фигуры по клеткам на доске. Если, например, белая фигура захватывает ряд черных между двумя белыми, все черные фигуры переворачиваются и ряд становится полностью белым.

Правила CSS изучить так же несложно, как правила «Реверси». Вы создаете селектор, чтобы сопоставить элементы; затем вы пишете пары «ключ/значение», которые форматируют эти элементы. Даже начинающим разработчикам не составит большого труда понять основной синтаксис. Секрет того, как стать *гуру* в CSS, как и в «Реверси»: нужно точно знать, *что* делать и *когда*.

CSS (каскадные таблицы стилей) — это один из языков для веб-разработки, но работа с ним не совсем то же самое, что программирование. В CSS мало логики и циклов. Математика ограничивается одной функцией. Лишь недавно стало возможно использовать в нем переменные. Вы почти не будете думать о безопасности. Каскадные таблицы стилей ближе к живописи, чем к языку программирования Python. С помощью CSS вы вольны делать все, что нравится. Каскадные таблицы стилей не будут озадачивать вас какими-то ошибками или сбоями в компиляции.

На пути к тому, чтобы стать гуру в CSS, вы должны изучить все, на что способны каскадные таблицы стилей. Чем больше вы знаете, тем более легким покажется старт. Чем больше тренируетесь, тем легче получите идеальную компоновку и расстояния между блоками. Чем больше читаете, тем увереннее будете чувствовать себя при создании любого дизайна.

Действительно хорошие разработчики CSS не ограничиваются единым дизайном. Каждая работа может стать зубодробительной головоломкой, которую нужно решить. Опытные разработчики CSS имеют полный и широкий спектр знаний о том, на что способны каскадные таблицы стилей. Эта книга станет этапом на пути к тому, чтобы вы могли стать хорошим разработчиком CSS. Прочитав ее, вы узнаете, как создавать макеты любой сложности.

Разрешите еще одну метафору: несмотря на то что каскадные таблицы стилей разрабатываются уже пару десятилетий, можно провести некую аналогию с Диким Западом. Вы можете делать все, что захотите, до тех пор, пока CSS делает то, что вы хотите. Нет никаких жестких правил. Но, поскольку вы предоставлены сами себе и не ограничены строгими рамками, вы должны быть предельно аккуратны. Мельчайшие изменения могут иметь огромные последствия. Каскадные таблицы стилей способны расти и становиться громоздкими. Вы можете испугаться собственных стилей!

Кит описывает в этой книге почти все особенности CSS, и каждый, даже маленький ее фрагмент поможет вам стать отличным разработчиком CSS и приручить этот Дикий Запад. Вы глубоко погрузитесь в сам язык, поймете, на что способны каскадные таблицы стилей. Вы будете лучше писать код — работоспособный и понятный.

Книга будет полезна даже опытным разработчикам. Обнаружив, что читаете о чем-то уже известном, вы укрепите свои навыки, подтвердите знания и дополните их недостающими фрагментами, которые удивят вас и расширят вашу теоретическую базу.

*Крис Койер (Chris Coyier), сооснователь
компании CodePen*

Введение

Впервые о каскадных таблицах стилей заговорили в 1994 году, и уже в 1996-м они были реализованы (частично) в браузере Internet Explorer 3. Примерно в то время я нажал прекрасную кнопку просмотра исходного кода и увидел, что все секреты веб-страницы расшифрованы в виде простого текста. Я сам изучил HTML и CSS, развлекаясь в текстовом редакторе и наблюдая за тем, что получалось. Это был приятный повод провести как можно больше времени в Интернете.

Тем временем мне нужно было найти настоящую работу. Я получил степень в области компьютерных наук. Но не знал, что в 2000-х годах это словосочетание потеряет смысл, так как появится профессия «веб-разработчик».

Я начал пользоваться каскадными таблицами стилей еще с момента их появления. Я работал и на стороне сервера, и с клиентскими интерфейсами, но в любой команде всегда оказывался экспертом в области CSS. Надо признать, что это часть Всемирной паутины, которой пренебрегают. Но как только вы поработаете над проектом с чистым кодом CSS, вы уже не захотите впредь обходиться без него. Увидев его в действии, даже опытные веб-разработчики спрашивают: «Как выучить CSS?»

На этот вопрос нет краткого и однозначного ответа. Недостаточно освоить один или два быстрых приема. Скорее, вам нужно разобраться в отдельных составляющих языка и понять, как они могут сочетаться. Одни книги дают общее представление о CSS, но многие разработчики уже что-то в нем понимают. В других книгах рассмотрено много полезных приемов, но предполагается, что читатель уже владеет языком.

В то же время темпы изменения CSS постоянно растут. Адаптивный дизайн сейчас стал стандартом де-факто. Веб-шрифты считаются банальщиной. В 2016 году мы наблюдали расцвет гибких контейнеров, а в 2017-м развитие получила CSS-сетка. Режимы смешивания, тени блоков, преобразования, переходы и анимация — все это нововведения в создании веб-страниц. Поскольку все больше и больше браузеров становятся «вечнозелеными», автоматически обновляясь до последней версии, новые возможности будут продолжать добавляться. Их достаточно, чтобы идти в ногу со временем.

Независимо от того, новичок вы в веб-дизайне или занимаетесь им уже некоторое время, вам нужно развивать или обновлять свои навыки работы с каскадными таблицами стилей. И я написал эту книгу, чтобы дать вам «волшебный пинок». Весь материал в нее отбирался по одному из трех оснований.

1. *Он важен.* Существует множество фундаментальных понятий языка, в которых, к сожалению, многие разработчики не до конца разбираются. К ним относятся каскадность, поведение элементов с обтеканием и позиционирование. Мы внимательно изучим их, и я объясню, как они работают.
2. *Он актуален.* За последние несколько лет появилось много новых функций. Я расскажу о последних улучшениях CSS и нескольких неочевидных возможностях. Это перспективная книга. Я укажу на проблемы обратной совместимости, когда это уместно, но я оптимистично отношусь к настоящему и будущему развития кросс-браузерности.
3. *Он не раскрыт в большинстве книг, посвященных CSS.* Мир каскадных таблиц стилей огромен. В современном мире разработки веб-приложений существуют важные передовые методы и общие подходы. Строго говоря, они являются частью не самого языка CSS, а скорее его культуры. И жизненно важны для современной веб-разработки.

Итак, как изучить CSS? Эта книга позиционируется как попытка ответить на данный вопрос.

Благодарности

Написание книги требует невероятных усилий. Я считаю, что это отличная книга, и надеюсь, вы согласитесь, но ее не было бы без помощи других людей.

Прежде всего я хотел бы поблагодарить свою супругу Кортни. Ты поддерживала и поощряла меня на протяжении всего процесса. Ты взяла на себя бремя написания этой книги вместе со мной. И даже помогла с редактурой в некоторых ключевых местах. Я не смог бы сделать это без тебя.

Я хотел бы поблагодарить своего начальника Марка Игла (Mark Eagle) и остальных членов моей команды в сети Intercontinental Exchange. Благодарю вас за то, что помогли мне на этом пути и позволили иногда отвлекаться от основной работы, чтобы писать книгу во время бесчисленных обеденных перерывов.

Благодарю редактора Грега Уайлда (Greg Wild), который нашел мои первые жалкие черновики в Интернете и убедил меня продолжать. Также благодарю главу издательства Manning Марьяна Баце (Marjan Bace), который почувствовал потенциал в этой идее. Всегда есть риск, связанный с изданием книги, особенно с новым автором. Спасибо, что дали шанс.

Хорошая книга не получится без редактора. Благодарю Кристен Уоттерсон (Kristen Watterson) за ее педантизм. Эта книга стала намного лучше благодаря вашим усилиям. И спасибо техническому редактору Робину Деусону (Robin Dewson), который играл роль защитника команды, за терпение и понимание на протяжении всего длительного процесса.

Благодарю Бирну Себарте (Birnou Sebarté) и Луиса Лазариса (Louis Lazaris) за то, что они тщательно откорректировали книгу от корки до корки. Спасибо Крису Койеру (Chris Coyer) за предисловие к ней.

Я также хотел бы поблагодарить технических рецензентов и друзей, которые нашли время, чтобы изучить мои проекты на разных этапах и отправить отзывы: Адама Раккиса (Adam Rackis), Ала Пезевски (Al Pezewski), Амита Ламба (Amit Lamba), Анто Аравинта (Anto Aravinth), Брайна Гайнеца (Brian Gaines), Дико Гольдони (Dico Goldoni), Джанкарло Массари (Giancarlo Massari), Гетца Хеллера (Goetz Heller), Арса Равалья (Harsh Raval), Джеймса Анаипакоса (James Anaipakos), Джеффри Лима (Jeffrey Lim), Джима Артура (Jim Arthur), Мэтью Хальверсона (Matthew Halverson),

Митчелл Роблес-мл. (Mitchell Robles, Jr.), Нитина Варму (Nitin Varma), Патрика Гетца (Patrick Goetz), Фили Австрия (Phily Austria), Пьерфранческо Д'Орсогну (Pierfrancesco D'Orsogna), Рафаэля Кассемиро Фрейре (Rafael Cassemiro Freire), Рафаэля Фрейре (Rafael Freire), Сачина Синхи (Sachin Singhi), Таню Вильке (Tanya Wilke), Трента Уайтли (Trent Whiteley) и Уильяма Э. Уилера (William E. Wheeler). Ваши отзывы оказались очень полезны, так как книга попадет в руки разработчиков всех уровней квалификации.

Наконец, я хотел бы выразить огромную благодарность толковым людям из рабочей группы каскадных таблиц стилей W3C за работу над спецификациями CSS. Вы трудитесь над решением очень сложных задач, благодаря чему разработчикам не приходится с ними разбираться самостоятельно. Спасибо за ваши усилия по улучшению каскадных таблиц стилей и Всемирной паутины.

Об этой книге

Мир каскадных таблиц стилей непрерывно совершенствуется. Все больше и больше веб-разработчиков осознают, что, хотя они, как им кажется, знают CSS, до полного понимания им далеко. В последние годы язык сильно развился, поэтому даже те разработчики, которые когда-то были искусны в CSS, сегодня могут получить абсолютно новые навыки, чтобы наверстать упущенное. Данная книга призвана удовлетворить эти потребности: обеспечить глубокое владение языком и привести к успеху в новых разработках и применении новейших возможностей CSS.

Эта книга называется «CSS для профи», но это также *всеобъемлющая* книга. В тех случаях, когда какие-то понятия или принципы трудны либо, как правило, трактуются неправильно, я подробно объясняю, что они обозначают или как работают и почему именно так. Другие главы, возможно, не настолько исчерпывающи, но я дам достаточно сведений, чтобы вы могли эффективно работать и двигаться в правильном направлении, если захотите расширить свои знания. В целом эта книга заполнит пробелы в ваших теоретических познаниях.

Некоторые темы: анимация, типографика, гибкие контейнеры и даже CSS-стек — достойны отдельных книг. Мои цели — конкретизировать ваши знания, помочь ликвидировать пробелы в них и привить вам любовь к языку CSS.

Для кого предназначена книга

Прежде всего эта книга предназначена для разработчиков, которые устали сражаться с каскадными таблицами стилей и хотят действительно разобраться, как они работают. И неважно, они новички или имеют за плечами многолетний опыт.

Я рассчитываю, что у вас есть поверхностное знание языка HTML, CSS и отчасти JavaScript. Если вы знакомы с основами синтаксиса CSS, то, скорее всего, сможете понять эту книгу. Но в первую очередь она написана для разработчиков, которые некоторое время работали с CSS, столкнулись с проблемами и разочаровались. Код JavaScript я максимально упрощал, поэтому, изучая короткие листинги, вы все поймете.

Если вы дизайнер, который решил приоткрыть для себя мир веб-дизайна, то, подозреваю, тоже многое почерпнете из этой книги, хотя я не писал ее непосредственно

для вас. По крайней мере вы научитесь лучше понимать разработчиков, с которыми будете сотрудничать.

Структура издания

Книга состоит из 16 глав, разделенных на четыре части. В части I мы поговорим об основах, сосредоточив внимание на некоторых деталях.

- ❑ Глава 1 охватывает каскадность и наследование. Эти понятия определяют, как стили применяются к элементам на странице.
- ❑ В главе 2 рассматриваются относительные единицы *em* и *rem*. Относительные единицы в CSS универсальны и важны, и эта глава научит вас работать с ними.
- ❑ В главе 3 рассматривается блочная модель, суть которой — управление размерами элементов на странице и объемом пространства между ними.

В части II я познакомлю вас с ключевыми инструментами для компоновки элементов на странице.

- ❑ Глава 4 погружает в принципы обтекания элементов макета. Мы создадим многоколоночную страницу и рассмотрим решения некоторых необычных проблем обтекания.
- ❑ В главе 5 говорится о Flexbox — способе гибкой верстки. Начнем мы с фундаментальных понятий, затем перейдем к практическим примерам.
- ❑ Глава 6 рассказывает о технологии верстки с помощью CSS-сетки. Она позволяет создавать макеты, которые ранее средствами каскадных таблиц стилей получить было невозможно.
- ❑ Глава 7 рассматривает позиционирование с использованием свойства **position**: абсолютное позиционирование, фиксированное позиционирование и многое другое. Это область, в которой многие разработчики сталкиваются с трудностями, поэтому им необходимо хорошо в ней разобраться.
- ❑ В главе 8 описывается адаптивный дизайн. Мы рассмотрим три ключевых принципа построения сайтов, работающих на экранах различных размеров и на широком спектре устройств.

В части III представлены некоторые передовые методы. Разместить элементы на странице желаемым образом — это одно, а организовать код так, чтобы его можно было понимать и поддерживать в будущем по мере развития веб-приложения, — совершенно другое. Из следующих двух глав вы узнаете о некоторых важных методах управления кодом.

- ❑ В главе 9 описано, как организовать каскадные таблицы стилей в модули, чтобы код стал многоразовым и удобным в сопровождении.
- ❑ В главе 10 рассказывается о создании библиотеки паттернов. Это важная часть использования и поддержки каскадных таблиц стилей в команде.

В части IV я познакомлю вас с миром дизайна. Мы рассмотрим важные соображения, касающиеся работы с дизайнером, и поговорим о том, как работать над дизайном самостоятельно, потому что иногда приходится и это делать.

- ❑ В главе 11 рассматриваются тени, градиенты и режимы смешивания. В совокупности эти эффекты позволяют создать элегантный пользовательский интерфейс.
- ❑ В главе 12 показано, как работать с контрастностью, цветом и пространством (воздухом). Эти детали — важный шаг на долгом пути от хорошего дизайна к отличному.
- ❑ Глава 13 посвящена типографике в Интернете — использованию веб-шрифтов для придания индивидуальности сайту или приложению.
- ❑ Глава 14 рассказывает, как наделить страницу переходами и анимацией, изменять форму, цвета и размеры элементов на странице.
- ❑ В главе 15 рассматриваются трансформации — важные инструменты, действующие совместно с переходами и анимацией. В этой главе обсуждаются также вопросы производительности анимации на странице.
- ❑ В главе 16 говорится об анимации ключевых кадров. Вы узнаете, как использовать сложную анимацию для взаимодействия с пользователем.

В конце книги приведены два приложения.

- ❑ Приложение А содержит перечень селекторов CSS всех типов.
- ❑ Приложение Б представляет собой введение в препроцессоры. Если вы еще не знакомы с ними, можете начать с этого приложения.

Я затратил много усилий для того, чтобы подробно раскрыть все темы книги. Начну с самых основ, которые вам нужно знать. Далее темы опираются друг на друга. Во многих местах я ссылаюсь на ранние концепции и стараюсь связать их вместе, если это необходимо. Хотя я и добавил полезные справочные приложения, рекомендую читать главы по порядку.

Условные обозначения и файлы примеров

Эта книга содержит много примеров исходного кода как в пронумерованных листингах, так и в обычном тексте. В обоих случаях исходный код обозначается **моноширинным шрифтом**, чтобы отделить его от основного текста. Иногда он выделен **полужирным моноширинным шрифтом**, чтобы показать код, который изменился с предыдущих шагов, приведенных в той же главе, например, если к строке кода добавляется новая функция или в нее вносятся какие-то изменения.

Во многих случаях исходный код был переформатирован: я добавил переносы и переделал отступы, чтобы сэкономить пространство страницы. Кроме того, во многих случаях из исходного кода были удалены комментарии, если он описан в тексте. Большинство листингов сопровождаются примечаниями, которые выделяют важные понятия.

Каскадные таблицы стилей работают в тандеме с HTML. Я всегда предоставляю листинги с кодом, один для HTML, а другой — для CSS. В большинстве разделов для нескольких листингов CSS используется один и тот же HTML-код. Я научу вас изменять каскадные таблицы стилей на многих этапах и рассчитываю, что вы будете менять свою таблицу стилей по мере перехода от одного листинга к другому.

Исходный код файлов примеров из этой книги размещен в репозитории `git` на странице github.com/CSSInDepth/css-in-depth и сайте издательства по адресу www.manning.com/books/css-in-depth. На первых порах может показаться, что некоторых листингов не хватает, так как примеры требуют указания как HTML-, так и CSS-кода. Учтите, что в большинстве примеров я поместил каскадные таблицы стилей в HTML-файлы, добавляя элементы `style`. Это означает, что оба листинга, HTML и CSS, в репозитории объединяются в один файл.

Например, в главе 1 листинг 1.1 — это HTML-код, а листинг 1.2 — CSS-код стилей, которые предназначены для применения к этому HTML-коду. В репозитории оба листинга объединены в один файл с именем `listing-1.2.html`. В листинге 1.3 в этот CSS-код вносятся изменения, он включен в файл `listing-1.3.html` вместе с соответствующим HTML-кодом из листинга 1.1.

Версии браузеров

Кросс-браузерное тестирование — важная часть веб-разработки. Большая часть кода в этой книге поддерживается браузерами Internet Explorer 10 и 11, Microsoft Edge, Chrome, Firefox, Safari, Opera и почти всеми мобильными браузерами. Новые функции будут работать не во всех этих браузерах, что я буду особо отмечать.

То, что функция не поддерживается в определенном браузере, не означает, что ее нельзя использовать. Часто можно включить *альтернативное* поведение для старых браузеров, которые не поддерживают какой-либо новый функционал. Я расскажу о таких случаях.

Если вы хотите придерживаться примеров кода, которые описаны в книге, на своем компьютере, рекомендую работать в последней версии браузера Firefox или Chrome.

Об авторе

Кит Дж. Грант (Keith J. Grant) в настоящее время трудится старшим веб-разработчиком в корпорации Intercontinental Exchange, Inc. (ICE), где пишет и поддерживает CSS-код для корпоративных клиентов, в том числе Нью-Йоркской фондовой биржи. У него за плечами 11-летний опыт создания и поддержки профессиональных веб-приложений и сайтов, сочетающих технологии HTML, CSS и JavaScript. Кит — классический самоучка в плане работы с HTML- и CSS-кодом: до того как его приняли в штат компании, он потратил несколько лет, самостоятельно приобретая опыт веб-разработчика.

Менеджер пригласил его в команду веб-разработчиков именно из-за опыта работы с CSS, когда компании ICE понадобился редизайн корпоративных сайтов. Каскадные таблицы стилей позволяют создавать уникальные креативные сайты, а также сложные веб-приложения с громоздкими макетами.

Хотя Кит в первую очередь занимался разработкой на языке JavaScript, для всех компаний, где он трудился, он становился крайне важным человеком, поскольку отлично разбирался в каскадных таблицах стилей.

Иллюстрация на обложке

Иллюстрация на обложке книги «CSS для профи» — это работа французского художника Октава Пингвилли Л'Харидона, которая называется *La Flamande* («Фламандка»). Картина входит в коллекцию произведений художников XIX века, собранную Луисом Курмером и впервые представленную в Париже в 1842 году. Название коллекции — *Les Français peints par eux-mêmes*, которое переводится как «Люди Франции, написанные самими собой». Каждая иллюстрация тонко прорисована и раскрашена вручную, а богатое разнообразие рисунков в коллекции напоминает нам о том, насколько культурно обособлялись районы, города, деревни и соседние области всего 200 лет назад. Изолированные друг от друга люди говорили на разных диалектах и языках. На улицах городов или в сельской местности было легко определить, где они жили, а об их профессии или месте в обществе говорила их одежда.

С тех пор такие кардинальные различия в одеяниях размылись и разнообразие, характеризующее регионы, столь богатое в прежние времена, исчезло. Сейчас трудно отличить друг от друга даже жителей разных континентов, не говоря уже о разных регионах страны и городах. Возможно, мы сменили культурные различия на более разнообразную личную и, безусловно, более многообразную и быстро развивающуюся технологическую жизнь.

В настоящее время, когда трудно отличить одну компьютерную книгу от другой, издательство Manning объединяет компьютерную литературу с выпущенными более двух столетий назад книгами, украшая обложки иллюстрациями, основанными на богатом разнообразии жизненного уклада людей тех времен.

Часть I

Обзор основных принципов

В части I подробно рассматриваются наиболее важные составляющие CSS — каскадность, относительные единицы и блочная модель. Из первых трех глав вы узнаете, как стили применяются к элементам на странице и как определяются размеры элементов. Всестороннее усвоение данных понятий — это база для всех последующих тем книги.

1

Каскадность, специфичность и наследование

В этой главе

- Четыре компонента каскадности.
- Разница между каскадностью и наследованием.
- Управление стилями, применяемыми к элементам.
- Распространенные недоразумения с сокращенными объявлениями.

CSS отличается от многих средств разработки программного обеспечения. Строго говоря, это не язык программирования, хотя он требует абстрактного мышления. Это не исключительно инструмент дизайна, хотя понадобится толика креатива. Каскадные таблицы стилей обеспечивают обманчиво простой декларативный синтаксис, но если вы использовали его при работе над крупными проектами, то знаете, что он способен перерасти в серьезную проблему.

Если требуется научиться делать что-то в обычном программировании, вы чаще всего знаете, что искать (например, «как найти элементы типа x в массиве»). С CSS не всегда легко решить проблему с помощью только одного запроса в Интернете. Даже когда вы можете это сделать, ответ часто от чего-то зависит. Чтобы добиться чего-либо, нередко нужно уметь обрабатывать разнообразные ситуации.

Первая часть книги начинается с рассмотрения фундаментальных принципов языка: каскадности, блочной модели и широкого набора доступных единиц измерения. Большинство веб-разработчиков знают о каскадности и блочной модели. Они знают о пикселях как о единицах, в которых можно измерять размер элементов, и, вероятно, слышали, что должны использовать единицы `em`. Но суть в том, что подобных тем очень много и поверхностного их рассмотрения недостаточно. Если вы решили изучить CSS, то должны сначала разобраться в фундаментальных понятиях, причем достаточно

Я знаю, что вам не терпится начать изучать современные и красивые правила CSS. Это очень интересный материал. Но сначала обратимся к основам. Я кратко расскажу о базовых вещах, с которыми вы, вероятно, уже знакомы, а затем мы подробно разберем каждую тему. Моя цель — укрепить фундамент, на котором строится ваше понимание CSS.

В этой главе начнем с каскадности. Я сформулирую, как она действует, а затем покажу, как работать с каскадностью на практике. После этого мы рассмотрим смежную тему — наследование. Я буду придерживаться этого плана, говоря о свойствах и некоторых распространенных недоразумениях, возникающих по поводу них.

Все эти темы касаются применения желаемых стилей к нужным элементам. Здесь можно допустить много нелепых ошибок. Глубокое понимание этих тем позволит лучше контролировать работу вашего CSS-кода, чтобы он делал то, что вы хотите. Если повезет, вам даже понравится работать с CSS.

1.1. Каскадность

По сути, CSS — это объявленные правила: мы хотим, чтобы в различных условиях происходили определенные вещи. Если этот класс добавлен к тому элементу — применить такие-то стили. Если элемент *X* является потомком элемента *Y* — вот эти. Затем браузер принимает эти правила, определяет, какие из них действуют, в каком порядке и в каком месте, и использует их при визуализации страницы.

Когда рассматриваются небольшие примеры, этот процесс обычно не вызывает сложностей. Но по мере увеличения размера таблицы стилей или количества веб-страниц, к которым вы ее применяете, код может неожиданно быстро усложниться. В CSS существует несколько способов делать одно и то же. При изменении структуры HTML или использовании стилей на разных страницах результаты могут оказаться совершенно разными. Это зависит от того, какое решение задействовано. Ключевая часть разработки CSS сводится к написанию правил таким образом, чтобы они были предсказуемыми.

Первый шаг — разобраться в том, как именно браузер понимает ваши правила. Каждое из них может быть простым, но что происходит, когда два правила противоречат друг другу в том, как форматировать элемент? Вы можете обнаружить, что одно из ваших правил не делает того, что вы ожидаете, поскольку с ним конфликтует другое правило.

Предсказывать, как поведут себя правила, реально, если понимать принципы каскадности. Продемонстрирую это. Создайте простенькую шапку типа той, которая находится в верхней части веб-страницы (рис. 1.1). В шапке указано название сайта, под ним — навигационные ссылки. Последняя ссылка окрашена в оранжевый цвет — так выделяются важные элементы. (Внимание: в печатной версии книги изображения черно-белые.)

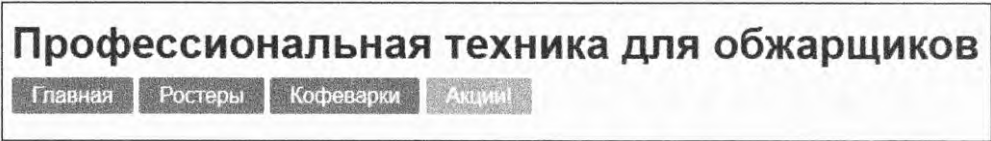


Рис. 1.1. Название страницы и навигационные ссылки

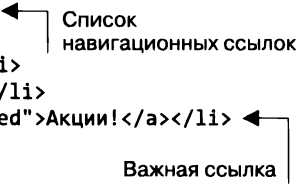
Для начала создайте HTML-документ и таблицу стилей с именем `styles.css`. Добавьте код, показанный в листинге 1.1, в HTML-файл.

ПРИМЕЧАНИЕ

Репозиторий, содержащий все примеры для этой книги, доступен для загрузки по адресу `github.com/CSSInDepth/css-in-depth`. В нем вы найдете HTML-файлы со встроенными правилами CSS.

Листинг 1.1. Разметка для шапки страницы

```
<!doctype html>
<head>
  <meta charset="utf-8">
  <link href="styles.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <header class="page-header">
    <h1 id="page-title" class="title">Профессиональная | Название страницы
      техника для обжарщиков</h1>
    <nav>
      <ul id="main-nav" class="nav">
        <li><a href="/">Главная</a></li>
        <li><a href="/coffees">Ростеры</a></li>
        <li><a href="/brewers">Кофеварки</a></li>
        <li><a href="/specials" class="featured">Акции!</a></li>
      </ul>
    </nav>
  </header>
</body>
```



Если к одному и тому же элементу страницы применяется два правила или больше, это может привести к конфликту объявлений. Листинг 1.2 показывает, как это происходит. В нем три набора правил задают шрифт для названия страницы. Название не может отображаться тремя разными шрифтами одновременно. Какой конкретно будет использован? Добавьте этот листинг в свой CSS-файл, чтобы узнать ответ.

Инструкции с конфликтующими объявлениями могут следовать одна за другой или оказаться разбросанными по всей таблице стилей. В любом случае, учитывая HTML-код, все они нацелены на один и тот же элемент.

Листинг 1.2. Конфликтующие объявления

```
h1 {  
  font-family: serif;  
}  
  
#page-title {  
  font-family: sans-serif;  
}  
  
.title {  
  font-family: monospace;  
}
```

Селектор тега

Селектор идентификатора

Селектор класса

Все три набора правил пытаются установить для названия собственное семейство шрифтов. Кто победит? Чтобы получить ответ, браузер следует конкретному набору правил, поэтому результат оказывается предсказуем. В этом случае правила указывают, что выигрывает второе объявление, с селектором идентификатора. Название будет оформлено рубленным шрифтом (sans-serif), применяемым по умолчанию (рис. 1.2).

Профессиональная техника для обжарщиков

- [Главная](#)
- [Ростеры](#)
- [Кофеварки](#)
- [Акции!](#)

Рис. 1.2. Селектор идентификатора выигрывает у других наборов правил, отображая название страницы рубленным шрифтом

Каскадность — так называется этот набор правил. Каскадность определяет способы разрешения конфликтов, и это фундаментальная часть работы CSS. Большинство опытных разработчиков имеют общее представление о каскадности, однако не всегда правильно интерпретируют ее.

Разберемся с каскадностью. Когда объявления конфликтуют, для устранения проблемы нужно учесть три показателя.

1. *Источник стилей* — место их расположения. Ваши стили накладываются на стили браузера, применяемые по умолчанию.
2. *Специфичность селекторов* — то, какие селекторы имеют приоритет над другими.
3. *Исходный порядок* — порядок, в котором стили объявляются в таблице стилей.

Правила каскадирования рассматривают их именно в этом порядке. Рисунок 1.3 обобщенно показывает, как они работают.

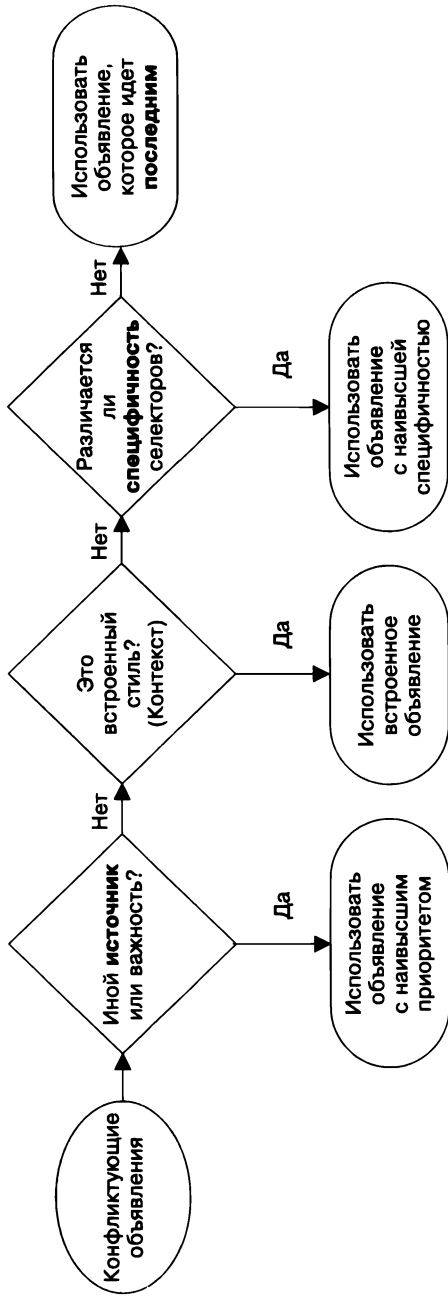


Рис. 1.3. Обобщенная блок-схема каскадности с указанием приоритета объявления

Эти правила определяют поведение браузера при появлении какой-либо неясности с CSS. Давайте разберемся с ними.

Быстрое ознакомление с терминологией

Возможно, вы уже знакомы с основной терминологией, относящейся к синтаксису CSS. Я не буду углубляться в эту тему, но, поскольку стану использовать термины в книге, напомним, что они означают.

Далее приведена строка CSS-кода. Она называется *объявлением*. *Объявление* состоит из *свойства* (*color*) и *значения* (*black*):

```
color: black;
```

Свойства не следует путать с *атрибутами*, которые относятся к синтаксису HTML. Например, в элементе `` часть `href` — это атрибут элемента `a`.

Группа объявлений внутри фигурных скобок называется *блоком объявлений*. Блоку объявлений предшествует *селектор* (в данном случае `body`):

```
body {  
  color: black;  
  font-family: Helvetica;  
}
```

В совокупности селектор и блок объявлений называются *набором правил*. Набор правил также называют *правилом*, хотя, по моему мнению, это слово редко используется в таком смысле. Обычно оно стоит во множественном числе и обозначает набор стилей.

Наконец, *@правила* — это языковые конструкции, начинающиеся с символа `@`, такие как запросы `@import` или `@media`.

1.1.1. Источник стилей

Таблицы стилей, добавляемые вами на вашу веб-страницу, не единственные, к которым обращается браузер. Существуют разные типы, или источники, таблиц стилей. Созданные вами таблицы называются *авторскими*, существуют также *браузерные* стили — стили браузера по умолчанию. Браузерные стили имеют более низкий приоритет, поэтому ваши стили заменяют их.

ПРИМЕЧАНИЕ

Некоторые браузеры позволяют пользователям определять свои стили. Это третий источник стилей, по приоритету они занимают промежуточное положение между браузерными и авторскими стилями. Пользовательские стили применяются редко и не поддаются контролю, поэтому для упрощения я их пропускаю.

Браузерные стили различаются в разных браузерах, но обычно выполняют одни и те же задачи: форматируют заголовки (от h1 до h6) и абзацы (p), задают верхнее и нижнее значения полей, форматируют списки (ol и ul), устанавливают отступ слева, определяют цвет ссылок и размеры шрифта по умолчанию.

Браузерные стили

Снова взглянем на страницу из примера (рис. 1.4). Название выполнено рубленным шрифтом, что определено теми стилями, которые вы добавили. Браузерные стили определяют ряд других параметров: для списка задан левый отступ, для отображения маркеров — свойство `list-style-type` со значением `disc`. Ссылки окрашены в синий цвет и подчеркнуты. У заголовка и списка есть верхние и нижние поля.



Рис. 1.4. Браузерные стили устанавливают форматирование по умолчанию для элементов на веб-странице

После применения браузерных стилей браузер задействует ваши авторские стили. Благодаря этому указанные вами объявления замещают браузерные. Если вы в своем HTML-файле ссылаетесь на несколько таблиц стилей, все они имеют один источник — авторский.

Обычно браузерные стили устанавливают подходящие параметры, поэтому ничего неожиданного не происходит. Если вам не нравится, как они влияют на то или иное свойство, задайте собственное значение в таблице стилей. Сделаем это сейчас. Можете переопределить некоторые браузерные стили, добавляющие не то форматирование, какое вы хотите, чтобы ваша страница выглядела, например, как на рис. 1.5.



Рис. 1.5. Авторские стили переопределяют браузерные, поскольку имеют более высокий приоритет

В листинге 1.3 я удалил конфликтующие объявления `font-family` из предыдущего примера, добавил новые для установки цветов и переопределил браузерные настройки полей, отступа списка и маркеров.

Листинг 1.3. Переопределение браузерных стилей

```
h1 {  
  color: #2f4f4f;  
  margin-bottom: 10px;  
}  
  
#main-nav {  
  margin-top: 10px;  
  list-style: none;  
  padding-left: 0;  
}  
  
#main-nav li {  
  display: inline-block;  
}  
  
#main-nav a {  
  color: white;  
  background-color: #13a4a4;  
  padding: 5px;  
  border-radius: 2px;  
  text-decoration: none;  
}
```

Уменьшение полей

Удаление браузерных стилей

Отображение элементов списка рядом друг с другом, а не один над другим

Оформление навигационных ссылок в виде кнопок

Отредактируйте таблицу стилей, чтобы она соответствовала этим изменениям.

Если вы уже давно работаете с CSS, то, вероятно, уже переопределяли браузерные стили. В этот момент вы пользовались таким принципом каскадности, как определение источника стиля. Ваш стиль всегда будет переопределять браузерный, потому что у них разные источники.

ПРИМЕЧАНИЕ

Возможно, вы заметили, что в данном коде я использовал селекторы идентификатора. Но в работе лучше не делать этого, о чем я вкратце еще расскажу.

Ключевое слово `important`

Объявления, отмеченные как *важные*, стоят особняком при определении источника стилей. Объявление может быть помечено как важное с помощью слова `!important`, указанного в конце, перед точкой с запятой:

```
color: red !important;
```

Объявление со словом `!important` рассматривается как источник с более высоким приоритетом. Далее стили перечислены в порядке убывания приоритета.

1. Важные авторские стили.
2. Авторские стили.
3. Браузерные стили.

Каскадность самостоятельно разрешает конфликты для всех свойств любого элемента на странице. Например, вы установили полужирный шрифт для абзацев текста. Тогда значения верхнего и нижнего полей будут применены согласно браузерным стилям, если вы явно не переопределите их. Приоритеты начинают играть важную роль, когда речь заходит об анимациях и переходах, так как те вводят дополнительные источники стилей. Аннотация `!important` — интересная причуда CSS, к которой мы еще вернемся.

1.1.2. Специфичность селекторов

Если конфликт объявлений не может быть разрешен на основании их источника, браузер попытается решить проблему, рассматривая их *специфичность*. Понять принципы специфичности очень важно. Вы можете долго работать, не вникая в источники стилей, потому что 99 % стилей на вашем сайте происходят из одного источника. Но если вы не знакомы с понятием специфичности, то дальше вам будет трудно. К сожалению, это часто упускают из виду.

Браузер оценивает специфичность в два этапа: сначала проверяет стили, встроенные в HTML-код (их еще называют строчными), затем стили, применяемые с помощью селекторов.

Встроенные стили

Если для задания стилей конкретного элемента в HTML-коде используется атрибут `style`, то объявления применяются только к данному элементу. Это, по сути, объявления с ограниченной областью действия, которые переопределяют любые объявления, задаваемые в вашей таблице стилей или теге `<style>`. Во встроенных стилях нет селектора, потому что они действуют непосредственно на элемент, на который нацелены.

Вы хотите, чтобы ссылка **Акции!** в меню навигации была оранжевой (рис. 1.6)? Рассмотрим несколько способов решения задачи, начиная с применения встроенных стилей (листинг 1.4).



Рис. 1.6. Встроенные стили переопределяют стили, применяемые с помощью селекторов

Листинг 1.4. Встроенные стили, переопределяющие объявления из других источников

```
<li>
  <a href="/specials" class="featured"
    style="background-color: orange;"> ← Встроенный стиль, применяемый
    Акции!                               с помощью атрибута style
  </a>
</li>
```

Чтобы увидеть результат в своем браузере, отредактируйте код в соответствии с приведенным листингом. (Вы отмените это изменение за секунду.)

Чтобы переопределить встроенные объявления стилей, необходимо добавить к нужному объявлению ключевое слово `!important`, изменив его приоритет на более высокий. Если же встроенные стили отмечены как важные, их не получится так переопределить. Рекомендуется сделать это из таблицы стилей. Отмените изменение, и мы рассмотрим более удачные способы.

Специфичность селекторов

Вторая составляющая специфичности касается селекторов. Например, селектор с двумя классами более специфичен, чем с одним. Если одно объявление задает оранжевый фон, а другое, более специфичное, меняет его, скажем, на бирюзовый, то в браузере вы увидите бирюзовый цвет.

Посмотрим, что произойдет, если мы попытаемся окрасить ссылку в оранжевый цвет с помощью простого селектора классов. Обновите заключительную часть таблицы стилей, чтобы она соответствовала коду, приведенному в листинге 1.5.

Листинг 1.5. Селекторы разной специфичности

```
#main-nav a {  
  color: white;  
  background-color: #13a4a4;  
  padding: 5px;  
  border-radius: 2px;  
  text-decoration: none;  
}  
  
.featured {  
  background-color: orange;  
}
```

Более специфичный селектор

Фоновый цвет — бирюзовый

Переопределение на оранжевый цвет не меняет бирюзовый цвет из-за специфичности селектора

Не сработало! Все ссылки остаются бирюзовыми. Почему? Первый селектор здесь специфичнее, чем второй. Он состоит из идентификатора и тега, в то время как второй — из класса. Однако можно сделать еще кое-что вместо того, чтобы учитывать, какой селектор длиннее.

Различные типы селекторов также имеют особенности. К примеру, селектор идентификатора специфичнее селектора класса. На самом деле один идентификатор специфичнее селектора с любым количеством классов. Аналогично селектор класса специфичнее селектора тега, называемого также *селектором типа*.

Итак, существуют следующие правила специфичности.

1. Наиболее специфичным будет селектор с идентификаторами. Чем больше идентификаторов, тем специфичнее будет селектор.
2. Далее идет селектор с наибольшим количеством классов.
3. Следующий по специфичности будет селектор с наибольшим количеством тегов.

Рассмотрим селекторы, показанные в листинге 1.6 (но не добавляйте их в свой код). Они написаны в порядке возрастания специфичности.

Листинг 1.6. Селекторы с возрастающей специфичностью

```

html body header h1 {      ← ❶ Четыре тега
    color: blue;
}

body header.page-header h1 { ← ❷ Два тега и один класс
    color: orange;
}

.page-header .title {      ← ❸ Два класса
    color: green;
}

#page-title {              ← ❹ Один идентификатор
    color: red;
}

```

Наиболее специфичный селектор — ❹, с одним идентификатором, поэтому его объявление красного цвета и применяется к названию. Менее специфичный — селектор ❸, с двумя классами. Он будет работать, если отсутствует селектор идентификатора ❹. Селектор ❸ специфичнее селектора ❷, несмотря на длину: два класса специфичнее одного. Наконец, селектор ❶ наименее специфичный, с четырьмя типами элементов (то есть именами тегов), но без идентификаторов и классов.

ПРИМЕЧАНИЕ

Селекторы псевдоклассов (например, `:hover`) и селекторы атрибутов (например, `[type = "input"]`) специфичны в той же степени, что и селектор класса. Универсальный селектор (`*`) и комбинаторы (`>`, `+`, `~`) не влияют на специфичность.

Если вы добавили в CSS объявление, но оно не вызвало никакого эффекта, то, скорее всего, более специфичное правило переопределяет его. Зачастую разработчики пишут селекторы идентификаторов, не понимая, что создают более высокий уровень специфичности, который позже будет трудно переопределить. Если нужно переопределить стиль, примененный с помощью селектора идентификатора, придется использовать другой идентификатор.

Это концепция проста, но если вы не разбираетесь в специфичности, то можете сойти с ума, пытаясь выяснить, почему одно работает, а другое — нет.

Замечание по поводу специфичности

Обычно специфичность выражается в числовой форме, чаще через запятую. Например, значение «1,2,2» указывает на специфичность одного идентификатора, двух классов и двух тегов. Сначала перечисляются идентификаторы с наивысшим приоритетом, за которыми следуют классы и далее теги.

Селектор `#page-header #page-title` содержит два идентификатора без классов и тегов. Мы можем представить эту специфичность в виде «2,0,0». Селектор `ul li` с двумя тегами, но без идентификаторов и классов имеет специфичность «0,0,2». В табл. 1.1 показаны селекторы из листинга 1.6.

Таблица 1.1. Различные селекторы и их особенности

Селектор	Идентификаторы	Классы	Теги	Нотация
html body header h1	0	0	4	0,0,4
body header.page-header h1	0	1	3	0,1,3
.page-header .title	0	2	0	0,2,0
#page-title	1	0	0	1,0,0

Теперь возникает вопрос: как сравнить числа, чтобы определить, какой селектор специфичнее? Специфичность «1,0,0» имеет приоритет над специфичностью «0,2,2» и даже «0,10,0», потому что первое число (идентификатор) имеет более высокий приоритет.

Иногда для записи используют четыре числа, добавляя цифру 0 или 1 в начале значения и указывая таким образом наличие встроенных стилей. В этом случае строчный стиль имеет специфичность «1,0,0,0». Он будет переопределять стили со специфичностью «0,1,2,0» и т. п.

Специфичность на практике

Попытка установить оранжевый фон с помощью селектора `.featured` успехом не увенчалась. Селектор `#main-nav` содержит идентификатор, который переопределяет селектор классов (специфичность «1,0,1» и «0,1,0»). Исправить это можно разными способами. Рассмотрим несколько вероятных изменений.

Самый быстрый способ решения проблемы заключается в том, чтобы добавить ключевое слово `!important` к объявлению, которое вы хотите применить. Измените код объявления следующим образом (листинг 1.7).

Листинг 1.7. Первое возможное решение проблемы

```
#main-nav a {
  color: white;
  background-color: #13a4a4;
  padding: 5px;
  border-radius: 2px;
  text-decoration: none;
}

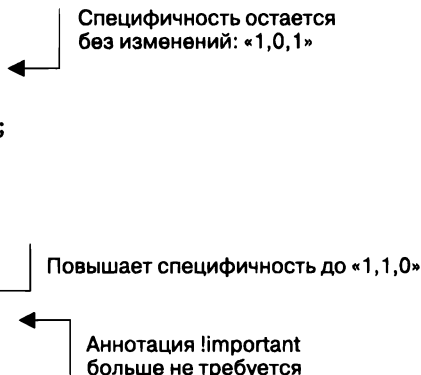
.featured {
  background-color: orange !important;
}
```

Объявление становится важным; теперь оно имеет более высокий приоритет

Прием работает, потому что аннотация `!important` придает объявлению более высокий приоритет. Способ простой, но наивный. Так поступить можно, но в будущем вероятны проблемы. Если вы добавите ключевое слово `!important` к нескольким объявлениям, что произойдет, если понадобится задать более высокий приоритет стилям, уже имеющим аннотацию `!important`? Если вы добавите несколько объявлений `!important`, то будут применяться обычные правила определения источника и специфичности. В конечном счете вы вернетесь в исходную позицию, только с аннотацией `!important` в множестве объявлений.

Определим способ получше. Вместо того чтобы пытаться обойти правила специфичности селектора, направим их работу в нужное русло. Как вы смотрите на то, чтобы повысить специфичность селектора? Обновите код правил в своей каскадной таблице стилей следующим образом (листинг 1.8).

Листинг 1.8. Второе возможное решение проблемы



```
#main-nav a {
  color: white;
  background-color: #13a4a4;
  padding: 5px;
  border-radius: 2px;
  text-decoration: none;
}

#main-nav .featured {
  background-color: orange;
}
```

Этот способ тоже работает. Теперь у вашего селектора есть один идентификатор и один класс, формирующие специфичность «1,1,0», что выше, чем `#main-nav` (специфичность «1,0,1»), поэтому оранжевый цвет фона применяется к элементу.

Тем не менее есть способ сделать еще лучше. Посмотрим, реально ли вместо *повышения* специфичности второго селектора *понижить* специфичность первого. У элемента также есть класс `<ul id="main-nav" class="nav">`, поэтому можно изменить CSS-код, чтобы нацелиться на элемент с помощью его имени класса, а не идентификатора. Измените строку `#main-nav` на `.nav` в своих селекторах, как показано в листинге 1.9.

Специфичность селекторов понижена. Специфичность оранжевого фона достаточно высока для того, чтобы переопределить бирюзовый цвет.

Как видно из этих примеров, специфичность нередко становится своего рода «гонкой вооружений». Это особенно актуально для крупных проектов. Обычно лучше всего сохранять более низкую специфичность, если возможно, и тогда, если нужно будет переопределить что-то, вы сможете решить проблему несколькими способами.

Листинг 1.9. Третье возможное решение проблемы

```
.nav {  
  margin-top: 10px;  
  list-style: none;  
  padding-left: 0;  
}  
  
.nav li {  
  display: inline-block;  
}  
  
.nav a {  
  color: white;  
  background-color: #13a4a4;  
  padding: 5px;  
  border-radius: 2px;  
  text-decoration: none;  
}  
  
.nav .featured {  
  background-color: orange;  
}
```

Код #main-nav меняется на .nav по всей таблице стилей

Понижаем первую специфичность («0,1,1»)

Повышаем вторую специфичность («0,2,0»)

1.1.3. Исходный порядок

Третий и последний показатель при определении каскадности — исходный порядок. Если источник и уровень специфичности одинаковы, то объявление, которое указано позже в таблице стилей или находится в таблице стилей, на которую ссылаются на странице позже, имеет больший приоритет.

Это означает, что вы можете менять исходный порядок, чтобы форматировать ссылку **Акции!**. Если добавите два конфликтующих селектора одинаковой специфичности, победит последний. Рассмотрим четвертый вариант решения проблемы (листинг 1.10).

Листинг 1.10. Четвертый способ решения проблемы

```
.nav a {  
  color: white;  
  background-color: #13a4a4;  
  padding: 5px;  
  border-radius: 2px;  
  text-decoration: none;  
}  
  
a.featured {  
  background-color: orange;  
}
```

Устанавливает специфичность равной «0,1,1»

В этом коде специфичности идентичны. Порядок определяет, какое объявление применяется к ссылке **Акции!**, в результате отображается оранжевый цвет.

Это решение проблемы может породить новую сложность: хотя код кнопки **Акции!** внутри объявления `nav` выглядит корректным, что произойдет, если вы захотите применить класс `featured` к другой ссылке в другой позиции страницы, вне `nav`? Получите странное сочетание стилей: белый текст на оранжевом фоне, увеличенные отступы и закругленные углы в оформлении ссылок на панели навигации (рис. 1.7).

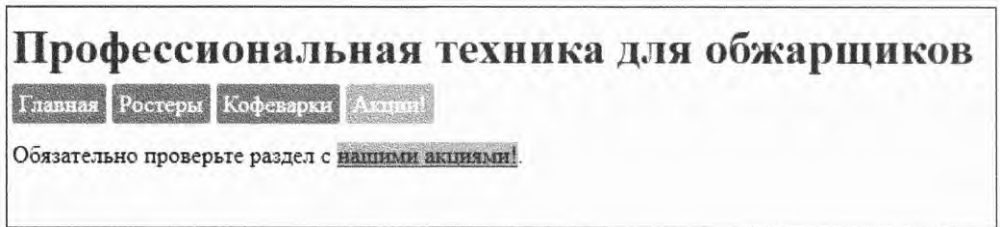


Рис. 1.7. Класс `featured` вне объявления `nav` дает неправильный результат

В листинге 1.11 показана разметка, создающая такое форматирование. Теперь на элемент нацелен только второй селектор, но не первый, что приводит к нежелательному результату. Нужно решить, хотите ли вы, чтобы этот оранжевый стиль использовался за пределами `nav`, и если да — убедиться, что к нему применяются все нужные стили.

Листинг 1.11. Ссылка `featured` за пределами `nav`

```
<header class="page-header">
  <h1 id="page-title" class="title">Профессиональная
    техника для обжарщиков</h1>
  <nav>
    <ul id="main-nav" class="nav">
      <li><a href="/">Главная</a></li>
      <li><a href="/coffees">Ростеры</a></li>
      <li><a href="/brewers">Кофеварки</a></li>
      <li><a href="/specials" class="featured">Акции!</a></li>
    </ul>
  </nav>
</header>
<main>
  <p>
    Обязательно проверьте раздел с
    <a href="/specials" class="featured">нашими акциями!</a>.
  </p>
</main>
```

Ссылка `featured`
за пределами `nav`
будет отформатирована
частично

На сайте, схожем с показанным в примере, я бы придерживался третьего варианта решения проблемы (см. листинг 1.9). Разрабатывая сайт, нужно продумывать и другие (в иных позициях сайта) элементы и их форматирование. Вероятно, вам следует поместить аналогичную ссылку в иных разделах сайта. В этом случае, думаю, как нельзя лучше подойдет четвертый вариант (см. листинг 1.10) с добавлением стилей для поддержки класса `featured` в других местах страницы.

Очень часто в CSS, как я уже говорил, все «зависит от обстоятельств». Существует много путей, ведущих к одному и тому же конечному результату. Рекомендуется рассмотреть несколько вариантов и продумать последствия каждого из них. При возникновении проблемы форматирования я часто решаю ее в два этапа: во-первых, выясняю, какие объявления будут применяться корректно, во-вторых, думаю о возможных способах структурирования селекторов и выбираю тот, который наилучшим образом решает мои задачи.

Форматирование ссылок и исходный порядок

Когда вы начали изучать каскадные таблицы стилей, то узнали, что селекторы для форматирования ссылок следует указывать в определенном порядке. Ведь порядок влияет на каскадность. В листинге 1.12 показаны стили ссылок на странице в правильном порядке.

Листинг 1.12. Стили ссылок

```
a:link {
    color: blue;
    text-decoration: none;
}

a:visited {
    color: purple;
}

a:hover {
    text-decoration: underline;
}

a:active {
    color: red;
}
```

Каскадность — причина того, что этот порядок имеет значение: стили, указанные позже, переопределяют ранние стили с той же специфичностью. Если два подобных стиля или более одновременно относятся к одному и тому же элементу, последний переопределяет предыдущие. Если пользователь наводит указатель мыши на посещенную ссылку, стиль наведения получает приоритет над стилем посещенной ссылки. Если пользователь активизирует ссылку, то есть щелкает на

ней кнопкой мыши, стиль активной ссылки становится приоритетным над стилем наведения.

Запомните порядок применения стилей ссылок: *непосещенная, посещенная, наведение, активная*. Обратите внимание: если вы измените один из селекторов для коррекции специфичности, это приведет к сбою и вы получите неожиданные результаты.

Каскадные значения

Браузер проверяет три параметра: *источник, специфичность и порядок* — для каждого свойства, применяемого к любому элементу на странице. Объявление, которое «побеждает» каскадность, называется *каскадным значением*. На каждое свойство любого элемента приходится не более одного каскадного значения. Определенный абзац (p) на странице может иметь верхнее и нижнее поля, но не два разных верхних или нижних поля. Если CSS задает различные значения для одного свойства, то при визуализации элемента каскадность выберет только одно. Это каскадное значение.



Каскадное значение — значение определенного свойства, применяемого к элементу в результате каскадности.

Если свойство для элемента не указано, то он не имеет каскадного значения для этого свойства. Например, тот же абзац не имеет границ или отступов.

1.1.4. Два правила

Существует два распространенных правила работы с каскадностью.

1. *Не используйте идентификаторы в селекторе.* Даже один идентификатор усиливает специфичность. Когда требуется переопределить селектор, может не оказаться другого идентификатора, который реально задействовать, поэтому вам придется скопировать исходный селектор и добавить другой класс, чтобы отличить его от того, который вы пытаетесь переопределить.
2. *Не используйте аннотацию !important.* Ее еще труднее переопределить, чем идентификатор, и, как только вы ее примените, нужно будет добавлять ее каждый раз, когда захотите переопределить исходное объявление, и тогда все равно придется иметь дело со специфичностью.

Эти два правила хороши, но годятся не на все случаи жизни. Никогда (за некоторыми исключениями) не используйте их, если нужно разобраться с проблемами специфичности.

Главное о главном

Я настоятельно рекомендую: создавая модуль JavaScript для последующего распространения (например, пакет NPM), не задействуйте стили, встроенные в JavaScript, когда этого можно избежать. Если вы это сделаете, разработчики, применяющие ваш пакет, будут вынуждены либо принять ваши стили без изменений, либо использовать аннотацию `!important` для каждого свойства, которое они захотят изменить.

Вместо этого добавьте в пакет таблицу стилей. Если ваш компонент должен динамически менять стили, почти всегда предпочтительнее задействовать JavaScript для добавления и удаления классов элементов. Кроме того, пользователи смогут отредактировать таблицу стилей как им нравится, не сталкиваясь с конфликтами специфичности.

За последние несколько лет было разработано немало практических методик, призванных помочь в управлении специфичностью селекторов. Мы подробно рассмотрим их в главе 9. Там я больше расскажу о работе со специфичностью, в том числе о том, когда приемлемо применять ключевое слово `!important`.

Теперь, когда мы разобрались с каскадностью, перейдем к наследованию.

1.2. Наследование

Существует еще один принцип, по которому к элементу применяются стили, — *наследование*. Каскадность часто упоминается наряду с наследованием. Хотя эти две темы взаимосвязаны, вы должны разбираться в каждой из них по отдельности.

Если элемент не имеет каскадного значения для данного свойства, он может наследовать его от родительского элемента (предка). Рядовой пример — применение свойства `font-family` к элементу `body`. Все дочерние элементы (потомки) внутри будут наследовать этот шрифт, и вам не нужно явно применять его к каждому элементу на странице. На рис. 1.8 показано, как наследование распространяется по дереву DOM.

Однако не все свойства наследуются, а по умолчанию — только определенные. В общем, это свойства, которые вы *хотите* унаследовать. Прежде всего относящиеся к шрифтам: `color`, `font`, `font-family`, `font-size`, `font-weight`, `font-variant`, `font-style`, `line-height`, `letter-spacing`, `text-align`, `text-indent`, `text-transform`, `white-space` и `word-spacing`.

Наследуются и некоторые другие, такие как свойства списков: `list-style`, `list-style-type`, `list-style-position` и `list-style-image`. Свойства границ таблицы `border-collapse` и `border-spacing` также наследуются. Обратите внимание на то, что они управляют поведением границ таблиц и отличаются от часто применяемых свойств для форматирования границ нетабличных элементов. (Нам же не нужно,

чтобы элемент `div` передавал свои стили границ каждому дочернему элементу.) Вот практически полный список.

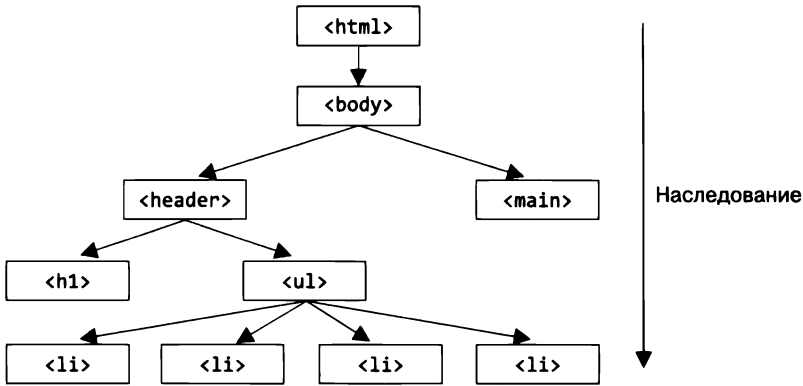


Рис. 1.8. Наследуемые свойства передаются по дереву DOM от предка к потомкам

Вы можете воспользоваться наследованием на своей странице, применив шрифт к элементу `body` и позволяя его дочерним элементам наследовать это значение (рис. 1.9).



Рис. 1.9. Добавьте свойство `font-family` к `body`, и пусть все дочерние элементы наследуют одно и то же значение

Добавьте этот код в начало таблицы стилей, чтобы применить правило к своей странице (листинг 1.13).

Листинг 1.13. Применение свойства `font-family` к родительскому элементу

```
body {
  font-family: sans-serif;
}
```

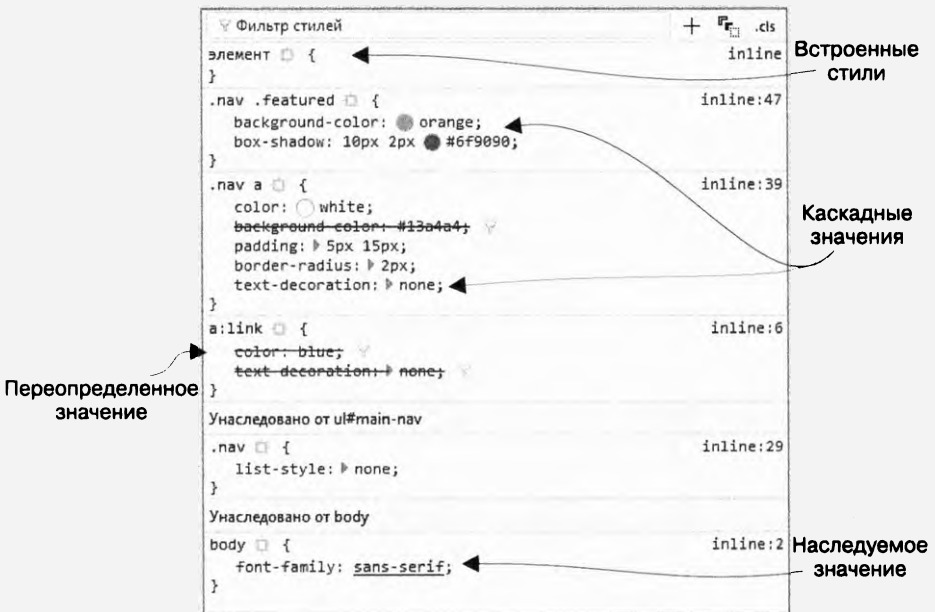
← Наследуемое свойство
будет применено
и к дочерним элементам

Стиль используется на всей странице, так как назначен всему ее телу. Но вы можете настроить определенный элемент на странице, и данный стиль будут наследовать только его потомки. Наследование станет применяться от элемента к элементу, пока не будет переопределено каскадным значением.

Инструменты разработчика

Уследить за наследуемыми значениями, переопределяющими друг друга, может быть очень сложно. Если вы еще не знакомы с инструментами разработчика своего браузера, научитесь их использовать.

Инструменты разработчика наглядно демонстрируют, какие правила к каким элементам применяются и почему. Каскадность и наследование — абстрактные понятия, инструменты разработчика — лучший из известных мне способов разобраться в них. Откройте панель инструментов разработчика, щелкнув правой кнопкой мыши на элементе на странице, и выберите в контекстном меню команду с названием наподобие **Исследовать элемент** (Inspect element). Далее представлен пример панели инструментов разработчика.



Инспектор стилей показывает все ориентированные на выбранный элемент селекторы, упорядоченные согласно специфичности. Внизу перечислены все наследуемые свойства. Так наглядно показаны каскадность и наследование выбранного элемента.

Есть много неявных признаков, которые помогут вам понять, что происходит со стилями элемента. Стили, расположенные ближе к верхней части панели, переопределяют следующие за ними. Переопределенные (замещенные) стили зачеркнуты. Имя файла таблицы стилей и номер строки в ней для каждого набора правил показаны справа, чтобы их легко было найти в исходном коде. Так вы узнаете, какой именно элемент какие стили унаследовал и где они расположены. Можете также ввести запрос в поле **Фильтр стилей**, чтобы скрыть все, кроме определенного набора объявлений.

1.3. Специальные значения

Существует два специальных значения, которые можно применять к любому свойству, чтобы управлять каскадностью: `inherit` и `initial`. Рассмотрим их.

1.3.1. Ключевое слово `inherit`

Иногда необходимо унаследовать стиль, когда каскадное значение замещает его. В этом случае используйте ключевое слово `inherit`. Замените одно значение другим, чтобы элемент унаследовал его от своего родителя.

Предположим, на страницу добавлен светло-серый подвал (*footer*, иногда еще называют колонтитулом). В нем могут быть размещены какие-то ссылки, но вы не хотите, чтобы они сильно выделялись, потому что подвал не слишком важная часть страницы. Таким образом, вы делаете ссылки в нем темно-серыми (рис. 1.10).

© 2019 Профессиональная техника для обжарщиков — Условия эксплуатации

Рис. 1.10. Ссылка Условия эксплуатации с унаследованным серым цветом шрифта

Добавьте приведенную далее разметку в конец страницы. На обычной странице будет больше контента между шапкой и подвалом, но для нашего примера этого достаточно.

Листинг 1.14. Подвал со ссылкой

```
<footer class="footer">
  &copy; 2019 Профессиональная техника для обжарщиков &mdash;
  <a href="/terms-of-use">Условия эксплуатации</a>
</footer>
```

По умолчанию шрифт ссылок будет того цвета, который установлен для всех ссылок на странице (если вы этого не сделаете, его определяют браузерные стили), в том числе ссылки *Условия эксплуатации*. Чтобы окрасить ссылку в подвале в серый цвет, нужно переопределить этот ее стиль. Добавьте в таблицу стилей следующий код (листинг 1.15).

Третий набор правил переопределяет синий цвет ссылки, передавая ссылке, расположенной в подвале, каскадное значение `inherit`. Таким образом, она наследует цвет от родительского элемента `footer`.

Преимущество здесь в том, что ссылка будет менять цвет вместе с остальной частью подвала, если такие изменения будут внесены. (На это способно повлиять изменение второго набора правил или указание нового стиля в другой позиции.) Если, например, текст в подвале на других страницах будет более темным, то и ссылка окрасится в тот же цвет.

Листинг 1.15. Значение inherit

```
a:link {  
    color: blue;  
    text-decoration: none;  
}  
  
...  
  
.footer {  
    color: #666;  
    background-color: #ccc;  
    padding: 15px 0;  
    text-align: center;  
    font-size: 14px;  
}  
  
.footer a {  
    color: inherit;  
    text-decoration: underline;  
}
```

Глобальное значение цвета
для всех ссылок на странице

Цвет шрифта текста
в подвале — серый

Наследует цвет шрифта
от элемента footer

Вы можете использовать ключевое слово `inherit` и для принудительного наследования свойства, которое обычно не наследуется, например параметров границ или отступов. Этот прием применяется на практике в нескольких случаях, и вы увидите один из них в главе 3, когда мы займемся блочной моделью.

1.3.2. Ключевое слово initial

Иногда к элементу применяются стили, которые хотелось бы отменить. Это делается при добавлении ключевого слова `initial`. Каждое свойство CSS имеет *начальное*, или применяемое по умолчанию, значение. Если вы присваиваете свойству значение `initial`, оно сбрасывается до начального. Как будет выглядеть подвал, если вы присвоите значение `initial`, а не `inherit`, показано на рис. 1.11.

© 2019 Профессиональная техника для обжарщиков — [Условия эксплуатации](#)

Рис. 1.11. Начальное значение свойства `color` — `black`

ВНИМАНИЕ!

Ключевое слово `initial` поддерживается не во всех версиях браузеров Internet Explorer или Opera Mini. Во всех других браузерах, включая Edge, созданный корпорацией Microsoft как преемник Internet Explorer 11, оно работает.

CSS-код для подвала на рис. 1.11 показан в листинге 1.16. Поскольку черный цвет — начальное значение свойства `color` в большинстве браузеров, `color: initial` — это эквивалент правила `color: black`.

Листинг 1.16. Значение `initial`

```
.footer a {  
  color: initial;  
  text-decoration: underline;  
}
```

Его преимущество в том, что вам не нужно задумываться об указании каких-то специальных значений в коде. Если вы хотите удалить заданную границу элемента, установите `border: initial`. Если нужно восстановить исходную ширину элемента, используйте код `width: initial`.

Возможно, у вас осталась привычка устанавливать значение `auto` для сброса. Фактически для достижения того же результата применяется правило `width: auto`. Это связано с тем, что ширина по умолчанию — `auto`.

Однако важно отметить, что `auto` — значение по умолчанию не для всех свойств. Например, строки `border-width: auto` и `padding: auto` недопустимы и, следовательно, не действуют. Вы могли бы потратить некоторое время, чтобы определить начальные значения этих свойств, но обычно проще задействовать ключевое слово `initial`.

ПРИМЕЧАНИЕ

Объявление `display: initial` эквивалентно коду `display: inline`. Оно не будет определено как `display: block` независимо от того, к элементу какого типа вы его применяете. Это потому, что ключевое слово `initial` выполняет сброс до начального значения для свойства, а не элемента, а `inline` — значение по умолчанию для свойства `display`.

1.4. Сокращенная запись свойств

Сокращенная запись свойств — это присвоение значений нескольких свойств одновременно. Например, `font` — это сокращенное свойство, позволяющее задать несколько свойств шрифта. Свойства `font-style`, `font-weight`, `font-size`, `lineheight` и `font-family` определяет следующее объявление:

```
font: italic bold 18px/1.2 "Helvetica", "Arial", sans-serif;
```

Аналогично:

- ❑ `background` — сокращенная запись свойств фона `background-color`, `background-image`, `background-size`, `background-repeat`, `background-position`, `background-origin`, `background-chip` и `background-attachment`;
- ❑ `border` — сокращенная запись свойств `border-width`, `border-style` и `border-color`;
- ❑ `border-width` — сокращенная запись свойств `top`, `right`, `bottom` и `left` ширины границ.

Сокращенные записи свойств полезны, они делают код кратким и понятным, но некоторые подводные камни здесь все-таки есть.

1.4.1. Остерегайтесь сокращений, скрыто переопределяющих другие стили

Большинство сокращенных записей свойств позволяют опускать определенные значения и указывать только нужные. Однако важно знать, что пропущенные свойства все равно задаются — им присваиваются начальные значения. Это может скрыто отменять стили, которые вы указали в других правилах. Если, например, нужно использовать запись `font` для названия страницы без указания свойства `font-weight`, значение `normal` для ширины шрифта все равно будет применено (рис. 1.12).

Профессиональная техника для обжарщиков

Рис. 1.12. Сокращенные записи свойств сбрасывают пропущенные свойства до их начальных значений

Добавьте код из листинга 1.17 в таблицу стилей, чтобы посмотреть, как это происходит.

Листинг 1.17. Сокращенная запись свойства, определяющая все соответствующие значения

```
h1 {
  color: #2f4f4f;
  margin-bottom: 10px;
  font-weight: bold;
}

.title {
  font: 32px Helvetica, Arial, sans-serif;
}
```

На первый взгляд может показаться, что код `<h1 class = "title">` сделает шрифт заголовка полужирным, но это не так. Примененные стили эквивалентны стилям из следующего кода (листинг 1.18).

Листинг 1.18. Полный эквивалент сокращенной записи свойства из листинга 1.17

```
h1 {
  color: #2f4f4f;
  margin-bottom: 10px;
  font-weight: bold;
}

.title {
  font-style: normal;
  font-variant: normal;
  font-weight: normal;
  font-stretch: normal;
  line-height: normal;
  font-size: 32px;
  font-family: Helvetica, Arial, sans-serif;
}
```

Начальные
значения свойств

Это означает, что в результате применения данных стилей к элементу **h1** шрифт отображается в обычном, а не в полужирном начертании. Запись способна переопределять и другие стили шрифта, которые в противном случае были бы унаследованы от родительского элемента. Из всех сокращенных записей свойств **font** чаще всего вызывает проблемы, поскольку определяет очень широкий набор параметров. По этой причине я избегаю использовать указанное свойство во всех случаях, кроме установки общих стилей в теле страницы. Вы можете столкнуться с данной проблемой, применяя и другие сокращенные записи свойств, поэтому учитывайте эту особенность.

1.4.2. Порядок записи сокращенных значений

Сокращенные записи свойств могут быть гибкими, когда речь заходит о порядке указания вами значений. Вы можете использовать код **border: 1px solid black** или **border: black 1px solid**, в обоих случаях результат будет идентичен. Так происходит потому, что браузеру ясно, какое значение определяет ширину, какое — цвет, а какое — стиль границы.

Но есть много свойств, когда значения могут трактоваться неоднозначно. В этих случаях порядок указания значений важен. Следует соблюдать его при вводе сокращенных записей свойств.

Вверху, справа, внизу, слева

Сокращенные записи свойств вынуждают разработчиков запоминать порядок перечисления значений, когда дело касается свойств **margin** и **padding** или некоторых свойств границ, которые определяют значения для каждой из четырех сторон элемента. Для них значения записываются по часовой стрелке, начиная сверху — **top**.

Порядок необходимо запомнить, чтобы избежать потенциальных проблем. Навигационные ссылки, показанные на рис. 1.13, имеют следующие отступы: верхний — 10 пикселей, справа — 15 пикселей, снизу — 0, слева — 5 пикселей. Выглядит некрасиво и неровно, но зато демонстрирует принцип работы сокращенной записи.



Рис. 1.13. Элементы с различными отступами со всех сторон

В листинге 1.19 показан CSS-код для этих ссылок.

Листинг 1.19. Указание отступов со всех сторон элемента

```
.nav a {
  color: white;
  background-color: #13a4a4;
  padding: 10px 15px 0 5px;
  border-radius: 2px;
  text-decoration: none;
}
```

Отступы сверху, справа,
снизу и слева

Свойства, значения которых указаны в этом порядке, также поддерживают сокращенную запись. Если объявление закончилось прежде, чем одной из четырех сторон было присвоено значение, сторона без значения использует указанное для противоположной стороны. Задайте три значения, и левая и правая стороны будут соответствовать второму значению. Укажите два значения, и верхняя и нижняя стороны будут соответствовать первому значению, а левая и правая — второму. Если вы укажете только одно значение, оно будет применяться ко всем четырём сторонам. Таким образом, следующие объявления идентичны:

```
padding: 1em 2em;  
padding: 1em 2em 1em;  
padding: 1em 2em 1em 2em;
```

Эти объявления также идентичны друг другу:

```
padding: 1em;  
padding: 1em 1em;  
padding: 1em 1em 1em;  
padding: 1em 1em 1em 1em;
```

Многие разработчики испытывают наибольшие затруднения, когда нужно задать три значения. В этом случае указываются верхняя, правая и нижняя стороны. Поскольку для левой значение не указано, оно будет таким же, как и для правой, — второе значение будет применено к обоим сторонам. Таким образом, правило `padding: 10px 15px 0` создает отступы по 15 пикселей как с левой, так и с правой стороны, тогда как верхний отступ равен 10 пикселям, а нижний — 0.

Однако чаще указывают два значения. Небольшие элементы лучше выглядят, когда слева и справа отступы больше, чем сверху и снизу. Этот подход идеален для форматирования кнопок или навигационных ссылок на странице (рис. 1.14).

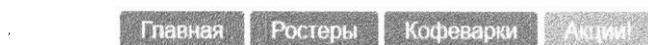


Рис. 1.14. Многие элементы выглядят лучше с большим отступом по горизонтали

Измените код таблицы стилей, чтобы он соответствовал листингу 1.20. Здесь используются сокращенные записи свойств, чтобы сначала применить вертикальный отступ, а затем горизонтальный.

Листинг 1.20. Указание двух значений

```
.nav a {  
  color: white;  
  background-color: #13a4a4;  
  padding: 5px 15px;  
  border-radius: 2px;  
  text-decoration: none;  
}
```

← Сначала отступы сверху и снизу,
а затем слева и справа

Существует много общих свойств, которые придерживаются этого порядка, поэтому рекомендуется запомнить его.

Горизонтально, вертикально

Мы рассмотрели свойства, которые назначают отдельные значения каждой из четырех сторон. Но существуют и свойства, поддерживающие только два значения. К ним относятся `background-position`, `box-shadow` и `text-shadow` (строго говоря, это несокращенные записи). По сравнению с четырьмя значениями таких свойств, как `padding`, порядок указания значений у двузначных свойств противоположен. Если `padding: 1em 2em` сначала определяет вертикальные отступы (сверху/снизу), а затем горизонтальные (справа/слева), то свойство `background-position: 25% 75%` определяет сначала горизонтальные значения (справа/слева) и только потом вертикальные (сверху/снизу).

Эта противоположность кажется нелогичной, но ее причина проста: два значения представляют собой прямоугольную систему координат. Значения в ней обычно задаются в порядке *x, y* (горизонтальное значение, а затем вертикальное). Если, например, вы хотите применить тень, подобную показанной на рис. 1.15, сначала укажите значение *x* (горизонтальное).

CSS-код показан в листинге 1.21.



Рис. 1.15. Свойству `box-shadow` присвоено значение `10px 2px`

Листинг 1.21. Свойство `box-shadow` определяет сначала значение *x*, а затем *y*

```
.nav .featured {
  background-color: orange;
  box-shadow: 10px 2px #6f9090;
}
```

← Смещение тени на 10 пикселей вправо и на 2 пикселя вниз

Первое (большее) значение определяет смещение по горизонтали, а второе (меньшее) — по вертикали.

Если вы работаете со свойством, которое принимает два значения, *x* и *y*, вспомните прямоугольную систему координат, если же со свойством, определяющим значения для *всех* сторон элемента, — часы.

Итоги главы

- ❑ Контролируйте специфичность селекторов.
- ❑ Не путайте каскадность с наследованием.
- ❑ Некоторые свойства наследуются в том числе для текста, списков и границ таблиц.
- ❑ Не путайте значения `initial` и `auto`.
- ❑ Остерегайтесь сокращений, скрыто переопределяющих другие стили.
- ❑ Представляйте прямоугольную систему координат, работая с двузначными свойствами, и часы, используя четырехзначные.

2

Работа с относительными единицами

В этой главе

- Гибкость относительных единиц.
- Как использовать единицы `em` и `rem` и не сойти с ума.
- Применение экранных единиц измерения.
- Введение в переменные CSS.

При указании значений CSS предоставляет широкий спектр возможностей для выбора единиц измерения. Одними из самых известных и, вероятно, самых простых для работы являются пикселы. Это *абсолютные единицы* — 5 пикселей всегда означают одно и то же. Другие единицы, такие как `em` и `rem`, не абсолютные, а *относительные*. Значение относительных единиц меняется под влиянием внешних факторов: размер 2 `em` меняется в зависимости от того, для какого элемента (а иногда даже свойства) вы его задаете. Естественно, это усложняет работу с относительными единицами.

Разработчикам CSS, даже опытным, не очень нравится работать с относительными единицами, включая пресловутую `em`. Значение `em` может меняться абсолютно по-разному, что делает ее непредсказуемой и менее четкой, чем пиксел. В данной главе я сниму завесу тайны вокруг относительных единиц. Сначала объясню, в чем их уникальная польза для CSS, затем помогу в них разобраться. Я расскажу, как они работают, и покажу, как укротить их сущность, на первый взгляд кажущуюся непредсказуемой. Вполне реально сделать так, чтобы относительные единицы работали на вас. Если правильно их применять, ваш код станет более простым и универсальным, а работа над ним упростится.

2.1. Мощь относительных значений

CSS позволяет *динамически связывать* стили с веб-страницей: контент и его стили не объединяются, пока их разработка не завершена. В процессе веб-дизайна это создает дополнительные проблемы, которых нет в других видах графического дизайна, но в то же время обеспечивает очень широкие возможности — одну таблицу стилей реально применить к сотням и даже тысячам страниц. Более того, пользователь может

напрямую изменить итоговую отрисовку страницы. Например, задать другой размер шрифта по умолчанию или размер окна браузера.

В ранних версиях компьютерных приложений разработчики знали точные ограничения своего носителя. Размер отдельного программного окна составлял 400 пикселей в ширину и 300 пикселей в высоту. Соответственно, когда разработчики приступали к разметке кнопок и текста приложения, они точно знали, насколько большими могут сделать эти элементы и сколько места останется у них для работы над другими объектами на странице. В Интернете дело обстоит иначе.

2.1.1. Борьба за pixel-perfect-дизайн

В интернет-среде окна браузера у пользователей могут быть любого размера, и CSS должен адаптироваться под них. Более того, пользователи способны изменить размер страницы после того, как ее откроют, и CSS должен подстроиться под новые ограничения. Это означает, что браузер должен рассчитывать стили в момент отображения страницы на экране.

Это повышает уровень абстрактности CSS. Мы не можем стилизовать элемент в соответствии с идеальным контекстом, нам нужны определенные правила, работающие в любом контексте, в котором способен оказаться этот элемент. В сегодняшнем Интернете страница должна отображаться на четырехдюймовом экране смартфона так же, как и на 30-дюймовом мониторе.

Дизайнеры долго смягчали эту сложность, концентрируясь на дизайне pixel-perfect. Они создавали строго ограниченный контейнер, зачастую это была выровненная по центру колонка около 800 пикселей шириной. Затем в рамках этих ограничений занимались дизайном приблизительно так же, как их предшественники — компоновкой ранних приложений.

2.1.2. Конец эпохи pixel-perfect

С развитием технологий и выпуском в производство мониторов более высокого разрешения подход pixel-perfect начал медленно терять актуальность. В начале 2000-х годов было много дискуссий о том, можем ли мы, разработчики, безопасно создавать дизайн для дисплеев шириной 1024 пикселей вместо 800. Затем тот же разговор велся о 1280 пикселях. Нам приходилось принимать спорные решения. Что было лучше: сделать сайт слишком широким под старые компьютеры или слишком узким под новые?

Когда появились смартфоны, разработчикам пришлось перестать делать вид, что можно добиться единого отображения их сайтов для каждого пользователя. Нравилось ли нам это, или мы это ненавидели, но пришлось отказаться от колонок фиксированной ширины и начать думать об *адаптивном* дизайне. Мы больше не могли избежать абстрактности, которая сопровождала CSS. Необходимо было ее принять.



Адаптивный — в CSS это определение относится к стилям, реакция которых зависит от размера окна браузера. Из-за этого необходимо принимать во внимание то, что экраны мобильных телефонов, планшетных или настольных компьютеров могут быть любого размера. Адаптивный дизайн подробно рассматривается в главе 8.

Введенная абстрактность означает дополнительную сложность. Если я задаю элементу ширину 800 пикселей, как он будет выглядеть в окне меньшего размера? Как будет выглядеть горизонтальное меню, если не уместится в одну строчку? Когда вы пишете CSS-код, вам необходимо уметь мыслить одновременно как в деталях, так и в общем. Когда есть множество способов решить конкретную проблему, следует отдать предпочтение тому, который будет работать в большинстве случаев в разнообразных обстоятельствах.

Относительные единицы — одно из средств CSS, обеспечивающих работу на данном уровне абстрактности. Вместо того чтобы устанавливать размер шрифта 14 пикселей, вы можете задать его в масштабе, пропорциональном размеру окна. Или установить размер всего, что есть на странице, относительно основного размера шрифта и затем менять размер всей страницы одной строкой кода. Рассмотрим, что CSS предлагает для реализации такого подхода.

Пиксели, пункты, пики

Каскадные таблицы стилей поддерживают некоторые абсолютные единицы измерения. Наиболее распространенные из них и основные — это пиксели (px). Менее распространенные абсолютные единицы: миллиметры (mm), сантиметры (cm), дюймы (in), пункты (pt, типографский термин для обозначения значения $1/72$ дюйма) и пики (pc, типографский термин для обозначения 12 пунктов). Любая из этих единиц может быть напрямую переведена в другую: 1 дюйм = 25,4 мм = 2,54 см = 6 пик = 72 пункта = 96 пикселей. Отсюда 16 пикселей — то же самое, что 12 пунктов ($16 / 96 \cdot 72$). Дизайнеры обычно используют пункты, тогда как разработчикам более привычны пиксели, поэтому вам, вероятно, придется делать перевод между этими значениями при общении с дизайнером.

Пиксел — понятие, немного вводящее в заблуждение: пиксел в CSS не совсем равен пикселу монитора. Это особенно характерно для дисплеев с высоким разрешением (retina). Несмотря на то что измерения в CSS могут быть в некоторой степени масштабированы в зависимости от браузера, операционной системы и аппаратного обеспечения, 96 пикселей обычно составляют приблизительно 1 физический дюйм экрана, хотя это значение может варьироваться на определенных устройствах или в зависимости от пользовательских настроек разрешения.

2.2. Единицы em и rem

Самая распространенная относительная единица измерения `em` — это мера, использовавшаяся изначально в полиграфии и ссылающаяся на определенный размер шрифта. В CSS `1 em` означает размер шрифта текущего элемента, его точное значение варьируется в зависимости от того, к какому элементу вы его применяете. Рисунок 2.1 показывает элемент `div` с отступами `1 em`.

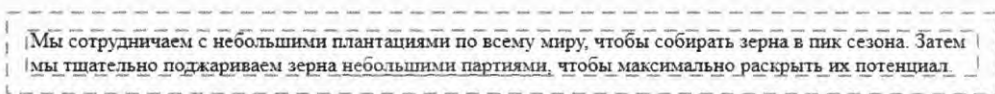


Рис. 2.1. Элемент, чье свойство `padding` имеет значение `1 em` (пунктирные линии добавлены для иллюстрации отступа)

Код для создания этого оформления показан в листинге 2.1. Набор правил определяет размер шрифта 16 пикселей, что становится локальным значением элемента величиной `1 em`. Затем код использует единицы `em`, чтобы определить отступы элемента. Добавьте код в новую таблицу стилей и вставьте любой текст в элемент `<div class="padded">`, чтобы увидеть пример в своем браузере.

Листинг 2.1. Добавление отступов со значениями в единицах `em`

```
.padded {
  font-size: 16px;
  padding: 1em;
}
```

Устанавливает отступы со всех сторон, равные размеру шрифта (свойству `font-size`)

Отступам присвоено значение `1em`. Оно умножается на размер шрифта, становясь при этом равным 16 пикселям. Это важно: значения, полученные с помощью относительных единиц, выражаются браузером как абсолютные значения и называются *вычисленными*.

В данном примере присвоение свойству `padding` значения `2em` увеличит вычисленное значение до 32 пикселей. Если другой селектор находит тот же элемент и переопределяет его с другим размером шрифта, он изменит локальное значение `em`, вычисленный размер отступов также изменится, чтобы отразить это.

Использовать единицу `em` удобно при настройке таких свойств, как `padding`, `height`, `width` или `border-radius`, поскольку они будут масштабироваться равномерно с элементом, если он наследует различные размеры шрифта или пользователь изменяет настройки шрифта.

На рис. 2.2 показаны два блока разных размеров. Размер шрифта, отступы и радиус скругления границ у них неодинаковы.

Можно отформатировать эти блоки, указав отступы и радиус скругления с помощью единиц `em`. Присвоив каждому отступу и радиусу скругления значение `1 em`, вы можете указать иной размер шрифта для каждого элемента, и другие свойства будут менять масштаб вместе со шрифтом.

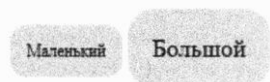


Рис. 2.2. Элементы с относительными размерами отступов и радиуса скругления

Создайте в HTML-коде два блока, как показано в листинге 2.2. Добавьте классы `box-small` и `box-large` в каждый из них в качестве модификаторов размера.

Листинг 2.2. Применение значений в единицах `em` к разным элементам (HTML)

```
<span class="box box-small">Маленький</span>
<span class="box box-large">Большой</span>
```

Теперь добавьте стили, приведенные в листинге 2.3, в свою таблицу стилей. Код будет задавать свойства блока с помощью единиц `em`. А также укажите модификаторы `box-small` и `box-large`, каждый из которых устанавливает свой размер шрифта.

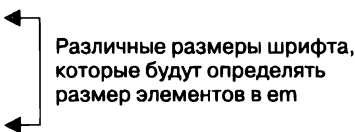
Листинг 2.3. Применение единиц `em` к различным элементам (CSS)

```
body {
  margin: 1.5em;
}

.box {
  padding: 1em;
  border-radius: 1em;
  background-color: lightgray;
}

.box-small {
  font-size: 12px;
}

.box-large {
  font-size: 18px;
}
```



Различные размеры шрифта, которые будут определять размер элементов в `em`

В этом мощь свойства `em`. Вы можете определить размер элемента в этих единицах, а затем увеличить или уменьшить масштаб с помощью всего одного объявления, которое изменит размер шрифта. Чуть позже мы рассмотрим еще один пример, но сначала поговорим о единицах `em` и размерах шрифтов.

2.2.1. Единицы `em` для указания размера шрифта

Когда речь идет о свойстве `font-size`, единицы `em` работают немного иначе. Как я говорил, значение `em` определяется текущим размером шрифта элемента. Но что означает объявление `font-size: 1.2em`? Размер шрифта не может равняться 1,2 самого себя. Вместо этого для свойства `font-size` значение `em` определяется в зависимости от наследуемого размера шрифта.

В качестве базового примера рассмотрим рис. 2.3. На нем показаны две строки текста, выполненные шрифтом разного размера. Вы зададите их с помощью единиц `em`, придерживаясь листинга 2.4.

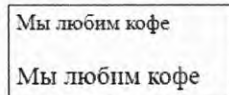


Рис. 2.3. Два разных размера шрифта, заданных с помощью единиц `em`

Измените код в соответствии со следующим листингом. Первая строка текста находится внутри элемента `body`, поэтому размер ее шрифта равен размеру шрифта тела. Вторая часть, слоган, наследует этот размер.

Листинг 2.4. Относительная разметка `font-size`

```
<body>
  Мы любим кофе
  <p class="slogan">Мы любим кофе</p>
</body>
```

Слоган наследует размер шрифта от элемента `body`

В листинге 2.5 CSS-код определяет размер шрифта тела. Для простоты я использовал здесь пиксели. Затем для увеличения размера слогана вы будете задавать значение в единицах `em`.

Листинг 2.5. Применение единиц `em` для свойства `font-size`

```
body {
  font-size: 16px;
}

.slogan {
  font-size: 1.2em;
}
```

Умножает на 1,2 наследуемый размер шрифта элемента

Установленный размер шрифта слогана — `1.2em`. Чтобы определить вычисленное значение в пикселах, необходимо сослаться на наследуемый размер шрифта 16 пикселей: $16 \cdot 1,2 = 19,2$, значит, вычисленный размер шрифта будет составлять 19,2 пиксела.

СОВЕТ

Если вы знаете, какой размер шрифта в пикселах требуется, но хотите указать объявление в `em`, вот простая формула: разделите желаемый размер в пикселах на родительский (наследуемый) размер в пикселах. Например, если нужно получить размер 10 пикселей, а ваш элемент наследует размер 12 пикселей, формула будет выглядеть так: $16 / 12 = 1,3333$ `em`. В данной главе похожие вычисления мы выполним несколько раз.

Это полезно знать, так как в большинстве браузеров размер шрифта по умолчанию — 16 пикселей. Технически это ключевое значение `medium`, которое вычисляется в 16 `px`.

Единицы `em` для указания размера шрифта и других свойств

Сейчас вы определили значение в единицах `em`, используемых для размера шрифта (на основе наследуемого размера шрифта). И нашли значение в `em` для других свойств, таких как `padding` и `border-radius` (на основе текущего размера шрифта элемента). Что затрудняет работу с единицами `em`, так это применение их одновременно и для размера шрифта, и для любых других свойств одного и того же элемента.

Когда вы так делаете, браузер должен сначала рассчитать размер шрифта, а затем использовать его для вычисления значений других свойств. Все свойства могут иметь одинаковое объявленное значение, но полученные значения окажутся разными.

В предыдущем примере мы рассчитали размер шрифта 19,2 пиксела (16 пикселов — наследуемый размер шрифта — умножили на 1,2 em). На рис. 2.4 показан тот же слоган, но с добавлением отступа 1,2 em и серым фоном, делающим его нагляднее. Этот отступ немного больше размера шрифта, несмотря на то что оба имеют одинаковое объявленное значение.




Рис. 2.4. Элемент с размером шрифта 1,2 em и отступом 1,2 em

Здесь происходит следующее: абзац наследует размер шрифта 16 пикселов от элемента `body`, получая вычисленный размер шрифта 19,2 пиксела. Это означает, что 19,2 пиксела теперь локальное значение для `em` и оно используется для расчета размера отступа. Необходимый CSS-код показан в листинге 2.6. Обновите свою таблицу стилей, чтобы увидеть пример в браузере.

Листинг 2.6. Применение единиц `em` для значений свойств `font-size` и `padding`

```
body {  
  font-size: 16px;  
}  
  
.slogan {  
  font-size: 1.2em;  
  padding: 1.2em;  
  background-color: #ccc;  
}
```

Вычисляется в 19,2 пиксела

Вычисляется в 23,04 пиксела

В данном примере заданное значение отступа — 1,2 em. Оно умножается на 19,2 пиксела (текущий размер шрифта элемента), и получается вычисленное значение 23,04 пиксела. Хотя свойства `font-size` и `padding` имеют одинаковое заданное значение, их вычисленные значения различны.

Проблема сжатия шрифтов

Единицы `em` могут давать неожиданные результаты, когда вы используете их для того, чтобы задать размер шрифта для множества вложенных элементов. Чтобы узнать точное значение для каждого элемента, нужно знать размер наследуемого шрифта, но если тот задан в родительском элементе в единицах `em`, это, в свою очередь, требует знания размера наследуемого шрифта родительского элемента, и так далее вверх по дереву.

Это быстро становится очевидным, когда вы задаете единицы `em` для размера шрифта списков с вложенными в них на несколько уровней списками. Практически каждый веб-разработчик в какой-то момент загружает страницу и обнаруживает

что-то похожее на рис. 2.5. Текст уменьшается! Именно эта проблема заставляет разработчиков избегать единиц em.

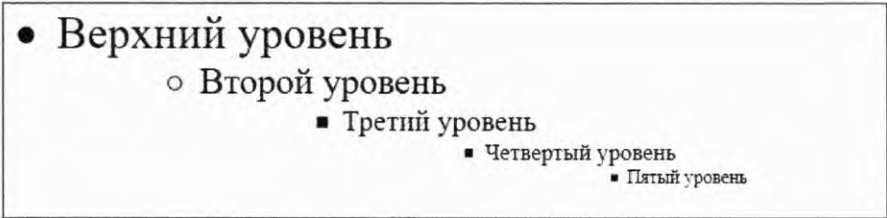


Рис. 2.5. Вложенные списки с уменьшающимся текстом

Текст уменьшается в тех случаях, когда вы вкладываете один в другой несколько уровней списков и указываете размер шрифта в единицах em для каждого уровня. Листинги 2.7 и 2.8 служат примером этого: размер шрифта в неупорядоченных списках устанавливается равным 8 em. Селектор нацеливается на каждый элемент ul на странице, поэтому, когда такие списки наследуют свой размер шрифта от других списков, вычисления в единицах em накладываются одно на другое.

Листинг 2.7. Применение значений в единицах em к списку

```
body {
  font-size: 16px;
}

ul {
  font-size: 0.8em;
}
```

Листинг 2.8. Вложенные списки

```
<ul>
  <li>Верхний уровень
    <ul>
      <li>Второй уровень
        <ul>
          <li>Третий уровень
            <ul>
              <li>Четвертый уровень
                <ul>
                  <li>Пятый уровень</li>
                </ul>
              </li>
            </ul>
          </li>
        </ul>
      </li>
    </ul>
  </li>
</ul>
```

Этот список вложен в первый, наследующий его размер шрифта...

...и этот список вложен в другой, наследующий размер шрифта второго списка...

...и так далее

Размер шрифта каждого уровня списка в 0,8 раза меньше родительского. Это означает, что размер шрифта на первом уровне — 12,8 пиксела, на следующем на один уровень ниже — 10,24 пиксела ($12,8 \cdot 0,8$), на третьем — 8,192 пиксела и т. д. Точно так же, если вы определяете размер больше 1 em, шрифт будет постоянно увеличиваться. Что мы хотим сделать, так это задать размер шрифта на верхнем уровне, а затем сохранить его на всех нижних (рис. 2.6).

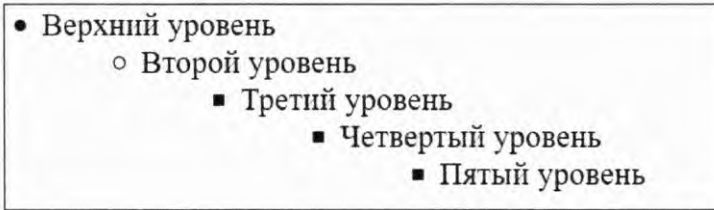


Рис. 2.6. Вложенные списки с исправленным шрифтом

Это можно сделать с помощью кода из листинга 2.9. Он задает размер шрифта первого списка равным 8 em, как и ранее (см. листинг 2.7). Второй селектор в листинге нацелен на все неупорядоченные списки внутри неупорядоченного списка — все, кроме списка верхнего уровня. Вложенные списки теперь имеют такой же размер шрифта, как и у их родителей (см. рис. 2.6).

Листинг 2.9. Исправление уменьшенного текста

```
ul {  
    font-size: 0.8em;  
}  
  
ul ul {  
    font-size: 1em;  
}
```

Списки внутри списков должны
иметь такой же размер шрифта,
как их родители

Это решает проблему, хотя и неидеально: вы задаете значение и затем сразу же переопределяете его, используя другое правило. Было бы лучше, если бы вы избегали переопределения правил, повышая специфичность селекторов.

К настоящему моменту должно стать понятно, что единицы em способны сыграть злую шутку, если проявить неосторожность. Они хороши для отступов, полей и размеров элемента, но, когда речь идет о размере шрифта, могут вызвать проблемы. К счастью, есть вариант лучше — единицы rem.

2.2.2. Указание размера шрифта в единицах rem

Когда браузер анализирует HTML-документ, он создает в памяти представление всех элементов на странице. Это представление называется *DOM (Document Object Model — объектная модель документа)*. Оно имеет древовидную структуру, где каждый элемент представлен узлом. Элемент `html` — это верхний (или корневой) узел. Далее идут дочерние узлы `head` и `body`, затем их потомки, потомки потомков и т. д.

Корневой узел — это предок всех остальных элементов документа. У него есть особый селектор псевдокласса (`:root`), который используется для его назначения. Этот прием эквивалентен задействию селектора типа `html` со специфичностью класса, а не тега.

Rem — это сокращение от `root em` (корневой `em`). Единицы `rem` относительно по отношению не к текущему элементу, а к корневому. Неважно, где вы примените их в документе, 1,2 `rem` будут иметь одинаковое вычисленное значение — 1,2 от размера шрифта корневого элемента. Листинг 2.10 устанавливает корневой размер шрифта и затем применяет единицы `rem`, чтобы определить размер шрифта относящихся к нему неупорядоченных списков.

Листинг 2.10. Указание размера шрифта с помощью единиц `rem`

```
:root {
  font-size: 1em;
}

ul {
  font-size: 0.8rem;
}
```

← Псевдокласс `:root` равен селектору типа `html`

← Использует установленный в браузере размер по умолчанию (16 пикселей)

В данном примере корневой размер шрифта соответствует установленному по умолчанию в браузере — 16 пикселей (`em` в корневом элементе относительно установленного в браузере по умолчанию). Неупорядоченные списки имеют заданный шрифт 8 `rem`, который пересчитывается в 12,8 пиксела. Поскольку все вычисления выполняются относительно корневого элемента, размер шрифта останется постоянным, даже если вы вложите списки.

Доступность: используйте относительные единицы для задания размера шрифта

Некоторые браузеры предлагают пользователю два способа настройки размера шрифта: масштабирование и установку по умолчанию. Нажатием сочетания клавиш `Ctrl++` (плюс) или `Ctrl+-` (минус) пользователь может увеличить или уменьшить размер страницы. Это визуально меняет масштаб всех шрифтов и изображений и в целом делает контент на странице больше или меньше. В некоторых браузерах это изменение применяется только к текущей вкладке и действует временно, то есть не переносится на новые вкладки.

Изменение размера шрифта по умолчанию несколько отличается от предыдущего способа. И дело не только в том, что сложнее найти, где его настроить (обычно на странице настроек браузера). Изменения на этом уровне становятся постоянными, пока пользователь снова не сменит значение. Подвох в том, что эта настройка не меняет размер шрифтов, определенных в пикселах или других абсолютных единицах. Поскольку размер шрифта по умолчанию крайне важен для некоторых пользователей, особенно тех, кто имеет недостатки зрения, всегда следует указывать размеры шрифта в относительных единицах или процентах.

Единицы `rem` упрощают многие сложности, связанные с использованием единиц `em`. По сути, это золотая середина между пикселями и единицами `em`: они обладают преимуществами относительных единиц, но с ними проще работать. Значит ли это, что везде нужно задействовать единицы `rem` и отказаться от других вариантов? Нет.

В CSS опять же чаще всего ответом будет: «Зависит от обстоятельств». Единицы `rem` — это всего лишь один из инструментов. Важной составляющей освоения каскадных таблиц стилей является изучение того, когда какой инструмент применять. Я по умолчанию задаю в единицах `rem` размер шрифта, в пикселах — ширину границ, в единицах `em` — большинство других свойств: отступы, поля, радиус скругления границ (но предпочитаю использовать проценты для определения ширины контейнеров, когда это необходимо).

Таким образом, размеры шрифта оказываются предсказуемыми, но при этом вы получаете возможности единиц `em`, которые будут менять масштаб отступов и полей, если другие факторы изменят размер шрифта элемента. Пиксеты подходят для границ, особенно когда вы хотите получить хорошую четкую линию. Вот мои коронные единицы для различных свойств, но это только инструменты и в некоторых обстоятельствах другие инструменты будут работать лучше.

СОВЕТ

Если сомневаетесь, используйте единицы `rem` для размера шрифта, пиксеты — для границ и единицы `em` — для большинства других свойств.

2.3. Перестаньте думать в пикселах

Один из шаблонов, или, скорее, антишаблонов, распространенных в последние несколько лет, — это установка размера шрифта корневого элемента на веб-странице до `0,625em`, или `62,5 %`.

Листинг 2.11. Антишаблон: глобальный сброс значения свойства `font-size` до 10 пикселей

```
html {  
  font-size: .625em;  
}
```

Я не советую так делать. Здесь берется установленный по умолчанию размер шрифта браузера 16 пикселей и уменьшается до 10 пикселей. Эту практику упрощает математика: если дизайнер говорит, что нужно сделать шрифт размером 14 пикселей, вы в уме делите это число на 10 и задаете `1,4 rem`, все еще используя относительные единицы.

Поначалу это может быть удобно, но у данного подхода есть две проблемы. Первая — приходится писать много повторяющихся стилей. Десять пикселей — это слишком мало для большинства текстового контента, поэтому придется переопределять его по всей странице. Вы обнаружите, что устанавливаете размер шрифта

абзаца равным 1,4 em, второстепенного контента — 1,4 em, навигационной панели — 1,4 em и т. д. Это увеличивает количество потенциальных уязвимостей и точек соприкосновения в коде при необходимости его изменения, растет и таблица стилей.

Вторая проблема в том, что, делая это, вы продолжаете мыслить в пикселах. Вы можете указать 1,4 em в коде, но все еще думаете: «14 пикселей». В адаптивном дизайне нужно освоиться с нечеткими значениями. Неважно, чему равны 1,2 em, вам нужно знать лишь, что это немного больше наследуемого шрифта. И если на экране все выглядит не так, как хочется, поменяйте значение. Делается это методом проб и ошибок, но то же самое происходит при работе с пикселями. (В главе 13 рассмотрим дополнительные правила для уточнения этого подхода.)

Работая с единицами em, легко увязнуть в вычислениях, скольким именно пикселям будут равняться объекты, особенно размер шрифта. Вы будете сходить с ума, деля и умножая значения в единицах em по ходу дела. Вместо этого я призываю вас взять за правило в первую очередь использовать единицы em. Если вы привыкли работать с пикселями, то для применения значений em придется попрактиковаться, но результат оправдает приложенные усилия.

Это не значит, что вам никогда не доведется работать с пикселями. Если вы трудитесь совместно с дизайнером, то, вероятно, в некоторых случаях нужно будет говорить о конкретном числе пикселей, и это нормально. В начале проекта следует установить базовый размер шрифта (и часто несколько общих размеров для заголовков и сносок). Абсолютные значения легче использовать при обсуждении размера объектов.

Преобразование в единицы em потребует вычислений, поэтому держите калькулятор под рукой. (Я нажимаю сочетание клавиш **Command+Пробел** на своем Mac и вбиваю уравнение в Spotlight.) Установка размера шрифта корневого элемента определяет значение единицы em. С этого момента работа в пикселях должна быть исключением, а не нормой.

Я продолжу упоминать пиксели в этой главе. Это поможет мне еще раз подчеркнуть, почему относительные единицы ведут себя именно так, а не иначе, а вам — привыкнуть к расчету единиц em. Позже я буду рассматривать размеры шрифтов с использованием относительных единиц.

2.3.1. Установка адекватного размера шрифта по умолчанию

Предположим, вы хотите установить размер шрифта, по умолчанию равный 14 пикселям. Вместо того чтобы задать значение по умолчанию 10 пикселей, а затем переопределять его по всей странице, присвойте это значение корневому элементу. Желаемое значение разделите на наследуемое значение — в данном случае установленное в браузере по умолчанию — 14 / 16, что равняется 0,875.

Добавьте код из листинга 2.12 в начало новой таблицы стилей, поскольку в дальнейшем будете с ней работать. Так будет установлен размер по умолчанию для корневого элемента (html).

Листинг 2.12. Установка истинного размера шрифта по умолчанию

```
:root {
  font-size: 0.875em;
}
```

Или используйте селектор html

14 / 16 (желаемый / наследуемый размер в пикселах) равняется 0,875

Теперь желаемый размер шрифта применен ко всей странице. Вам не придется указывать его где-то еще. Нужно будет только менять его там, где дизайн отклоняется от данного значения, например в заголовках.

Создадим панель с текстом, показанную на рис. 2.7. За основу возьмем шрифт размером 14 пикселей и будем использовать относительные измерения.

МОНОСОРТА

Мы сотрудничаем с небольшими плантациями по всему миру, чтобы собирать зерна в пик сезона. Затем мы тщательно поджариваем зерна небольшими партиями, чтобы максимально раскрыть их потенциал.

Рис. 2.7. Панель с текстом, для которого используются относительные единицы и наследуемый размер шрифта

Разметка для панели показана в листинге 2.13. Добавьте ее на свою страницу.

Листинг 2.13. Разметка панели

```
<div class="panel">
  <h2>Моносорта</h2>
  <div class="panel-body">
    Мы сотрудничаем с небольшими плантациями по всему миру, чтобы
    собирать зерна в пик сезона. Затем мы тщательно поджариваем зерна
    <a href="/batch-size.html">небольшими партиями</a>, чтобы максимально
    раскрыть их потенциал.
  </div>
</div>
```

Листинг 2.14 показывает стили. Вы будете использовать единицы em для определения отступов и радиуса скругления границ, rem — для размера шрифта заголовка и пиксели — для задания ширины границы. Добавьте код в свою таблицу стилей.

Листинг 2.14. Панель с относительными единицами

```
.panel {
  padding: 1em;
  border-radius: 0.5em;
  border: 1px solid #999;
}

.panel > h2 {
  margin-top: 0;
  font-size: 0.8rem;
  font-weight: bold;
  text-transform: uppercase;
}
```

Использует единицы em для определения отступов и радиуса скругления границ

Использует 1 пиксел для создания тонкой границы

Убирает лишнее пространство над панелью; больше об этом см. в главе 3

Задаст стиль шрифта заголовка, используя единицы rem для размера шрифта

Код рисует тонкую границу вокруг панели и задает стиль заголовка. Я создал заголовок меньшего размера, но полужирный и набранный прописными буквами. (Вы можете сделать его больше или выбрать другую гарнитуру шрифта, если хотите.)

Символ > во втором селекторе — это *прямой комбинатор дочерних элементов*. Он находит элемент `h2` — дочерний для элемента `.panel` (см. в приложении А полный перечень селекторов и комбинаторов).

В листинге 2.13 я для ясности добавил класс `panel-body` к внутренней части (контенту) панели, но вы можете не использовать этот прием. Поскольку данный элемент наследует размер шрифта корневого элемента, он уже появляется в нужном виде.

2.3.2. Делаем панель адаптивной

Пойдем дальше. Можно задействовать медиазапросы для изменения базового размера шрифта в соответствии с размером экрана. Будет отображаться панель разной величины в зависимости от размера экрана пользователя (рис. 2.8).



Медиазапрос — правило `@media` — используется для задания стилей, которые будут применяться только к определенным размерам экрана или типам устройств (например, принтер или экран). Это ключевой компонент адаптивного дизайна (см. листинг 2.15 для примера; рассмотрим этот вопрос подробнее в главе 8).

МОНОСОРТА

Мы сотрудничаем с небольшими плантациями по всему миру, чтобы собирать зерна в пик сезона. Затем мы тщательно поджариваем зерна небольшими партиями, чтобы максимально раскрыть их потенциал.

МОНОСОРТА

Мы сотрудничаем с небольшими плантациями по всему миру, чтобы собирать зерна в пик сезона. Затем мы тщательно поджариваем зерна небольшими партиями, чтобы максимально раскрыть их потенциал.

МОНОСОРТА

Мы сотрудничаем с небольшими плантациями по всему миру, чтобы собирать зерна в пик сезона. Затем мы тщательно поджариваем зерна небольшими партиями, чтобы максимально раскрыть их потенциал.

Рис. 2.8. Адаптивная панель, меняющая размер при разной ширине экрана: 300 пикселей (вверху слева), 800 пикселей (вверху справа) и 1440 пикселей (внизу)

Чтобы увидеть такой результат, отредактируйте эту часть таблицы стилей в соответствии с листингом 2.15.

Первый набор правил задает маленький размер шрифта по умолчанию. Это тот размер, который мы хотим видеть на маленьких экранах. Затем добавлены медиазапросы, чтобы переопределить это значение с целью постепенного увеличения размеров шрифта для экранов шириной 800, 1200 пикселей и более.

Применяя эти размеры шрифта к корневому элементу страницы, вы адаптивно переопределяете значение единиц `em` и `rem` на всей странице. Это означает, что панель теперь адаптивна, несмотря на то что вы напрямую не вносили в нее никаких изменений. На маленьком экране, например, смартфона шрифт будет выглядеть меньше (12 пикселей), уменьшатся также отступы и радиус скругления границ, чтобы ему соответствовать. А на экранах шириной 800 и 1200 пикселей эта составляющая будет увеличивать размер шрифта до 14 и 16 пикселей соответственно. Измените размер окна браузера, чтобы увидеть эти метаморфозы.

Листинг 2.15. Адаптивное базовое свойство `font-size`

<pre>:root { font-size: 0.75em; }</pre>	Применяется ко всем экранам, но переопределяется для больших размеров экрана
<pre>@media (min-width: 800px) { :root { font-size: 0.875em; } }</pre>	Применяется только к экранам шириной 800 пикселей и больше, переопределяя оригинальное значение
<pre>@media (min-width: 1200px) { :root { font-size: 1em; } }</pre>	Применяется только к экранам шириной 1200 пикселей и больше, переопределяя оба значения

Если вы достаточно организованны, чтобы задать стили для всей страницы в относительных единицах, как сделано здесь, вся она будет увеличиваться и уменьшаться в масштабе в зависимости от размера экрана. Это может быть значимой частью вашей стратегии адаптации. Два медиазапроса в верхней части таблицы стилей могут избавить от необходимости делать десятки медиазапросов в остальной части CSS-кода. Но это не работает, если вы определяете размеры в пикселах.

Таким же образом, если ваш начальник или клиент решает, что размер шрифта на сайте, который вы создали, слишком маленький или слишком большой, вы можете изменить его повсеместно, затронув лишь одну строчку кода. Особых усилий от вас это не потребует, а изменение скажется на всей остальной части страницы.

2.3.3. Изменение размера отдельного компонента

Можно использовать единицы `em` также для масштабирования отдельно взятого компонента на странице. Иногда оказывается нужна увеличенная версия той же части интерфейса в определенных местах на странице. Сделаем это с нашей панелью. Добавим класс `large` к панели: `<div class="panel large">`.

На рис. 2.9 для сравнения показаны нормальная и большая панели. Эффект такой же, как и с адаптивными панелями, но оба размера могут быть задействованы одновременно на одной и той же странице.

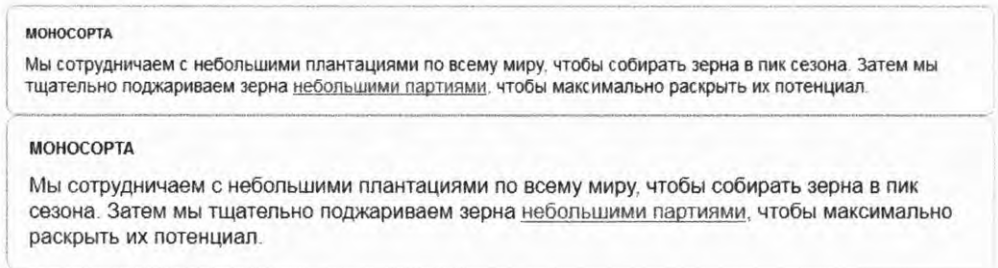


Рис. 2.9. Нормальная и большая панели, заданные на одной странице

Внесем небольшое изменение в то, как определяются размеры шрифта панели. Вы все равно будете использовать относительные единицы, но настроите то, к чему они относятся. Сначала добавьте объявление `font-size: 1rem` к родительскому элементу обеих панелей. Это означает, что для каждой из них будет установлен предсказуемый размер шрифта независимо от положения на странице.

Затем переопределите размер шрифта заголовка, применив единицы `em` вместо `rem`, чтобы сделать изменение выполняющимся относительно родительского размера шрифта, который вы только что установили равным `1 rem`. Код приведен в листинге 2.16. Обновите свою таблицу стилей.

Листинг 2.16. Создание увеличенной версии панели

```
.panel {
  font-size: 1rem;
  padding: 1em;
  border: 1px solid #999;
  border-radius: 0.5em;
}

.panel > h2 {
  margin-top: 0;
  font-size: 0.8em;
  font-weight: bold;
  text-transform: uppercase;
}
```

← Устанавливает ожидаемый размер шрифта для компонента

← Использует единицы `em` для того, чтобы остальной шрифт стал относительным к установленному родительскому размеру шрифта

Это изменение никак не влияет на внешний вид панели, но подталкивает вас создать увеличенную версию панели всего одной строкой CSS. Вам всего лишь нужно переопределить родительский размер элемента `1 rem` в другое значение. Поскольку все параметры компонента выполняются относительно него, переопределение изменит размер всей панели. Добавьте CSS-код из листинга 2.17 в свою таблицу стилей для определения увеличенной версии.

Листинг 2.17. Масштабирование всей панели одним объявлением

```
.panel.large {  
  font-size: 1.2em;  
}
```

← Составной селектор находит
элементы с классами `panel` и `large`

Теперь вы можете использовать код `class="panel"` для панели среднего размера и `class="panel large"` для большой. Аналогично, установив меньший размер шрифта, вы зададите уменьшенную версию панели. Если панель будет представлять собой более сложный компонент с различными размерами шрифта или отступами, по-прежнему понадобится лишь одно это объявление, чтобы изменить размер, в то время как все, что находится внутри нее, будет определяться с помощью единиц `em`.

2.4. Единицы измерения, относящиеся к размеру экрана устройства

Вы узнали, что единицы `em` и `rem` относительные для `font-size`, но это не единственные типы относительных единиц. Существуют также единицы измерения, относящиеся к размеру экрана.



Viewport (область просмотра) — ограниченная область в окне браузера, где отображается веб-страница, за исключением адресной строки, панели инструментов и панели текущего состояния, если они есть.

Если вы не знакомы с единицами измерения, относящимися к размеру области просмотра устройства, прочитайте далее краткое объяснение:

- ❑ *vh* — 1/100 высоты области просмотра;
- ❑ *vw* — 1/100 ширины области просмотра;
- ❑ *vmin* — 1/100 наименьших высоты или ширины (Internet Explorer 9 поддерживает `vm` вместо `vmin`);
- ❑ *vmax* — 1/100 наибольших высоты или ширины (не поддерживается в Internet Explorer и на момент написания в Edge).

Например, 50 *vw* равняется половине ширины области просмотра, а 25 *vh* — 25 % высоты области просмотра. *Vmin* берет за основу меньшую из двух величин — высоты и ширины. Это помогает убедиться, что элемент поместится в области просмотра независимо от ориентации: если устройство в режиме альбомного просмотра, он возьмет за основу высоту, а если книжного — ширину.

На рис. 2.10 показан квадратный элемент, который появляется на различных экранах разных размеров. Он определен высотой и шириной 90 *vmin*, что равняется 90 %

наименьшей из двух величин — 90 % высоты при просмотре в альбомной ориентации или 90 % ширины — в книжной.

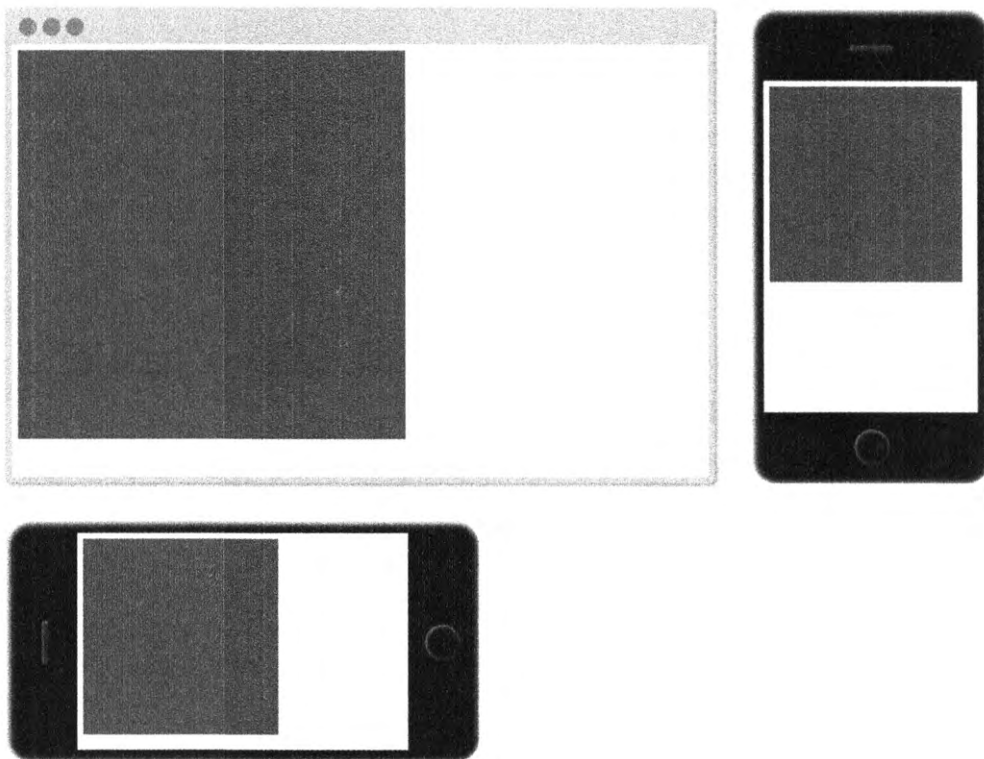


Рис. 2.10. Элемент высотой и шириной 90 vmin всегда будет отображаться как квадрат размером чуть меньше, чем область просмотра, независимо от размера экрана и ориентации устройства

Листинг 2.18 показывает стили для этого элемента. Он создает большой квадрат, который всегда соответствует размеру области просмотра устройства вне зависимости от размеров браузера. Можете добавить код `<div class="square">` на свою страницу, чтобы увидеть это.

Листинг 2.18. Квадратный элемент с размерами, заданными в единицах vmin

```
.square {  
  width: 90vmin;  
  height: 90vmin;  
  background-color: #369;  
}
```

Длины, относящиеся к размеру области просмотра устройства, прекрасно подходят, например, для того, чтобы обеспечить соответствие большого ключевого

изображения продукта области просмотра. Ваше изображение может находиться внутри длинного контейнера, но установка высоты изображения 100 vh делает его высоту именно такой, как в области просмотра.

ПРИМЕЧАНИЕ

Единицы измерения, относящиеся к размеру области просмотра устройства, для большинства браузеров являются совсем новой функцией, поэтому могут возникнуть какие-нибудь странные ошибки, если вы станете их использовать в необычных комбинациях с другими стилями (полный список см. на сайте caniuse.com/#feat=viewport-units).

CSS3

Некоторых типов единиц, рассматриваемых в данной главе, не было в предыдущих версиях CSS (в частности, `rem` и единиц, относящихся к размеру области просмотра). Их добавили в рамках серии изменений в языке, который часто называют CSS3.

В конце 1990-х и начале 2000-х годов, после завершения первоначальной работы над спецификацией CSS, на протяжении длительного времени мало что менялось. W3C (консорциум Всемирной паутины) выпустил спецификацию CSS2 в мае 1998-го. Вскоре после этого началась работа над версией 2.1, призванная исправить проблемы и ошибки версии 2. Она длилась много лет, было сделано несколько значимых дополнений к языку. Он не был окончательно доработан до апреля 2011 года. К этому моменту в браузерах уже реализовали многие изменения CSS 2.1 и все было готово к добавлению новых функций так называемого CSS3.

Цифра 3 — это неофициальный порядковый номер версии, спецификации CSS3 не существует. Скорее, спецификация была разбита на отдельные модули, каждый из которых не зависел от версии. Спецификация для фонов и ширины границ была отделена от спецификации для блочных моделей и еще от одной — для каскадирования и наследования. Это позволяло W3C вносить новые изменения в одну область CSS, не выполняя ненужное обновление областей, которые не менялись. Многие из этих спецификаций остались в версии 3 (теперь называется level 3), но некоторые, например спецификация селекторов, содержатся в level 4, а такие, как гибкая `flexbox`-верстка, — в level 1.

По мере того как вводились эти изменения, мы наблюдали взрывное увеличение количества новых функций, выходящих в браузерах с 2009 по 2013 год. Существенными дополнениями в это время стали единицы `rem` и `viewport`-относительные единицы, а также новые селекторы, медиазапросы, веб-шрифты, закругленные границы, анимация, переходы, трансформации и различные способы указания цветов. С тех пор новые функции появляются постоянно.

Это означает, что мы больше не работаем с одной конкретной версией CSS. Это новый уровень жизни. Все браузеры постоянно добавляют поддержку новых функций. Разработчики используют данные изменения и приспосабливаются к ним. Не будет CSS4, за исключением, возможно, более общего маркетингового термина. Хотя эта книга охватывает функции CSS3, я не обязательно называю их таковыми, поскольку то, что касается Интернета, — это все CSS.

2.4.1. Единицы vw для указания размера шрифта

Один из аспектов применения единиц, относящихся к размеру области просмотра, не всегда очевидный, — это указание размера шрифта. На самом деле я считаю его более практичным, чем задействование единиц vh и vw для задания высоты и ширины элемента.

Рассмотрим, что произойдет, если вы примените свойство `font-size: 2vw` к элементу. На экране монитора шириной 1200 пикселей элемент равен 24 пикселям (2 % от 1200). На планшете с экраном шириной 768 пикселей — примерно 15 пикселей (2 % от 768). И прекрасно, что он без проблем масштабируется между двумя размерами. Это значит, что нет резких изменений в контрольных точках, все происходит постепенно в соответствии с изменением размера области просмотра.

К сожалению, 24 пикселя — это слишком много для большого экрана. И что хуже, элемент меняет масштаб вплоть до 7,5 пикселя на iPhone 6. Было бы прекрасно сохранить такой эффект масштабирования, но с меньшими крайностями. В CSS вы можете добиться этого, работая с функцией `calc()`.

2.4.2. Функция `calc()` для указания размера шрифта

Функция `calc()` позволяет выполнять базовые арифметические вычисления с двумя и более значениями. Это особенно полезно для объединения значений, измеряемых в разных единицах. Функция поддерживает сложение (+), вычитание (-), умножение (*) и деление (/). Операторы сложения и вычитания должны быть окружены пробелами, поэтому стоит выработать привычку всегда ставить пробел до и после каждого оператора, например: `calc(1em + 10px)`.

Применение функции `calc()` показано в листинге 2.19, где объединяются единицы em и vw. Удалите предыдущий основной размер шрифта и соответствующие медиазапросы из своей таблицы стилей. Вставьте на их место следующий код.

Листинг 2.19. Использование функции `calc()` для вычисления значения свойства `font-size` в единицах em и vw

```
:root {  
  font-size: calc(0.5em + 1vw);  
}
```

Теперь откройте страницу и медленно меняйте размер окна браузера. Вы увидите, как шрифт плавно подстраивается под него. В качестве своего рода минимального размера шрифта здесь выступает 0,5 em, а 1 vw добавляет гибкую скалярную величину. Это даст вам основной размер шрифта, который меняется от 11,75 пикселя на iPhone 6 до 20 пикселей в окне браузера размером 1200 пикселей. Можете настроить эти значения как хотите.

Теперь вы реализовали большую часть своей адаптивной стратегии без единого медиазапроса. Вместо применения трех или четырех контрольных точек со сложным кодом все элементы страницы масштабируются плавно в соответствии с размером области просмотра.

2.5. Числа без единиц измерения и свойство `line-height`

Некоторые свойства допускают указание значений *без единиц измерения*. К ним относятся `line-height`, `z-index` и `font-weight` (700 — это полужирный шрифт, 400 — обычный и т. д.). Вы также можете задействовать значение 0 без единиц измерения везде, где требуются единицы длины (пиксели, `em` или `rem`), потому что в таких случаях единица неважна — 0 пикселей равняется 0 % и 0 `em`.

ВНИМАНИЕ!

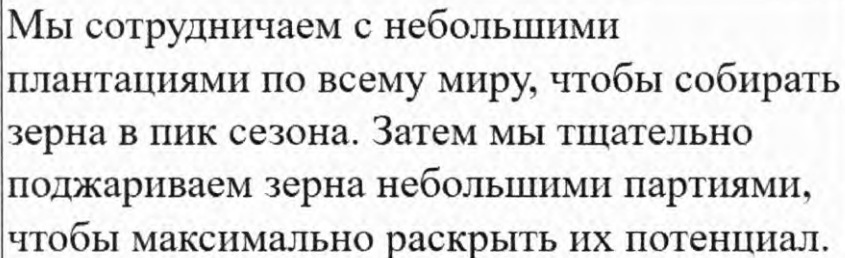
Ноль без единиц измерения используется только для значений длины и процентов, например, для настройки отступов, границ и ширины. Его нельзя применять с угловыми величинами, такими как градусы, или временными значениями, такими как секунды.

Свойство `line-height` (высота строки) необычно тем, что позволяет как использовать единицы измерения, так и не использовать. Вам стоит применять числа без единиц, поскольку они наследуются по-другому. Добавим текст на страницу и посмотрим, как это работает. Вставьте код из листинга 2.20 в свою таблицу стилей.

Листинг 2.20. Наследуемая разметка `line-height`

```
<body>
  <p class="about-us">
    Мы сотрудничаем с небольшими плантациями по всему миру, чтобы
    собирать зерна в пик сезона. Затем мы тщательно поджариваем зерна
    небольшими партиями, чтобы максимально раскрыть их потенциал.
  </p>
</body>
```

Вы зададите высоту строки для элемента `body` и разрешите наследовать это свойство в остальной части документа. Прием будет работать в соответствии с ожиданиями вне зависимости от того, что вы сделаете с размером шрифта на странице (рис. 2.11).



Мы сотрудничаем с небольшими
плантациями по всему миру, чтобы собирать
зерна в пик сезона. Затем мы тщательно
поджариваем зерна небольшими партиями,
чтобы максимально раскрыть их потенциал.

Рис. 2.11. Высота строки без единиц измерения рассчитывается заново для каждого дочернего элемента

Добавьте код из листинга 2.21 в свою таблицу стилей для перенесения этих стилей. Абзац наследует высоту строки, равную 1,2. Поскольку размер шрифта 32 пиксела ($2\text{ em} \cdot 16$ пикселей, установленных в браузере по умолчанию), высота строки вычисляется как 38,4 пиксела ($32\text{ пиксела} \cdot 1,2$). Это значение оставит соответствующий промежуток между строками текста.

Листинг 2.21. Свойство `line-height` с числовым значением без единиц измерения

```
body {
  line-height: 1.2;
}

.about-us {
  font-size: 2em;
}
```

← Дочерние элементы наследуют значение без единиц измерения

Если вместо этого вы зададите высоту строки, указав единицы измерения, то, возможно, столкнетесь с непредвиденным результатом (рис. 2.12): строки текста наползают одна на другую. Листинг 2.22 содержит CSS-код, который сгенерировал этот эффект.

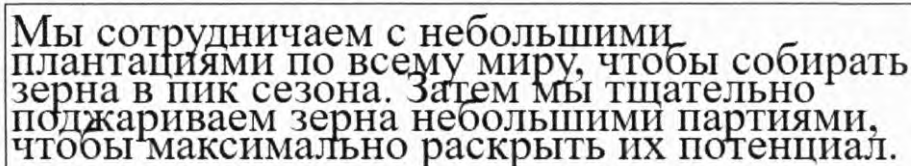


Рис. 2.12. Наложение строк из-за наследуемого значения свойства `line-height`

Листинг 2.22. Свойство `line-height` с числовым значением с единицами измерения дает неожиданный результат

```
body {
  line-height: 1.2em;
}

.about-us {
  font-size: 2em;
}
```

← Дочерние элементы наследуют вычисленное значение (19,2 пиксела)

← Получается 32 пиксела

Этот результат обусловлен своеобразной причудой наследования: когда значение элемента задано с помощью единиц измерения длины (пикселей, `em`, `rem` и т. д.), его вычисленное значение наследуется дочерними элементами. Если такие единицы, как `em`, заданы для высоты строки, их значение вычисляется и передается всем наследуемым потомкам. Если дочерний элемент имеет другой размер шрифта, то

добавление свойства `line-height` может привести к непредвиденным результатам, таким как наклаывающийся текст.



Длина — формальное наименование значения CSS, которое обозначает меру расстояния. Это число, за которым следует единица измерения, например 5 пикселей. Длина бывает абсолютная и относительная. Проценты схожи с длиной, но, строго говоря, ею не считаются.

Когда вы используете числа без единиц измерения, объявленное значение наследуется, то есть его вычисленное значение пересчитывается для каждого наследующего дочернего элемента. Это практически всегда приносит ожидаемый результат. С помощью значений без единиц измерения можно установить высоту строки для тела страницы, а затем забыть об этом для остальной ее части, если только нет определенных частей контента, для которых нужно сделать исключение.

2.6. Пользовательские свойства (или CSS-переменные)

В 2015 году долгожданная спецификация CSS «Пользовательские свойства для каскадных переменных» была опубликована в качестве кандидата в рекомендации. Она ввела понятие переменных в язык CSS, который теперь обеспечил новый уровень динамических, основанных на контексте стилей. Можно объявить переменную и присвоить ей значение, а затем ссылаться на него в таблице стилей. Этот прием сокращает повторение кода в таблице стилей, оказываясь полезным и в иных случаях, как вы вскоре увидите.

На момент написания книги поддержка пользовательских свойств была добавлена во всех ведущих браузерах, кроме Internet Explorer. Получить актуальную информацию о поддержке менее известных браузеров можно в разделе Can I Use на странице caniuse.com/#feat=css-variables.

ПРИМЕЧАНИЕ

Если вы используете препроцессор CSS, который поддерживает собственные переменные, такие как Sass (Syntactically Awesome Stylesheets) или Less, у вас может появиться соблазн игнорировать CSS-переменные. Не делайте этого. Новые CSS-переменные отличаются от прежних по своей природе и стали гораздо более универсальными. Я склонен называть их пользовательскими свойствами, а не переменными, чтобы подчеркнуть это различие.

Чтобы задать пользовательское свойство, вы объявляете его практически так же, как любое другое свойство CSS. Листинг 2.23 служит примером объявления переменной. Начните новую страницу и таблицу стилей, добавьте этот CSS.

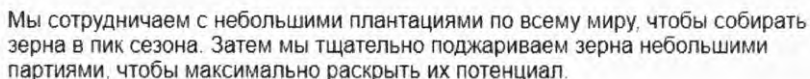
Листинг 2.23. Определение пользовательского свойства

```
:root {
  --main-font: Helvetica, Arial, sans-serif;
}
```

Код задает переменную с именем `--main-font` и устанавливает ее значение — набор распространенных рубленых шрифтов. Имя должно начинаться с двух дефисов (`--`), чтобы отличаться от свойств CSS, далее идет любое имя, которое вы хотите использовать.

Переменные должны быть объявлены внутри блока объявления. Я взял селектор `:root`, который устанавливает переменную для всей страницы (вскоре это поясню).

Само по себе объявление переменной ничего не дает, пока она не используется. Применим ее к абзацу, чтобы это дало результат, показанный на рис. 2.13.



Мы сотрудничаем с небольшими плантациями по всему миру, чтобы собирать зерна в пик сезона. Затем мы тщательно поджариваем зерна небольшими партиями, чтобы максимально раскрыть их потенциал.

Рис. 2.13. Простой абзац с применением переменной с рубленным шрифтом

Функция `var()` позволяет использовать переменные. Вы будете применять ее, чтобы сослаться на только что заданную переменную `--main-font`. Добавьте набор правил из листинга 2.24, разрешая задействовать переменную.

Листинг 2.24. Применение пользовательского свойства

```
:root {
  --main-font: Helvetica, Arial, sans-serif;
}

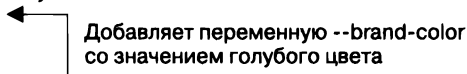
p {
  font-family: var(--main-font);
}
```

Устанавливает семейство шрифтов
для абзаца: Helvetica, Arial, sans-serif

Пользовательские свойства позволяют задать значение один раз как эталон и использовать его повторно в любом месте таблицы стилей. Это особенно полезно для повторяющихся значений, таких как цвета. В листинге 2.25 добавляется пользовательское свойство `--brand-color`. Можете задействовать эту переменную в таблице стилей десятки раз, а если захотите изменить ее, отредактировать нужно будет только в одном месте.

Листинг 2.25. Применение пользовательских свойств для цветов

```
:root {  
  --main-font: Helvetica, Arial, sans-serif;  
  --brand-color: #369;  
}  
  
p {  
  font-family: var(--main-font);  
  color: var(--brand-color);  
}
```

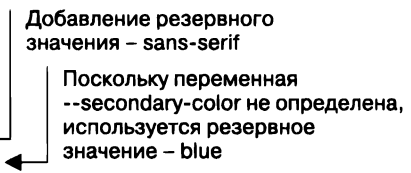


Добавляет переменную `--brand-color` со значением голубого цвета

Функция `var()` принимает второй параметр, который задает резервное значение. Если переменная, указанная в первом параметре, не определена, используется второе значение (листинг 2.26).

Листинг 2.26. Добавление резервных значений

```
:root {  
  --main-font: Helvetica, Arial, sans-serif;  
  --brand-color: #369;  
}  
  
p {  
  font-family: var(--main-font, sans-serif);  
  color: var(--brand-color, blue);  
}
```



Добавление резервного значения – `sans-serif`

Поскольку переменная `--secondary-color` не определена, используется резервное значение – `blue`

Листинг содержит резервные значения в двух различных объявлениях. В первом переменная `--main-font` определена со значением `Helvetica, Arial, sans-serif`, так что используется именно оно. Во втором переменная `--secondary-color` не определена, поэтому берется резервное значение `blue`.

ПРИМЕЧАНИЕ

Если функция `var()` имеет недопустимое значение, свойству будет присвоено его первоначальное значение. Например, так как переменная в коде `padding: var(--brand-color)` вычисляет цвет, это будет недопустимое значение отступа. В этом случае отступ будет установлен со значением `0`.

2.6.1. Динамическое изменение пользовательских свойств

В приведенных примерах пользовательские свойства — просто-напросто удобное средство, они могут избавить вас от множества повторений в коде. Но особенно интересными их делает то, что объявления пользовательских свойств каскадируются и наследуются: если определить одну и ту же переменную внутри

нескольких селекторов, она будет иметь различные значения для разных частей страницы.

Вы можете сделать переменную, например, черной, а затем переопределить ее в белую внутри конкретного контейнера. Затем любые стили, основанные на этой переменной, будут динамически задавать черный цвет, если они вне этого контейнера, и белый, если внутри. Используем описанный прием, чтобы достичь такого же результата, как на рис. 2.14.

Данная панель похожа на ту, что вы видели ранее (см. рис. 2.7). Ее HTML-код показан в листинге 2.27. В нем есть две вариации панели: одна — внутри тела, другая — внутри контейнера `dark`. Обновите свой HTML-код, чтобы он соответствовал данному.

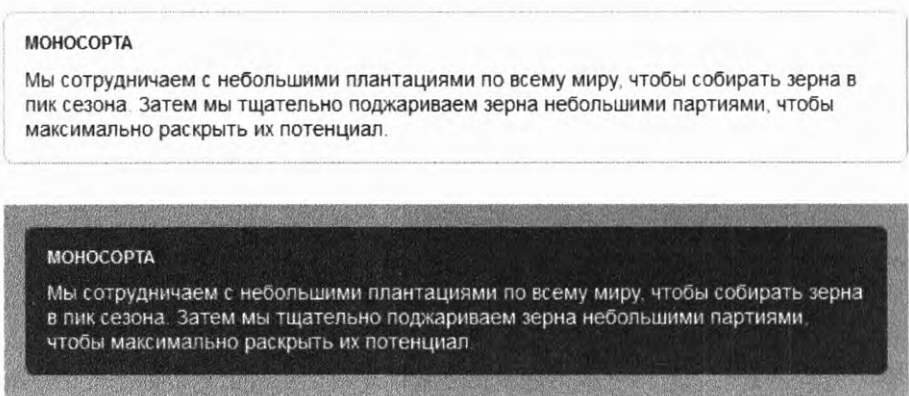


Рис. 2.14. Пользовательские свойства порождают панели разного цвета в зависимости от локальных значений переменных

Листинг 2.27. Две панели в различном контексте на странице

```
<body>
  <div class="panel">           ← Обычная панель на странице
    <h2>Моносорта</h2>
    <div class="body">
      Мы сотрудничаем с небольшими плантациями
      по всему миру, чтобы собирать зерна в пик сезона.
      Затем мы тщательно поджариваем зерна небольшими
      партиями, чтобы максимально раскрыть их потенциал.
    </div>
  </div>

  <aside class="dark">         | Вторая панель
    <div class="panel">         | внутри контейнера dark
      <h2>Моносорта</h2>
      <div class="body">
        Мы сотрудничаем с небольшими плантациями
        по всему миру, чтобы собирать зерна в пик сезона.
```

```
        Затем мы тщательно поджариваем зерна небольшими
        партиями, чтобы максимально раскрыть их потенциал.
    </div>
</div>
</aside>
</body>
```

Переопределим панель для использования переменных для текста и фоновой цвета. Добавьте листинг 2.28 в свою таблицу стилей. Так вы сделаете фоновый цвет белым, а текст — черным. Я объясню, как это работает, прежде чем вы добавите стили для темного варианта.

И снова вы определили переменные внутри набора правил селектора `:root`. Это важно, поскольку подразумевает, что эти значения установлены для всего содержимого корневого элемента (на целой странице).

Листинг 2.28. Определение цвета панели с помощью переменных

```
:root {
  font-size: calc(0.5em + 1vw);
  --main-bg: #fff;
  --main-color: #000;
}

.panel {
  font-size: 1rem;
  padding: 1em;
  border: 1px solid #999;
  border-radius: 0.5em;
  background-color: var(--main-bg);
  color: var(--main-color);
}

.panel > h2 {
  margin-top: 0;
  font-size: 0.8em;
  font-weight: bold;
  text-transform: uppercase;
}
```

		Определяет переменный цвет фона и текста как белый и черный соответственно
		Использует переменные в стилях панели

У вас есть две панели, но они по-прежнему выглядят одинаково. Теперь снова определим переменные, но на этот раз с другим селектором. В листинге 2.29 указаны стили для контейнера `dark`. Он устанавливает темно-серый фон контейнера, а также маленькие отступы и поля. И переопределяет обе переменные. Добавьте этот код в свою таблицу стилей.

Обновите страницу, и вторая панель станет выглядеть иначе: будет белый текст на темном фоне. Это произойдет потому, что, когда панель использует указанные переменные, они превращаются в значения, определенные в контейнере `dark`, а не в корневом элементе. Обратите внимание: вам не пришлось менять стили панели или применять какие-то дополнительные классы.

В данном примере вы определили пользовательские свойства дважды: первый раз в корневом элементе, где переменная `--main-color` имеет черный цвет, второй — в контейнере `dark`, где переменная `--main-color` имеет белый цвет. Пользовательские свойства ведут себя как своего рода переменные с областью видимости, поскольку значения наследуются потомственными элементами. Внутри контейнера `dark` цвет переменной `--main-color` белый, а где-либо еще на странице — черный.

Листинг 2.29. Форматирование контейнера `dark`

```
.dark {
  margin-top: 2em;
  padding: 1em;
  background-color: #999;
  --main-bg: #333;
  --main-color: #fff;
}
```

Устанавливает поле между контейнером `dark` и предыдущей панелью

Применяет темно-серый фон к контейнеру `dark`

Переопределяет переменные `--main-bg` и `--main-color` в области контейнера

2.6.2. Изменение пользовательских свойств с помощью JavaScript

Можно получить доступ к пользовательским свойствам и настроить их в реальном времени в браузере также с помощью JavaScript-сценариев. Поскольку эта книга не о JavaScript, я кратко ознакомлю вас с данной темой. Изучите ее сами, чтобы получить возможность интегрировать функционал в свои проекты JavaScript.

Листинг 2.30 показывает, как получить доступ к свойству элемента. Он добавляет на страницу скрипт, который записывает значение свойства `--main-bg` корневого элемента.

Листинг 2.30. Получение доступа к пользовательским свойствам в JavaScript

```
<script type="text/javascript">
  let rootElement = document.documentElement;
  let styles = getComputedStyle(rootElement);
  let mainColor = styles.getPropertyValue('--main-bg');
  console.log(String(mainColor).trim());
</script>
```

Получает объект `styles` для элемента

Получает значение `--main-bg` из объекта `styles`

Определяет, что свойство `mainColor` имеет значение строкового типа и удаляет пробельные символы

Поскольку можно в реальном времени задавать новые значения пользовательских свойств, примените JavaScript для динамического присвоения нового значения переменной `--main-bg`. Если это будет значение голубого цвета, контейнер появится в таком виде, как на рис. 2.15.

МОНОСОРТА

Мы сотрудничаем с небольшими плантациями по всему миру, чтобы собирать зерна в пик сезона. Затем мы тщательно поджариваем зерна небольшими партиями, чтобы максимально раскрыть их потенциал.

Рис. 2.15. JavaScript-сценарий способен установить фон панели, изменив значение переменной `--main-bg`

Код из листинга 2.31 присваивает новое значение свойству `--main-bg` корневого элемента. Добавьте этот код в конец элемента `script`.

Листинг 2.31. Установка пользовательского свойства в JavaScript

```
rootElement = document.documentElement;  
rootElement.style.setProperty('--main-bg', '#cdf'); ← Присваивает  
                                                         переменной --main-bg  
                                                         голубой цвет
```

Если вы запустите этот скрипт, любые элементы, которые наследуют свойство `--main-bg`, обновятся и будут использовать новое значение. На вашей странице фон первой панели изменится на голубой. Вторая панель останется неизменной, так как она по-прежнему наследует свойство от контейнера `dark`.

Эта техника позволяет применять JavaScript-сценарии для изменения темы сайта в режиме реального времени в браузере, или выделять отдельные части страницы, или вносить любые другие изменения на ходу. Используя всего несколько строк JavaScript, вы внесете изменения, которые повлияют на огромное количество элементов на странице.

2.6.3. Экспериментирование с пользовательскими свойствами

Пользовательские свойства — это новая область CSS, которую разработчики только начинают открывать, и ее поддержка браузерами еще ограничена. Я уверен, что спустя некоторое время вы услышите о передовых методах и новых способах применения. Экспериментируйте с пользовательскими свойствами и смотрите, к чему это может привести.

Имейте в виду, что любое объявление с помощью функции `var()` проигнорируют старые браузеры, которые его не понимают. Обеспечьте резервное поведение для них, когда это возможно:

```
color: black;  
color: var(--main-color);
```

А возможно это не всегда, если учесть динамический характер пользовательских свойств. Следите за тем, какие из новых технологий поддерживают браузеры, на сайте caniuse.com.

Итоги главы

- ❑ Используйте относительные единицы измерения, чтобы структура страницы влияла на стили.
- ❑ Применяйте единицы `rem` для указания размера шрифта и избирательно — единицы `em` для простого масштабирования компонентов на странице.
- ❑ Можно сделать так, чтобы вся страница адаптивно меняла масштаб без каких-либо медиазапросов.
- ❑ При указании высоты строки используйте значения без единиц измерения.
- ❑ Постарайтесь ознакомиться с одной из новейших возможностей CSS — с пользовательскими свойствами.

3

Знакомство с блочной моделью

В этой главе

- Практический совет по установке размеров элементов и их позиционированию на странице.
- Вертикальное центрирование.
- Колонки равной высоты.
- Отрицательные значения полей, схлопывание полей.
- Согласованное расстояние между компонентами на странице.

Когда дело дойдет до расположения элементов на странице, вы поймете, что здесь нужно учитывать множество параметров. При создании сложного сайта вы столкнетесь с плавающими или абсолютно позиционированными элементами, а также с элементами различных размеров. Могут встретиться макеты страниц с использованием новых CSS-конструкций, таких как *гибкая flexbox-верстка* или *CSS-сетка*. Существует много нюансов, за которыми нужно следить, и изучить все, что связано с макетом, практически невозможно.

Мы разберем несколько тем и внимательно рассмотрим несколько методов расположения элементов на странице. Но прежде, чем дело дойдет до них, нужно получить четкое представление об основах — размерах окна браузера и позиционировании элементов. Более сложные темы охватывают такие понятия, как поток документа и блочная модель, определяющие положение и размер элементов на странице.

В этой главе мы сверстаем двухколоночный макет страницы. Может быть, вы уже видели примеры таких макетов, так как это классическое упражнение для тех, кто начинает изучать каскадные таблицы стилей, но я особое внимание уделю нюансам, о которых часто забывают при верстке. Мы рассмотрим случаи работы с блочной моделью, и я дам практические советы по определению размеров и выравниванию элементов. Разберем также две наиболее известные проблемы: вертикальное центрирование и колонки одинаковой высоты.

3.1. Трудности с шириной элемента

В этой главе создадим простую страницу с шапкой сверху и двумя колонками под ней. В конце главы она будет выглядеть так, как показано на рис. 3.1. Я намеренно сделал дизайн страницы блочным, поэтому вы легко увидите расположение всех элементов и определите их размеры.

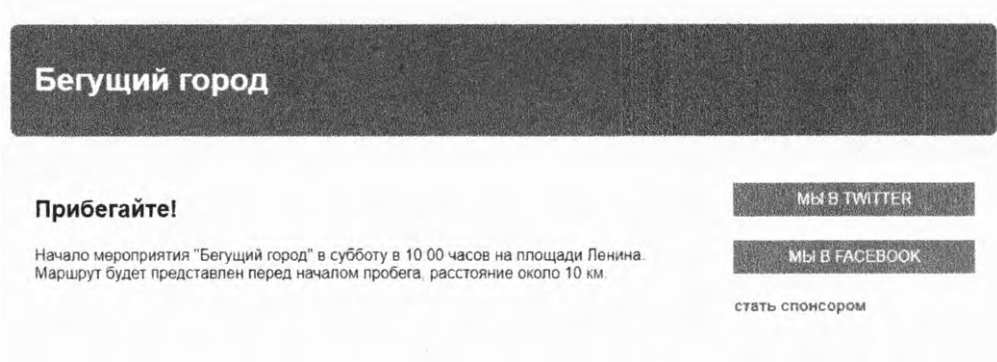


Рис. 3.1. Макет страницы с шапкой и двумя колонками

Создайте новую страницу и пустую таблицу стилей, а затем свяжите их. Теперь добавьте разметку. На странице разместятся шапка в виде одного контейнера и область основного контента, состоящая из двух колонок — основной и боковой панелей. Эти две колонки обернуты во второй контейнер (листинг 3.1).

Листинг 3.1. HTML-код страницы с двухколоночным макетом

```
<body>
  <header>
    <h1>Бегущий город</h1>
  </header>
  <div class="container">
    <main class="main">
      <h2>Прибегайте!</h2>
      <p>
        Начало мероприятия "Бегущий город" в субботу в 10:00
        на площади Ленина. Маршрут будет представлен перед началом
        пробега, расстояние около 10 км.
      </p>
    </main>
    <aside class="sidebar">
      <div class="widget"></div>
      <div class="widget"></div>
      <div class="gizmo"></div>
    </aside>
  </div>
</body>
```

Начнем с самого простого. Установите шрифт для страницы и цвет фона, определите основные блоки. Это поможет увидеть расположение и размер каждого элемента. После этого страница будет выглядеть так, как показано на рис. 3.2.

Бегущий город

Прибегайте!

Начало мероприятия "Бегущий город" в субботу в 10 00 часов на площади Ленина. Маршрут будет представлен перед началом пробега, расстояние около 10 км.

Рис. 3.2. Три основных контейнера на фоне

Для некоторых видов дизайна сайтов цвет фона контейнеров может быть прозрачным. В этом случае было бы полезно временно применить какой-либо цвет фона к контейнеру, пока вы не позиционируете его на странице и не определите его размеры.

Стили показаны в листинге 3.2. Сейчас боковая панель пуста, поэтому по умолчанию не имеет высоты. Добавьте отступы, чтобы установить для панели некоторую высоту. В других контейнерах тоже нужно задать отступы, но мы вернемся к этому позже. Пока добавьте в таблицу стилей следующий код.

Листинг 3.2. Применение шрифта и цветов

```
body {  
    background-color: #eee;  
    font-family: Helvetica, Arial, sans-serif;  
}  
  
header {  
    color: #fff;  
    background-color: #0072b0;  
    border-radius: .5em;  
}  
  
main {  
    display: block;  
}  
  
.main {  
    background-color: #fff;  
    border-radius: .5em;  
}  
  
.sidebar {  
    padding: 1.5em;  
    background-color: #fff;  
    border-radius: .5em;  
}
```

Исправляет баг,
связанный с Internet Explorer

Добавляет отступы
для панели sidebar

ПРИМЕЧАНИЕ

В Internet Explorer есть ошибка, из-за которой элементы `main` по умолчанию отображаются как строчные, а не как блочные. Здесь мы ее исправили, добавив объявление `display: block`.

Затем разместим две колонки на нужных местах. Для начала примените плавающий макет. Выровняйте блоки `main` и `sidebar` по левой стороне и задайте им ширину 70 и 30 % соответственно. Обновите таблицу стилей в соответствии с показанным далее CSS-кодом (листинг 3.3).

Листинг 3.3. Выравнивание двух колонок

```
.main {
  float: left;
  width: 70%;
  background-color: #fff;
  border-radius: .5em;
}
```

← Выравнивание основной панели по левой стороне и установка ей ширины, равной 70 %

```
.sidebar {
  float: left;
  width: 30%;
  padding: 1.5em;
  background-color: #fff;
  border-radius: .5em;
}
```

← Выравнивание боковой панели по левой стороне и установка ей ширины, равной 30 %

Результат показан на рис. 3.3, но это не совсем то, что нужно.

Бегущий город

Прибегайте!

Начало мероприятия "Бегущий город" в субботу в 10.00 часов на площади Ленина. Маршрут будет представлен перед началом пробега, расстояние около 10 км.

Рис. 3.3. Основная и боковая панели шириной 70 и 30 % соответственно

Две колонки, вместо того чтобы разместиться рядом, расположились одна под другой. Даже если вы указали ширину 70 и 30 %, колонки заняли более 100 % доступного пространства. Это произошло из-за поведения по умолчанию блочной модели (рис. 3.4). Когда вы устанавливаете ширину или высоту элемента, то указываете ширину или высоту его контента, а к ней добавляются любые отступы, границы и поля.

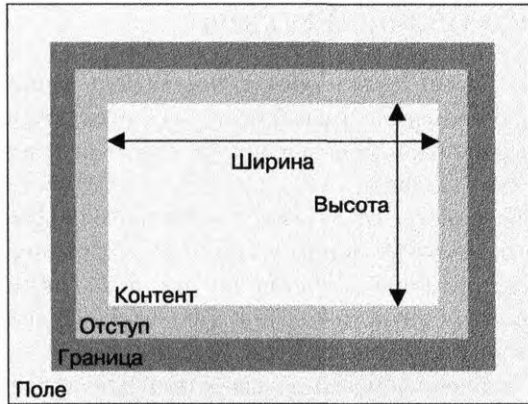


Рис. 3.4. Блочная модель по умолчанию

Такое поведение означает, что элемент шириной 300 пикселей с отступом 10 пикселей и границей 1 пиксел имеет общую ширину 322 пиксела (ширина плюс левый и правый отступы плюс левая и правая границы). Путаница увеличивается, когда все значения разные.

В примере (см. листинг 3.3) боковая панель имеет ширину 30 % плюс 1,5 em левый и правый отступы, ширина основного блока — 70 %. Вместе две колонки займут 100 % области просмотра экрана плюс 3 em. Для подгонки контейнеры нужно обернуть.

3.1.1. Избегаем волшебных чисел

Самый простой способ решить проблему заключается в уменьшении ширины одной из колонок (например, боковой панели). На моем экране для боковой панели выделено 26 % ширины, но это ненадежно; 26 % — это так называемое *волшебное число*. Вместо того чтобы задействовать желаемое значение, я нашел значение 26 %, внося разные изменения в свои стили, пока не получил результат.

В программировании в целом использовать волшебные числа нежелательно. Часто бывает трудно объяснить, почему они работают. Если вы не понимаете, откуда появляется это число, вы не поймете, как оно будет вести себя в разных обстоятельствах. Ширина моего экрана — 1440 пикселей, поэтому на небольших экранах боковая панель все равно будет вылезать за пределы экрана. Хотя в CSS широко пользуются методом проб и ошибок, обычно так поступают, когда нужно определиться со стилями, а не для того, чтобы заставить элементы вписаться в определенное место.

Одна из альтернатив магическому числу — позволить браузеру вычислять значения самому. В данном случае ширина колонок — 3 em (из-за отступов), поэтому можете использовать функцию `calc()`, чтобы уменьшить ширину так, как требуется вам. Ширина боковой панели `calc(30% - 3em)` дает именно то, что нужно. Но есть способ и лучше.

3.1.2. Настройка блочной модели

Столкнувшись с проблемами, вы поняли: блочная модель со значениями по умолчанию — это не то, с чем бы вам хотелось работать. Вы хотите, чтобы указанная ширина включала отступы и границы. CSS позволяет настроить поведение блочной модели с помощью свойства `box-sizing`.

По умолчанию для свойства `box-sizing` задано значение `content-box`. Это означает, что любая указанная высота или ширина задает размер области контента. Вместо этого можете присвоить значение `border-box`. Таким образом, свойства `height` и `width` задают суммарный размер контента, отступов и границ, и это именно то, чего вы хотели добиться в этом примере.

На рис. 3.5 показана блочная модель со свойством `box-sizing: border-box`. С этой моделью отступ не расширяет элемент, а сужает внутренний контент (то же самое для высоты).

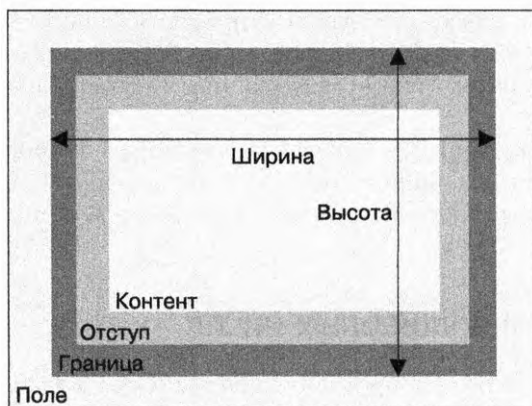


Рис. 3.5. Блочная модель со свойством `box-sizing: border-box`

Если вы обновите код, чтобы использовать свойство `box-sizing: border-box`, контейнеры будут располагаться на одной строке независимо от левого и правого отступов (рис. 3.6).

Бегущий город

Прибегайте!

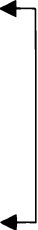
Начало мероприятия "Бегущий город" в субботу в 10.00 часов на площади Ленина. Маршрут будет представлен перед началом пробега, расстояние около 10 км.

Рис. 3.6. Колонки с настроенной блочной моделью расположены бок о бок

Чтобы настроить блочную модель для двух элементов, `main` (основного контента) и `sidebar` (боковой панели), обновите таблицу стилей, чтобы CSS-код соответствовал листингу 3.4.

Листинг 3.4. Плавающие колонки с настроенной блочной моделью

```
.main {  
  box-sizing: border-box;  
  float: left;  
  width: 70%;  
  background-color: #fff;  
  border-radius: .5em;  
}  
  
.sidebar {  
  box-sizing: border-box;  
  float: left;  
  width: 30%;  
  padding: 1.5em;  
  background-color: #fff;  
  border-radius: .5em;  
}
```



Применяем
свойство `box-sizing:`
`border-box`


Результат использования свойства `box-sizing: border-box`: два элемента занимают не более 100 % ширины. В представленные 70 и 30 % теперь входят отступы, поэтому они вписываются в одну и ту же строку.

3.1.3. Глобальное применение свойства `box-sizing: border-box`

Теперь размер блоков более удобный, но вы наверняка столкнетесь с этой же проблемой, только уже с другими элементами. Было бы неплохо исправить ее один раз, сразу для всех элементов, чтобы не пришлось снова лезть в код. Это делается с помощью универсального селектора (*), предназначенного для всех элементов на странице, как показано в листинге 3.5. Я добавил селекторы для нацеливания на каждый псевдоэлемент на странице. Поместите этот код в верхнюю часть таблицы стилей.

Листинг 3.5. Глобальное применение свойства `box-sizing: border-box`

```
*,  
::before,  
::after {  
  box-sizing: border-box;  
}
```



Изменение значения
свойства `box-sizing`
у всех элементов
и псевдоэлементов на странице

После внесения изменений свойства `height` и `width` всегда будут определять актуальные высоту и ширину элемента. Отступы не изменят их.

ПРИМЕЧАНИЕ

Добавление фрагментов кода в начало таблицы стилей — распространенная практика.

Если вы добавляете сторонние компоненты с собственным CSS-кодом на свою страницу, то можете увидеть некоторые неправильно расположенные макеты этих компонентов, особенно если их CSS-код не был написан с учетом этого исправления. Поскольку универсальное решение по изменению значения свойства `border-box` направлено на каждый элемент компонента с помощью универсального селектора, исправить это может быть проблематично. Вам нужно настроить каждый элемент внутри компонента, чтобы вернуться к значению `content-size`.

Упростить процесс можно иным способом с учетом наследования. Обновите эту часть кода таблицы стилей, чтобы она соответствовала листингу 3.6.

Листинг 3.6. Более надежное применение значения `border-box`

```
:root {
  box-sizing: border-box;
}

*,
::before,
::after {
  box-sizing: inherit;
}
```

Применение значения `border-box` к корневому элементу

Указывает всем другим элементам и псевдоэлементам наследовать значение свойства `box-sizing`

Свойство `box-sizing` обычно не наследуется, но, используя ключевое слово `inherit`, вы можете заставить его быть таковым. В приведенном далее коде стороннему компоненту назначают свойство `content-box`, когда это необходимо, настройкой его контейнера верхнего уровня. Все элементы внутри компонента будут наследовать значение свойства `box-sizing`:

```
.third-party-component {
  box-sizing: content-box;
}
```

Теперь все элементы вашего сайта получают более предсказуемую блочную модель. Рекомендую добавлять код из листинга 3.6 в CSS-код каждый раз, когда начинаете создавать сайт, — это застрахует вас от массы неприятностей в долгосрочной перспективе. Однако могут возникнуть проблемы в существующей таблице стилей, особенно если уже написаны множество стилей, основанных на стандартной модели, завязанной на размер контента. После добавления кода в существующий проект обязательно проверьте сайт на наличие любых ошибок.

ПРИМЕЧАНИЕ

С этого момента в каждом примере в книге предполагается, что данный фрагмент кода с решением проблемы свойства `box-sizing` находится в начале таблицы стилей.

3.1.4. Добавление зазора между колонками

Часто между колонками следует оставить небольшой промежуток (зазор). Иногда его можно создать, добавив отступ к одной из колонок, но в некоторых случаях такой подход не работает. Если обе колонки имеют фоновый цвет или границу, как в примере, нужно, чтобы зазор появился между границами двух элементов (рис. 3.7). Обратите внимание на серое пространство между двумя областями с белым фоном. Вы можете добиться этого несколькими способами. Рассмотрим пару из них (листинги 3.7 и 3.8).

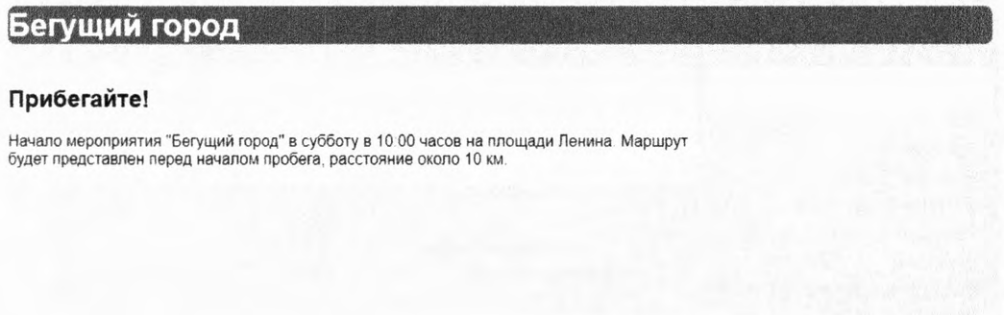


Рис. 3.7. Добавлен зазор между двумя колонками

Можно добавить поле к одной из колонок и настроить ширину элементов для контроля добавленного пространства. В листинге 3.7 показано, как вычесть 1 % из ширины боковой колонки и переместить его в поле. Обновите свой CSS-код до соответствия листингу.

Листинг 3.7. Зазор на основе поля с процентным размером

```
.main {
  float: left;
  width: 70%;
  background-color: #fff;
  border-radius: .5em;
}

.sidebar {
  float: left;
  width: 29%;
  margin-left: 1%;
  padding: 1.5em;
  background-color: #fff;
  border-radius: .5em;
}
```

Вычитает 1 % из ширины...

...и добавляет его в качестве поля

Прием добавляет зазор, но его ширина основывается на ширине внешнего контейнера — 1 % от полной ширины родительского элемента. Что делать, если вы хотите

указать размер зазора в единицах, отличных от процентов? (Я предпочитаю делать зазоры, размер которых основан на единицах em, мне так удобнее.) Выполните это с помощью функции `calc()`.

Вместо того чтобы перемещать в поле 1 % от ширины, перемещайте 1,5 em. В листинге 3.8 показана реализация с помощью функции `calc()`. Снова измените CSS-код, чтобы он соответствовал листингу.

Листинг 3.8. Использование функции `calc()` для вычитания зазора из ширины

```
.main {
  float: left;
  width: 70%;
  background-color: #fff;
  border-radius: .5em;
}

.sidebar {
  float: left;
  width: calc(30% - 1.5em);
  margin-left: 1.5em;
  padding: 1.5em;
  background-color: #fff;
  border-radius: .5em;
}
```

Вычитает 1.5 em из ширины...

...и добавляет его в качестве поля

Это не только позволяет вам использовать единицы em вместо процентов для создания зазоров, но и делает код чуть более ясным.

3.2. Проблемы высоты элементов

Управление высотой элементов отличается от изменения их ширины. Решения проблем со свойством `box-sizing`, рассмотренные ранее, по-прежнему применимы и полезны, но, как правило, лучше избегать явной установки высоты элемента. Обычный поток документа предназначен для работы с ограниченной шириной и неограниченной высотой. Контент заполняет ширину области просмотра, а затем при необходимости строки переносятся. Из-за этого высота контейнера определяется его контентом, а не самим контейнером.



Обычный поток документа определяет поведение по умолчанию элементов макета на странице. Строчные элементы перемещаются вместе с текстом страницы слева направо с переносом строк, когда те достигают края контейнера. Блочные элементы располагаются на отдельных строках с промежутками сверху и снизу.

3.2.1. Управление переполнением

Когда вы явно определяете высоту элемента, то рискуете *переполнить* контейнер контентом. Такое происходит, когда контент не соответствует указанному ограничению по размеру и визуализируется вне родительского элемента. На рис. 3.8 показано это поведение. Поток документа не учитывает переполнение, и любой контент под контейнером будет отображаться поверх контента, «вытекшего» из контейнера.

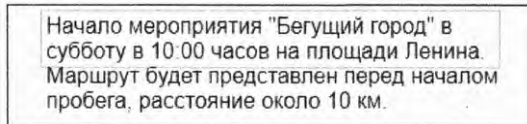


Рис. 3.8. Контейнер, переполненный контентом

Вы можете контролировать точное поведение переполняющего контента с помощью свойства `overflow`, которое поддерживает четыре значения:

- ☐ `visible` (значение по умолчанию) — весь контент отображается, даже если выходит за края контейнера;
- ☐ `hidden` — контент вне контейнера будет скрыт;
- ☐ `scroll` — в контейнер добавляются полосы прокрутки, чтобы пользователь мог увидеть оставшийся контент. В некоторых операционных системах выполняются как горизонтальные, так и вертикальные полосы прокрутки, даже если весь контент уместается. В этом случае они будут отключены (затенены);
- ☐ `auto` — полосы прокрутки добавляются в контейнер только при переполнении контентом.

Обычно я предпочитаю значение `auto`, а не `scroll`, потому что в большинстве случаев не хочу, чтобы полосы прокрутки отображались, если нет такой необходимости. На рис. 3.9 показаны четыре блока с разными значениями свойства `overflow`.

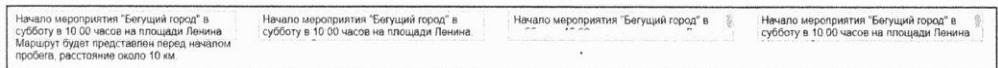


Рис. 3.9. Поведение контента в случае переполнения: `visible`, `hidden`, `scroll` и `auto` (слева направо)

Будьте внимательны с применением полос прокрутки. Браузеры вставляют их для прокрутки страницы, а добавление вложенных прокручиваемых областей на страницу может оказаться неудобным для пользователя. Если он использует колесико мыши, чтобы прокручивать страницу вниз, и указатель мыши попадает на малую прокручиваемую область, прокрутка основной страницы прекращается и вместо этого прокручивается меньшая область с контентом.

Горизонтальное переполнение

Контент способен переполняться не только по вертикали, но и по горизонтали. Типичная ситуация — длинный URL-адрес помещен в узкий блок. Тогда действуют те же правила, что и при вертикальном переполнении.

Вы можете управлять только горизонтальным переполнением, используя свойство `overflow-x`, или только вертикальным с помощью `overflow-y`. Они поддерживают те же значения, что и свойство `overflow`. Обратите внимание: явное определение различных значений как по оси *X*, так и по оси *Y*, как правило, приводит к непредсказуемым результатам.

3.2.2. Применение альтернатив к высотам, указанным в процентах

Указание высоты в виде значений в процентах способно вызвать проблемы. Проценты относятся к размеру блока, содержащего элемент, а высота этого контейнера обычно определяется высотой его потомков. В итоге браузер не может распознать определение, поэтому он будет игнорировать объявление. Для того чтобы работать с высотой в процентах, для родительского элемента нужно явно указать определенную высоту.

Одна из причин, по которой разработчики пытаются указывать высоту в процентах, состоит в следующем: нужно, чтобы контейнер заполнял экран. Лучший подход — использование единиц `vh`, относящихся к области просмотра, которые мы рассмотрели в главе 2. Высота `100 vh` — это высота области просмотра. Чаще всего колонки создаются равной высоты. Это тоже может быть достигнуто без значений в процентах.

Колонки равной высоты

Проблема с колонками равной высоты — это одна слабость, которая изначально преследовала CSS. В начале 2000-х годов каскадные таблицы стилей вытеснили HTML-таблицы, использовавшиеся для размещения контента. В то время таблицы были единственным способом создания двух колонок равной высоты или более конкретно колонок одинаковой высоты без явного указания размеров. Вы можете легко установить для всех колонок высоту 500 пикселей или задать другое произвольное значение. Но если позволите колонкам самостоятельно определять свою высоту, то каждый элемент станет оценивать ее, основываясь на контенте.

Для решения этой проблемы потребовалось схитрить. По мере развития CSS возникали способы с применением псевдоэлементов или отрицательных значений полей. Если вы все еще задействуете какой-либо из этих сложных методов, при-

шло время забыть о них. Современные браузеры делают намного проще — они поддерживают CSS-таблицы. Например, Internet Explorer 8 и выше поддерживает свойство `display: table`, а Internet Explorer 10 и выше позволяет работать с гибкими flex-контейнерами, дающими возможность по умолчанию создавать колонки равной высоты.

ПРИМЕЧАНИЕ

Когда я говорю о современных браузерах, я имею в виду последние версии автоматически обновляющихся («вечнозеленых») браузеров. К ним относятся Chrome, Firefox, Edge, Opera и в большинстве случаев Safari. Internet Explorer — это самая большая головная боль.

Большинство современных видов дизайна страниц требуют колонок равной высоты. Отличный пример — ваша страница с двумя колонками. Она будет выглядеть аккуратнее, если выровнять высоту основной и боковой колонок (рис. 3.10). По мере того как контент в любой из них увеличивается, растут и они (при необходимости), чтобы нижние их границы всегда оставались на одном уровне.

Бегущий город

Прибегайте!

Начало мероприятия "Бегущий город" в субботу в 10:00 часов на площади Ленина. Маршрут будет представлен перед началом пробега, расстояние около 10 км

Рис. 3.10. Колонки одинаковой высоты

Вы можете решить задачу, установив произвольную высоту для обеих колонок, но какое значение вы бы выбрали? Если слишком большое, то внизу контейнеров окажется большой промежуток, а если слишком маленькое — произойдет переполнение контентом.

Лучше всего позволить колонкам самостоятельно управлять высотой, чтобы меньшая колонка удлинялась, выравниваясь по высоте с большей. Я покажу, как это сделать, задействуя макеты на основе CSS-таблиц и flex-контейнеров.

CSS-таблицы

Вы будете работать с макетом страницы на основе CSS-таблиц. Вместо использования плавающих элементов создайте контейнер `display: table` и две колонки `display: table-cell`. Обновите свои стили в соответствии с листингом 3.9. (Обратите внимание на отсутствие элемента `table-row`. В CSS-таблицах положение элемента в строке не такое строгое, как в HTML-таблицах.)

Листинг 3.9. Колонки равной высоты с использованием CSS-табличной верстки

```

.container {
  display: table;
  width: 100%;
}

.main {
  display: table-cell;
  width: 70%;
  background-color: #fff;
  border-radius: .5em;
}

.sidebar {
  display: table-cell;
  width: 30%;
  margin-left: 1.5em;
  padding: 1.5em;
  background-color: #fff;
  border-radius: .5em;
}

```

Делает макет контейнера похожим на таблицу

Заставляет таблицу заполнять всю ширину контейнера

Позволяет макету колонки имитировать ячейки таблицы

Поле больше не применяется

По умолчанию элемент со значением `display: table` не будет расширяться до 100%-ной ширины, как элемент `block`, поэтому вам придется объявить ширину явно ❶. Уже лучше, но теперь нет зазора. Это связано с тем, что поля ❷ не могут быть применены к элементам `table-cell`. Нужно будет внести дополнительные изменения, чтобы получить именно то, что хочется.

Чтобы обозначить пространство между ячейками таблицы, можно использовать свойство `border-spacing` элемента таблицы. Оно принимает два значения длины: для горизонтального и вертикального интервалов. (Можно указать лишь одно значение для обоих интервалов.) Если применить свойство `border-spacing: 1.5em 0` к контейнеру, возникнет побочный эффект: это значение воздействует и на внешние края таблицы. Теперь колонки больше не выровнены по отношению к левой и правой сторонам шапки (рис. 3.11).

Бегущий город

Прибегайте!

Начало мероприятия "Бегущий город" в субботу в 10.00 часов на площади Ленина. Маршрут будет представлен перед началом пробега, расстояние около 10 км.

Рис. 3.11. Свойство `border-spacing` применяется между ячейками таблицы и влияет на внешние края

Проблему можно решить с помощью *отрицательных* значений полей, но для этого нужен новый контейнер, который обтекает всю таблицу. Добавьте вокруг

контейнера код `<div class = "wrapper">` и установите левое и правое поля равными 1,5 em, чтобы избавиться от интервала 1,5 em между границами на боковых панелях. Этот фрагмент кода таблицы стилей должен выглядеть как в листинге 3.10.

Листинг 3.10. Колонки на основе табличного макета с исправленным зазором

```
.wrapper {
  margin-left: -1.5em;
  margin-right: -1.5em;
}

.container {
  display: table;
  width: 100%;
  border-spacing: 1.5em 0;
}

.main {
  display: table-cell;
  width: 70%;
  background-color: #fff;
  border-radius: .5em;
}

.sidebar {
  display: table-cell;
  width: 30%;
  padding: 1.5em;
  background-color: #fff;
  border-radius: .5em;
}
```

Добавляет новый обертывающий элемент с отрицательными полями

Устанавливает горизонтальное расстояние между ячейками таблицы

Вместо положительных значений полей, которые «давят» на края контейнера, отрицательные значения полей расширяют края контейнера. В сочетании со свойством `border-spacing` внешние стороны колонок теперь выравниваются с границами элемента `body` (контейнер внутри обертки). Вы получили макет, который хотели сделать изначально: две колонки одинаковой высоты, зазор 1,5 em и внешние края, выровненные по краям шапки (рис. 3.12).

Бегущий город

Прибегайте!

Начало мероприятия "Бегущий город" в субботу в 10.00 часов на площади Ленина. Маршрут будет представлен перед началом пробега, расстояние около 10 км.

Рис. 3.12. Колонки равной высоты наконец-то верны

У отрицательных значений есть несколько интересных способов применения, которые мы рассмотрим немного позже.

Табличные макеты?

Если вы уже работали над веб-проектами, то, вероятно, слышали, что использование HTML-таблиц при верстке макетов — плохая идея. Многие веб-дизайнеры в начале 2000-х годов создавали сайты с помощью элементов `table`. Часто бывало проще создать страницы с таблицами вместо того, чтобы решать проблемы с отступами (единственная реальная альтернатива в то время). В конце концов решили отказаться от таблиц при верстке макетов, потому что это означало уход от семантики HTML. Вместо того чтобы быть HTML-элементами, представляющими контент, они играли роль инструментов макетирования, а за это должен отвечать CSS.

Браузеры теперь поддерживают отображение таблиц для всех видов элементов, отличных от `table`, поэтому вы можете пользоваться преимуществами табличных макетов и поддерживать семантическую разметку. Однако это не панацея при решении наших проблем. В HTML-таблицах атрибуты `colspan` и `rowspan` не имеют эквивалента, а плавающие элементы, `flex`-контейнеры и строчные блоки позволяют размещать контент так, как не могут таблицы.

Flexbox-верстка

Двухколоночный макет с колонками равной высоты можно создать также с помощью *flexbox-верстки* (листинг 3.11). Примечательно, что этот способ не требует дополнительного обертыwania в контейнеры `div`. По умолчанию `flexbox`-верстка создает элементы одинаковой высоты, вам не придется задействовать отрицательные значения полей.

СОВЕТ

Используйте `flexbox`-верстку вместо таблиц, если не собираетесь поддерживать Internet Explorer 9 и более старые версии.

Удалите обертку `div`, добавленную в макет таблицы, и обновите таблицу стилей, чтобы она соответствовала листингу 3.11. Если вы новичок в `flexbox`-верстке, это послужит введением в основные действия.

Применяя свойства `display: flex` к контейнеру, вы делаете его гибким *flex-контейнером*. Его дочерние элементы по умолчанию станут одинаковыми по высоте. Вы можете указывать ширину элементов и поля для них. Даже если значения будут превышать 100 %, `flexbox`-верстка справится с этим. Листинг 3.11 попиксельно визуализирует страницу, как и табличный макет, но не требует дополнительного обертыwania. Да и каскадные таблицы стилей здесь немного проще.

Листинг 3.11. Колонки одинаковой высоты с помощью flexbox-верстки

```
.container {  
  display: flex;  
}  
  
.main {  
  width: 70%;  
  background-color: #fff;  
  border-radius: .5em;  
}  
  
.sidebar {  
  width: 30%;  
  padding: 1.5em;  
  margin-left: 1.5em;  
  background-color: #fff;  
  border-radius: .5em;  
}
```

Присвоение значения flex свойству display контейнера

Элементам внутри flex-контейнера не нужно задавать свойства display и float

Поля работают так же, как и раньше с плавающими элементами

Flexbox-верстка предоставляет множество возможностей, о которых я расскажу в главе 5. В этом примере показано все, что нужно для создания первого flex-макета. (Internet Explorer 10 дополнительно требует указания некоторых свойств с вендорными префиксами, также рассмотрим их в главе 5.)

ВНИМАНИЕ!

Указывайте высоту элемента, только если нет другого выхода. Всегда ищите альтернативный подход. Установка фиксированной высоты впоследствии приводит к осложнениям.

3.2.3. Свойства min-height и max-height

Два свойства, которые могут оказаться чрезвычайно полезными, — это `min-height` и `max-height`. Вместо того чтобы явно определять высоту, можете использовать их для указания минимального или максимального значения, позволяющего элементу естественным образом варьироваться в пределах данного диапазона.

Предположим, вы хотите разместить изображение после большого абзаца, но обеспокоены тем, что оно переполнит контейнер. Вместо установки определенной высоты укажите минимальную с помощью свойства `min-height`. Это означает, что элемент будет как минимум выше, чем вы укажете, и, если контент не уместается, браузер позволит элементу расшириться, чтобы предотвратить переполнение.

На рис. 3.13 показаны три элемента. Тот, что слева, не имеет свойства `min-height`, поэтому его высота определяется естественным образом, а минимальная высота двух других — 3 em. Элемент посередине должен быть ниже, чем показано на рисунке, но свойство `min-height: 3em` обусловило высоту 3 em. Элемент справа

содержит контент, которого достаточно для того, чтобы превысить 3 em, и контейнер расширился, чтобы вместить весь текст.

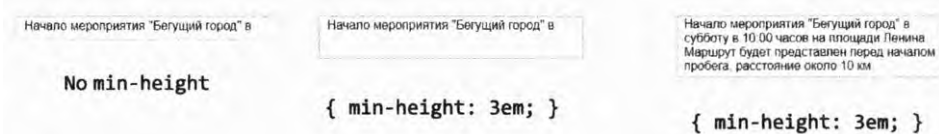


Рис. 3.13. Три элемента: один без ограничения минимальной высоты и два со свойством `min-height: 3em`

Точно так же `max-height` позволяет варьировать размер элемента вплоть до указанного. Если он достигнут, элемент не становится выше и будет переполнен контентом. Аналогичные свойства `min-width` и `max-width` ограничивают ширину элемента.

3.2.4. Центрирование контента по вертикали

Вертикальное центрирование в CSS — еще одна распространенная проблема. Существует несколько старых способов добиться центрирования по вертикали, причем каждый из них работает только при определенных обстоятельствах. В случае с CSS ответ на вопрос о том, как решить какую-либо проблему, часто такой: «Зависит от обстоятельств». И это именно та ситуация.

Почему не работает `vertical-align`?

Разработчики часто разочаровываются, когда применяют свойство `vertical-align: middle` к блочному элементу, ожидая, что он будет центрировать контент блока. Но браузер игнорирует это объявление.

Свойство `vertical-align` влияет только на строчные и табличные элементы. В случае со строчными элементами оно контролирует выравнивание между элементами в одной строке. Например, вы можете использовать его для управления тем, как строчное изображение выравнивается с соседним текстом.

В случае табличных элементов свойство `vertical-align` управляет выравниванием контента внутри ячейки. Если страница сверстана на основе CSS-таблицы, можете выполнить вертикальное центрирование с помощью свойства `vertical-align`.

Большинство проблем связано с установкой фиксированной высоты контейнера, а затем попыткой динамически центрировать контент внутри него. Когда это возможно, попробуйте добиться желаемого эффекта, позволяя браузеру самостоятельно определять высоту.

Вот самый простой способ вертикального центрирования в CSS: установите в контейнере равные значения верхнего и нижнего отступов, и пусть контейнер и его контент сами определяют высоту (рис. 3.14). В листинге 3.12 показан код решения. Вы можете временно добавить его в свою таблицу стилей, чтобы просмотреть на

своей странице (потом обязательно удалите его, так как он не является частью нашего дизайна).



Бегущий город

Рис. 3.14. Использование отступов для вертикального центрирования контента

Листинг 3.12. Применение отступов для вертикального центрирования контента

```
header {  
  padding-top: 4em;  
  padding-bottom: 4em;  
  color: #fff;  
  background-color: #0072b0;  
  border-radius: .5em;  
}
```

Равные значения верхнего и нижнего отступов
центрируют по вертикали контент элемента
без фиксации высоты

Этот подход работает независимо от того, какой контент внутри контейнера — строчный, блочный или любой другой. Иногда, однако, нужно установить определенную высоту контейнера, а то не будет возможности использовать отступы, так как потребуются, чтобы другой потомок располагался в контейнере сверху или снизу.

Это тоже распространенная проблема с колонками равной высоты, особенно если вы применяете более старый метод плавающих элементов. К счастью, оба подхода, и CSS-таблицы, и flexbox-верстка, упрощают центрирование. (Если используется один из старых методов, придется найти другой способ позиционирования контента.) Для получения справки по различным сценариям смотрите следующую врезку.

Руководство по вертикальному центрированию

Наилучший подход к центрированию контента внутри контейнера может опираться на ряд факторов, основанных на конкретном сценарии. Выполняйте следующие шаги, пока не наткнетесь на решение, которое ищете. Некоторые из этих методов будут рассмотрены в последующих главах (их номера указаны).

- *Можете ли вы использовать контейнер определенной высоты?* Установите одинаковые верхние и нижние отступы у контейнера, чтобы центрировать его контент.
- *Вам требуется определенная высота контейнера или нужно избегать использования отступов?* Примените свойства `display: table-cell` и `vertical-align: middle` к контейнеру.

- *Можете ли вы задействовать flexbox-верстку?* Если вам не нужна поддержка браузера Internet Explorer 9, центрируйте контент с помощью flexbox-верстки (см. главу 5).
- *Является ли внутренний контент только одной строкой текста?* Установите высоту строки, равную высоте нужного контейнера. Это заставит контейнер увеличиться, чтобы вместить ее. Если контент не встроенный, может потребоваться значение `inline-block`.
- *Знаете ли вы высоту как контейнера, так и внутреннего контента?* Центрируйте контент посредством абсолютного позиционирования (см. главу 7). Я рекомендую применять это решение, только если все подходы, упомянутые ранее, не дали нужного результата.
- *Что делать, если высота внутреннего элемента неизвестна?* Задействуйте абсолютное позиционирование в сочетании с трансформациями (см. главу 15 в качестве примера). И вновь рекомендую делать так лишь в том случае, если вы исключили все другие варианты.

Если вы затрудняетесь, какое решение принять, посетите сайт howtocenterincss.com. Это отличный ресурс. Выберите несколько вариантов на основе своего сценария и сверстайте код, который сможете использовать для вертикального центрирования.

3.3. Отрицательные значения полей

Вместо того чтобы работать с отступами и управлять шириной границ, можете назначать отрицательные значения полей. Этот способ удобен, например, когда элементы должны перекрывать друг друга или иметь бо́льшую ширину, чем содержащие их контейнеры.

Как поведут себя поля с отрицательными значениями, зависит от того, с какой стороны элемента вы их примените. Результат приведен на рис. 3.15. Если отрицательное значение задано левому или верхнему полю, то элемент перемещается влево или вверх соответственно. Это может привести к тому, что элемент будет перекрывать другой, предшествующий ему в потоке документа. Если отрицательное значение устанавливается для правого или нижнего поля, то элемент не сдвигается, а перекрывается любым последующим элементом. Задать элементу отрицательное значение нижнего поля — то же самое, что и применить к элементу (элементам) под ним отрицательное значение верхнего поля.

Когда блочный элемент не имеет указанной ширины, он, естественно, заполняет ширину своего контейнера. Отрицательное значение правого поля, однако, может изменить это: до тех пор пока не указана ширина, он станет тянуть край элемента вправо, выводя его за пределы контейнера. Задайте равное отрицательное значение левому полю, и обе стороны элемента выйдут за границы контейнера. Этот трюк позволяет изменить размер таблицы, показанной на рис. 3.12, чтобы заполнить ширину элемента `body`, несмотря на пустое пространство между границами.

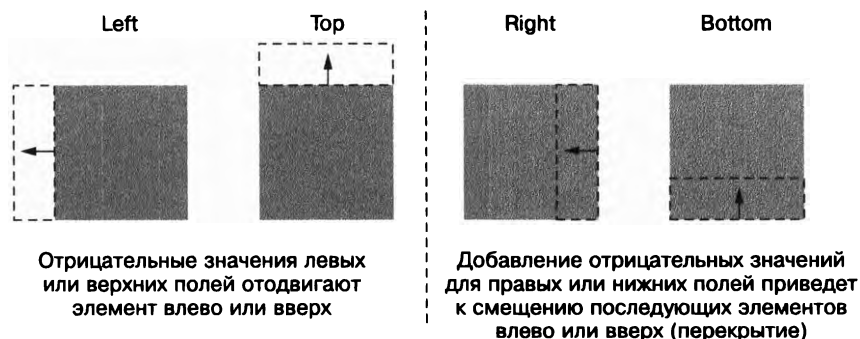


Рис. 3.15. Поведение отрицательных значений полей

ВНИМАНИЕ

Использование отрицательных значений полей для перекрытия элементов даст возможность визуализировать некоторые элементы, не позволяя взаимодействовать с ними, если они расположены под другими элементами.

Отрицательные значения полей редко бывают востребованы, но в некоторых случаях оказываются полезными. В частности, пригождаются при построении макетов с колонками. Убедитесь, что они используются не слишком часто, иначе вы можете быстро потерять контроль над ситуацией на странице.

3.4. Схлопывание полей

Взгляните еще раз на свою страницу. Видите, с полями происходит что-то странное? Вы не применяли никаких полей к шапке или контейнеру, но между ними есть разрыв (рис. 3.16). Почему появился этот промежуток?

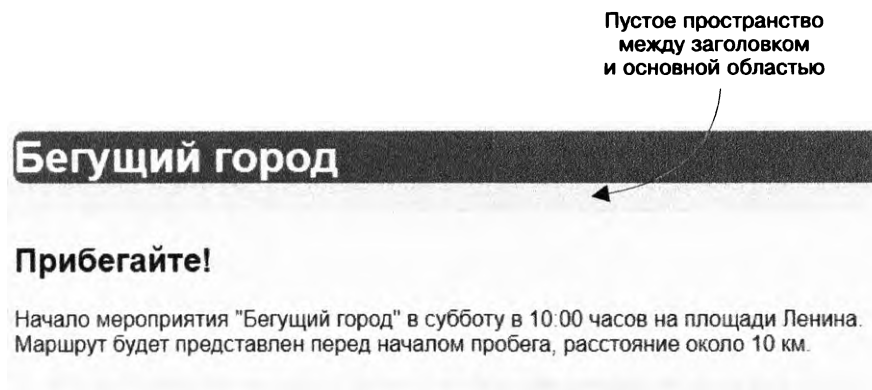


Рис. 3.16. Промежуток, вызванный схлопыванием полей

Когда верхнее и/или нижнее поля прилегают друг к другу, они перекрываются, объединяясь для формирования единого поля. Это называется *схлопыванием*. Пространство под шапкой на рис. 3.16 является результатом схлопывания полей. Посмотрим, как это работает.

3.4.1. Схлопывание между текстом

Основная причина схлопывания полей — это расстояние между текстовыми блоками. Абзацы (p) по умолчанию имеют верхнее поле 1 em и нижнее поле 1 em. Эти значения применяются браузерными правилами стилей. Но когда два абзаца следуют один за другим, их поля не складываются до пустого пространства размером 2 em. Вместо этого они схлопываются и перекрывают друг друга, оставляя только 1 em пустого пространства между абзацами.

Вы увидите схлопнувшееся поле в левой колонке на странице. Заголовок («Прибегайте!») в элементе h2 имеет нижнее поле 0,83 em, которое совмещается с верхним полем следующего абзаца. Их поля показаны на рис. 3.17. Обратите внимание, что поля обоих элементов занимают одно и то же пространство на странице.



Рис. 3.17. Выделенные контуром поля заголовка (слева) и абзаца (справа)

Размер схлопнувшегося поля равен наибольшему из объединенных полей. В примере у заголовка есть нижнее поле 19,92 пиксела ($24 \text{ пиксела (размер шрифта)} \cdot 0,83 \text{ em (поле)}$), а у абзаца — верхнее поле 16 пикселей ($16 \text{ пикселей (размер шрифта)} \cdot 1 \text{ em (поле)}$). Большее значение, 19,92 пиксела, представляет собой видимое пространство между двумя элементами.

3.4.2. Схлопывание многочисленных полей

Смежные элементы не обязательно должны быть родственниками, чтобы их поля схлопывались. Даже если вы обернете абзац дополнительным контейнером div, как показано в листинге 3.13, результат не изменится. В отсутствие каких-либо других вмешательств со стороны CSS все смежные верхние и нижние поля будут схлопываться.

В этом случае схлопываются три поля: нижнее поле заголовка h2, верхнее поле контейнера div и верхнее поле абзаца p. Вычисляемые их значения равны 19,92 пиксела, 0 пикселей и 16 пикселей соответственно, поэтому пространство между элементами составит 19,92 пиксела — самое большое значение из трех. Фактически вы можете вложить абзац внутрь нескольких контейнеров div, и контент все равно будет выглядеть одинаково — все поля схлопнутся.

Листинг 3.13. Абзац, обернутый в контейнер `div`, с тем же результатом

```
<main class="main">
  <h2>Прибегайте!</h2>
  <div>
    <p>
      Начало мероприятия "Бегущий город" в субботу
      в 10:00 часов на площади Ленина. Маршрут будет
      представлен перед началом пробега,
      расстояние около 10 км.
    </p>
  </div>
</main>
```

Поля по-прежнему
схлопываются,
даже если абзац обернут
в другой контейнер `div`

Короче говоря, любые смежные верхние и нижние поля будут схлопываться. Если вы добавите пустой, неотформатированный контейнер `div` (без настройки высоты, границ или отступов) на страницу, его собственные верхние и нижние поля схлопнутся.

ПРИМЕЧАНИЕ

Схлопываются только верхние и нижние поля. С левыми и правыми такого не происходит.

Смежные поля действуют как своего рода личное пространство. Если двум людям на автобусной остановке удобно стоять на расстоянии 2 м друг от друга (каждому), им не нужно отступать на 4 м друг от друга, чтобы оба были счастливы.

Такое поведение обычно означает, что вы можете создавать поля для разных элементов, не заботясь о том, какой контент появляется выше или ниже их. Если вы назначите заголовкам нижнее поле 1,5 em, то получите этот промежуток независимо от того, будет следующий элемент абзацем `p` с верхним полем 1 em или контейнером `div` вообще без верхнего поля. Схлопнувшееся поле между элементами становится больше, только если следующий элемент требует больше пространства.

3.4.3. Схлопывание вне контейнера

Схлопывание трех последовательных полей может заставить вас врасплох. Когда поле элемента схлопывается за пределами его контейнера, это обычно создает нежелательный эффект, если у контейнера есть фон.

Еще раз взгляните на пустое пространство под заголовком на рис. 3.16. Заголовок страницы — элемент `h1` с нижним полем 0,67 em (21,44 пиксела), задаваемым браузерными стилями. Этот заголовок находится внутри элемента `header` без полей. Нижние поля обоих элементов смежные, поэтому они схлопываются, что дает нижнее поле 21,44 пиксела у заголовка. То же самое происходит и с верхними полями обоих элементов.

Возможно, вы предпочитаете, чтобы заголовок `h1` оставался внутри элемента `header`. Но поля не всегда схлопываются именно так, как вы хотите. К счастью,

существует ряд способов предотвратить их нежелательное поведение. Фактически вы уже изменили его для основного раздела страницы: обратите внимание, что поле выше текста «Прибегайте!» не схлопывается вверх, за пределами его контейнера. Это потому, что поля flex-элементов не схлопываются, а вы написали данную часть страницы с помощью flex-элементов.

Отступы обеспечивают и другое решение. Если вы добавите верхние и нижние отступы к заголовку, поля внутри него не будут схлопываться наружу. Раз так, обновим заголовок, чтобы он выглядел как на рис. 3.18, и правильно применим левые и правые отступы. Для этого обновите таблицу стилей в соответствии с листингом 3.14. Вы увидите результат: между заголовком и основным контентом нет поля. Мы вернемся к этому вопросу в ближайшее время.

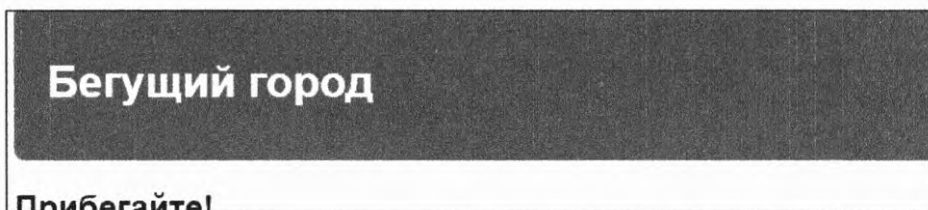


Рис. 3.18. Добавление отступов в заголовок предотвращает схлопывание

Листинг 3.14. Задание отступов для заголовка

```
header {
  padding: 1em 1.5em;
  color: #fff;
  background-color: #0072b0;
  border-radius: .5em;
}
```

Далее перечислены способы предотвращения схлопывания полей.

- ❑ Применение свойства `overflow: auto` (или любого значения, отличного от `visible`) к контейнеру предотвращает схлопывание полей внутри него с полями контента вне контейнера. Это основное решение.
- ❑ Добавление границ или отступов между двумя полями предотвращает схлопывание.
- ❑ Поля не будут схлопываться вне плавающего контейнера, строчного блока или имеющего абсолютное или фиксированное позиционирование.
- ❑ В случае flexbox-верстки поля не будут схлопываться между элементами, являющимися частью flex-макета. Это относится и к CSS-сеткам (см. главу 6).
- ❑ Элементы со значениями `table-cell` не имеют полей, поэтому они не будут схлопываться. Это относится также к элементам `table-row` и большинству других типов отображения таблиц. Исключения — элементы `table`, `table-inline` и `table-caption`.

Однако многие из них влияют на компоновку элементов макета, поэтому вы, вероятно, не захотите применять их, если они не затрагивают желаемый макет.

3.5. Расстояние между элементами в контейнере

Взаимосвязь между отступами контейнера и полями его контента тоже нужно задавать с умом. Поместим несколько элементов на боковую панель и проработаем проблемы, которые могут возникнуть. В конце я опишу полезный прием, способный значительно облегчить вам жизнь.

Вы разместите на боковой панели две кнопки, которые ссылаются на страницы социальных сетей, и одну менее важную ссылку. Боковая панель должна выглядеть как на рис. 3.19.



Рис. 3.19. Боковая панель с правильно позиционированным контентом

Начнем с двух ссылок на социальные сети. Добавьте их на боковую панель, как показано в листинге 3.15. Класс `button-link` отлично сыграет роль селектора CSS.

Листинг 3.15. Добавление двух кнопок социальных сетей на боковую панель

```
<aside class="sidebar">
  <a href="/twitter" class="button-link">мы в Twitter</a>
  <a href="/facebook" class="button-link">мы в Facebook</a>
</aside>
```

Затем отформатируйте с помощью стилей строки, чтобы они стали похожи на кнопки. Сделайте так, чтобы блочные элементы заполнили ширину контейнера и каждый из них отображался в отдельной строке. Добавьте следующий CSS-код (листинг 3.16) в таблицу стилей.

Листинг 3.16. Настройка размера, шрифтов и цвета для кнопок боковой панели

```
.button-link {
  display: block;
  padding: .5em;
  color: #fff;
  background-color: #0090C9;
  text-align: center;
  text-decoration: none;
  text-transform: uppercase;
}
```

← Блочное представление элементов заполняет доступную ширину и помещает каждую ссылку на отдельную строку

Теперь ссылки оформлены правильно, но вам все равно нужно настроить расстояние между ними. Без полей они будут накладываться друг на друга, как сейчас. У вас есть два варианта: можете задать отдельные значения для верхнего и нижнего полей или сразу для обоих, при этом происходит схлопывание полей между двумя кнопками.

Какой бы подход вы ни выбрали, все равно столкнетесь с проблемой: поля должны сочетаться с отступами боковой панели. Если добавите свойство `margin-top: 1.5em`, то получите результат, показанный на рис. 3.20.

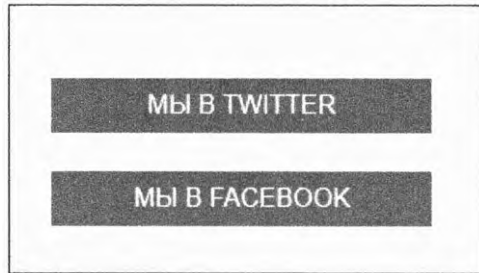


Рис. 3.20. Верхнее поле добавляет расстояние к отступам контейнера

Теперь появилось лишнее пространство в верхней части контейнера. Верхнее поле первой кнопки плюс верхний отступ контейнера создают промежуток, который выглядит непропорционально по сравнению с расстоянием до трех остальных сторон контейнера.

Эту ситуацию можно исправить несколькими способами. В листинге 3.17 показано одно из простых решений. В нем используется комбинатор смежных элементов (+) для настройки элемента `button-link` и его смежных элементов. Теперь поле есть только между двумя кнопками.

Листинг 3.17. Использование комбинатора для установки поля между кнопками

```
.button-link {
  display: block;
  padding: .5em;
  color: #fff;
  background-color: #0090C9;
  text-align: center;
  text-decoration: none;
  text-transform: uppercase;
}

.button-link + .button-link {
  margin-top: 1.5em;
}
```

Применяем только верхнее поле к элементам `button-link`, которые стоят сразу за другим `button-link`

Прием сработал (рис. 3.21). У верхней кнопки больше нет верхнего поля, поэтому расстояния до границ контейнера одинаковы.

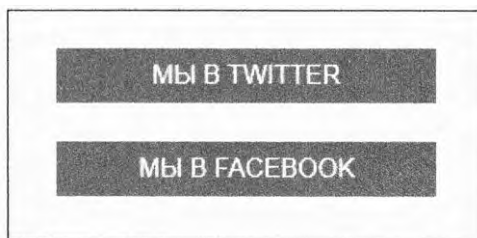


Рис. 3.21. Кнопки на равном расстоянии от границ контейнера

3.5.1. Учет изменения контента

Вы на правильном пути, но проблема с пустым пространством возникает вновь, если поместить больше контента на боковую панель. Добавьте третью ссылку на свою страницу, как показано в листинге 3.18. К ней применяется класс `sponsor-link`, поэтому можете задавать разные стили для ссылок.

Листинг 3.18. Добавление на боковую панель ссылки другого типа

```
<aside class="sidebar">
  <a href="/twitter" class="button-link">
    мы в Twitter
  </a>
  <a href="/facebook" class="button-link">
    мы в Facebook
  </a>
  <a href="/sponsors" class="sponsor-link">
    стать спонсором
  </a>
</aside>
```

Добавляем ссылку другого типа
на боковую панель

Вы отформатируете ее, но вновь нужно будет поднять вопрос о расстоянии между ним и другими кнопками. На рис. 3.22 показано, как будет выглядеть ссылка, *прежде* чем вы исправите поля.



Рис. 3.22. Расстояние между второй кнопкой и нижней ссылкой отсутствует

CSS-код показан в листинге 3.19. Включите его в таблицу стилей. Вероятно, вы собрались добавить верхнее поле к ссылке — не делайте этого. Я покажу вам интересную альтернативу.

Листинг 3.19. Добавление стилей для `sponsor-link`

```
.sponsor-link {  
  display: block;  
  color: #0072b0;  
  font-weight: bold;  
  text-decoration: none;  
}
```

Вы можете добавить верхнее поле, и это будет выглядеть правильно. Но учтите следующее: HTML-код имеет неприятную привычку меняться. В какой-то момент, например в следующем месяце или году, что-то на боковой панели потребуется переместить или заменить. Допустим, спонсорскую ссылку нужно будет перенести выше. Или добавить виджет, чтобы оформлять подписку на рассылку по электронной почте.

Каждый раз, когда что-то меняется, вам придется просматривать эти поля. Потребуется убедиться, что между всеми элементами есть промежутки, но нет излишнего пустого пространства в верхней или нижней части контейнера.

3.5.2. Универсальное решение: селектор лоботомированной совы

Веб-дизайнер Хейдон Пикеринг (Heydon Pickering) однажды сказал, что поля — это как «намазать клеем одну сторону объекта до того, как вы решите его приклеить и определиться с тем, куда именно». Вместо того чтобы фиксировать поля для текущего контента страницы, исправим его способом, который работает независимо от того, как страница будет реструктурирована. Вы сделаете это с помощью механизма, который Пикеринг назвал *селектором лоботомированной совы*. Он выглядит так: `* + *`.

Это универсальный селектор (`*`), который предназначен для всех элементов и за которым следует комбинатор смежных элементов (`+`), а затем еще один универсальный селектор. Селектор получил такое название, потому что он похож на сову с пустым взглядом. Лоботомированная сова не похожа на селектор, который вы использовали ранее: `.social-button + .social-button`. Кроме того, вместо кнопок для настроек, которые стоят сразу после других кнопок, он нацелен на любой элемент, который расположен сразу же за любым другим элементом. То есть он выбирает все элементы на странице, которые не являются первым дочерним элементом их родителя.

Задействуем селектор лоботомированной совы, чтобы добавить верхние поля к элементам на вашей странице. Способ позволит равномерно расположить все элементы на боковой панели. Селектор будет применен также к основному контейнеру, потому что смежный элемент стоит сразу после заголовка, создавая необходимое пространство. Результат показан на рис. 3.23.

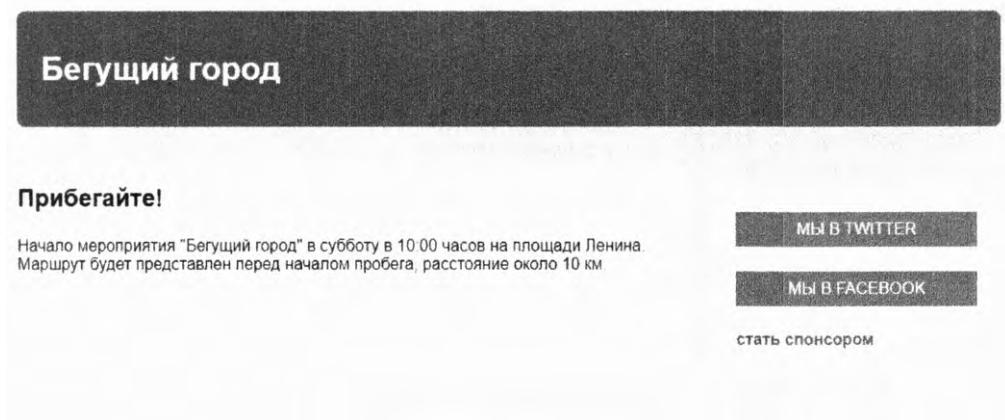


Рис. 3.23. Все смежные родственные элементы теперь имеют верхнее поле

Вставьте код из листинга 3.20 в верхнюю часть таблицы стилей. Я добавил ключевое слово `body` в начало кода селектора. Оно ограничивает селектор — теперь он будет применяться только к элементам внутри тела страницы. Если вы используете селектор лоботомированной совы самостоятельно, он будет нацелен на элемент `body`, потому что тот является смежным родственником элемента `head`.

Листинг 3.20. Селектор лоботомированной совы

```
body * + * {  
    margin-top: 1.5em;  
}
```

ПРИМЕЧАНИЕ

Вас могут беспокоить результаты работы универсального селектора (*). В браузере Internet Explorer 6 он был невероятно медленным, поэтому разработчики избегали его. Сегодня это уже не проблема — современные браузеры хорошо с ним справляются. Кроме того, использование его в селекторе лоботомированной совы способно уменьшить количество селекторов в таблице стилей, поскольку он глобально фиксирует большинство элементов. На самом деле этот способ может быть более результативным, что зависит от особенностей таблицы стилей.

После применения селектора лоботомированной совы на боковой панели возникает нежелательный побочный эффект. Так как она смежна с основной колонкой и родственна ей, она тоже получает верхнее поле. Придется указать для него нулевое значение. Нужно также добавить отступы к основным колонкам, потому что вы еще не сделали этого. Обновите соответствующую часть таблицы стилей, чтобы код соответствовал листингу 3.21.

Листинг 3.21. Заключительные штрихи

```

.main {
  width: 70%;
  padding: 1em 1.5em;
  background-color: #fff;
  border-radius: .5em;
}

.sidebar {
  width: 30%;
  padding: 1.5em;
  margin-top: 0;
  margin-left: 1.5em;
  background-color: #fff;
  border-radius: .5em;
}

```

Добавляет отступ к основной колонке

Удаляет верхнее поле, примененное селектором лоботомированной совы

Это последние штрихи для вашей страницы. Теперь она должна выглядеть так, как на рис. 3.24.

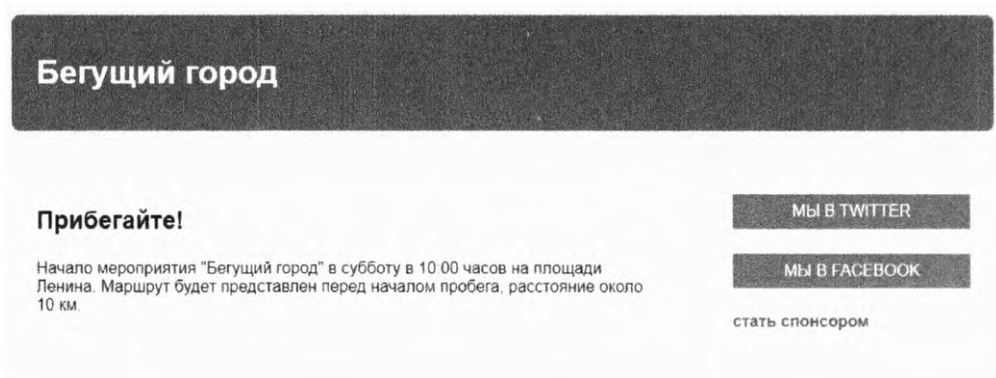


Рис. 3.24. Готовая страница с двухколоночным макетом

Селектор лоботомированной совы — это компромисс. Он упрощает обработку множества полей на всей странице, но вам придется переопределить их в тех местах, где не нужно, чтобы они применялись, — обычно только там, где есть элементы, расположенные бок о бок, как в многоколоночном макете. В зависимости от дизайна необходимо также задать желаемые поля для абзацев и заголовков.

Мы используем селектор лоботомированной совы в нескольких примерах в последующих главах, чтобы вы оценили компромиссы. Этот селектор не всегда правильное решение для проекта. К тому же его сложно добавить в существующий проект, не нарушая макет, так что учитывайте это при запуске нового сайта или веб-приложения. Далее приведена полная таблица стилей (листинг 3.22).

Листинг 3.22. Итоговая таблица стилей

```
:root {
  box-sizing: border-box;
}

*,
::before,
::after {
  box-sizing: inherit;
}

body {
  background-color: #eee;
  font-family: Helvetica, Arial, sans-serif;
}

body * + * {
  margin-top: 1.5em;
}

header {
  padding: 1em 1.5em;
  color: #fff;
  background-color: #0072b0;
  border-radius: .5em;
}

.container {
  display: flex;
}

.main {
  width: 70 %;
  padding: 1em 1.5em;
  background-color: #fff;
  border-radius: .5em;
}

.sidebar {
  width: 30 %;
  padding: 1.5em;
  margin-top: 0;
  margin-left: 1.5em;
  background-color: #fff;
  border-radius: .5em;
}

.button-link {
  display: block;
```

```
padding: .5em;
color: #fff;
background-color: #0090C9;
text-align: center;
text-decoration: none;
text-transform: uppercase;
}

.sponsor-link {
display: block;
color: #0072b0;
font-weight: bold;
text-decoration: none;
}
```

Итоги главы

- ❑ Всегда работайте с универсальным значением **border-box**, чтобы размеры элементов были предсказуемыми.
- ❑ Избегайте явной установки высоты элемента во избежание проблем из-за переполнения.
- ❑ Применяйте современные методы компоновки, такие как **display: table**, либо flex-контейнеры для создания колонок равной высоты или вертикально центрированного контента.
- ❑ Если поля ведут себя странно, выполните нужные действия, чтобы предотвратить их схлопывание.
- ❑ Попробуйте использовать на своей странице селектор лоботомированной совы, чтобы задать поля между смежными элементами.

Часть II

Разметка

Технология CSS предоставляет несколько инструментов для управления разметкой веб-страницы. В части II (главы 4–8) мы рассмотрим наиболее важные из этих инструментов — от плавающих элементов до flex-контейнеров и позиционирования. Ни один из них не обладает принципиальным преимуществом над остальными, они скорее позволяют выполнять различные задачи. Я покажу вам, как работает каждый из этих инструментов, чтобы вы могли совмещать их на своей странице и достигать необходимых результатов.

4

Плавающие элементы

В этой главе

- Как работают плавающие элементы, и как избежать распространенных ошибок с ними.
- Схлопывание контейнера и очистка потока с помощью метода `clearfix`.
- Медиаобъекты и шаблон двойного контейнера.
- Блочный контекст форматирования.
- Создание и анализ CSS-сеток.

В конце части I мы обсудили несколько фундаментальных концепций задания размеров элементов и промежутков между ними. В части II продолжим и углубим их обсуждение, более подробно рассмотрев основные методы разметки страницы. Поговорим о трех важнейших методах изменения потока документа: плавающих элементах, флекс-контейнерах и CSS-сетках. Флекс-контейнеры и CSS-сетки используются в CSS не так давно, но уже доказали свою значимость как обязательные инструменты. Несмотря на то что плавающие элементы и позиционирование не представляют собой ничего нового, зачастую их неправильно воспринимают.

В этой главе в первую очередь рассмотрим плавающие элементы — старейший метод разметки веб-страницы, единственный на протяжении многих лет. Впрочем, плавающие элементы немного странные. Начинать разбираться в них следует с изучения их изначального предназначения. Так и поступим. Я покажу, как справиться с некоторыми их странностями, в том числе с применением инструмента `clearfix`. Это принесет в поведение плавающих элементов некую конкретику.

По ходу обсуждения вы также узнаете о двух шаблонах, которые часто встречаются в макетах страниц: двойном контейнере и медиаобъекте. Закрепить знания вам поможет создание CSS-сетки — универсального инструмента для структуризации страницы.

4.1. Предназначение плавающих элементов

Несмотря на то что плавающие элементы не были предназначены для конструирования макета страницы, они хорошо справлялись с этой работой. Однако, чтобы понять смысл плавающих элементов, мы должны разобраться в их изначальном предназначении.

Плавающий элемент — объект (зачастую изображение), выровненный по одной из сторон контейнера, что позволяет потоку документа обтекать его (рис. 4.1). Такая компоновка часто встречается в газетах и журналах, поэтому для достижения описанного эффекта плавающие элементы были добавлены и в CSS.

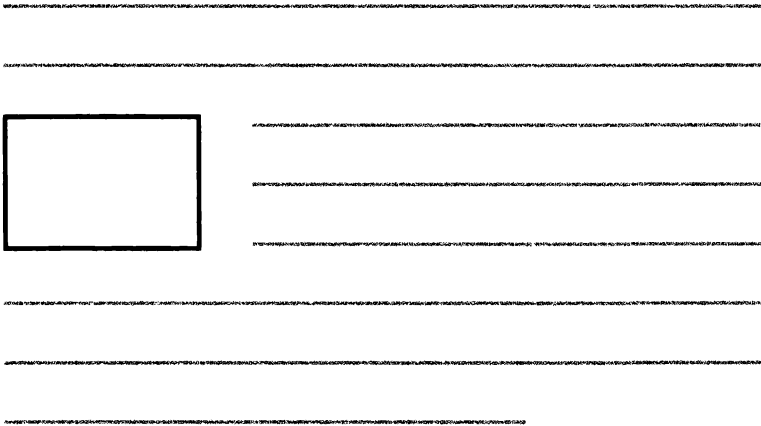


Рис. 4.1. Строки текста обтекают плавающий элемент

На этой иллюстрации элемент выровнен по левому краю, но вы могли бы разместить его и у правого края. Плавающий элемент извлекается из нормального потока документа и выравнивается по одной из сторон контейнера, после чего поток документа станет обтекать участок, где теперь располагается этот элемент. Если вы выравниваете два плавающих элемента по одной стороне, то они выстраиваются один за другим (рис. 4.2).

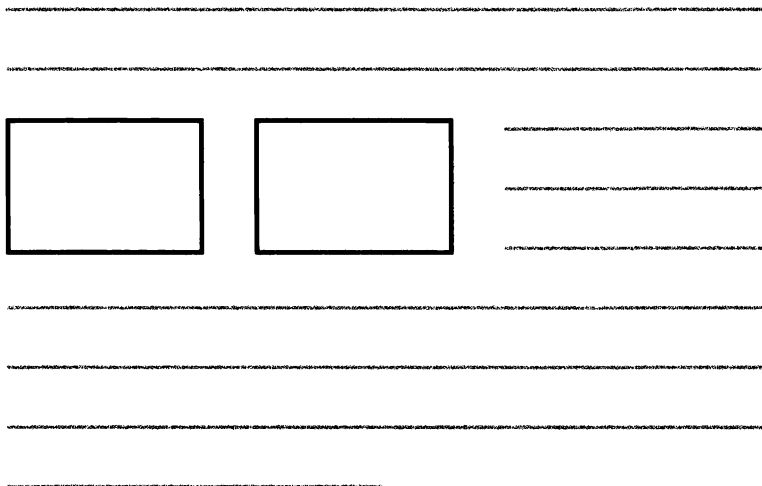


Рис. 4.2. Два плавающих элемента, выстроившихся один за другим

Если вы уже какое-то время пишете CSS-код, то такое поведение для вас не новость, однако нужно заметить: не всегда плавающие элементы применяются таким образом, несмотря на то что в этом и есть их первоначальное предназначение.

Уже в первые дни существования CSS разработчики поняли, что могут использовать этот простой инструмент для перемещения разделов страницы при верстке всех видов макетов. Плавающие элементы не были инструментом компоновки страниц, но почти два десятилетия мы задействовали их именно для этих целей.

Мы делали это, потому что это был единственный вариант. В конце концов появилась возможность задействовать свойства `display: inline-block` или `display: table` — альтернативные, хотя и весьма ограниченные. До последнего времени, пока не были добавлены flex-контейнеры и CSS-сетки, плавающие элементы оставались козырем при верстке макетов страниц. Посмотрим, как они работают. Для примера создайте страницу, показанную на рис. 4.3.

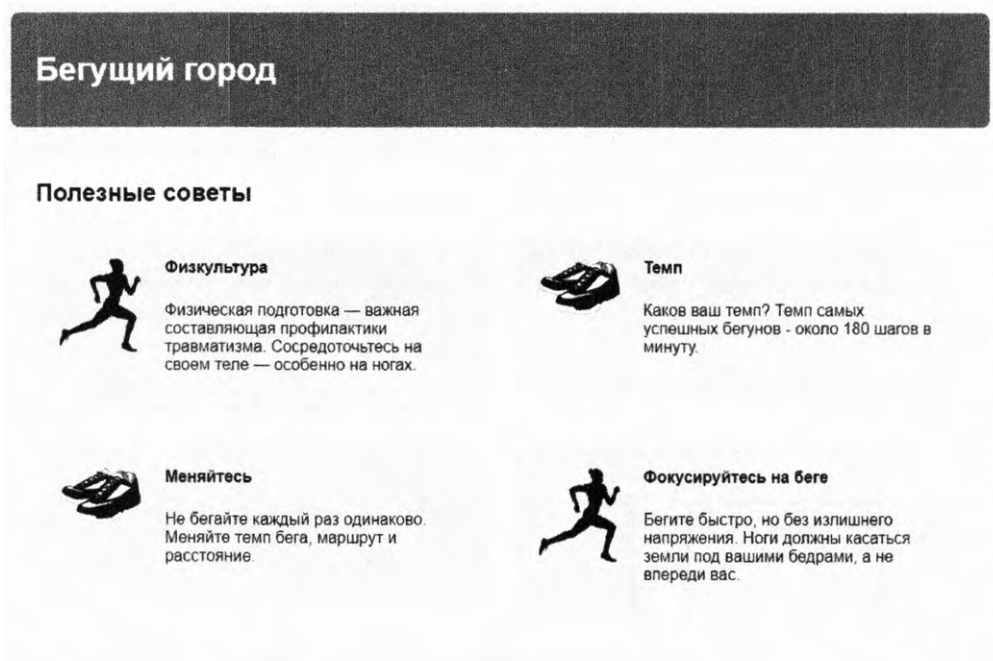


Рис. 4.3. Веб-страница с макетом, основанным на плавающих элементах

В примерах из этой главы вы станете использовать плавающие элементы для позиционирования каждого из четырех серых полей. А внутри полей расположите плавающие изображения рядом с текстом. Создайте пустую страницу и подключите ее к созданной ранее таблице стилей, затем добавьте на страницу код, приведенный в листинге 4.1.

Листинг 4.1. HTML-код страницы с разметкой, основанной на плавающих элементах

```

<body>
  <div class="container">
    <header>
      <h1>Бегущий город</h1>
    </header>

    <main class="main clearfix">
      <h2>Полезные советы</h2>

      <div>
        <div class="media">
          
          <div class="media-body">
            <h4>Физкультура</h4>
            <p>
              Физическая подготовка &mdash; важная составляющая
              профилактики травматизма. Сосредоточьтесь
              на своем теле &mdash; особенно на ногах.
            </p>
          </div>
        </div>

        <div class="media">
          
          <div class="media-body">
            <h4>Темп</h4>
            <p>
              Каков ваш темп? Темп самых успешных бегунов —
              около 180 шагов в минуту.
            </p>
          </div>
        </div>

        <div class="media">
          
          <div class="media-body">
            <h4>Меняйтесь</h4>
            <p>
              Не бегайте каждый раз одинаково. Меняйте темп
              бега, маршрут и расстояние.
            </p>
          </div>
        </div>

        <div class="media">
          
          <div class="media-body">
            <h4>Фокусируйтесь на беге</h4>
            <p>
              Бегите быстро, но без излишнего напряжения.
              Ноги должны касаться земли под вашими бедрами,
              а не впереди вас.
            </p>
          </div>
        </div>
      </div>
    </main>
  </div>

```

Разметка шапки аналогична разметке из главы 3

Основной элемент — белое поле, содержащее большую часть страницы

Четыре медиаобъекта для каждого из серых полей


```

        </p>
      </div>
    </div>

    </div>
  </main>
</div>
</body>

```

Этот листинг дает структуру страницы: шапку и основной элемент, содержащий весь материал страницы. Внутри основного элемента находятся название страницы, а также *анонимный контейнер* `div`, то есть элемент `div`, не имеющий класса или идентификатора. Это позволяет сгруппировать четыре серых медиаобъекта, в элементе тела каждого из которых есть изображение.

СОВЕТ

Обычно проще сначала разметить большие разделы страницы, а затем приступить к работе над более мелкими, находящимися внутри крупных.

Прежде чем начать работать с плавающими элементами, нужно выстроить внешнюю структуру страницы. Добавьте листинг 4.2 в файл таблицы стилей.

Листинг 4.2. Базовые стили для страницы

<pre> :root { box-sizing: border-box; } *, ::before, ::after { box-sizing: inherit; } body { background-color: #eee; font-family: Helvetica, Arial, sans-serif; } body * + * { margin-top: 1.5em; } </pre>	<div style="border-left: 1px solid black; padding-left: 10px;"> <p>Глобальное исправление свойства <code>border-box</code> (из главы 3)</p> </div>
<pre> header { padding: 1em 1.5em; color: #fff; background-color: #0072b0; border-radius: .5em; margin-bottom: 1.5em; } </pre>	<div style="border-left: 1px solid black; padding-left: 10px;"> <p>Глобальные поля с помощью селектора лоботомированной совы (из главы 3)</p> <p>Цвета и отступы шапки</p> </div>

```
.main {  
  padding: 0 1.5em;  
  background-color: #fff;  
  border-radius: .5em;  
}
```

Цвета и отступы
основного (белого)
контейнера

Этот код позволяет установить некоторые базовые стили для страницы, в том числе исправление свойства `box-sizing` и селектор лоботомированной совы (см. главу 3). Далее необходимо ограничить ширину контента страницы (рис. 4.4). Обратите внимание на светло-серые поля по обеим сторонам, а также на то, что шапка и основной контейнер одинаковой ширины.

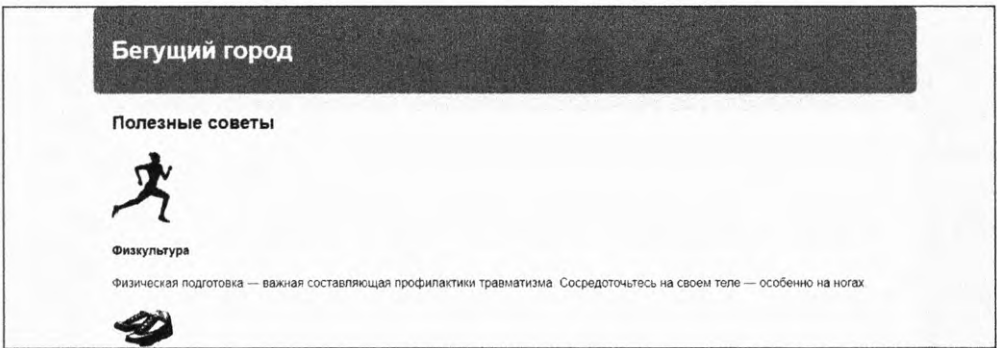


Рис. 4.4. Страница ограниченной ширины

Такая разметка часто используется для центрирования контента страницы. Вы можете добиться этого, расположив контент внутри двух вложенных контейнеров и настроив поля внутреннего контейнера так, чтобы он не выходил за пределы внешнего (рис. 4.5). Веб-разработчик Брэд Вестфол (Brad Westfall) называет такой подход *шаблоном двойного контейнера*.

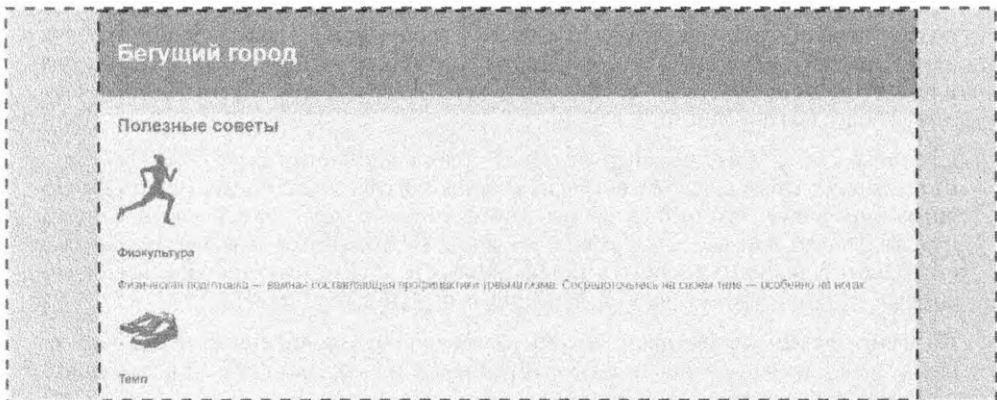


Рис. 4.5. Шаблон двойного контейнера

В нашем примере элемент `body` играет роль внешнего контейнера. По умолчанию данный элемент занимает 100 % ширины страницы, так что вам не придется применять к нему новые стили. Мы упаковали весь контент страницы внутрь этого элемента, в элемент `<div class="container">`, играющий роль внутреннего контейнера. Добавьте код из листинга 4.3 в свою таблицу стилей.

Листинг 4.3. Стили двойного контейнера

```
.container {  
  max-width: 1080px;  
  margin: 0 auto;  
}
```

Ограничение максимальной ширины в 1080 пикселей

Автоматизация увеличения левого и правого полей для заполнения доступного пространства и центрирования элемента относительно внешнего контейнера

Благодаря использованию свойства `max-width` вместо свойства `width` элемент сжимается до значений менее 1080 пикселей, если разрешение области просмотра ниже этого значения. Иными словами, контейнер заполнит меньшие области просмотра, но в больших областях просмотра он расширится до 1080 пикселей. Это важно для того, чтобы избежать горизонтального прокручивания на устройствах с небольшой областью просмотра.

Вам все еще нужно знать, как работать с плавающими элементами?

Flexbox-верстка быстро замещает плавающие элементы при разметке страницы. Начинаям разработчикам поведение этого инструмента кажется более прямолинейным и предсказуемым. Задайтесь вопросом: а нужно ли вам знать о плавающих элементах? Не оставила ли технология CSS этот этап развития позади?

С современными браузерами вы способны добиться лучших результатов без плавающих элементов, чем было возможно раньше. Вероятно, сможете не использовать плавающие элементы вовсе. Но если нужно обеспечить поддержку браузера Internet Explorer, то, наверное, не стоит забывать о плавающих элементах. Flex-контейнеры поддерживаются только в браузерах Internet Explorer 10 и 11, но даже в этих версиях не обходится без ошибок. Если вы не хотите переживать по поводу ошибок браузеров или вам требуется поддержка старых браузеров, то плавающие элементы могут оказаться более подходящими.

На старых сайтах наверняка используются плавающие элементы. Чтобы обеспечить их поддержку, вы должны знать, как они работают. Кроме того, макеты, основанные на плавающих элементах, требуют меньшего количества кода, тогда как новым методам нужны дополнительные контейнерные элементы. Если вы лишь частично контролируете разметку, форматированием которой занимаетесь, то плавающие элементы могут обеспечить более эффективное выполнение необходимых действий.

Ко всему прочему, плавающие элементы — до сих пор единственный метод перемещения изображения к одной из сторон страницы, к тому же позволяющий выполнить обтекание этого изображения текстом.

4.2. Схлопывание контейнера и clearfix

В прошлом ошибки браузеров искажали поведение плавающих элементов, впрочем, это было характерно в основном для Internet Explorer 6 и 7. Я почти убежден, что вам больше не нужно поддерживать эти браузеры, поэтому не следует и переживать по поводу этих ошибок. Теперь вы можете быть уверены в том, что браузеры будут правильно обрабатывать плавающие элементы.

Однако некоторые особенности поведения плавающих элементов могут заставить вас врасплох и сегодня. Это не ошибки, плавающие элементы ведут себя именно так, как должны. Рассмотрим, как они работают и как можно настроить их поведение для создания нужной разметки.

4.2.1. Что такое схлопывание контейнера

Выворачиваем четыре плавающих медиаобъекта на странице по левому краю — и проблемы не заставят себя долго ждать (рис. 4.6).



Рис. 4.6. Контейнер с четырьмя плавающими потомками

Что случилось с белым фоном? Мы видим его позади названия страницы («Полезные советы»), однако этим он и ограничивается, вместо того чтобы расширяться вниз и включать все медиаобъекты. Чтобы увидеть это на своей странице, добавьте код из листинга 4.4 в таблицу стилей. Затем рассмотрим, почему так происходит и как это исправить.

Листинг 4.4. Выравнивание четырех плавающих элементов по левому краю

```
.media {  
  float: left;  
  width: 50%;  
  padding: 1.5em;  
  background-color: #eee;  
  border-radius: 0.5em;  
}
```

Выравнивание каждого из четырех плавающих элементов по левому краю

Установка ширины так, чтобы по горизонтали вмещались два блока

Мы установили светло-серый фон для каждого медиаобъекта, ожидая, что белый фон контейнера окажется позади (или, точнее, вокруг) них. Однако вместо этого белый фон исчезает у верхнего края медиаобъектов. Почему так?

Проблема в том, что, в отличие от элементов нормального потока документа, плавающие элементы не добавляют высоту родительским элементам. Это может показаться странным, но данное свойство восходит к изначальному предназначению плавающих элементов.

Как вы узнали в начале этой главы, плавающие элементы предназначены для того, чтобы текст мог обтекать их. Когда вы помещаете плавающее изображение в абзац текста, этот абзац не увеличивается, чтобы уместить изображение. А значит, если высота изображения больше высоты абзаца, то текст следующего абзаца начнется сразу же под текстом первого абзаца, следовательно, текст обоих абзацев будет обтекать данное плавающее изображение. Описанное проиллюстрировано на рис. 4.7.

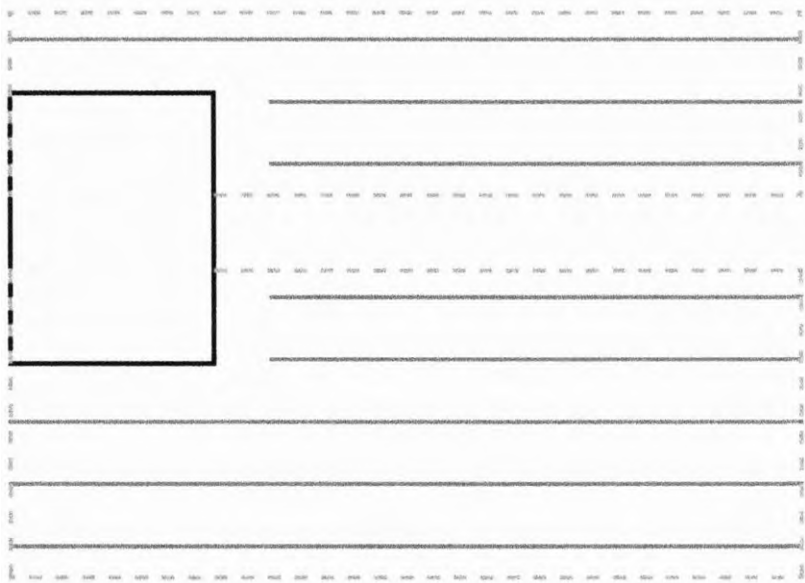


Рис. 4.7. Плавающий элемент из одного контейнера распространяется на следующий контейнер, что позволяет находящемуся в них тексту обтекать этот элемент (контейнеры выделены пунктирными линиями)

На вашей странице все расположенное внутри основного элемента, за исключением названия страницы, представляет собой плавающие элементы. Таким образом, только название вносит вклад в высоту контейнера, оставляя все плавающие медиа-объекты расширяться вниз с выходом за границы белого фона основного контейнера. Для нас такое поведение нежелательно, поэтому исправим его. Основной элемент должен расширяться вниз и содержать серые блоки (рис. 4.8).

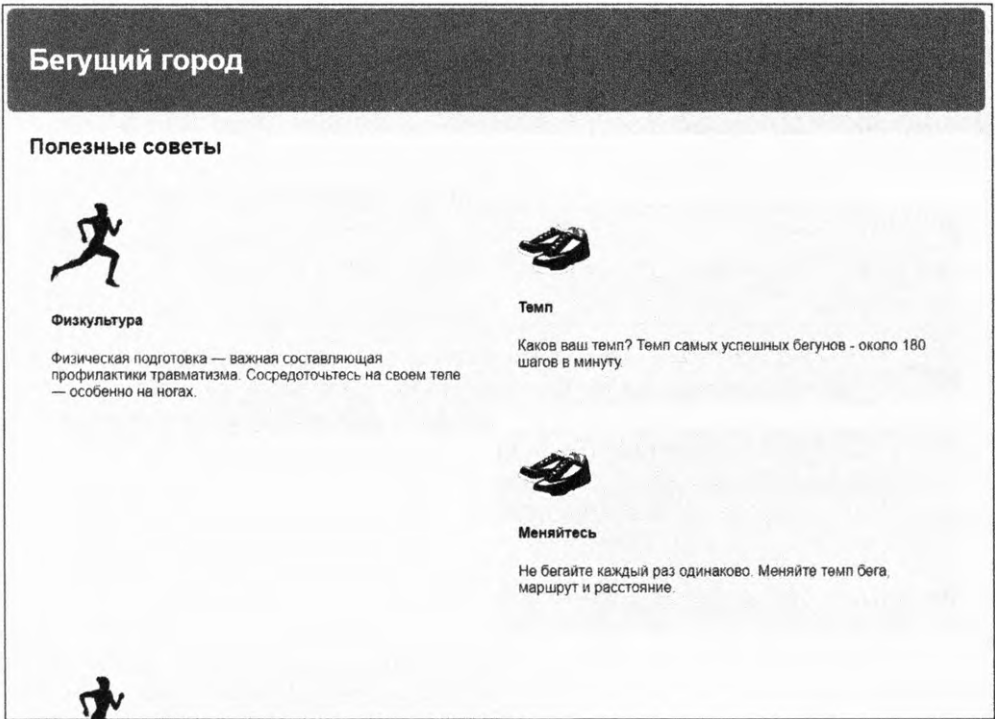


Рис. 4.8. Контейнер расширился, чтобы уместить плавающие элементы

Один из способов корректировки — использование `clear` — смежного свойства плавающего элемента. Если вы поместите элемент в конец основного контейнера и воспользуетесь свойством `clear`, это приведет к тому, что контейнер расширится до нижнего края плавающих элементов. Код в листинге 4.5, в принципе, иллюстрирует то, что нужно сделать. Вы можете на время добавить его на свою страницу, чтобы посмотреть, как это работает.

Листинг 4.5. Контейнер расширяется, чтобы уместить элемент, сбрасывающий обтекание плавающих элементов

```
<main class="main clearfix">
```

```
...
```

```
  <div style="clear: both"></div>
```

```
</main>
```

Добавляется пустой контейнер `div` со свойством `clear` в конец основного контейнера

Объявление `clear:both` приводит к тому, что этот элемент смещается ниже плавающих элементов, вместо того чтобы расположиться рядом с ними. Вы можете присвоить данному свойству одно из значений, `left` или `right`, для сброса обтекания только тех плавающих элементов, которые выровнены по левому или правому краю соответственно.

Это позволяет установить нужный размер, однако такой подход немного похож на хакерство, так как добавляет нежелательную разметку в HTML-код, из-за чего тот берет на себя функции CSS. Поэтому следует удалить пустой контейнер `div`. Рассмотрим способ выполнить это же задание с помощью только CSS-кода.

4.2.2. Что такое `clearfix`

Вместо добавления еще одного контейнера `div` в разметку воспользуемся *псевдоэлементом*. С помощью селектора псевдоэлемента `::after` можно эффективно вставить элемент в DOM в конце контейнера без добавления его в разметку.



Псевдоэлементы — специальные селекторы, воздействующие на разные части документа. Их объявление начинается с двойного двоеточия (`::`), причем, в целях обратной совместимости многие браузеры поддерживают синтаксис с одним двоеточием. Самые распространенные псевдоэлементы — это `::before` и `::after` для вставки контента в начало или конец элемента (см. приложение А для получения дополнительных сведений).

В листинге 4.6 показан общий подход к решению проблемы умещения плавающих элементов, называемый *clearfix*. (Некоторые разработчики предпочитают сокращать имя класса до `cf`, что удобно, так как одновременно это сокращение фразы `containing floats` — умещение плавающих элементов.) Добавьте следующий код в свою таблицу стилей.

Листинг 4.6. Использование метода `clearfix` для умещения плавающих элементов

```
.clearfix::after {
  display: block;
  content: " ";
  clear: both;
}
```

Нацеливается на псевдоэлемент, находящийся в конце контейнера

Свойства приводят к появлению псевдоэлемента в документе

Псевдоэлемент сбрасывает обтекание всех плавающих элементов в контейнере

Важно знать, что класс `clearfix` применяется к элементу, содержащему плавающие элементы. Распространенная ошибка — применение данного класса к неподходящему элементу, например непосредственно к плавающему элементу или контейнеру, стоящему после контейнера с плавающими элементами.

ПРИМЕЧАНИЕ

За время своего существования метод clearfix прошел десятки итераций, некоторые из них были сложнее предыдущих. Многие версии включали нюансы, корректирующие различные ошибки браузеров. В большинстве таких изменений необходимости больше нет, однако приведенный пример все же включает одну корректировку — пробел в качестве значения свойства content. Пустая строка ("") сработала бы так же хорошо, однако добавление пробела исправляет незаметную ошибку в старых версиях браузера Opera. Я предпочитаю использовать это исправление, так как оно неdestructивно.

Впрочем, у метода clearfix остается одна неувязка: поля вложенных плавающих элементов не будут схлопываться вне контейнера с примененным clearfix, а поля неплавающих элементов схлопнутся, как обычно. Вы видите это на своей странице, где заголовок «Полезные советы» плотно прижат к верхнему краю белого элемента main (см. рис. 4.8): его поле схлопнулось вне контейнера.

Иногда разработчики предпочитают использовать модифицированную версию метода clearfix, которая будет вмещать все поля и окажется более предсказуемой. Добавление этой версии на вашу страницу предотвратит схлопывание верхнего поля названия страницы вне элемента main (рис. 4.9), что оставит нужное расстояние над заголовком.

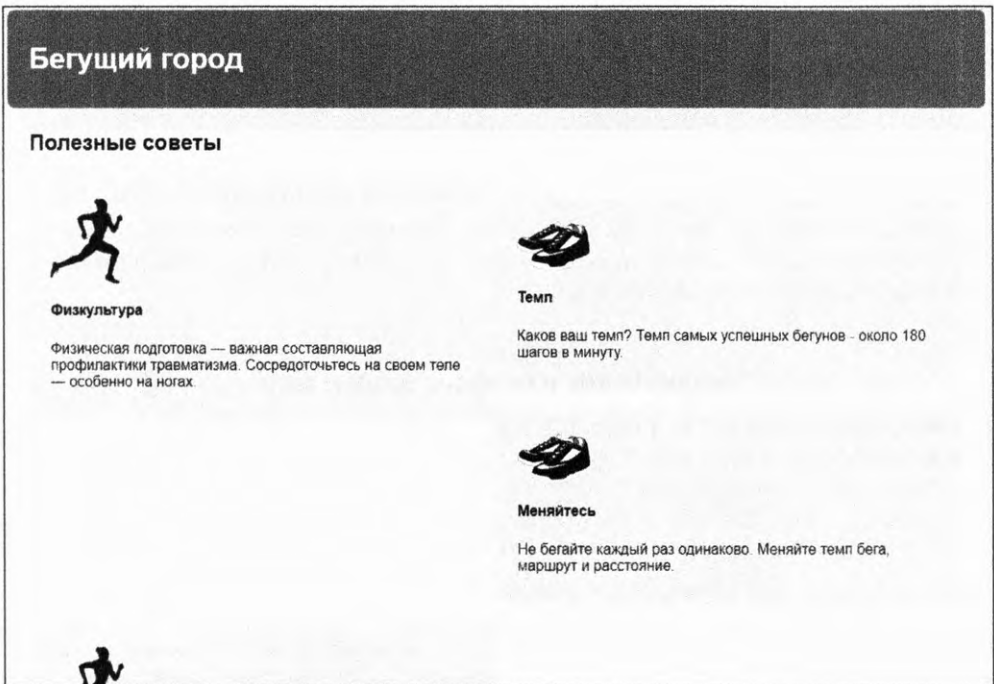


Рис. 4.9. Модифицированная версия метода clearfix вмещает все плавающие элементы и поля. Обратите внимание на то, что теперь верхнее поле заголовка «Полезные советы» содержится внутри белого элемента main

Для использования модифицированной версии обновите код метода `clearfix` в своей таблице стилей, как показано в листинге 4.7.

Листинг 4.7. Модификация метода `clearfix` для умиротворения всех полей

<pre>.clearfix::before, .clearfix::after { display: table; content: " "; }</pre>	<div style="border-left: 1px solid black; padding-left: 5px; margin-bottom: 10px;"> Приводит к появлению псевдоэлементов <code>::before</code> и <code>::after</code> </div> <div style="border-left: 1px solid black; padding-left: 5px;"> С помощью псевдоэлементов предотвращает схлопывание полей </div>
<pre>.clearfix::after { clear: both; }</pre>	<div style="border-left: 1px solid black; padding-left: 5px;"> Только псевдоэлемент <code>::after</code> нуждается в сбросе обтекания плавающих элементов </div>

В этой версии используется свойство `display: table`, а не `display: block`. Применяв его к обоим псевдоэлементам, `::before` и `::after`, вы уместите любые поля дочерних элементов, располагающихся в верхней и нижней частях контейнера. Во врезке «Метод `clearfix` и свойство `display: table`» ниже более подробно объясняется, как это работает.

СОВЕТ

Данная версия метода `clearfix` также предотвращает нежелательное схлопывание полей.

Вам решать, какую версию метода `clearfix` использовать в своих проектах. Некоторые разработчики приводят такой аргумент: схлопывание полей — это фундаментальная особенность CSS, поэтому они предпочитают не уместить поля в контейнер. Но поскольку ни одна версия не уместит поля плавающих элементов, другие разработчики предпочитают более отлаженное поведение модифицированной версии. У каждого аргумента есть свой резон.

Метод `clearfix` и свойство `display: table`

Применение свойства `display: table` в методе `clearfix` уместит поля благодаря некоторым особенностям CSS. Создание табличного элемента (или в данном случае псевдоэлемента) неявно создает строку таблицы внутри этого элемента, а также ячейку таблицы в строке. Так как поля не схлопываются через табличные элементы (как упоминалось в главе 3), поля не будут схлопываться и через табличный псевдоэлемент.

Может показаться, что при использовании свойства `display: table-cell` вы получили бы тот же эффект. Однако свойство `clear` срабатывает только применительно к блочным элементам, коим ячейка таблицы не является. Таким образом, свойство `clear` нельзя применять со свойством `display: table-cell`. В общем, чтобы уместить поля, сбросить обтекание всех плавающих элементов и подразумеваемой ячейки, воспользуйтесь свойством `display: table`.

4.3. Неожиданный «захват» плавающего элемента

Теперь, когда белый контейнер содержит плавающие медиаобъекты, становится очевидной другая проблема: четыре медиаобъекта не образуют две одинаковые строки, как нам нужно. Вместо этого первые два блока («Физкультура» и «Темп») стоят в одной строке, как и ожидалось, но третий блок («Меняйтесь») расположен справа, под вторым блоком. При такой компоновке под первым блоком остается большой промежуток, это происходит из-за того, что браузер размещает плавающие элементы как можно выше.

На рис. 4.10 приведена упрощенная схема.

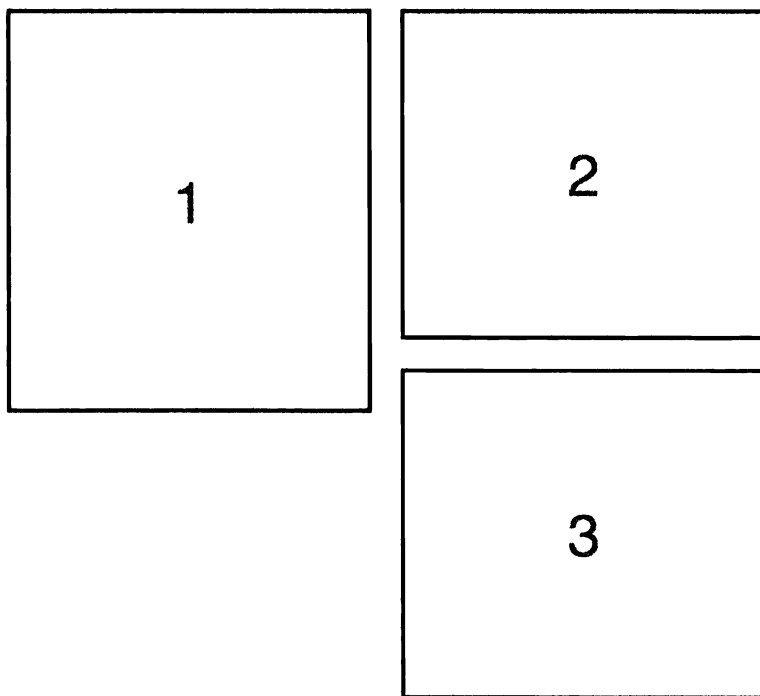


Рис. 4.10. Три плавающих блока, выровненные по левому краю: блок 3 не выравнивается по левому краю, если высота блока 1 превышает высоту блока 2

Так как блок 2 ниже блока 1, для блока 3 попросту нет места под блоком 1. Вместо обхода блока 1 блок 3 «захватывает» его. Иными словами, блок 3 не выравнивается по левому краю, а обтекает нижний угол блока 1.

Нюансы этого поведения зависят от высоты каждого плавающего блока. Даже разница в 1 пиксел может вызвать эту проблему. В то же время если блок 1 короче блока 2, то для блока 3 не будет края, за который он мог бы зацепиться, и вы не встретитесь

с описанной проблемой до тех пор, пока не поменяется контент, что приведет к изменению высоты элементов.

Вываливая несколько плавающих элементов по одному краю, вы можете получить любой из множества вариантов компоновки в зависимости от высоты каждого блока. Даже изменение ширины окна браузера способно все перестроить, поскольку повлияет на обтекание строк текста и, соответственно, изменит высоту элементов. Мы же на странице хотим увидеть по два плавающих блока на строку (рис. 4.11).

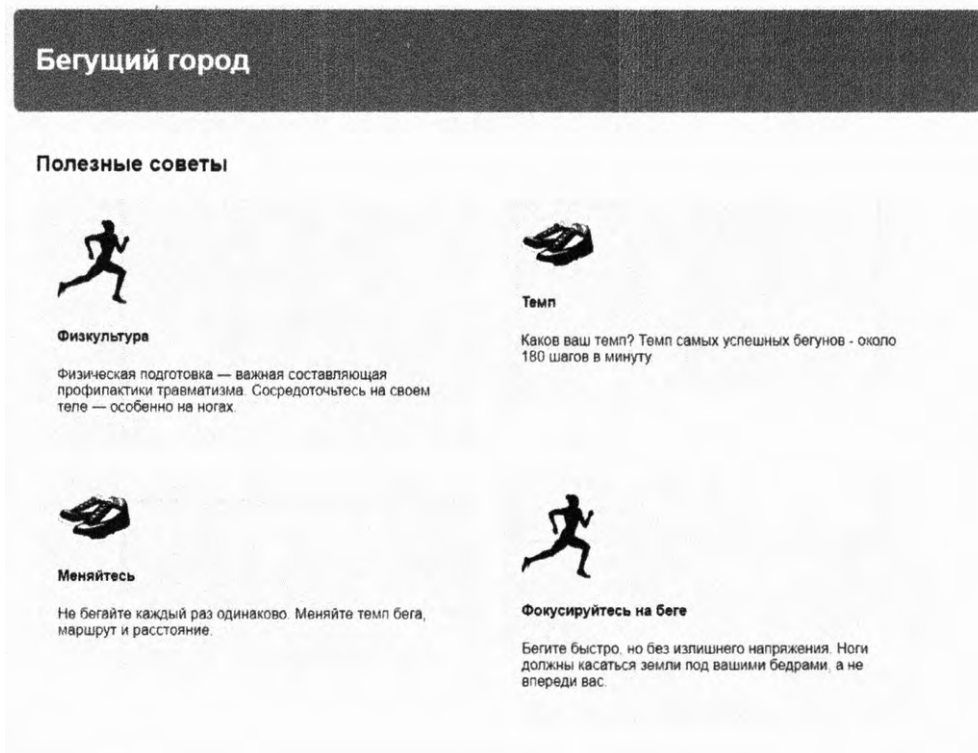


Рис. 4.11. Два элемента на строку: вторая строка медиаобъектов должна сбрасывать обтекание элементов из первой строки

Исправить эту проблему просто: третьему плавающему элементу необходимо сбрасывать обтекание плавающих элементов, расположенных над ним. Или, иными словами, первый элемент в каждой строке должен сбрасывать обтекание плавающего элемента, расположенного над ним. Так как у вас по два блока в строке, то потребуются, чтобы каждый нечетный элемент сбрасывал обтекание вышестоящей строки. Вы можете адресно выбрать такие элементы с помощью селектора псевдокласса `:nth-child()`. Добавьте в таблицу стилей следующий набор правил (листинг 4.8).

Листинг 4.8. Использование селектора `:nth-child()` для нацеливания на нечетные медиаобъекты

```
.media {  
  float: left;  
  width: 50%;  
  padding: 1.5em;  
  background-color: #eee;  
  border-radius: 0.5em;  
}
```

```
.media:nth-child(odd) {  
  clear: left;  
}
```

Теперь каждая строка сбрасывает
отбегание предшествующей строки

Этот код будет работать, даже если позднее вы добавите на страницу новые элементы. Код применяется к первому, третьему, пятому элементам и т. д. Если бы требовалось разместить по три элемента в строке, то вы могли бы нацелить селектор на каждый третий элемент: `.media:nth-child(3n+1)` (см. приложение А для получения дополнительной информации об использовании селектора `:nth-child`).

ПРИМЕЧАНИЕ

Техника сброса отбегания любой строки работает только тогда, когда известно, сколько элементов будет находиться в каждой строке. Если ширина устанавливается не в процентах, то количество элементов может различаться в зависимости от ширины области просмотра. В этом случае лучше воспользоваться другой техникой разметки страницы, например флекс-контейнерами или строчно-блочными элементами.

Добавим поля для медиаобъектов, чтобы между ними образовался зазор. Селектор лоботомированной совы также задаст верхнее поле всем элементам, за исключением первого. Это вызовет сбой выравнивания элементов в первой строке, поэтому потребуется сбросить верхнее поле этих элементов. Обновите свою таблицу стилей, как показано в листинге 4.9.

Листинг 4.9. Добавление полей к медиаобъектам

```
.media {  
  float: left;  
  margin: 0 1.5em 1.5em 0;  
  width: calc(50% - 1.5em);  
  padding: 1.5em;  
  background-color: #eee;  
  border-radius: 0.5em;  
}
```

Добавляет правое и нижнее поля
к каждому элементу

Вычитает значение поля из ширины
во избежание нежелательного
отбегания строки

```
.media:nth-child(odd) {  
  clear: left;  
}
```

После того как к элементам добавили поля, они больше не умещаются по два в строку, поэтому потребуется вычестить соответствующее значение из ширины элемента с помощью функции `calc()`.

4.4. Медиаобъекты и блочный контекст форматирования

Теперь, когда залиты все четыре серых блока, посмотрим на их контент. В нашем образце дизайна страницы изображение располагается с одной стороны, а блок текста — рядом с ним (рис. 4.12). Это еще один распространенный шаблон верстки макетов страниц, который веб-разработчик Николь Салливан (Nicole Sullivan) назвала медиаобъектом.

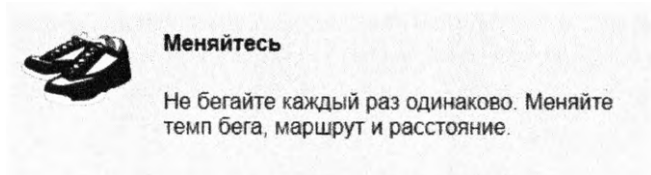


Рис. 4.12. Шаблон медиаобъекта: изображение слева, описательный контент — справа

Данный шаблон может быть реализован несколькими способами, в том числе посредством флекс-контейнера и табличного дизайна, но мы сделаем это с помощью плавающих элементов. Разметка для одного из медиаобъектов на вашей странице выглядит следующим образом:

```
<div class="media">
  
  <div class="media-body">
    <h4>Меняйтесь</h4>
    <p>
      Не бегайте каждый раз одинаково. Меняйте темп
      бега, маршрут и расстояние.
    </p>
  </div>
</div>
```

В левую и правую части всех медиаобъектов мы добавили классы `media-image` (изображение медиаобъекта) и `media-body` (тело медиаобъекта), которые вы будете использовать для позиционирования этих элементов. Начните с выравнивания плавающего изображения по левому краю. Как вы видите на рис. 4.13, одного лишь выравнивания плавающего изображения недостаточно. Если текст довольно длинный, он обтекает плавающий элемент. Это нормальное поведение плавающего элемента, но не то, что нужно в данном случае.

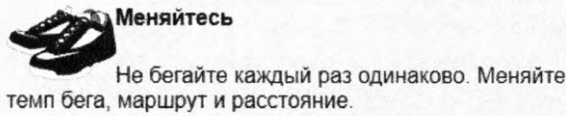


Рис. 4.13. Нежелательное обтекание текстом плавающего изображения

Добавьте в свою таблицу стилей код из листинга 4.10, чтобы страница походила на то, что показано на рис. 4.13, затем мы посмотрим, как это можно исправить. Данный код также убирает верхние поля из тела и заголовка медиаобъекта.

Листинг 4.10. Выравнивание плавающего изображения по левому краю

<pre>.media-image { float: left; }</pre>	Выравнивание плавающего изображения по левому краю
<pre>.media-body { margin-top: 0; }</pre>	Удаление верхнего поля с помощью селектора лоботомированной совы
<pre>.media-body h4 { margin-top: 0; }</pre>	Переопределение верхнего поля, создаваемого с помощью браузерных стилей

Чтобы исправить поведение текста, вам потребуется еще немного разобраться в том, как работают плавающие элементы.

4.4.1. Что такое блочный контекст форматирования

Если вы рассмотрите тело медиаобъекта на панели инструментов разработчика своего браузера (щелкните правой кнопкой мыши и выберите команду с названием наподобие *Исследовать элемент* (Inspect element)), то увидите, что поле этого элемента расширяется влево до конца, захватывая таким образом и плавающее изображение (рис. 4.14, *слева*). Текст внутри тела обтекает изображение, но, достигнув его нижнего края, смещается до конца влево. Нужно лишь поместить левый край тела медиаобъекта справа от плавающего изображения (см. рис. 4.14, *справа*).

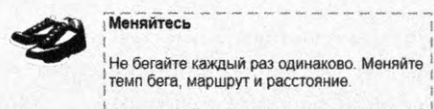
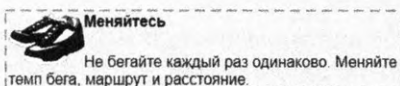


Рис. 4.14. По умолчанию тело медиаобъекта обтекает плавающее изображение (*слева*). После установки блочного контекста форматирования для тела текст не выходит за границы блока (*справа*)

Для достижения компоновки, показанной справа, вам потребуется настроить *блок контекста форматирования* для тела медиаобъекта. Так называется область страницы, в которой размещаются элементы. Сам по себе блок контекста форматирования — часть окружающего потока документа, но при этом он изолирует свой контент от внешнего контекста. В результате изоляции выполняются три действия для элемента, устанавливающего блок контекста форматирования.

- ❑ Умещаются верхние и нижние поля всех элементов. Они не будут схлопываться с полями элементов вне блочного контекста форматирования.
- ❑ Умещаются все плавающие элементы.
- ❑ Не перекрываются плавающие элементы вне блочного контекста форматирования.

Проще говоря, контент в блоке контекста форматирования не будет накладываться на элементы вне блока или взаимодействовать с ними, как ожидалось бы в нормальной ситуации. В результате применения к элементу свойства `clear` этот элемент сбросит обтекание только плавающих элементов внутри собственного блока контекста форматирования. Если же вы установите для элемента новый контекст форматирования, он не будет накладываться на блоки контекста форматирования других элементов.

Установить новый блок контекста форматирования можно несколькими способами. Применение к элементу любого из следующих значений свойств приводит к установке контекста:

- ❑ `float: left` или `float: right` — все, кроме `none`;
- ❑ `overflow: hidden, auto` или `scroll` — все, кроме `visible`;
- ❑ `display: inline-block, table-cell, table-caption, flex, inline-flex, grid` или `inline-grid` — это *блочные контейнеры*;
- ❑ `position: absolute` или `position: fixed`.

ПРИМЕЧАНИЕ

Корневой элемент страницы также создает для страницы блок контекста форматирования самого высокого уровня.

4.4.2. Использование блочного контекста форматирования для разметки медиаобъектов

По установке собственного блока контекста форматирования для тела каждого медиаобъекта ваша страница получит необходимую разметку (рис. 4.15). Зачастую наилучший способ сделать это — присвоить значение `hidden` или `auto` свойству `overflow`.

Установите значения свойства `overflow` в таблице стилей. Обновите соответствующую часть таблицы стилей так, как показано в листинге 4.11.

Бегущий город

Полезные советы



Физкультура

Физическая подготовка — важная составляющая профилактики травматизма. Сосредоточьтесь на своем теле — особенно на ногах.



Темп

Каков ваш темп? Темп самых успешных бегунов — около 180 шагов в минуту.



Меняйтесь

Не бегайте каждый раз одинаково. Меняйте темп бега, маршрут и расстояние.



Фокусируйтесь на беге

Бегите быстро, но без излишнего напряжения. Ноги должны касаться земли под вашими бедрами, а не впереди вас.

Рис. 4.15. Блочный контекст форматирования применен к телам всех медиаобъектов

Листинг 4.11. Добавление свойства `overflow: auto` запускает новый блок контекста форматирования

```
.media {
  float: left;
  margin: 0 1.5em 1.5em 0;
  width: calc(50% - 1.5em);
  padding: 1.5em;
  background-color: #eee;
  border-radius: 0.5em;
}
```

```
.media:nth-child(odd) {
  clear: left;
}
```

```
.media-image {
  float: left;
  margin-right: 1.5em;
}
```

Добавление поля к изображению
для вставки промежутка между ним и телом

```
.media-body {
  overflow: auto;
  margin-top: 0;
}
```

Установка нового блока контекста
форматирования с тем, чтобы тело
не накладывалось на плавающее изображение

```
.media-body h4 {
  margin-top: 0;
}
```


Использование свойства `overflow: auto` для блока контекста форматирования — как правило, самый простой подход. Можете применять его вместо упомянутых ранее свойств, но кое-что следует учитывать: плавающий или строчно-блочный элемент увеличится до 100 % ширины, поэтому потребуется ограничить ширину элемента, дабы предотвратить обертывание строки ниже плавающего элемента. Напротив, ячейка таблицы увеличится только до размера, достаточного для вмещения своего контента, поэтому, чтобы такой элемент заполнил все оставшееся пространство, потребуется установить большую ширину.

ПРИМЕЧАНИЕ

В некоторых ситуациях контент одного блока контекста форматирования все же может накладываться на содержимое другого. Это случится, если контент переполнит контейнер (например, ширина контента слишком велика) или если отрицательные поля вытягивают контент из контейнера.

Для получения дополнительной информации о медиаобъектах прочтите посвященный им эпохальный пост Николь Салливан, размещенный по адресу: mng.bz/6wj3w. В нем затрагивается методология под названием «объектно-ориентированные каскадные таблицы стилей» (object-oriented CSS, OOCSS), которую мы более детально рассмотрим в главе 9.

4.5. CSS-сетки

К настоящему моменту вы уже создали разметку всей страницы, но она не без изъянов. Самый заметный — неготовность к многократному использованию определенных стилей. Вы запрограммировали медиаобъекты таким образом, чтобы их ширина составляла 50 %, а значит, эти объекты всегда будут размещаться по два в строке. Но что, если позднее вы захотите применить этот же дизайн, но для строк с тремя объектами?

Один из популярных способов многократного использования кода — прибегнуть к помощи *CSS-сетки* — последовательности имен классов, которую можно добавить в разметку для структурирования частей страницы на строки и колонки. Сетка не должна задавать никаких визуальных стилей, таких как цвета или рамки, — лишь содержать наборы значений ширины и позиций контейнеров. Внутри каждого из таких контейнеров вы можете добавлять любые элементы для задания визуального стиля по своему желанию.

Большинство популярных *CSS-фреймворков* содержат CSS-сетку того или иного типа. Детали варьируются, но обычно общий принцип остается одним и тем же: поместить контейнер строки вокруг одного или нескольких контейнеров колонок. Классы, применяемые к контейнерам колонок, определяют соответствующие значения ширины каждой из этих колонок. Создадим собственную CSS-сетку, чтобы вы поняли, как система работает, и впоследствии смогли задействовать ее на своей странице.



CSS-фреймворк — это библиотека предварительно написанного CSS-кода, содержащая стили и шаблоны, часто используемые в веб-разработке. Они применяются для быстрого прототипирования или обеспечения прочной основы, на которую вы можете надстроить дополнительные стили. Часто используемые фреймворки — это, в частности, Bootstrap, Foundation и Pure.

4.5.1. Принципы CSS-сетки

Прежде чем вы создадите CSS-сетку, рассмотрим, какого поведения следует от нее ожидать. Обычно *CSS-сетка* определяется таким образом, чтобы она содержала в каждой строке определенное количество колонок — как правило, 12, но это количество может варьироваться. Дочерние элементы строки могут иметь любую ширину в диапазоне от 1 до 12 колонок.

На рис. 4.16 приведены две строки сетки с 12 колонками. В первой строке шесть элементов шириной в одну колонку и три элемента шириной в две колонки. На следующей строке располагаются один элемент на четыре колонки и еще один — на восемь колонок.

1 колонка	1 колонка	1 колонка	1 колонка	1 колонка	1 колонка	2 колонки	2 колонки	2 колонки
4 колонки				8 колонок				

Рис. 4.16. Две строки CSS-сетки на 12 колонок: дочерние элементы всех строк могут иметь любую ширину в диапазоне от 1 до 12 колонок

Двенадцать — это хорошее число для задания количества колонок, поскольку оно делится на два, три, четыре и шесть, что обеспечивает большую пластичность. Такой подход упрощает создание разметки на три колонки (три элемента на четыре колонки) или четыре колонки (четыре элемента на три колонки). Также можно выполнять асимметричную разметку, например, основной элемент на девять колонок и боковая панель на три колонки. Внутрь каждого элемента можете поместить любую разметку.

Разметка для данного примера очень проста. Каждая строка содержит контейнер `div`, внутрь которого вы поместите контейнер `div` для каждого элемента колонки с классом `column-n` (где `n` — количество колонок в сетке):

```
<div class="row">
  <div class="column-4">4 колонки</div>
  <div class="column-8">8 колонок</div>
</div>
```

4.5.2. Создание CSS-сетки

Конвертируем страницу, чтобы на ней использовалась CSS-сетка. Такой подход многословнее, чем то, что мы делали раньше, но это достойная цена за более пригодный для многократного применения CSS-код. Отредактируйте HTML-файл, как показано в листинге 4.12.

Листинг 4.12. Реструктуризация HTML для использования CSS-сетки

```
<main class="main clearfix">
  <h2>Полезные советы</h2>
```

```
  <div class="row">
    <div class="column-6">
      <div class="media">
        
        <div class="media-body">
          <h4>Физкультура</h4>
          <p>
            Физическая подготовка &mdash; важная составляющая
            профилактики травматизма. Сосредоточьтесь
            на своем теле &mdash; особенно на ногах.
          </p>
        </div>
      </div>
    </div>
  </div>
```

Добавляет
строку
вокруг
каждого
набора
из двух
медиа-
объектов

```
  <div class="column-6">
    <div class="media">
      
      <div class="media-body">
        <h4>Темп</h4>
        <p>
          Каков ваш темп? Темп самых успешных бегунов –
          около 180 шагов в минуту.
        </p>
      </div>
    </div>
  </div>
```

Закрывает первую строку
перед открытием второй

Добавляет
column-6
вокруг
каждого
медиаобъекта,
размеща
при этом
каждый
медиаобъект
в его
собственной
колонке

```
  <div class="row">
    <div class="column-6">
      <div class="media">
        
        <div class="media-body">
          <h4>Меняйтесь</h4>
          <p>
            Не бегайте каждый раз одинаково. Меняйте темп
            бега, маршрут и расстояние.
          </p>
        </div>
      </div>
    </div>
  </div>
```

```
  <div class="column-6">
    <div class="media">
      
      <div class="media-body">
        <h4>Фокусируйтесь на беге</h4>
```

```

    <p>
      Бегите быстро, но без излишнего напряжения.
      Ноги должны касаться земли под вашими бедрами,
      а не впереди вас.
    </p>
  </div>
</div>
</div>
</div>
</div>
</main>

```

Этот листинг создает строку вокруг каждого набора из двух медиаобъектов. Внутри мы упаковали каждый медиаобъект в собственный контейнер шириной шесть колонок.

Добавим стили для разметки сетки. Сначала необходимо определить класс строки. Добавьте следующий код (листинг 4.13) в свою таблицу стилей.

Листинг 4.13. CSS-код для строк сетки

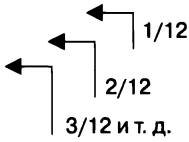
<pre> .row::after { content: " "; display: block; clear: both; } </pre>	<p>Репликация метода <code>clearfix</code>, чтобы строка содержала плавающие колонки</p>
---	--

Это не что иное, как метод `clearfix`. Вы добавляете этот код сюда, чтобы не было необходимости добавлять класс `clearfix` каждый раз, когда создается новая строка. Совсем скоро мы добавим еще код, но главное, что делает строка, — предоставляет обертку для всех колонок. Именно эту задачу и выполняет метод `clearfix`.

Далее добавим изначальные стили для колонок. Именно здесь предстоит «тяжелая работа», но, как вы убедитесь, это не так уж и трудно. Вы выровняете плавающие колонки по левому краю и укажете значения ширины для каждой колонки. Добавьте код из листинга 4.14 в свою таблицу стилей.

Листинг 4.14. CSS-код для колонок сетки

<pre> [class*="column-"] { float: left; } </pre>	<p>Нацеливается на все элементы с атрибутом класса <code>"column-"</code></p>
--	---

<pre> .column-1 { width: 8.3333%; } .column-2 { width: 16.6667%; } .column-3 { width: 25%; } .column-4 { width: 33.3333%; } .column-5 { width: 41.6667%; } .column-6 { width: 50%; } .column-7 { width: 58.3333%; } .column-8 { width: 66.6667%; } .column-9 { width: 75%; } .column-10 { width: 83.3333%; } .column-11 { width: 91.6667%; } .column-12 { width: 100%; } </pre>	 <p>1/12 2/12 3/12 и т. д.</p>
---	---

Первый селектор может быть вам незнаком. Это *селектор атрибутов*, нацеливающийся на элементы по их атрибуту `class`. Он позволяет выполнять несколько более сложные задачи по сравнению с обычным селектором классов. Компаратор `*` устанавливает любые значения, в которых встречается указанная подстрока — любые элементы с подстрокой `column-` в любом месте атрибута `class`. Это позволяет нацеливаться на элементы `<div class="column-2">`, а также `<div class="column-6">` — проще говоря, на все классы колонок. Теперь все плавающие колонки вне зависимости от их размера будут выровнены по левому краю (см. приложение А для получения дополнительных сведений о селекторах атрибутов).

ПРИМЕЧАНИЕ

Этот селектор атрибутов захватывает больше элементов, чем нам требуется, так как данный атрибут нацеливается также на такие элементы, как `<div class="column-header">`. Имейте это в виду при написании новых стилей. Для наших целей наилучшим решением будет воспринимать подстроку `"column"` в имени класса как нечто вроде зарезервированного слова, чтобы в дальнейшем не допускать конфликта с этими правилами.

После применения свойства `float: left` ко всем колонкам вы нацеливаетесь на каждую из них, указывая их ширину. Определение всех значений может потребовать небольшой работы с калькулятором: желаемое количество колонок нужно разделить на общее количество колонок (12). Убедитесь в том, что сохраняются хотя бы несколько знаков после запятой, чтобы избежать погрешностей округления.

На данном этапе вы уже создали основы CSS-сетки. Страница должна выглядеть как на рис. 4.17. Сейчас она немного несуразная, так как к плавающим объектам все еще применяются стили, дублирующие работу CSS-сетки.

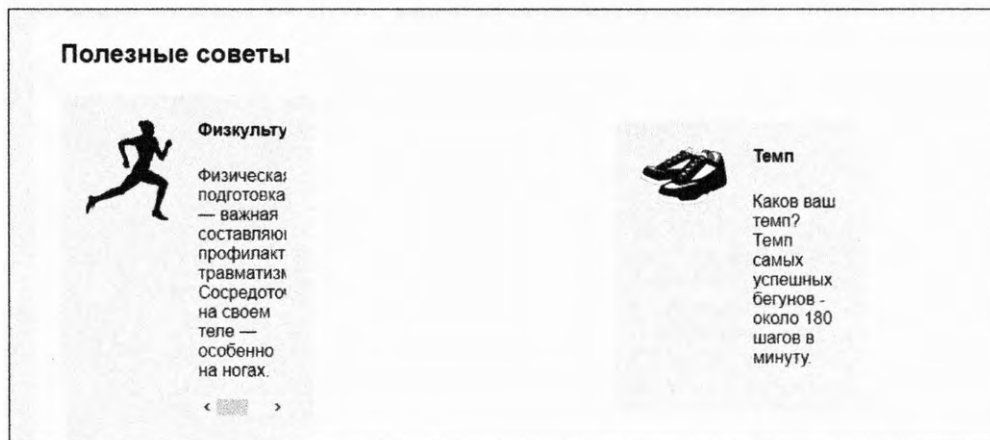


Рис. 4.17. При использовании CSS-сетки медиаобъекты больше не нуждаются в некоторых из ранее задействованных стилей

Сейчас можно упростить медиаобъект. Эти объекты больше не нужно выравнивать по левому краю, так как эту работу за вас выполнит CSS-сетка. Также объектам больше не нужна ширина: если ее не задавать, они автоматически заполняют 100 % контейнера. Контейнер — это элемент `column-6` как раз нужного размера. Вы можете удалить поля и селектор n -го потомка, что отменит обтекание всех строк. После выполнения этих операций страница должна выглядеть как на рис. 4.18.

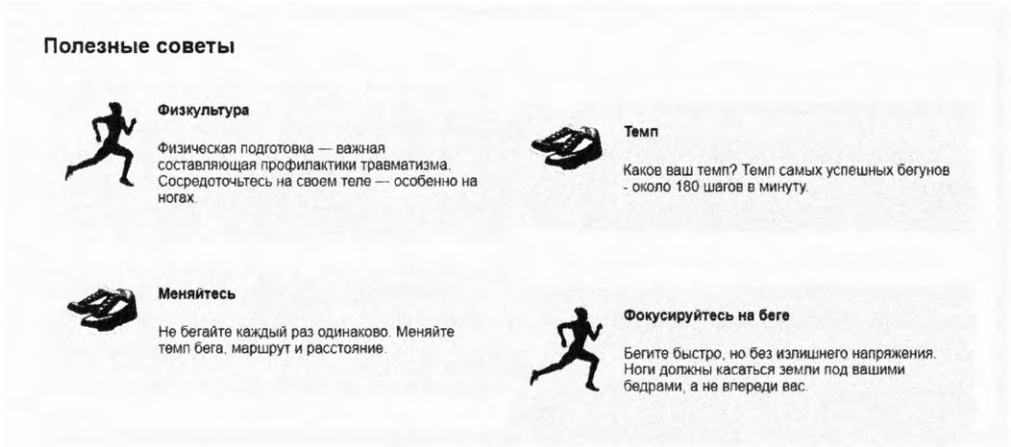


Рис. 4.18. Удаление связанных с позиционированием свойств медиаобъектов позволяет CSS-сетке правильно их расположить

После удаления этих участков кода стили ваших медиаобъектов должны выглядеть так, как показано в листинге 4.15.

Листинг 4.15. Удаление объявлений о размерности и позиционировании медиаобъектов

```
.media {
  padding: 1.5em;
  background-color: #eee;
  border-radius: 0.5em;
}

.media-image {
  float: left;
  margin-right: 1.5em;
}

.media-body {
  overflow: auto;
  margin-top: 0;
}

.media-body h4 {
  margin-top: 0;
}
```

Удалены объявления
float, margin и width

Удален набор правил
.media:nth-child(odd)
с объявлением clear: left

Поскольку вы удалили все поля медиаобъекта, в том числе нижнее, теперь между последней строкой медиаобъектов и нижним краем их контейнера больше нет промежутка. Воспользуемся свойством `padding` контейнера, чтобы вернуть его (листинг 4.16).

Листинг 4.16. Добавление нижнего отступа в основной контейнер

```
.main {
  padding: 0 1.5em 1.5em;
  background-color: #fff;
  border-radius: .5em;
}
```

← Добавляет нижний отступ размером 1,5 em, равный отступам слева и справа

Дизайн страницы почти готов, хотя еще остались детали, которые нужно выполнить.

4.5.3. Добавление зазоров

Вашей CSS-сетке по-прежнему не хватает одной вещи — зазоров между колонками. Добавим их и еще пару деталей. По окончании страница должна выглядеть так, как на рис. 4.19.



Рис. 4.19. Законченная страница с полнофункциональной CSS-сеткой

Можно создавать зазоры, добавляя левый и правый отступы каждой колонке сетки. Добавив зазор в CSS-сетку вместо отдельных компонентов, например медиаобъектов, используйте CSS-сетку вновь и вновь, не волнуясь о создании зазоров.

Вам необходим зазор размером 1,5 em, поэтому разделите это значение пополам, а затем задайте половину для левой и правой частей каждого элемента колонки. Обновите стили сетки как показано в листинге 4.17. Этот код также убирает верхнее поле у всех колонок, еще раз переопределяя селектор лоботомированной совы.

Листинг 4.17. Добавление промежутков в CSS-сетку

```
[class*="column-"] {  
    float: left;  
    padding: 0 0.75em;  
    margin-top: 0;  
}
```

Добавляет отступ размером 0,75 em
справа каждому элементу колонки

Удаляет верхние поля колонок

Теперь между элементами колонок всегда будет симпатичный зазор размером 1,5 em, и все выглядит довольно хорошо. Однако этот код создает небольшую асимметрию выравнивания между колонкой сетки и контентом вне строки сетки. На рис. 4.20 продемонстрировано, где именно это наблюдается на странице: верхний край названия страницы («Полезные советы») должен быть выровнен с краем медиаобъекта в первой колонке. Вместо этого отступ колонки выталкивает серое поле медиаобъекта чуть вправо.



Рис. 4.20. Левая сторона названия страницы выровнена с краем элемента колонки (пунктирная линия), вместо этого она должна быть выровнена с контентом колонки

Мы можем исправить это, удалив левый отступ первой колонки и правый отступ последней колонки всех строк, но вместо применения целого набора специальных правил для реализации этого приема просто настроим ширину строки.

Можно растянуть строку, используя отрицательные значения полей. Заданное для левого поля строки значение $-0,75$ em вытянет левую сторону строки из контейнера. После выполнения этой операции отступ колонки вытолкнет ее контент на 0,75 em вправо, благодаря чему первая колонка будет выровнена по названию (рис. 4.21). Применение отрицательного поля справа сделает то же самое для правой стороны.



Рис. 4.21. Отрицательное значение поля строки вытягивает ее влево, что компенсирует отступ колонки. Теперь контент колонки выровнен по названию

Код для этих действий показан в листинге 4.18. Теперь вне зависимости от того, где вы поместите строку, она будет на 1,5 em шире своего контейнера, отступы колонок затем будут смещать их контент обратно вправо для выравнивания с внешним контейнером. По сути, это модифицированная версия шаблона двойного контейнера, где строка — внутренний контейнер внутри обертки.

Листинг 4.18. Добавление отрицательных полей для строки сетки

```
.row {
  margin-left: -0.75em;
  margin-right: -0.75em;
}
```

Теперь у вас есть полнофункциональная CSS-сетка, основанная на плавающих элементах. Воспользуетесь вы ею или же сеткой, которую найдете в CSS-фреймворке, вне зависимости от этого вы уже понимаете, как система выполняет свои задачи. Полная таблица стилей должна совпадать с листингом 4.19.

Листинг 4.19. Готовая таблица стилей

```
:root {
  box-sizing: border-box;
}

*,
::before,
::after {
  box-sizing: inherit;
}

body {
  background-color: #eee;
  font-family: Helvetica, Arial, sans-serif;
}

body * + * {
  margin-top: 1.5em;
}

.row {
  margin-left: -0.75em;
  margin-right: -0.75em;
}

.row::after {
  content: " ";
  display: block;
  clear: both;
}

[class*="column-"] {
  float: left;
```

```
padding: 0 0.75em;
margin-top: 0;
}

.column-1 { width: 8.3333%; }
.column-2 { width: 16.6667%; }
.column-3 { width: 25%; }
.column-4 { width: 33.3333%; }
.column-5 { width: 41.6667%; }
.column-6 { width: 50%; }
.column-7 { width: 58.3333%; }
.column-8 { width: 66.6667%; }
.column-9 { width: 75%; }
.column-10 { width: 83.3333%; }
.column-11 { width: 91.6667%; }
.column-12 { width: 100%; }

header {
padding: 1em 1.5em;
color: #fff;
background-color: #0072b0;
border-radius: .5em;
margin-bottom: 1.5em;
}

.main {
padding: 0 1.5em 1.5em;
background-color: #fff;
border-radius: .5em;
}

.container {
max-width: 1080px;
margin: 0 auto;
}

.media {
padding: 1.5em;
background-color: #eee;
border-radius: 0.5em;
}

.media-image {
float: left;
margin-right: 1.5em;
}

.media-body {
overflow: auto;
margin-top: 0;
}

.media-body h4 {
```

```
margin-top: 0;
}

.clearfix::before,
.clearfix::after {
  display: table;
  content: " ";
}
.clearfix::after {
  clear: both;
}
```

Вы разметили страницу, используя для этого только плавающие элементы. Несмотря на присущие им странности, они выполняют свою работу. Глубже погрузившись в принципы их поведения, я надеюсь, вы окончательно перестанете бояться плавающих элементов. Как я упоминал ранее, существуют более легкие для понимания альтернативы разметок, основанных на плавающих элементах. Рассмотрим их в следующих главах.

Итоги главы

- ❑ Плавающие элементы созданы для того, чтобы текст обтекал их, но зачастую это не тот эффект, который вам нужен.
- ❑ Для того чтобы разместить плавающие элементы, используйте метод `clearfix`.
- ❑ Запомните три правила применения блочного контекста форматирования: уметь плавающие элементы, предотвратить схлопывание полей, предотвратить обтекание плавающего элемента потоком документа.
- ❑ Центрируйте контент страницы с помощью шаблона двойного контейнера.
- ❑ Задействуйте шаблон медиаобъекта для позиционирования текста описания рядом с изображением.
- ❑ Используйте CSS-сетку для создания массива разметок страниц.

5

Flexbox-верстка

В этой главе

- Flex-контейнеры и flex-элементы.
- Главная и поперечная оси.
- Размеры элементов при flexbox-верстке.
- Выравнивание элементов при flexbox-верстке.

Если последние несколько лет вы пребывали в мире CSS, то почти наверняка слышали, как кто-то воспевал flexbox-сайты. Flexbox-верстка (формально разметка с помощью пластичных блоков) — это новый метод компоновки элементов на странице. По сравнению с плавающими элементами flexbox-блоки более предсказуемы и обеспечивают более специфичные возможности управления. Также это простое решение давних проблем получения вертикального центрирования и колонок одинаковой высоты.

Flexbox-верстка маячила на горизонте на протяжении нескольких лет, ею пользовались только разработчики, которым необходимо было поддерживать новейшие браузеры. Но теперь flexbox-верстка добилась такого положения, когда ее поддерживают все основные браузеры, в том числе (частично) Internet Explorer 10. Фактически flexbox-верстка имеет более широкую поддержку, чем даже свойство `border-radius` (не поддерживается Opera Mini). Если вы ждали подходящего момента, чтобы изучить flexbox-верстку, он наступил. В этой главе вы с ней познакомитесь.

Если у flexbox-верстки и есть слабая сторона (одна), то это количество реализуемых ею возможностей. Данная технология добавляет 12 новых свойств в CSS, в том числе несколько сокращенных. За раз такое осилить может быть тяжело. Когда я только начал изучать flexbox-верстку, это больше походило на попытку напиться из пожарного гидранта: мне тоже было сложно запомнить все новые свойства. Обучая вас этой технологии, я использую иной подход — и вы легко освоите ее.

Сначала рассмотрим несколько базовых принципов, которые вы должны понимать, после чего разберем практические примеры. Чтобы верстать flexbox-сайты, не нужно заучивать все 12 свойств. Я обнаружил, что только несколько из них делают всю основную работу, на них мы и сконцентрируемся. Остальные свойства

опциональны при выравнивании элементов и создании промежутков между ними. Ближе к концу главы мы рассмотрим и их, также будет представлен справочный путеводитель, к которому вы сможете вернуться при необходимости.

5.1. Принципы flexbox-верстки

Flexbox-разметка начинается со знакомого свойства `display`. В результате применения свойства `display: flex` элемент превращается во *flex-контейнер*, а его потомки — во *flex-элементы*. По умолчанию flex-элементы выстраиваются друг за другом в одну строку слева направо. Flex-контейнер заполняет доступную ширину, как и блочный элемент, однако flex-элементы не обязательно заполняют ширину своего flex-контейнера. Flex-элементы имеют одинаковую высоту, что естественным образом определяется их контентом.

СОВЕТ

Можете использовать также свойство `display: inline-flex`. Это создаст flex-контейнер, который ведет себя скорее как строчно-блочный элемент, нежели как просто блок. Такой блок встраивается в поток вместе с другими встроенными элементами, но его ширина не растет до 100 %. Flex-элементы внутри этого блока, как правило, ведут себя так же, как и в случае применения свойства `display: flex`. Исходя из практических соображений, вы нечасто будете пользоваться этим приемом.

Flexbox-верстка отличается от ранее рассмотренных вариантов отображения (строчной, строчно-блочной и т. д.), которые влияют лишь на те элементы, к которым применены. В отличие от них flex-контейнер контролирует разметку элементов, находящихся внутри него. Flex-контейнер и его элементы приведены на рис. 5.1.

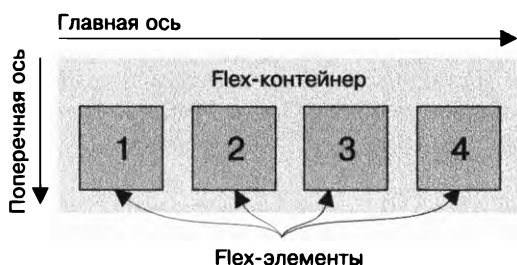


Рис. 5.1. Flex-контейнер и его элементы

Элементы размещаются вдоль линии, называемой *главной осью*, проходящей от *начала главной оси* (слева) до *конца главной оси* (справа). Перпендикулярно главной оси располагается *поперечная ось*. Она проходит от *начала поперечной оси* (сверху) до *конца поперечной оси* (снизу). Направление осей может быть изменено, в дальнейшем я продемонстрирую это.

ПРИМЕЧАНИЕ

Поскольку flexbox-разметка определяется понятиями главной и поперечной осей, то по отношению к осям мы будем использовать термины «начало» и «конец», а не «слева» и «справа» или «сверху» и «снизу».

Эти концепции (flex-контейнер, flex-элементы и две оси) составляют немалую часть того, что вы должны знать о flexbox-верстке. Применение свойства `display: flex` заведет вас уже довольно далеко, прежде чем потребуется задействовать одно из вышеупомянутых 12 новых свойств. Для того чтобы опробовать все это, создадим страницу, приведенную на рис. 5.2.

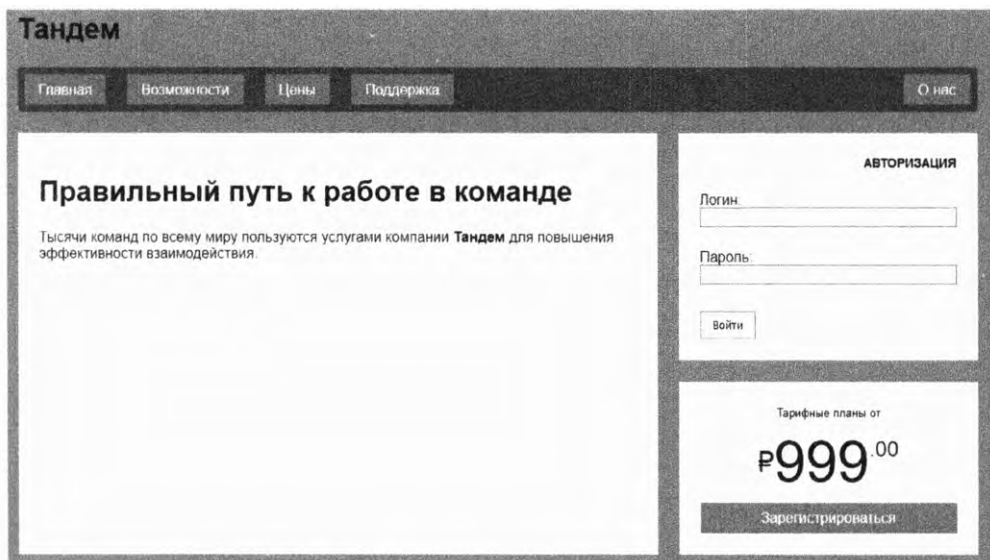


Рис. 5.2. Законченная страница с flexbox-разметкой

Я структурировал эту страницу таким образом, чтобы затронуть несколько способов применения flexbox-верстки. Мы воспользуемся flexbox-разметкой для создания панели навигации в верхней части страницы, а также компоновки трех белых блоков и отформатированного с помощью стилей текста **₽999.00** справа внизу.

Начните создавать страницу и свяжите ее с новой таблицей стилей. Затем добавьте на страницу следующую разметку (листинг 5.1).

Листинг 5.1. Разметка страницы

```
<!doctype html>
<head>
<meta charset="utf-8">
</head>
<body>
  <div class="container">
```

```

</header>
<nav>
  <ul class="site-nav">
    <li><a href="/">Главная</a></li>
    <li><a href="/features">Возможности</a></li>
    <li><a href="/pricing">Цены</a></li>
    <li><a href="/support">Поддержка</a></li>
    <li class="nav-right">
      <a href="/about">О нас</a>
    </li>
  </ul>
</nav>

<main class="flex">
  <div class="column-main tile">
    <h1>Правильный путь к работе в команде</h1>
    <p>Тысячи команд по всему миру пользуются
      услугами компании <b>Тандем</b> для повышения
      эффективности взаимодействия.</p>
  </div>

  <div class="column-sidebar">
    <div class="tile">
      <form class="login-form">
        <h3>Авторизация</h3>
        <p>
          <label for="username">Логин:</label>
          <input id="username" type="text"
            name="username"/>
        </p>
        <p>
          <label for="password">Пароль:</label>
          <input id="password" type="password"
            name="password"/>
        </p>
        <button type="submit">Войти</button>
      </form>
    </div>
    <div class="tile centered">
      <small>Тарифные планы от</small>
      <div class="cost">
        <span class="cost-currency">₽</span>
        <span class="cost-roubles">999</span>
        <span class="cost-kopecks">.00</span>
      </div>
      <a class="cta-button" href="/pricing">
        Зарегистрироваться
      </a>
    </div>
  </div>
</main>
</div>
</body>

```

Панель навигации

Большой основной блок

Боковая панель, содержащая два блока, расположенных стопкой

Чтобы положить начало таблице стилей, введите следующий CSS-код (листинг 5.2). (Надеюсь, к данному моменту эти стили уже кажутся вам знакомыми.)

Листинг 5.2. Базовые стили страницы

<pre>:root { box-sizing: border-box; } *, ::before, ::after { box-sizing: inherit; } body { background-color: #709b90; font-family: Helvetica, Arial, sans-serif; } body * + * { margin-top: 1.5em; } .container { max-width: 1080px; margin: 0 auto; }</pre>	<p>Глобальное исправление с помощью свойства <code>box-sizing</code> (глава 3)</p> <p>Устанавливает для страницы зеленый фон и рубленый шрифт</p> <p>Глобальные поля заданы с помощью селектора лоботомированной совы (глава 3)</p> <p>Двойной контейнер для центрирования контента страницы (глава 4)</p>
---	--

Теперь, приступив к созданию страницы, скомпируем некоторые элементы с помощью flexbox-верстки. Начнем с панели навигации, расположенной в верхней части страницы.

5.1.1. Создание базовой flex-навигации

Для данного примера требуется, чтобы навигационная панель выглядела так, как на рис. 5.3. Большинство элементов панели будут выровнены по левому краю, но один элемент нужно выровнять по правому.

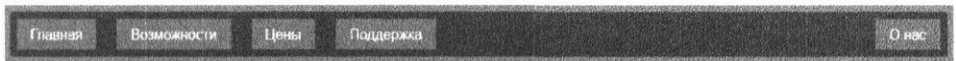


Рис. 5.3. Навигационная панель, элементы которой созданы с помощью flexbox-верстки

Работая над этой панелью, вы должны подумать, какой элемент будет flex-контейнером. Имейте в виду, что его дочерние элементы превратятся во flex-элементы. В случае с панелью нашей страницы flex-контейнером должен быть неупорядоченный список (элемент `ul`). Потомки этого контейнера — элементы списка (`li`) — станут flex-элементами. Вот как это будет выглядеть:

```
<ul class="site-nav">
  <li><a href="/">Главная</a></li>
```



```

<li><a href="/features">Возможности</a></li>
<li><a href="/pricing">Цены</a></li>
<li><a href="/support">Поддержка</a></li>
<li class="nav-right">
  <a href="/about">О нас</a>
</li>
</ul>

```

Мы повторим эти действия несколько раз по мере того, как я шаг за шагом проведу вас через создание панели. В первую очередь вы примените к списку свойство `display: flex`. Также потребуется переопределить используемые по умолчанию браузерные стили списков и верхние поля после метода лоботомированной совы. А еще — применить цвета. На рис. 5.4 показан результат выполнения этих шагов.



Рис. 5.4. Flex-навигация с применением цветов

В данной разметке вы присвоили элементу `ul` класс `site-nav`, который можно использовать для таргетирования стилей. Добавьте в таблицу стилей следующие объявления (листинг 5.3).

Листинг 5.3. Применение flexbox-верстки и цветов к панели навигации

```

.site-nav {
  display: flex;
  padding-left: 0;
  background-color: #5f4b44;
  list-style-type: none;
}

.site-nav > li {
  margin-top: 0;
}

.site-nav > li > a {
  background-color: #cc6b5a;
  color: white;
  text-decoration: none;
}

```

Класс `site-nav` становится flex-контейнером, а его потомки — flex-элементами

Убирает левый отступ и маркеры списка из браузерной таблицы стилей

Переопределяет верхнее поле, созданное методом лоботомированной совы

Убирает подчеркивания у ссылок, добавленных браузерной таблицей стилей

Обратите внимание на то, что в данном случае вы работаете с элементами трех уровней со списком `site-nav` (flex-контейнер), элементами списка (flex-элементы) и элементами привязки в них. Мы использовали комбинаторы прямых потомков (`>`), чтобы гарантировать нацеливание только на элементы, являющиеся прямыми потомками. Возможно, это не строгое требование, так как вряд ли в будущем при внесении изменений на навигационную панель будет добавлен вложенный список, однако осторожность не повредит. Если вы не знакомы с данным комбинатором, обратитесь к приложению А для получения дополнительной информации.

Вендорные префиксы

Если вы откроете flexbox-сайт в старых браузерах, таких как Internet Explorer 10 или Safari 8, то обнаружите, что он не работает. Это происходит потому, что старые браузеры требуют наличия *вендорных префиксов* для flexbox-атрибутов. Именно таким образом они обеспечивали поддержку некоторых новых функций CSS до появления стабильной спецификации. Так, например, вместо реализации `display: flex` в старых версиях браузера Safari было задействовано свойство `display: -webkit-flex`. Для того чтобы технология flex-контейнер заработала в браузере Safari 8, потребуется добавить указанное объявление, за ним должно следовать обычное объявление:

```
.site-nav {  
  display: -webkit-flex;  
  display: flex;  
}
```

Браузер игнорирует объявления, которые не понимает. Таким образом, в Safari 8 `-webkit-flex` будет каскадным значением, поведение которого аналогично поведению свойства `flex` в современных версиях. Это правило работает как для имен свойств, так и для их значений. Например, вы должны объявить значение свойства `flex` (которое мы рассмотрим чуть позже в этой главе) следующим образом:

```
-webkit-flex: 1;  
flex: 1;
```

В случае с Internet Explorer 10 все несколько сложнее, так как в нем используется более старая спецификация технологии flexbox. Чтобы реализовать поддержку этой версии, потребуется знать старые имена свойств (например, `flexbox` вместо `flex`), а также добавить версии этих свойств с префиксами:

```
display: -ms-flexbox;  
display: -webkit-flex;  
display: flex;
```

За этими особенностями сложно уследить, кроме того, они требуют большого количества повторений в таблице стилей. Поэтому я *настоятельно* рекомендую автоматизировать данный процесс с использованием инструмента Autoprefixer, доступного на сайте github.com/postcss/autoprefixer. Этот инструмент разбирает ваш CSS-код и выводит новый файл, в который по мере необходимости будут добавлены все нужные префиксы. Данный инструмент работает с большим количеством инструментов разработки, поэтому можете внедрить его в любой комфортный для вас рабочий процесс.

Для упрощения я не применяю префиксы в примерах, рассматриваемых в главе. Эти примеры будут работать в современных браузерах, но когда придет время подготовки кода для выпуска в реальный мир, прогоните его через Autoprefixer, чтобы обеспечить поддержку более широкого диапазона браузеров.

Также важно знать, что концепция префиксов уходит в прошлое. Все основные браузеры уже перешли или переходят на новый метод поддержки зарождающихся нестабильных функций с помощью опции «экспериментальные функции». Мы рассмотрим его в главе 6.

5.1.2. Добавление отступов и промежутков

На данном этапе наша навигационная панель выглядит несколько худосочной. Расширим ее с помощью отступов. Вы добавите отступы и для контейнера, и для ссылок навигации. После выполнения этого шага область навигации будет выглядеть так, как на рис. 5.5.



Рис. 5.5. Панель навигации с добавлением отступов и стилей ссылок

Если вы не очень хорошо знакомы с тем, как создавать панели навигации такого типа (с помощью flexbox-верстки или любого другого метода разметки), обратите внимание на то, как это делается. В примерах вы примените отступы панели навигации к внутренним элементам `a`, но не к элементам `li`. Вам потребуется, чтобы вся область, выглядящая как панель навигации, вела себя как ссылка, когда пользователь щелкает на ней кнопкой мыши. Поскольку ссылочное поведение задается элементом `a`, не нужно превращать элементы `li` в одну большую красивую кнопку, следует создать внутри них лишь небольшой чувствительный к щелчку кнопкой мыши целевой участок (`a`).

Обновите стили, как показано в листинге 5.4. Это заполнит отступы на панели навигации.

Листинг 5.4. Добавление отступов к панели навигации и ссылкам

```
.site-nav {
  display: flex;
  padding: .5em;
  background-color: #5f4b44;
  list-style-type: none;
  border-radius: .2em;
}

.site-nav > li {
  margin-top: 0;
}

.site-nav > li > a {
  display: block;
  padding: .5em 1em;
  background-color: #cc6b5a;
  color: white;
  text-decoration: none;
}
```

Добавление отступа к панели навигации вне ссылок

Блочный вид ссылок, чтобы они увеличивали высоту родительских элементов

Добавление отступа внутри ссылок

Обратите внимание на то, что вы превратили ссылки в блок. Если бы они оставались строчными элементами, то высота, которую они добавляли своим родительским элементам, была бы соразмерна высоте строки, но не их контенту или отступам, од-

нако это именно то поведение, которое необходимо для создания данной страницы. Примененный горизонтальный отступ также несколько больше вертикального, что, как правило, выглядит более эстетично.

Далее добавьте промежутки между пунктами навигации. Обычные старые поля подойдут. Более того, flexbox-верстка позволяет использовать свойство `margin: auto` для заполнения всего доступного пространства между flex-элементами. Также можно воспользоваться этим ходом, чтобы перенести последний элемент навигации вправо. Завершенная панель навигации после добавления полей показана на рис. 5.6.



Рис. 5.6. Поля создают промежутки между flex-элементами

В листинге 5.5 приведены описанные стили. В этом листинге поля задаются между всеми элементами, но не применяются к внешним краям. Можете создать такую разметку, используя свойство `margin-left` и комбинатор смежных элементов. Этот метод аналогичен методу лоботомированной совы из главы 3. Для последней кнопки также задается автоматическое левое поле (`auto`), оно заполняет все доступное пространство, тем самым выталкивая ее вправо. Добавьте следующий код в свою таблицу стилей.

Листинг 5.5. Использование полей для создания промежутков между элементами

```
.site-nav > li + li {
  margin-left: 1.5em;
}

.site-nav > .nav-right {
  margin-left: auto;
}
```

Нацеливается на все элементы списка после данного элемента списка (то есть все, кроме первого)

Автоматические поля внутри flex-блока заполняют все доступное пространство

Вы применили автоматические поля только к элементу **О нас**. Вместо этого могли бы задать их для пункта **Поддержка** панели навигации, чтобы перенести вправо два последних элемента.

В данном случае поля работают хорошо, так как между этими элементами требуются разные промежутки. Если бы нужны были одинаковые, то лучшим подходом было бы свойство `justify-content`. Мы к этому вскоре вернемся.

5.2. Размеры flex-элементов

В листинге 5.5 мы настраивали поля для создания промежутков между flex-элементами. Для задания их размеров вы могли бы воспользоваться свойствами `width` и `height`, но flex-блоки предоставляют больше возможностей для задания размеров и промежутков, чем уже знакомые свойства `margin`, `width` и `height` по отдельности.

Свойство `flex` управляет размером flex-элементов вдоль главной оси (то есть шириной). В листинге 5.6 вы примените flex-разметку для основной области страницы, затем увидите, каким образом свойство `flex` управляет размером колонок. Изначально основная область страницы будет выглядеть как на рис. 5.7.

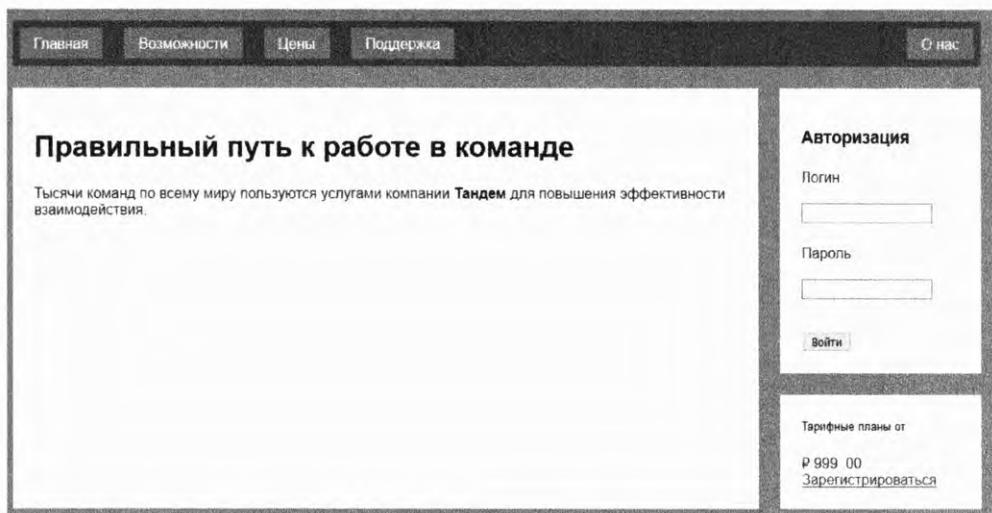


Рис. 5.7. Основная область с flex-разметкой

Добавьте стили из следующего листинга в свою таблицу стилей. Он задает белый фон для трех блоков с помощью класса `tile` и flex-разметки элемента `main`, нацеливаясь на класс `flex`.

Листинг 5.6. Применение flexbox-верстки к основному контейнеру

<pre><code>.tile { padding: 1.5em; background-color: #fff; }</code></pre>	<p>Добавление цвета фона и отступов к трем блокам</p>
<pre><code>.flex { display: flex; }</code></pre>	<p>Применение flex-разметки к основному контейнеру</p>
<pre><code>.flex > * + * { margin-top: 0; margin-left: 1.5em; }</code></pre>	<p>Удаление верхнего поля и создание промежутка между flex-элементами</p>

Теперь контент разделен на две колонки: слева находится более крупная область с основным содержанием страницы, а справа — форма авторизации и небольшой

блок для указания цены. Вы еще не задавали ширину двух колонок, поэтому они сделают это самостоятельно и исходя из контента. На моем экране (см. рис. 5.7) они не заполняют все доступное пространство, впрочем, при меньших размерах окна все может быть иначе.

ПРИМЕЧАНИЕ

В случае с CSS важно рассматривать не только контент, который находится на странице в данный момент, но и то, что случится со страницей, если контент изменится или эта же таблица стилей будет применена к нескольким подобным страницам. Вам нужно определить, каким будет желательное поведение двух колонок в разных ситуациях.

Свойство `flex`, задаваемое для flex-элементов, предоставляет несколько вариантов. Сначала рассмотрим наиболее простой из них. Для нацеливания на колонки вы будете задействовать имена классов `column-main` и `column-sidebar`, чтобы затем с помощью свойства `flex` задать ширину размером одна треть и две трети. Добавьте в таблицу стилей следующий код (листинг 5.7).

Листинг 5.7. Использование свойства `flex` для задания ширины колонок

```
.column-main {
  flex: 2;
}

.column-sidebar {
  flex: 1;
}
```

Теперь, когда колонки стали шире и вместе имеют такую же ширину, как и панель `nav`, основная колонка шире боковой панели в два раза. Технология flexbox выполнила все математические расчеты за вас. Рассмотрим, что произошло, подробнее.

Свойство `flex` — это сокращение трех различных свойств задания размера: `flex-grow`, `flex-shrink` и `flex-basis`. В листинге 5.7 вы определили только свойство `flex-grow`, оставив значения двух других установленными по умолчанию (1 и 0% соответственно). Таким образом, выражение `flex: 2` эквивалентно выражению `flex: 2 1 0%`. Как правило, предпочтительно краткое объявление, однако можно объявить эти свойства и по отдельности:

```
flex-grow: 2;
flex-shrink: 1;
flex-basis: 0%;
```

Рассмотрим, что означает каждое из этих свойств. Начнем с `flex-basis`, так как остальные два свойства основаны на нем.

5.2.1. Свойство flex-basis

Свойство `flex-basis` определяет что-то наподобие начальной точки для размера элемента: изначальный основной размер. Свойству `flex-basis` может быть присвоено любое значение, которое применимо к ширине (`width`), в том числе значения в пикселах, единицах `em` и процентах. Изначальное значение этого свойства — `auto`, которое говорит о том, что браузер будет искать определение свойства `width` данного элемента. Если такое свойство не определено, то браузер установит размер элемента в зависимости от контента. То есть для любого элемента значение свойства `width` будет проигнорировано, если значение `flex-basis` не равняется `auto`. Это правило проиллюстрировано на рис. 5.8.

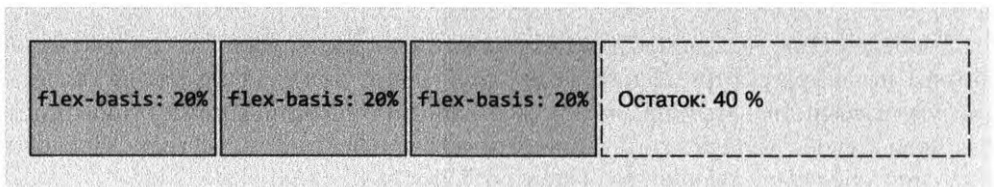


Рис. 5.8. Три flex-элемента со значением свойства `flex-basis`, равным 20 %, что задает для каждого элемента изначальный размер (ширину) 20 %

После установления изначального размера по главной оси для каждого из flex-элементов им может потребоваться стать шире или уже для того, чтобы уместиться во flex-контейнер или заполнить его. Именно здесь и используются свойства `flex-grow` и `flex-shrink`.

5.2.2. Свойство flex-grow

После того как свойство `flex-basis` выполнит вычисления для всех flex-элементов, эти элементы, а также поля между ними внесут свой вклад в ширину. Получившаяся ширина не обязательно заполнит ширину flex-контейнера, формируя тем самым некий остаток (см. рис. 5.8).

Оставшееся пространство (или остаток) будет поглощено flex-элементами в зависимости от значений их свойства `flex-grow`, которому всегда присваивается неотрицательное число. Если значение свойства `flex-grow` некоего элемента равняется 0, то он не расширится вне значения свойства `flex-basis`. Если элементам задан положительный фактор расширения, то они будут увеличиваться до тех пор, пока не используют все нераспределенное пространство. Это означает, что flex-элементы заполняют всю ширину контейнера (рис. 5.9).

Объявление больших значений свойства `flex-grow` говорит о том, что элемент будет иметь больший вес — займет большую долю остатка. Элемент со значением свойства `flex-grow: 2` станет в два раза шире по сравнению с элементом `flex-grow: 1` (рис. 5.10).



Рис. 5.9. Оставшаяся ширина распределяется равными долями между элементами с одинаковыми значениями свойства `flex-grow`

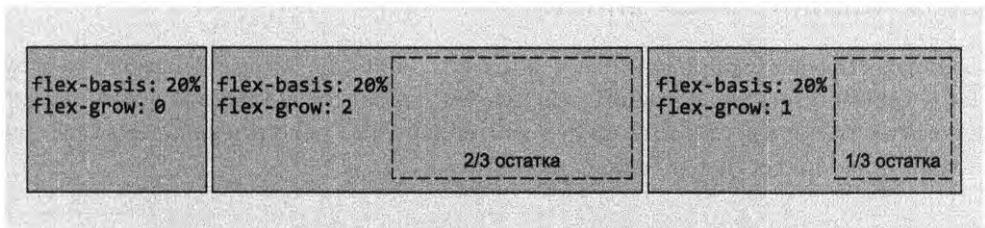


Рис. 5.10. Элементы с большим значением свойства `flex-grow` займут большую долю остатка доступной ширины

Это именно то, что вы сделали на своей странице. Краткие объявления `flex: 2` и `flex: 1` устанавливают значение свойства `flex-basis: 0`. Таким образом, 100 % ширины контейнера — это остаток за вычетом 1,5 em — поля между двумя колонками. После этого остаток распределяется между двумя колонками: две трети отходят к первой, оставшаяся треть — ко второй (рис. 5.11).

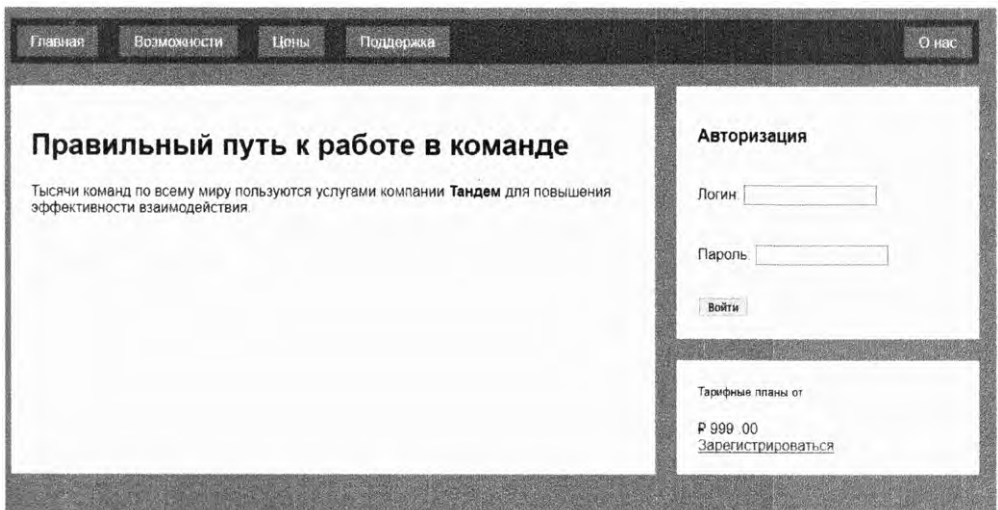


Рис. 5.11. Две колонки заполняют ширину flex-контейнера

СОВЕТ

Рекомендую использовать краткое свойство `flex` вместо отдельного объявления свойств `flex-grow`, `flex-shrink` и `flex-basis`. Если опустить эти свойства, то им, в отличие от большинства кратких свойств, не будут присвоены изначальные значения. Вместо этого они приобретут полезные значения по умолчанию: `flex-grow: 1`, `flex-shrink: 1` и `flex-basis: 0 %`. Как правило, это именно нужные вам значения.

5.2.3. Свойство `flex-shrink`

Свойство `flex-shrink` придерживается тех же принципов, что и свойство `flex-grow`. После определения изначального размера `flex`-элементов по главной оси они могут превысить размер, доступный во `flex`-контейнере. Без использования `flex-shrink` это способно привести к переполнению (рис. 5.12).

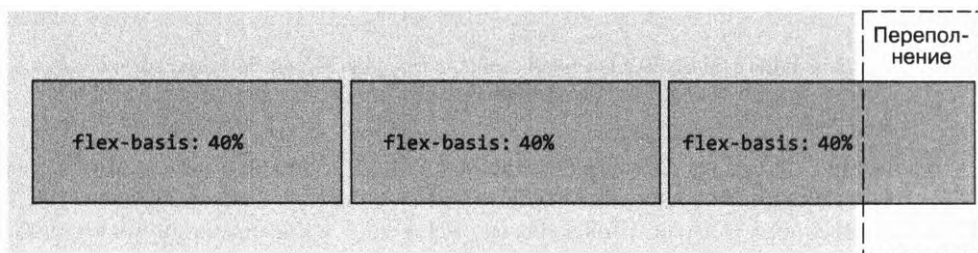


Рис. 5.12. Изначальный размер `flex`-элементов по главной оси может превысить размер `flex`-контейнера

Значение свойства `flex-shrink` каждого элемента показывает, должен ли он сжиматься для предотвращения переполнения. Если для элемента установлено значение `flex-shrink: 0`, то он не будет сжиматься. Элементы со значениями данного свойства больше нуля станут сжиматься до тех пор, пока переполнение не будет устранено. Элемент с большим значением сжимается сильнее, чем элемент с меньшим значением, пропорционально значениям свойства `flex-shrink`.

В качестве альтернативного подхода к созданию страницы вы могли бы положиться на свойство `flex-shrink`. Чтобы сделать это, установите свойство `flex-basis` для обеих колонок в виде желаемого процента (66.67% и 33.33%). Ширина плюс промежуток 1,5 em означает переполнение на 1,5 em. Установите для обеих колонок значение свойства `flex-shrink: 1`, а значение 0,75 em будет вычтено из ширины обеих колонок, что позволит им поместиться в контейнер. В листинге 5.8 показано, как бы выглядел такой код.

Перед вами другой подход для получения того же результата, что и раньше (см. листинг 5.7). Любой из этих вариантов подходит для наших целей.

Листинг 5.8. Использование свойства flex для установки ширины

```
.column-main {  
  flex: 66.67%;  
}  
  
.column-sidebar {  
  flex: 33.33%;  
}
```

← Эквивалентно свойству flex:
1 1 66.67%

← Эквивалентно свойству flex:
1 1 33.33%

ПРИМЕЧАНИЕ

Если присмотреться к мельчайшим подробностям, то между результатами работы листингов 5.7 и 5.8 обнаружится небольшое несоответствие. Причина довольно сложна, но если вкратце, это происходит из-за того, что элементу `column-main` задан отступ, а элементу `column-sidebar` — нет. Отступ изменяет способ определения изначального размера flex-элемента по главной оси, когда свойству `flex-basis` присвоено значение 0 %. Таким образом, элемент `column-main` из листинга 5.7 на 3 px шире, чем соответствующий элемент из листинга 5.8 — размер левого и правого отступов. Если хотите, чтобы ваши измерения были точны, то либо убедитесь, что значения отступа равны, либо используйте метод `flex-basis` из листинга 5.8.

5.2.4. Практические примеры

Свойство `flex` применяется бесчисленным количеством способов. Вы можете определять пропорциональные колонки с помощью задания процентных значений свойствам `flex-grow` и `flex-basis`, как мы делали на нашей странице. Или определить колонки фиксированной ширины и пластичные¹, размер которых изменяется в зависимости от ширины области просмотра. Используя flex-блоки вместо плавающих элементов, можете сгенерировать CSS-сетку наподобие той, что мы создали в главе 4. На рис. 5.13 приведены несколько видов разметки, создаваемых с помощью flex-блоков.

Третий пример иллюстрирует часть разметки «Святой Грааль». Она известна сложностью реализации в CSS. Две боковые панели фиксированной ширины, тогда как центральная — пластичная, то есть увеличивающаяся в зависимости от количества доступного пространства. Несмотря на то что такую разметку реально воссоздать, используя плавающие элементы, она требует реализации некоторых малоизвестных и требующих скрупулезности тактик. Данную разметку можно любым количеством способов сочетать с любым количеством flex-элементов.

¹ Также называемые жидкими. — *Примеч. пер.*

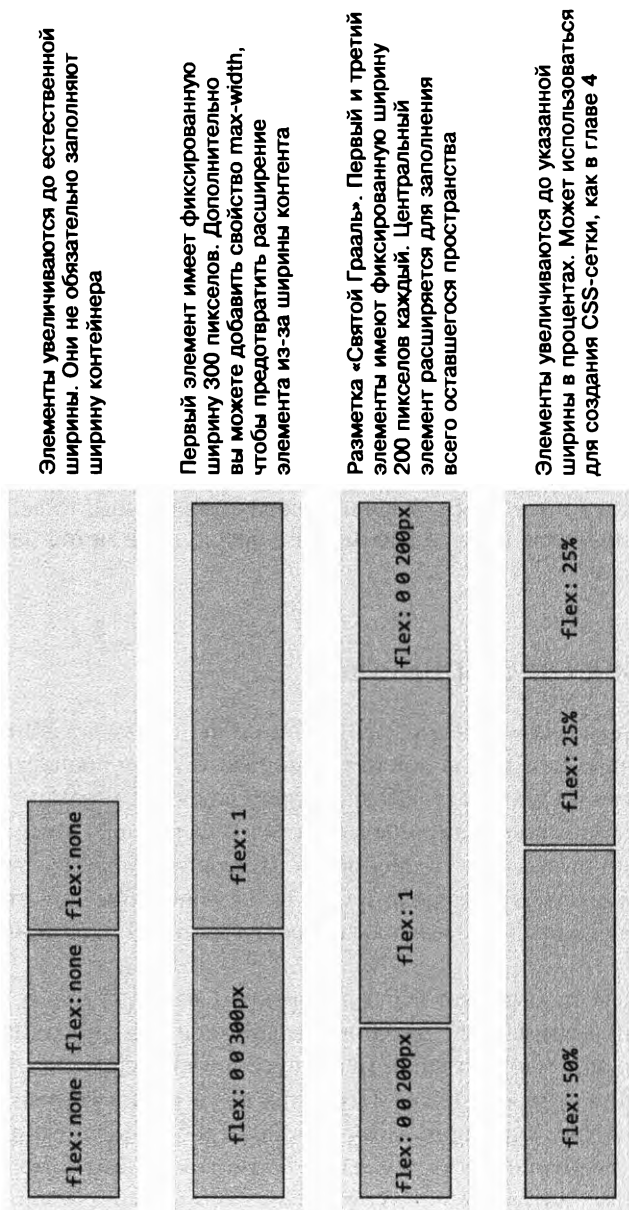


Рис. 5.13. Способы определения размеров с помощью свойства flex

5.3. Направление flex-элементов

Еще одна важная функция flexbox-верстки — возможность изменить направление осей. Управляет этим свойство `flex-direction`, применяемое к flex-контейнеру. Начальное значение этого свойства (`row`) заставляет элементы располагаться слева направо, как мы уже видели. Указание свойства `flex-direction: column` приводит к тому, что элементы выстраиваются по вертикали сверху вниз. Flex-блоки также поддерживают свойство `row-reverse`, позволяющее элементам выстраиваться справа налево, и `column-reverse`, располагающее элементы снизу вверх (рис. 5.14).

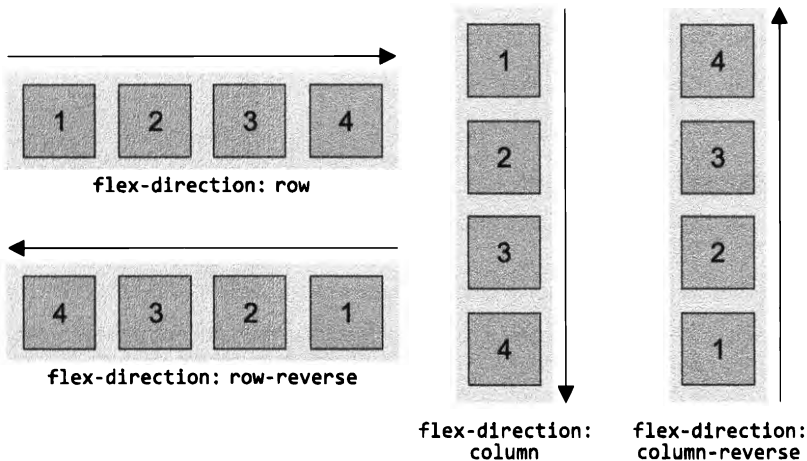


Рис. 5.14. Изменение направления flex меняет главную ось. Изменяется также поперечная ось, чтобы оставаться перпендикулярной главной

Воспользуйтесь свойством `flex-direction` для правой колонки страницы, в которой два блока расположены один под другим. Может показаться, что в этом нет необходимости, в конце концов, блоки в правой колонке уже находятся в стеке. Обычные блочные элементы так себя и ведут, однако с разметкой страницы есть одна не вполне очевидная проблема, проявляющаяся по мере добавления контента (рис. 5.15).

Добавьте несколько заголовков и абзацев в код элемента `column-main` — и вы увидите, что основной блок увеличится по высоте и зайдет за нижнюю границу блока справа. Технология flexbox-верстки должна обеспечивать одинаковую высоту колонок, так почему же это не работает?

На рис. 5.15 пунктирной линией показано, что фактически flex-элементы одинаковы по высоте. Проблема в том, что блоки внутри flex-элемента справа не увеличиваются и не заполняют его.

Идеальная разметка была бы такой, как показано на рис. 5.16. Два блока справа увеличиваются, но не заполняют колонку даже при увеличении контента левой колонки. До появления flexbox-верстки этого эффекта невозможно было достичь с помощью CSS, но реально — прибегнув к JavaScript.

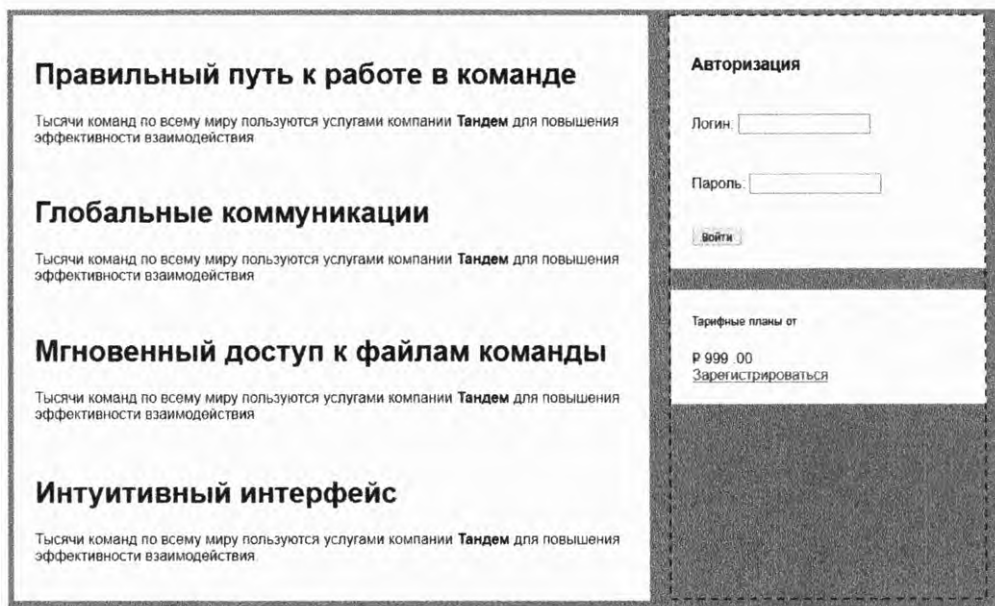


Рис. 5.15. Основной блок увеличивается по высоте и заходит за нижнюю границу блока в колонке справа (пунктиром обозначен размер элемента column-sidebar)

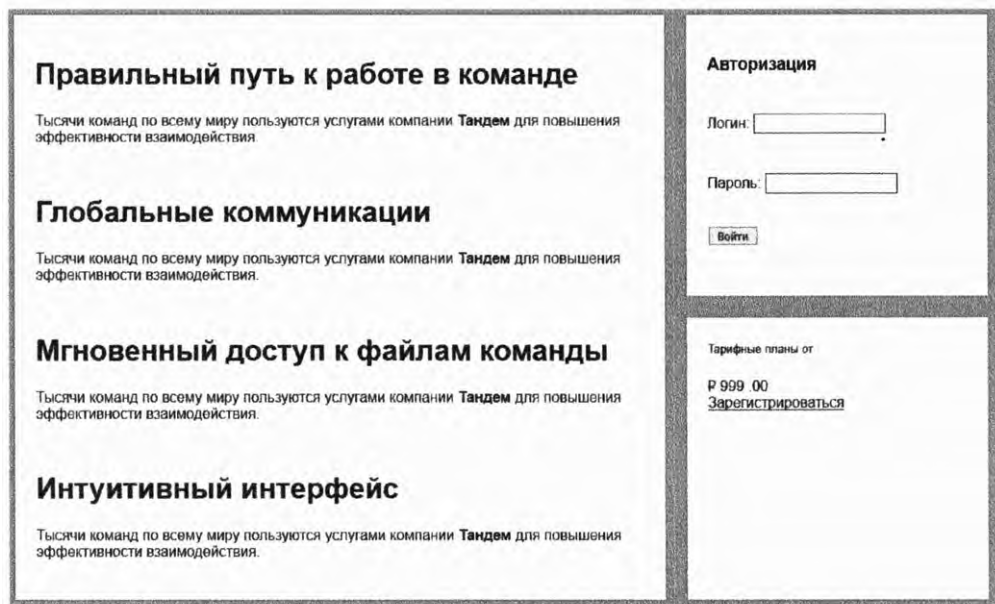


Рис. 5.16. Идеальная разметка: блоки справа выровнены по высоте с большим блоком слева

5.3.1. Изменение направления flex-элементов

Вам нужно всего лишь заставить колонки при необходимости увеличиваться в размерах, чтобы заполнить высоту контейнера. Для этого превратите правую колонку (`column-sidebar`) во flex-контейнер, задав свойство `flex-direction: column`. После этого присвойте ненулевое значение свойству `flex-grow` обоим блокам внутри этого контейнера. В листинге 5.9 приведен программный код для этих шагов. Обновите соответствующим образом свою таблицу стилей.

Листинг 5.9. Создание flex-колонки справа

<pre>.column-sidebar { flex: 1; display: flex; flex-direction: column; }</pre>	<div></div>	<p>Flex-элемент для внешнего flex-блока и flex-контейнер для внутреннего</p>
<pre>.column-sidebar > .tile { flex: 1; }</pre>	<div>←</div>	<p>Применение свойства flex-grow для элементов внутри</p>

Теперь у вас есть *вложенные flex-блоки*. Элемент `div class="column-sidebar"` — это flex-элемент для внешнего flex-блока, при этом данный элемент также flex-контейнер для внутреннего flex-блока. Общая структура этих элементов выглядит следующим образом (текст удален для краткости):

```
<main class="flex">  
  <div class="column-main tile">  
    ...  
  </div>  
  <div class="column-sidebar">  
    <div class="tile">...</div>  
    <div class="tile">...</div>  
  </div>  
</div>
```

Свойству `flex-direction` внутреннего flex-блока присвоено значение `column`, благодаря чему главная ось повернута и направлена сверху вниз, а поперечная ось идет слева направо. Это означает, что теперь свойства `flex-basis`, `flex-grow` и `flex-shrink` данных элементов влияют на их высоту, а не на ширину. Указание свойства `flex: 1` приведет к тому, что при необходимости высота этих элементов увеличится и они заполнят контейнер. Теперь вне зависимости от того, высота какого блока, большого или малого, больше, их нижние края выровнены.

В работе с вертикальными flex-блоками (`column` или `column-reverse`) применяются те же общие концепции, что и для строк, но всегда необходимо учитывать одно различие: в CSS работа с высотой фундаментально отличается от работы с шириной. В ширину flex-контейнер будет равняться 100 % доступного пространства, но его высота естественным образом определяется контентом. Это поведение не изменяется при повороте главной оси.

Высота flex-контейнера определяется flex-элементами, которые полностью заполняют высоту. В случае с вертикальными flex-блоками свойства `flex-grow` и `flex-shrink` не дадут никакого эффекта, если нечто иное не задает конкретную высоту flex-контейнера. На вашей странице это нечто — высота, производная от внешнего flex-блока.

5.3.2. Стилизовое форматирование формы авторизации

К этому моменту вы уже создали общую разметку всей страницы. Осталось отформатировать малые элементы — форму авторизации и ссылку регистрации. Для формы авторизации flex-блок не нужен, но для ощущения завершенности я все же вкратце проведу вас по нему. После выполнения всех действий форма должна выглядеть как на рис. 5.17.

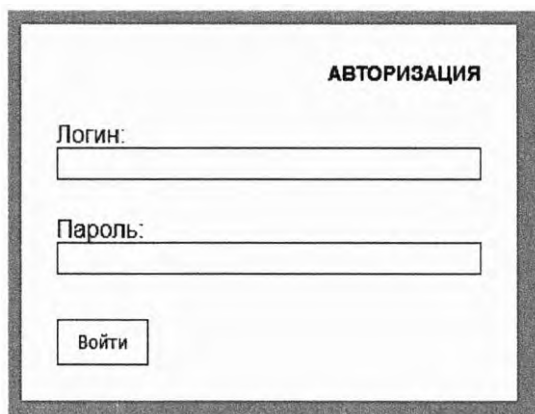


Рис. 5.17. Форма авторизации

Элементу `form` присвоен класс `login-form`, и вы будете использовать его для нацеливания в CSS. Добавьте в таблицу стилей код из листинга 5.10. Он форматировует форму авторизации в виде трех частей: заголовка, полей ввода и кнопки.

Первое — это заголовок, в котором используются, должно быть, уже знакомые вам свойства шрифтов. Вы применили свойство `text-align` для выравнивания текста по правому краю и `text-transform` для смены регистра букв на прописные. Обратите внимание на то, что текст не был набран прописными буквами в HTML. Если запись прописными буквами — это лишь вопрос форматирования, то вам нужно набрать заголовок строчными буквами в соответствии с правилами грамматики в HTML, а затем задействовать CSS для преобразования всех букв в прописные. Таким образом вы сможете изменять данное свойство стиля без необходимости перепечатывать текст в HTML-коде в нужном регистре.

Второй набор правил форматировует поля ввода текста. В данном случае используется необычный селектор, в основном потому, что сам элемент `input` весьма

необычен. Он применяется для полей ввода текста, пароля, а также других средств ввода HTML5, выглядящих подобно перечисленным. Кроме того, с его помощью создаются компоненты ввода данных формы, абсолютно непохожие на текстовые поля, в частности переключатели и флажки.

Листинг 5.10. Стили формы авторизации

```
.login-form h3 {  
  margin: 0;  
  font-size: .9em;  
  font-weight: bold;  
  text-align: right;  
  text-transform: uppercase;  
}
```

Установка полужирного начертания шрифта,
выравнивание по правому краю
и преобразование в прописные всех букв заголовка

```
.login-form input:not([type=checkbox]):not([type=radio]) {  
  display: block;  
  margin-top: 0;  
  width: 100%;  
}
```

Стилевое форматирование
всех полей ввода текста
(не флажков
и не переключателей)

```
.login-form button {  
  margin-top: 1em;  
  border: 1px solid #cc6b5a;  
  background-color: white;  
  padding: .5em 1em;  
  cursor: pointer;  
}
```

Стилевое
форматирование
кнопки

Я объединил псевдокласс `:not()` с селекторами атрибутов `[type=checkbox]` и `[type=radio]` (см. приложение А для получения дополнительных сведений). Этот шаг позволяет нацеливаться на все элементы ввода, за исключением флажков и переключателей. Данный подход аналогичен добавлению в черный список. Можно также воспользоваться подходом «белый список», подразумевающим использование множества селекторов для поименного перечисления всех элементов, на которые необходимо настроить таргетинг (список может оказаться довольно длинным).

ПРИМЕЧАНИЕ

В форме на этой странице есть только поля ввода текста и пароля, однако важно рассматривать и иные элементы разметки, к которым может быть применен CSS-код в будущем, и принимать их во внимание при создании таблицы стилей.

В наборе правил вы преобразуете элементы ввода в блочные, благодаря чему они выводятся каждый на своей строке. Также потребовалось указать ширину 100 %. Как правило, блочные элементы автоматически заполняют всю ширину, но элемент `input` не таков. Его ширина определяется атрибутом `size`, указывающим количество





символов, которое данное поле должно вмещать без прокрутки. Этому атрибуту присваивается значение по умолчанию, если не указано иное. Устанавливается нужная ширина с помощью CSS-свойства `width`.

Третий набор правил определяет стилевое форматирование кнопки **Войти**. Стили в большинстве своем интуитивно понятны, однако свойство `cursor` может быть вам незнакомо. Оно управляет внешним видом указателя мыши, когда его помещают над соответствующим элементом. Значение `pointer` превращает его в руку с вытянутым указательным пальцем, что соответствует указателю мыши при наведении на ссылку. Это преобразование сообщает пользователям, что они могут щелкнуть на кнопке, и является последней деталью полировки ее внешнего вида.

5.4. Выравнивание, промежутки и другие штрихи

К данному моменту вы уже должны четко понимать основные части flexbox-верстки. Но, как я уже упоминал, существует большой массив свойств, требующихся время от времени. Это относится в основном к выравниванию и распределению flex-элементов во flex-контейнерах. Можете также задействовать обтекание строк для переупорядочения отдельных flex-элементов. Свойства, управляющие данными аспектами, приведены далее: в табл. 5.1 перечисляются все применяемые к flex-контейнерам, а в табл. 5.2 — применяемые к flex-элементам.

Таблица 5.1. Свойства flex-контейнеров

Свойство	Значения (начальные значения выделены полужирным шрифтом)
flex-direction — определяет направление главной оси. Поперечная ось будет перпендикулярна главной	<div>row</div> 
	<div>row-reverse</div> 
	<div>column</div> 
	<div>column-reverse</div> 



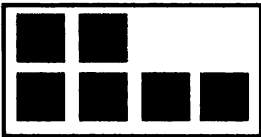









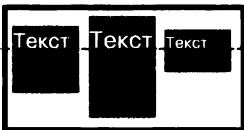
Свойство	Значения (начальные значения выделены полужирным шрифтом)
flex-wrap — определяет, будут ли flex-элементы переноситься на следующую строку flex-контейнера (или в новую колонку, если свойству flex-direction присвоено значение column или column-reverse)	<div><div>nowrap</div><div>wrap</div><div>wrap-reverse</div></div>
flex-flow	Сокращение для <flex-direction><flex-wrap>
justify-content — управляет тем, как элементы распределяются вдоль главной оси	<div><div>flex-start</div><div>flex-end</div><div>center</div><div>space-between</div><div>space-around</div></div>
align-items — управляет тем, как элементы распределяются вдоль поперечной оси	<div><div>flex-start</div><div>flex-end</div><div>center</div><div>stretch</div><div>baseline</div></div>

Таблица 5.1 (продолжение)

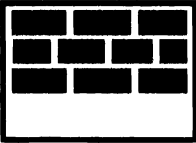
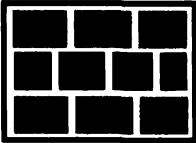


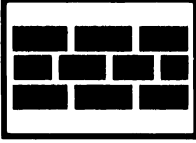


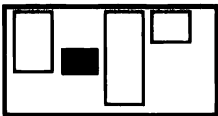
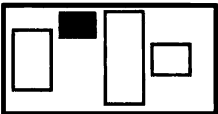

Свойство	Значения (начальные значения выделены полужирным шрифтом)			
align-content — если свойство flex-wrap задействовано, то оно управляет промежутками flex-строк вдоль поперечной оси. Если элементы не переносятся, то данное свойство игнорируется	flex-start		stretch	
	flex-end		space-between	
	center		space-around	

Таблица 5.2. Свойства flex-элементов

Свойство	Значения (начальные значения выделены полужирным шрифтом)
flex-grow — целое число — «фактор роста», определяющий, насколько увеличится элемент вдоль главной оси для заполнения нераспределенного пространства	<div><div>1</div><div>1</div><div>2</div></div>
flex-shrink — целое число — «фактор уменьшения», определяющий, насколько уменьшится элемент вдоль главной оси, если это необходимо, во избежание переполнения. Данное свойство игнорируется, если задействовано свойство flex-wrap	<div><div>2</div><div>2</div><div>1</div></div>

Свойство	Значения (начальные значения выделены полужирным шрифтом)
flex-basis — определяет начальный размер элемента перед применением свойств flex-grow или flex-shrink	<length> или <percent>
flex	Сокращение для <flex-grow><flex-shrink><flex-basis>
align-self — управляет выравниванием элемента по поперечной оси. Это свойство переопределяет значение свойства align-items контейнера для конкретного элемента (-ов). Игнорируется при присвоении значения auto полю поперечной оси	<div><div><div>auto</div></div><div><div>center</div></div></div> <div><div>flex-start</div></div> <div><div>stretch</div></div>

flex-end



baseline



Как правило, flexbox-верстку начинают с уже озвученных шагов.

- ❑ Идентифицировать контейнер и содержащиеся в нем элементы, применить свойство `display: flex` для контейнера.
- ❑ При необходимости установить для контейнера свойство `flex-direction`.
- ❑ Если нужно, объявить поля и/или значения `flex` для управления размерами элементов.

Поместив все элементы примерно туда, где они должны располагаться, добавьте, если требуется, другие свойства flex-блоков. Я бы порекомендовал максимально подробно ознакомиться с уже рассмотренными концепциями. Продолжайте читать

эту главу, чтобы понять, какие еще возможности предлагает flexbox-верстка, но не волнуйтесь, что все забудете до тех пор, пока не потребуется их применить. Когда они вам понадобятся, вернитесь сюда для получения справки. Большинство этих возможностей интуитивно понятны, хотя вам редко придется ими пользоваться.

5.4.1. Свойства flex-контейнеров

Для управления компоновкой flex-элементов внутри flex-контейнеров к последним можно применять несколько свойств. Первое, `flex-direction`, мы рассмотрели в разделе 5.3. Давайте взглянем на другие.

Свойство flex-wrap

С помощью свойства `flex-wrap` можно заверстывать flex-элементы на новую строку (или строки). Ему присваивается значение `nowrap` (по умолчанию), `wrap` или `wrap-reverse`. При заверстывании элементы не сужаются в соответствии со значениями свойства `flex-shrink`. Вместо этого любой элемент, который мог бы переполнить flex-контейнер, переносится на следующую строку.

Если для свойства `flex-direction` установлено значение `column` или `column-reverse`, то свойство `flex-wrap` позволяет flex-элементам перетечь в новую колонку. Однако это происходит только при ограничении высоты контейнера, в противном случае последний увеличится, чтобы уместить свои flex-элементы.

Свойство flex-flow

Свойство `flex-flow` — это сокращение для свойств `flex-direction` и `flex-wrap`. Например, выражение `flex-flow: column wrap` указывает, что поток flex-элементов будет направлен сверху вниз, при необходимости элементы будут заверстываться в новую колонку.

Свойство justify-content

Свойство `justify-content` управляет распределением элементов вдоль главной оси, если они не заполняют контейнер. Среди поддерживаемых значений есть несколько новых ключевых слов: `flex-start`, `flex-end`, `center`, `space-between` и `space-around`. Значение `flex-start` (по умолчанию) собирает элементы к началу главной оси — с левой стороны при нормальном направлении строки. В этом случае между элементами не будет промежутков, если для них не указаны размеры полей. Значение `flex-end` сдвигает элементы к концу главной оси, а `center` соответственно центрирует их.

Значение `space-between` помещает первый flex-элемент в начало главной оси, а последний — в конец. Оставшиеся элементы размещаются между первым и последним элементами с равными промежутками. Значение `space-around` выполняет подобную функцию с той лишь разницей, что перед первым элементом и после последнего добавляется такой же промежуток.

Промежутки применяются после вычисления полей и значений `flex-grow`. Это значит, что если элементам присвоено ненулевое значение свойства `flex-grow` или для любых элементов установлены автоматические поля по главной оси, то свойство `justify-content` не будет оказывать никакого воздействия.

Свойство `align-items`

В то время как свойство `justify-content` управляет распределением элементов вдоль главной оси, свойство `align-items` настраивает их распределение вдоль поперечной. Изначально ему присвоено значение `stretch`, благодаря которому все элементы заполняют высоту контейнера при горизонтальной разметке или ширину — при вертикальной. Это дает возможность создавать колонки одинаковой высоты.

Остальные значения позволяют `flex`-элементам задавать свой размер естественным образом, при этом не заполняя контейнер (что концептуально аналогично свойству `vertical-align`).

- ❑ Значения `flex-start` и `flex-end` выравнивают элементы вдоль начала или конца поперечной оси (верха или низа строки соответственно).
- ❑ Значение `center` центрирует элементы.
- ❑ Значение `baseline` выравнивает элементы таким образом, что базовая линия первой строки каждого `flex`-элемента оказывается выровненной.

Значение `baseline` полезно, если вы хотите, чтобы базовая линия шапки в одном `flex`-элементе с большим кеглем шрифта была выровнена с базовой линией текста меньшего шрифта в других `flex`-элементах.

СОВЕТ

Очень легко перепутать свойства `justify-content` и `align-items`. Я запоминаю их в терминах форматирования текста: вы можете распределить текст по ширине (англ. `justify`) от края до края. Подобно этому с помощью свойства `vertical-align` встроенные элементы распределяются по вертикали.

Свойство `align-content`

Если вы задействуете заверствывание с помощью свойства `text-wrap`, оно будет управлять междустрочными интервалами внутри `flex`-контейнера по поперечной оси. Поддерживаются значения `flex-start`, `flex-end`, `center`, `stretch` (изначальное значение), `space-between` и `space-around`. Они применяют интервалы аналогично тому, как это делает свойство `justify-content`.

5.4.2. Свойства `flex`-элементов

Я уже описал свойства `flex-grow`, `flex-shrink`, `flex-basis` и привел их общее сокращение `flex` (см. раздел 5.2). Далее рассмотрим два дополнительных свойства `flex`-элементов — `align-self` и `order`.

Свойство align-self

Данное свойство управляет выравниванием flex-элементов вдоль поперечной оси. Оно выполняет ту же функцию, что и свойство flex-контейнера `align-items`, с той лишь разницей, что позволяет задать разные свойства выравнивания для отдельных элементов. Задание значения `auto` (изначального) ссылается на свойство контейнера `align-items`. Любое другое значение переопределяет установки контейнера. Свойство `align-self` поддерживает те же самые ключевые слова для задания значений, что и свойство `align-items`: `flex-start`, `flex-end`, `center`, `stretch` и `baseline`.

Свойство order

Обычно flex-элементы компонуются в том порядке, в каком появляются в файле исходного кода HTML. Элементы располагаются вдоль главной оси. Используя свойство `order`, можно изменить порядок компоновки элементов. Указавайте любые целые числа — положительные или отрицательные. Если нескольким flex-элементам присвоены одинаковые значения, то они будут скомпонованы в порядке появления в исходном коде.

Изначально у всех flex-элементов порядковый номер 0. Присвоение элементу значения `-1` перенесет его в начало списка, а значения `1` — в конец. Для упорядочения элементов можете давать им любые номера. Они не обязательно должны быть последовательными.

ВНИМАНИЕ!

Будьте осторожны при использовании свойства `order`. Значительные отличия порядка компоновки элементов визуальной разметки от порядка в исходном коде могут навредить доступности вашего сайта. В некоторых браузерах навигация с помощью клавиши `Tab` по-прежнему будет производиться в порядке, указанном в файле исходного кода, что может запутать. Программы экранного доступа для людей с ограничениями зрения в большинстве случаев также будут использовать порядок из исходного кода.

5.4.3. Выравнивание flex-блоков

Чтобы завершить нашу страницу, займемся выравниванием. В последнем блоке имеются отформатированный ценник и кнопка побуждения к действию. Последний шаг при создании страницы по завершении написания кода должен выглядеть так, как показано на рис. 5.18.

Разметка для этой секции уже есть на странице. Соответствующий фрагмент кода выглядит следующим образом:

```
<div class="tile centered">
  <small>Тарифные планы от</small>
  <div class="cost">
    <span class="cost-currency">P</span>
    <span class="cost-roubles">999</span>
```

```

    <span class="cost-kopecs">.00</span>
  </div>
  <a class="cta-button" href="/pricing">
    Зарегистрироваться
  </a>
</div>

```



Рис. 5.18. Отформатированный с помощью flexbox-верстки текст

Текст **₽999.00** упакован в элемент `div class="cost"`, который вы будете использовать как flex-контейнер. Текст состоит из трех flex-элементов для трех выравниваемых частей текста: **₽**, **999** и **.00**. Для них я решил взять элементы `span`, а не `div`, так как элементы `span` встроены по умолчанию. Если по какой-либо причине CSS-файл не загрузится или браузер не поддерживает flexbox-верстку, то текст **₽999.00** по-прежнему будет выведен на одной строке.

В листинге 5.11 вы воспользуетесь свойством `justify-content` для горизонтального центрирования flex-элементов в контейнере. Далее задействуете свойства `align-items` и `align-self`, чтобы выровнять элементы по вертикали. Добавьте в свою таблицу стилей следующий код.

Листинг 5.11. Установка стилей для текста с ценой

```

.centered {
  text-align: center;
}

.cost {
  display: flex;
  justify-content: center;
  align-items: center;
  line-height: .7;
}

.cost > span {
  margin-top: 0;
}

.cost-currency {

```

Центрирование flex-элементов по главной и поперечной осям

← Переопределение полей после лоботомированной совы


```

font-size: 2rem;
}
.cost-roubles {
  font-size: 4rem;
}
.cost-kopecks {
  font-size: 1.5rem;
  align-self: flex-start;
}

.cta-button {
  display: block;
  background-color: #cc6b5a;
  color: white;
  padding: .5em 1em;
  text-decoration: none;
}

```

Установка разных размеров шрифта для каждой части ценника

Переопределение свойства align-items для данного элемента выравниванием его по верхнему краю вместо центра

Данный код создает разметку для отформатированного с помощью стилей текста **9999.00**, а также определяет класс **centered** для центрирования остального текста и класс **cta-button** для кнопки побуждения к действию.

В коде есть одно странное объявление — **line-height: .7**. Оно необходимо по той причине, что высота текста внутри каждого flex-элемента — это именно то, что определяет высоту каждого отдельного элемента. То есть высота элементов несколько больше, чем текста: высота в единицах **em** включает потомков, коих у текста нет, поэтому символы текста в данном случае несколько ниже, чем **1 em**. Я шел к этому значению методом проб и ошибок до тех пор, пока верхние края символов **20** и **.00** визуально не выровнялись (см. главу 13 для получения дополнительной информации о работе с текстом).

5.5. Пара вещей, о которых следует знать

Для технологии CSS flexbox-верстка — это огромный шаг вперед. Велик соблазн, познакомившись с ней, начать применять ее для всех элементов страницы. Но все же следует использовать нормальный поток документа и не прибегать к flexbox-верстке только потому, что вы уверены в ее необходимости. Не нужно избегать flexbox-верстки, но не стоит и предаваться излишествам.

5.5.1. Flex-баги

Не во всех браузерах flexbox-верстка реализована идеально, особенно это касается Internet Explorer 10 и 11. В большинстве случаев технология работает нормально, но при неблагоприятном стечении обстоятельств вы можете столкнуться с несколькими программными ошибками на странице. Всегда тестируйте свои flexbox-сайты во всех устаревших браузерах, поддержка которых может потребоваться.

Вместо того чтобы тратить много времени на обсуждение программных ошибок, с которыми вы можете столкнуться или никогда не встретиться, я дам ссылку на замечательный ресурс под названием Flexbugs (github.com/philipwalton/flexbugs). Это актуальный список всех известных браузерных ошибок, связанных с flex-блоками (на момент написания данной книги их было 17). В ресурсе приводятся точное объяснение того, какие именно обстоятельства вызвали ошибки, и в большинстве случаев — способы их обхода. Если когда-нибудь вы обнаружите, что в каком-то браузере flex-блоки ведут себя странно, посетите эту страницу, чтобы проверить, не столкнулись ли вы с одной из браузерных ошибок, связанных с flexbox-версткой.

5.5.2. Полноформатная разметка

Интересная особенность flexbox-верстки — то, каким образом вычисляются flex-размеры в зависимости от количества flex-элементов и объема (и размера) контента в них. Это способно обусловить странное поведение, если страница большая или загружается через медленное соединение.

По мере загрузки контента браузер постепенно выводит его на экран, даже если страница загружена не полностью. Предположим, что выполнена разметка на три колонки (`flex-direction: row`). Если загрузится контент двух из них, то браузер может обработать страницу прежде, чем загрузится контент третьей колонки. Затем по мере загрузки оставшегося контента браузер пересчитывает размеры всех flex-элементов и многократно обрабатывает страницу. Пользователь на мгновение увидит разметку в две колонки, после чего их размер изменится (и даже весьма значительно) и появится третья колонка.

Джейк Арчибальд (Jake Archibald), разработчик и ярый приверженец Google Chrome, написал об этом в статье, размещенной на jakearchibald.com/2014/dont-use-flexbox-for-page-layout/. В ней можно увидеть примеры описанного поведения. Он дает одну рекомендацию — для полноформатной разметки применять CSS-сетки (о чем мы поговорим в следующей главе).

ПРИМЕЧАНИЕ

Описанное ранее поведение — это проблема только при наличии нескольких колонок в строке. Такого не происходит при использовании конфигурации «несколько строк в одной колонке» (`flex-direction: column`) для основной разметки страницы.

Итоги главы

- ❑ Применяйте flexbox-верстку для создания адаптивной, легкой в управлении разметки контента страницы.
- ❑ Инструмент Autoprefixer может упростить поддержку flexbox-сайтов в старых браузерах.

- ❑ Используйте значение `flex` для задания практически любой комбинации размеров flex-элементов.
- ❑ Вкладывайте flex-блоки друг в друга для сочетания более сложных видов разметки и заполнения блоков, чья высота формируется естественным образом.
- ❑ Flexbox-верстка автоматически создает колонки одинаковой высоты.
- ❑ Задействуйте свойство `align-items` или `align-self` для вертикального центрирования flex-элементов внутри flex-контейнера.

6

CSS-сетки

В этой главе

- Использование первой реальной системы разметки CSS — сетки.
- Возможности CSS-сетки.
- Компоновка элементов с помощью сетки.
- Совместное применение flex-блоков и CSS-сетки для создания целостной разметки веб-страницы.

Flexbox-верстка произвела революцию в том, как создается разметка во Всемирной паутине, но это только начало. У данной технологии есть «большой брат» — новая спецификация под названием «модуль CSS3 Grid Layout». Совместно эти две спецификации представляют собой полнофункциональный механизм для верстки сайтов, какого вы не видели ранее.

В этой главе я покажу, как начать верстать с помощью CSS-сетки уже сегодня. Я представлю общую информацию о том, как работает эта технология, затем проведу вас по нескольким примерам, чтобы проиллюстрировать некоторые возможности верстки с помощью CSS-сетки. Создать базовую сетку просто. Мощности этой технологии достаточно и для создания сложной разметки, но это уже требует изучения новых свойств и ключевых слов. Данная глава познакомит вас с ними.

CSS-сетка позволяет определить двумерную разметку колонок и строк, а затем поместить в нее элементы. Некоторые из них заполняют только одну ячейку сетки, а другие способны расширяться и захватывать соседние колонки и строки. Можно задать размер сетки точно или позволить ей автоматически определять его по мере необходимости, чтобы уместить весь нужный контент. Помещайте элементы на точные места внутри сетки или позволяйте им плавать естественным образом для заполнения промежутков. Сетка позволяет создавать сложную разметку, пример приведен на рис. 6.1.

мире, конечно, если вы проведете небольшую работу для поддержки резервной конструкции. Мы рассмотрим это ближе к концу главы.

ПРИМЕЧАНИЕ

Компания Microsoft реализовала раннюю версию CSS-сетки с вендорными префиксами. Это означает, что браузеры Internet Explorer 10 и 11 частично поддерживают CSS-сетку с использованием префикса `-ms-`. Для поддержки этих браузеров задействуйте инструмент Autoprefixer, как поясняется в главе 5 (см. врезку «Вендорные префиксы»).

Задействование экспериментальных возможностей

Прежде чем в браузерах начали поддерживать CSS-сетку по умолчанию, они позволяли разработчикам задействовать ее. Несмотря на то что CSS-сетка сейчас поддерживается, все же важно знать, как получить доступ к экспериментальным возможностям, если вы захотите изучить их в будущем.

В браузерах Chrome и Opera это осуществляется установкой соответствующего флажка в настройках. В браузере Chrome введите в адресную строку `chrome://flags` и нажмите клавишу **Enter**. В браузере Chrome введите `opera://flags`. Затем пролистайте список до флажка **Experimental Web Platform Features** (Экспериментальные возможности веб-платформы) или воспользуйтесь функцией поиска в браузере и выберите пункт раскрывающегося списка **Enabled** (Включено).

Если вы предпочитаете браузер Firefox, то скачайте и установите версию браузера для разработчиков Firefox Developer Edition (www.mozilla.org/ru-RU/firefox/developer/) или Firefox Nightly (nightly.mozilla.org). Если используете браузер Safari, сможете установить версию Safari Technology Preview или Webkit Nightly Builds.

6.1.1. Создание базовой сетки

Выполним простейшую CSS-сетку, чтобы удостовериться, что она работает в вашем браузере. Скомпонуйте шесть блоков в три колонки (рис. 6.2). Код разметки для этой сетки приведен в листинге 6.1.

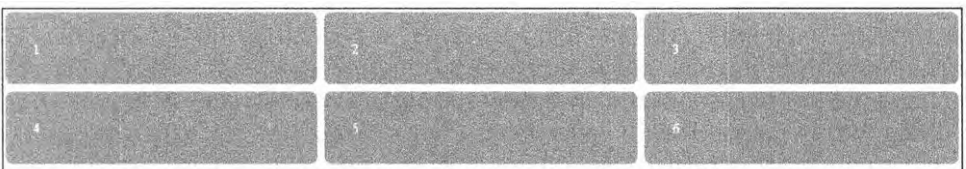



Рис. 6.2. Простая сетка с тремя колонками и двумя строками

Создайте новую страницу и подключите ее к новой таблице стилей. Добавьте на страницу код, приведенный в листинге 6.1. В данном коде я указал числа от 1 до 6, чтобы то, где окажется каждый элемент, было очевидным.

Листинг 6.1. Сетка с шестью элементами

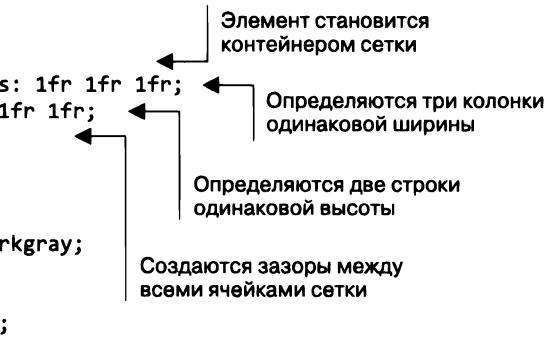


```
<div class="grid">
  <div class="1">1</div>
  <div class="2">2</div>
  <div class="3">3</div>
  <div class="4">4</div>
  <div class="5">5</div>
  <div class="6">6</div>
</div>
```

Как и при flexbox-верстке, CSS-сетка применяется к двум уровням архитектуры DOM. Элемент со свойством `display: grid` становится *контейнером сетки*, его потомки — *элементами сетки*.

Далее вы задействуете несколько новых свойств для определения особенностей сетки. Добавьте стили из листинга 6.2 в таблицу стилей.

Листинг 6.2. Выкладываем базовую сетку



```
.grid {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  grid-template-rows: 1fr 1fr;
  grid-gap: 0.5em;
}

.grid > * {
  background-color: darkgray;
  color: white;
  padding: 2em;
  border-radius: 0.5em;
}
```

Если ваш браузер поддерживает CSS-сетку, то приведенный код будет обработан для создания шести блоков равного размера, расположенных в трех колонках (см. рис. 6.2). В коде производятся несколько новых операций. Рассмотрим их поближе.

Вначале вы применили свойство `display: grid` для определения контейнера сетки. Контейнер ведет себя как блочный элемент, заполняя 100 % доступной ширины. Хотя это и не показано в листинге 6.2, но можно использовать и значение `inline-grid`: в этом случае элемент будет строчным, а его ширина — не большей, чем необходимо, чтобы уместить его потомков. Скорее всего, вы нечасто будете работать со значением `inline-grid`.

Далее идут новые свойства — `grid-template-columns` и `grid-template-rows`. Они устанавливают размер каждой колонки и каждой строки сетки. В данном

примере используется новая единица измерения *fr*, представляющая собой доли колонки или строки. Поведение этой единицы аналогично фактору *flex-grow* для *flex*-блоков. Объявление `grid-template-columns: 1fr 1fr 1fr` создает три колонки одинакового размера.

Применять единицы *fr* для каждой колонки или строки не обязательно. Можете брать и другие единицы, например пиксели, *em* или проценты, или смешивать их. В частности, выражение `grid-template-columns: 300px 1fr` определит колонку фиксированной ширины 300 пикселей, рядом с которой будет колонка, заполняющая все доступное пространство. Колонка *2fr* будет в два раза шире колонки *1fr*.

Наконец, свойство `grid-gap` определяет пространство, которое необходимо добавить для создания промежутков между каждой парой ячеек. Опционально вы можете указать два значения для индивидуальной установки горизонтального и вертикального промежутков, например: `grid-gap: 0.5em 1em`.

Рекомендую поэкспериментировать с этими значениями и понаблюдать, как они влияют на окончательную разметку. Добавляйте новые колонки или изменяйте их ширину. Добавляйте или удаляйте элементы сетки. Продолжайте исследовать другие виды разметки по мере чтения главы, так как это наилучший способ ознакомиться со всеми ее возможностями.

6.2. Анатомия сетки

Важно разбираться, как действуют различные части сетки. Я уже упоминал контейнеры и элементы сетки, из которых она скомпонована. На рис. 6.3 проиллюстрированы четыре других важных термина, которые нужно знать.

- ❑ *Линия сетки* формирует структуру сетки. Она может быть вертикальной или горизонтальной и лежать на любой из двух сторон строки или колонки. Свойство `grid-gap`, если оно определено, накладывается сверху линий сетки.
- ❑ *Полоса сетки* — расстояние между двумя соседними линиями сетки. У сетки есть горизонтальные (строки) и вертикальные (колонки) полосы.
- ❑ *Ячейка сетки* — единичное пространство в сетке, где пересекаются горизонтальная и вертикальная полосы сетки.
- ❑ *Область сетки* — прямоугольный участок, составленный из одной или нескольких ячеек сетки. Область определяется между двумя вертикальными и двумя горизонтальными линиями сетки.

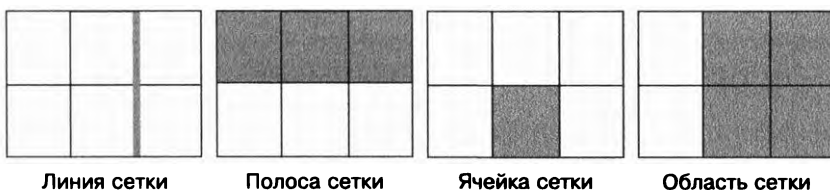


Рис. 6.3. Части сетки

Вы будете говорить об этих частях при создании CSS-сетки. В частности, объявляя свойство `grid-template-columns: 1fr 1fr 1fr`, вы создаете три вертикальные *полосы сетки* одинаковой ширины. Это выражение определяет также четыре *линии сетки*: одну с левого края, две между каждой парой ее полос и еще одну вдоль правого края.

В главе 5 вы создали страницу с помощью flex-блоков. Еще раз посмотрим на ее дизайн и подумаем о том, как реализовать его с помощью CSS-сетки. Дизайн страницы показан на рис. 6.4. Я добавил пунктирные линии для обозначения расположения ячеек сетки. Обратите внимание на то, что некоторые разделы занимают несколько ячеек, заполняя большую *область сетки*.

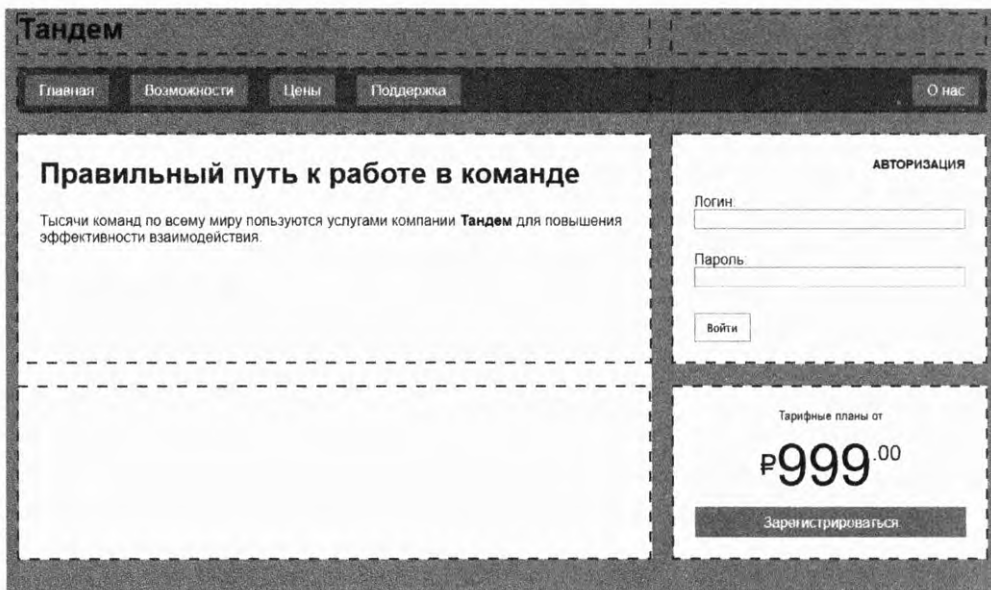


Рис. 6.4. Разметка страницы, созданная с помощью CSS-сетки

Эта сетка состоит из двух колонок и четырех строк. Две верхние полосы отведены под название страницы («Тандем») и основную навигационную панель. Основная область заполняет оставшиеся две ячейки первой вертикальной полосы, два блока боковой панели помещены в две оставшиеся ячейки второй вертикальной полосы.

ПРИМЕЧАНИЕ

Дизайну совершенно не обязательно охватывать все ячейки сетки. Оставьте пустую ячейку там, где хотите, чтобы было свободное пространство.

Важно заметить, что CSS-сетка не доказывает бесполезность flex-блоков. По мере разработки страницы вы убедитесь, что flex-блоки — все еще важная составляющая разметки. Я отдельно выделяю те области страницы, где их использование выглядит разумным.

При создании этой страницы с применением flex-блоков приходилось вкладывать одни элементы в другие, так вы использовали flex-блоки для определения колонок и вкладывали в них другие flex-блоки для определения строк (см. листинг 5.1). Разработка этой разметки с помощью CSS-сетки требует другой структуры HTML: нужно сделать HTML-код более плоским. Каждый элемент, помещаемый в сетку, должен быть дочерним по отношению к основному контейнеру сетки. Новая разметка приведена далее. Создайте новую страницу (или отредактируйте страницу из главы 5), как показано в листинге 6.3.

Листинг 6.3. Структура HTML для CSS-сетки

```
<body>
  <div class="container">
    <header>
      <h1 class="page-heading">Тандем</h1>
    </header>
    <nav>
      <ul class="site-nav">
        <li><a href="/">Главная</a></li>
        <li><a href="/features">Возможности</a></li>
        <li><a href="/pricing">Цены</a></li>
        <li><a href="/support">Поддержка</a></li>
        <li class="nav-right">
          <a href="/about">О нас</a>
        </li>
      </ul>
    </nav>
    <main class="main tile">
      <h1>Правильный путь к работе в команде</h1>
      <p>Тысячи команд по всему миру пользуются
        услугами компании <b>Тандем</b> для повышения
        эффективности взаимодействия.</p>
    </main>
    <div class="sidebar-top tile">
      <form class="login-form">
        <h3>Авторизация</h3>
        <p>
          <label for="username">Логин:</label>
          <input id="username" type="text"
            name="username"/>
        </p>
        <p>
          <label for="password">Пароль:</label>
          <input id="password" type="password"
            name="password"/>
        </p>
        <button type="submit">Войти</button>
      </form>
    </div>
    <div class="sidebar-bottom tile centered">
      <small>Тарифные планы от</small>
```

"container" становится контейнером сетки

Каждый элемент сетки должен быть потомком контейнера сетки

```

<div class="cost">
  <span class="cost-currency">Р</span>
  <span class="cost-roubles">999</span>
  <span class="cost-kopecks">.00</span>
</div>
<a class="cta-button" href="/pricing">
  Зарегистрироваться
</a>
</div>
</div>
</body>

```

В этой версии страницы мы разметили каждый раздел страницы как отдельный элемент сетки: шапка, панель навигации (nav), основной раздел и две боковые панели. Также добавили класс title в основной раздел и две боковые панели, чтобы создать белый фон и отступы, одинаковые для этих элементов.

Применим к странице CSS-сетку и поместим каждый раздел туда, где он должен находиться. Воспользуемся большим количеством стилей из главы 5, но для начала определим общую форму страницы. (Мне проще создавать страницу, вначале решая большие задачи, а затем прорабатывая детали.) После создания базовой сетки страница будет выглядеть так, как на рис. 6.5.

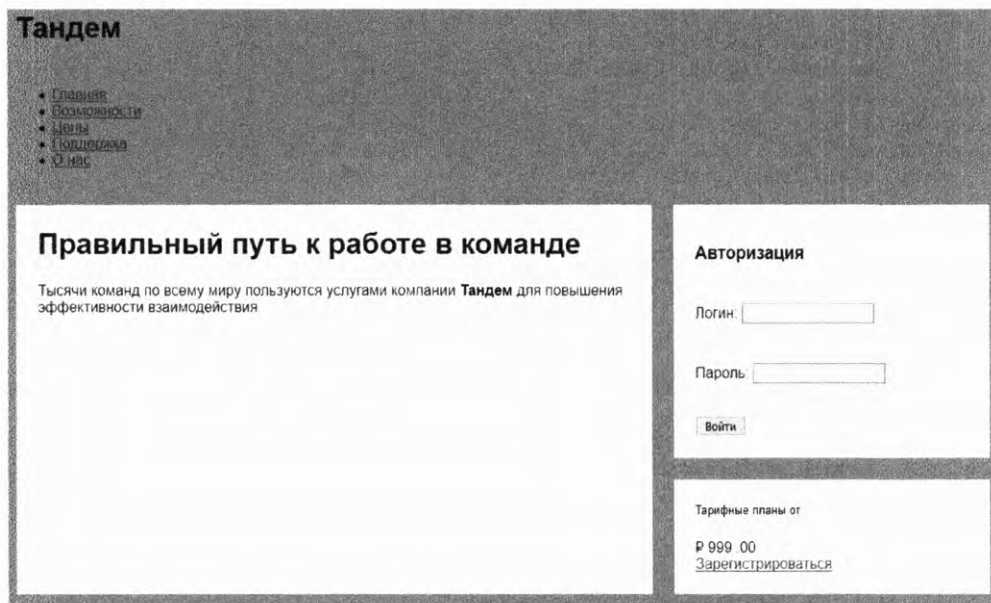


Рис. 6.5. Страница с воссозданной базовой сеткой

Создайте пустую таблицу стилей и подключите к ней страницу. Добавьте в новую таблицу стилей код из листинга 6.4. В этом коде появляются несколько новых концепций, о которых мы вскоре поговорим.

Листинг 6.4. Разметка страницы верхнего уровня с помощью CSS-сетки

```

:root {
    box-sizing: border-box;
}

*,
::before,
::after {
    box-sizing: inherit;
}

body {
    background-color: #709b90;
    font-family: Helvetica, Arial, sans-serif;
}

.container {
    display: grid;
    grid-template-columns: 2fr 1fr;
    grid-template-rows: repeat(4, auto);
    grid-gap: 1.5em;
    max-width: 1080px;
    margin: 0 auto;
}

header,
nav {
    grid-column: 1 / 3;
    grid-row: span 1;
}

.main {
    grid-column: 1 / 2;
    grid-row: 3 / 5;
}

.sidebar-top {
    grid-column: 2 / 3;
    grid-row: 3 / 4;
}

.sidebar-bottom {
    grid-column: 2 / 3;
    grid-row: 4 / 5;
}

.tile {
    padding: 1.5em;
    background-color: #fff;
}

.tile > :first-child {

```

Определяются две вертикальные полосы сетки

Определяются четыре горизонтальные полосы сетки, размер задается автоматически

Занимает промежуток между вертикальными линиями сетки 1 и 3

Занимает промежуток, совпадающий с одной горизонтальной полосой сетки

Помещает другие элементы сетки между различными линиями сетки

```

margin-top: 0;
}

.tile * + * {
margin-top: 1.5em;
}

```

В этом коде появляется сразу несколько новых концепций. Рассмотрим их по очереди.

Вы установили контейнер сетки и определили полосы сетки внутри него с помощью свойств `grid-template-columns` и `grid-template-rows`. Колонки задаются с использованием долей — 2 fr и 1 fr, таким образом, первая колонка увеличится и станет в два раза больше второй. Для строк применяется кое-что новое — функция `repeat()`. Это сокращенный вариант для определения нескольких полос сетки.

Объявление `grid-template-rows: repeat(4, auto);` определяет четыре горизонтальные полосы сетки со значением высоты `auto`. Такое объявление эквивалентно объявлению `grid-template-rows: auto auto auto auto`. Полоса со значением высоты `auto` будет увеличиваться по мере необходимости в соответствии с размером контента.

С помощью нотации `repeat()` можно определять также повторяющиеся последовательности. В частности, выражение `repeat(3, 2fr 1fr)` определяет шесть полос сетки путем трехкратного повторения последовательности, выводя результат в виде `2fr 1fr 2fr 1fr 2fr 1fr`. На рис. 6.6 проиллюстрированы получившиеся колонки.

`grid-template-columns: repeat(3, 2fr 1fr);`

2fr	1fr	2fr	1fr	2fr	1fr
-----	-----	-----	-----	-----	-----

`grid-template-columns: 1fr repeat(3, 3fr) 1fr;`

1fr	3fr	3fr	3fr	1fr
-----	-----	-----	-----	-----

Рис. 6.6. Использование функции `repeat()` для определения повторяющейся последовательности в шаблоне

Можете также взять функцию `repeat()` как часть более длинной последовательности. Так, выражение `grid-template-columns: 1fr repeat(3, 3fr) 1fr` определяет колонку размером 1 fr, после которой стоят три колонки размером 3 fr, а затем еще одна колонка размером 1 fr (или `1fr 3fr 3fr 3fr 1fr`). Как видите, развернутая версия выражения тяжела для визуального восприятия, поэтому функция `repeat()` очень полезна.

6.2.1. Нумерация линий сетки

После определения полос сетки следующая порция кода помещает каждый элемент сетки в некую конкретную позицию сетки. Браузер присваивает каждой линии сетки номера (рис. 6.7). В CSS они обозначают, куда должен быть помещен тот или иной элемент.

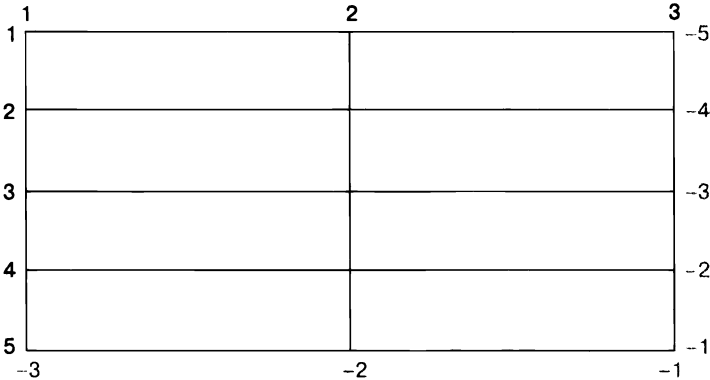


Рис. 6.7. Линии сетки пронумерованы, начиная с 1, в верхнем левом углу. Отрицательные числа показывают положение относительно нижнего правого угла

Вы можете использовать номера сетки для обозначения того, куда должен быть помещен тот или иной элемент, с помощью свойств `grid-column` или `grid-row`. Если хотите, чтобы элемент сетки занимал пространство от линии 1 до линии 3, примените к нему свойство `grid-column: 1 / 3`. Или задействуйте свойство `grid-column: 3 / 5`, чтобы этот элемент занимал пространство от горизонтальной линии сетки 3 до линии 5. Совместная работа этих свойств устанавливает область сетки, используемую для элемента.

На вашей странице некоторые элементы сетки размещены следующим образом:

```
.main {  
  grid-column: 1 / 2;  
  grid-row: 3 / 5;  
}  
.sidebar-top {  
  grid-column: 2 / 3;  
  grid-row: 3 / 4;  
}  
.sidebar-bottom {  
  grid-column: 2 / 3;  
  grid-row: 4 / 5;  
}
```

Этот код помещает элемент `main` в первую колонку (между линиями сетки 1 и 2), при этом расширяя его на третью и четвертую строки (между линиями сетки 3 и 5). Код помещает каждый блок боковой панели в правую колонку (между линиями сетки 2 и 3) друг над другом — в третью и четвертую строки.

ПРИМЕЧАНИЕ

Эти свойства, по сути, являются сокращениями: `grid-column` — для свойств `grid-column-start` и `grid-column-end`, `grid-row` — для свойств `grid-row-start` и `grid-row-end`. Косая черта необходима только в нотации сокращенного свойства для разделения двух значений. Пробелы перед косой чертой и после нее необязательны.

Набор правил, помещающий элементы `header` и `nav` в верхнюю часть страницы, выглядит несколько иначе. Далее я воспользовался одним и тем же набором правил для нацеливания на оба элемента.

```
header,  
nav {  
  grid-column: 1 / 3;  
  grid-row: span 1;  
}
```

В этом примере используется продемонстрированное ранее свойство `grid-column`, благодаря которому элемент расширяется и заполняет всю ширину сетки. Вы можете указать значения свойств `grid-column` и `grid-row` с помощью специального ключевого слова `span` (применялось здесь для свойства `grid-row`). Это ключевое слово сообщает браузеру, что элемент будет занимать одну полосу сетки. Я не указывал явно, с какой строки следует начать или на какой закончить, поэтому элемент будет размещен автоматически с помощью *алгоритма размещения* элементов сетки. Алгоритм выложит элементы таким образом, чтобы они заполнили первое свободное пространство на сетке, которого для них будет достаточно. В данном случае это первая и вторая строки. Рассмотрим автоматическое размещение чуть позже в этой главе.

6.2.2. Совместная работа с flex-блоками

Узнав про CSS-сетку, разработчики часто задают вопросы об *flex-блоках*, в частности: являются ли эти два метода разметки конкурирующими? Ответ на этот вопрос: нет, они взаимодополняющие. Эти два метода были разработаны по большей части совместно. Несмотря на то что существует некое наложение возможностей технологий, все же они наиболее эффективны в разных сценариях. Выбор между *flexbox-версткой* и CSS-сеткой должен основываться на конкретных потребностях при создании того или иного элемента дизайна. Между этими двумя методами есть два существенных различия.

- ❑ *Flexbox-верстка* в основном одномерная, тогда как CSS-сетка — двумерная.
- ❑ *Flexbox-верстка* отталкивается от контента, а CSS-сетка — от разметки.

Так как *flexbox-верстка* одномерная, она идеальна для строк (или колонок) подобных элементов. Данная технология содержит перенос строк с помощью свойства `flex-wrap`, однако не существует какого-либо способа выровнять элементы одной строки с элементами другой. CSS-сетка же двумерная и предназначена для ситуа-

ций, когда необходимо выравнивать элементы в одной полосе с элементами в другой. Данное различие проиллюстрировано на рис. 6.8.

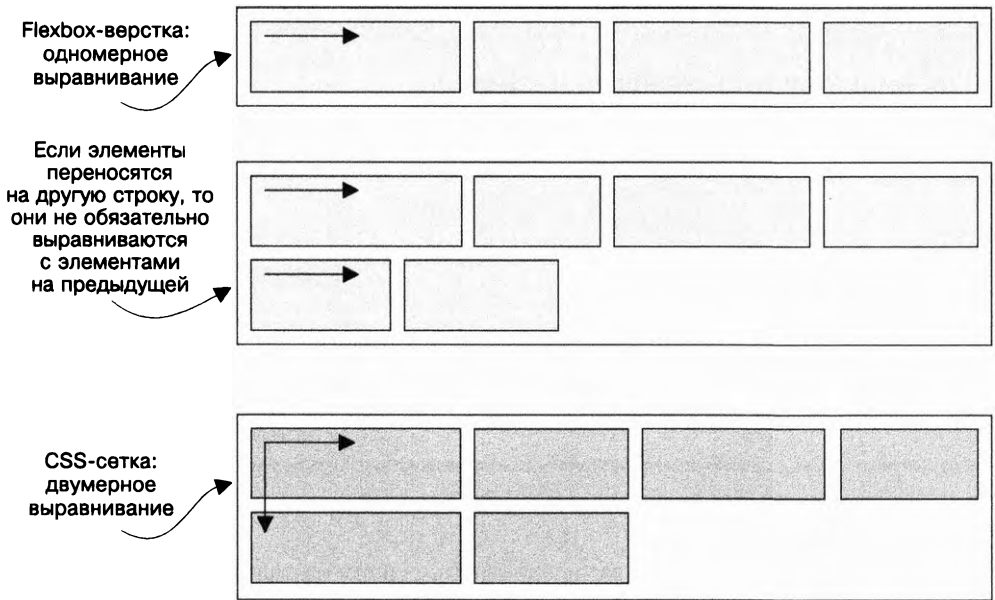


Рис. 6.8. Flexbox-верстка выравнивает элементы в одном измерении, тогда как CSS-сетка — в двух

Второе существенное различие, на которое обращает внимание представитель CSS WG Рэйчел Эндрю (Rachel Andrew), состоит в том, что flexbox-верстка отталкивается от контента, тогда как CSS-сетка — от разметки. Flexbox-верстка позволяет упорядочить последовательность элементов в строке или колонке, однако при этом нет необходимости явно устанавливать их размерность: контент элементов устанавливает объем пространства, требующийся каждому из них.

Работая же с CSS-сеткой, вы в первую очередь описываете разметку, а затем помещаете в эту структуру элементы.

Мы позиционировали основные разделы страницы, используя CSS-сетку, так как хотим, чтобы контент строго придерживался сетки в том виде, в каком она определена. В то же время можем позволить, чтобы контент некоторых других элементов страницы, таких как навигационная панель, сильнее влиял на результат, таким образом, элементы с большим количеством текста могут быть шире, а с меньшим — уже. Это горизонтальная (одномерная) разметка. По этой причине для данных элементов flexbox-верстка — более оптимальное решение. Для завершения страницы отформатируем их с помощью flexbox-верстки.

На рис. 6.9 показана страница с верхней навигационной панелью, состоящей из списка ссылок, выровненных по горизонтали. Воспользуемся flex-контейнером также для цифр отформатированного с помощью стилей ценника справа внизу. После добавления стилей страница примет заверченный вид.

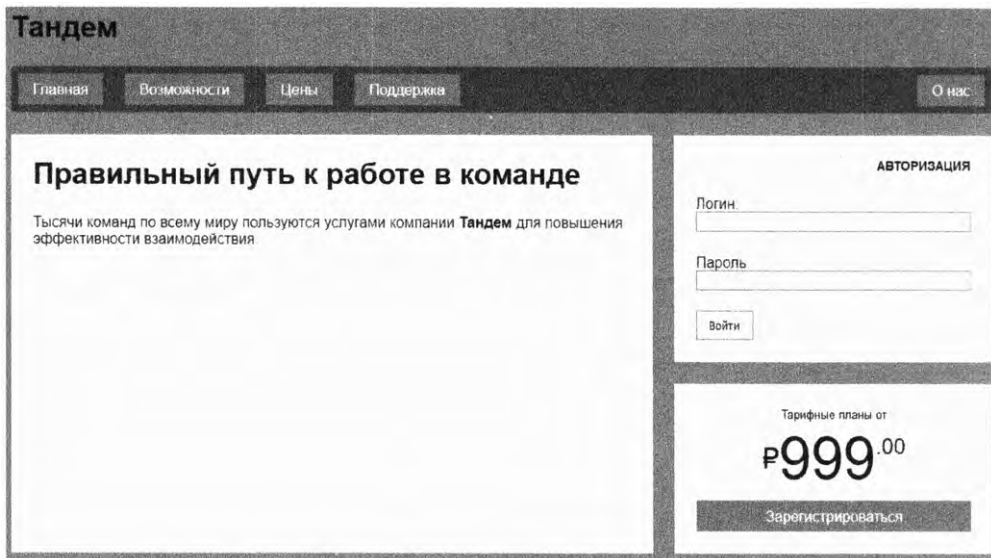


Рис. 6.9. Полностью отформатированная с помощью стилей страница

Стили для выполнения этих задач идентичны стилям из главы 5, за вычетом высокоуровневой разметки, которую мы применили с использованием CSS-сетки (см. листинг 6.4). Я повторил их в листинге 6.5. Добавьте этот код в свою таблицу стилей.

Листинг 6.5. Заключительные стили страницы

```
.page-heading {
  margin: 0;
}

.site-nav {
  display: flex;
  margin: 0;
  padding: .5em;
  background-color: #5f4b44;
  list-style-type: none;
  border-radius: .2em;
}

.site-nav > li {
  margin-top: 0;
}

.site-nav > li > a {
  display: block;
  padding: .5em 1em;
  background-color: #cc6b5a;
  color: white;
}
```

← Flex-навигация

```
text-decoration: none;
}

.site-nav > li + li {
  margin-left: 1.5em;
}

.site-nav > .nav-right {
  margin-left: auto;
}

.login-form h3 {
  margin: 0;
  font-size: .9em;
  font-weight: bold;
  text-align: right;
  text-transform: uppercase;
}

.login-form input:not([type=checkbox]):not([type=radio]) {
  display: block;
  margin-top: 0;
  width: 100%;
}

.login-form button {
  margin-top: 1em;
  border: 1px solid #cc6b5a;
  background-color: white;
  padding: .5em 1em;
  cursor: pointer;
}

.centered {
  text-align: center;
}

.cost {
  display: flex;
  justify-content: center;
  align-items: center;
  line-height: .7;
}

.cost > span {
  margin-top: 0;
}

.cost-currency {
  font-size: 2rem;
}

.cost-roubles {
```

← Стилизованный с помощью
flexbox-верстки «ценник»

```
font-size: 4rem;
}
.cost-kopecks {
  font-size: 1.5rem;
  align-self: flex-start;
}

.cta-button {
  display: block;
  background-color: #cc6b5a;
  color: white;
  padding: .5em 1em;
  text-decoration: none;
}
```

Когда дизайн требует выравнивания элементов в двух измерениях, используйте CSS-сетку, если же ваши заботы ограничиваются одномерным потоком, то flexbox-верстку. На практике это часто (но не всегда) обозначает, что CSS-сетка будет более разумной для высокоуровневой разметки, тогда как flexbox-верстка — лишь для нескольких элементов во всех областях сетки. По мере продолжения работы с обеими технологиями вы научитесь определять, которая из них лучше подходит в том или ином случае.

6.3. Альтернативный синтаксис

Существует два альтернативных синтаксиса для компоновки элементов сетки: именованные линии сетки и именованные области сетки. Выбор между ними — это вопрос ваших приоритетов. Для некоторых видов дизайна один синтаксис может быть более удобен, чем другой. Давайте рассмотрим оба.

6.3.1. Присвоение имен линиям сетки

Иногда в ходе работы довольно сложно отслеживать все пронумерованные линии сетки, особенно если полос в ней много. Для упрощения можете давать линиям сетки имена и использовать их вместо номеров. При объявлении полос сетки следует поместить имя в скобки для того, чтобы присвоить имя линии между любыми двумя полосами:

```
grid-template-columns: [start] 2fr [center] 1fr [end];
```

Данное объявление определяет сетку с двумя колонками и тремя вертикальными линиями, названными `start`, `center` и `end`. В дальнейшем при помещении элементов на сетку можете ссылаться на эти имена вместо номеров, например:

```
grid-column: start / center;
```

Это объявление помещает элемент на сетку таким образом, что он занимает пространство от линии 1 (`start`) до линии 2 (`center`). Вы также можете присвоить

разные имена одним и тем же линиям сетки, как показано в следующем примере (я добавил переносы строк для удобства чтения):

```
grid-template-columns: [left-start] 2fr
                      [left-end right-start] 1fr
                      [right-end];
```

В этом объявлении линии 2 сетки присвоены имена `left-end` и `right-start`. Используйте любое из них при размещении элемента сетки. Данное объявление позволяет выполнить и еще один трюк: присвоив линиям сетки имена `left-start` и `left-end`, вы определили область с именем `left`, занимающую пространство между ними. Суффиксы `-start` и `-end` играют роль своего рода ключевых слов, определяющих промежуточный участок. Если вы примените к элементу свойство `grid-column: left`, то данный элемент займет пространство от `left-start` до `left-end`.

В CSS в листинге 6.6 для разметки страницы используются именованные линии сетки. Это дает тот же результат, что и листинг 6.4. Обновите данный фрагмент кода в своей таблице стилей.

Листинг 6.6. CSS-сетка с именованными линиями

```
.container {
  display: grid;
  grid-template-columns: [left-start] 2fr
                        [left-end right-start] 1fr
                        [right-end];
  grid-template-rows: repeat(4, [row] auto);
  grid-gap: 1.5em;
  max-width: 1080px;
  margin: 0 auto;
}

header,
nav {
  grid-column: left-start / right-end;
  grid-row: span 1;
}

.main {
  grid-column: left;
  grid-row: row 3 / span 2;
}

.sidebar-top {
  grid-column: right;
  grid-row: 3 / 4;
}

.sidebar-bottom {
  grid-column: right;
  grid-row: 4 / 5;
}
```

Именованная каждая вертикальная линия сетки

Именованная горизонтальных линий сетки словом row

Занимает пространство от left-start до left-end

Помещение элемента сетки, начинающегося с третьей строки и занимающего пространство в две полосы сетки

Занимает пространство от right-start до right-end

В этом примере мы поместили элементы в нужные колонки сетки, используя именованные линии. В данном коде также объявлена именованная горизонтальная линия сетки внутри функции `repeat()`. Это действие именует все горизонтальные линии `row`, за исключением последней. Многократное применение одного и того же имени может показаться необычным, однако в этом нет ничего криминального. Далее поместите элемент `main` так, чтобы он начинался на строке 3 (третья линия сетки с именем `row`) и занимал пространство двух полос, начиная с этой линии.

Именованные линии сетки используются множеством способов. То, как вы это делаете, может различаться от сетки к сетке в зависимости от структуры каждой конкретной сетки. Один из вариантов приведен на рис. 6.10.

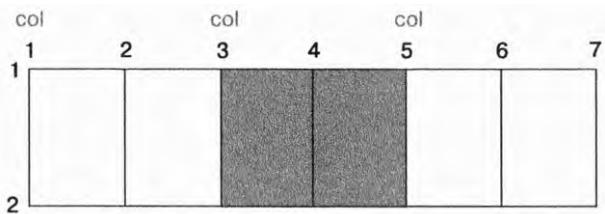


Рис. 6.10. Размещение у второй линии сетки `col` элемента, занимающего две полосы (`col 2 / span 2`)

В этом сценарии представлена повторяющаяся последовательность двух колонок сетки, при которой присваиваются имена линиям перед каждой парой полос (`grid-template-columns: repeat(3, [col] 1fr 1fr)`). Затем можете задействовать именованные линии для позиционирования элемента во втором наборе колонок (`grid-column: col 2 / span 2`).

6.3.2. Присвоение имен областям сетки

Существует и другой подход — присвоить имена областям сетки. Вместо подсчета и именования линий используйте именованные области для компоновки элементов в сетке. Это можно сделать с помощью свойств `grid-template` контейнера сетки и `grid-area` элементов сетки.

Код, приведенный в листинге 6.7, — пример этого приема. Результат выполнения кода — точно такая же разметка, как и прежде (см. листинги 6.4 и 6.6). Данный код — альтернативный синтаксис. Обновите таблицу стилей, чтобы она совпадала со стилями, приведенными далее.

Свойство `grid-template-areas` позволяет создать визуальное представление сетки напрямую в коде CSS, используя короткий синтаксис в духе ASCII-графики. Это объявление выводит набор строк, заключенных в кавычки, при этом каждое такое значение представляет собой одну строку сетки с промежутком между колонками.

В этом примере первая строка полностью присваивается области сетки `title`. Вторая строка присваивается `nav`. Левая колонка двух оставшихся строк отнесена к области `main`, а блоки боковой панели — к областям `aside1` и `aside2`. Затем каждый элемент сетки помещается в именованные области с помощью свойства `grid-area`.

Листинг 6.7. Использование именованных участков сетки

```

.container {
  display: grid;
  grid-template-areas: "title title"
                       "nav  nav"
                       "main aside1"
                       "main aside2";
  grid-template-columns: 2fr 1fr;
  grid-template-rows: repeat(4, auto);
  grid-gap: 1.5em;
  max-width: 1080px;
  margin: 0 auto;
}

header {
  grid-area: title;
}

nav {
  grid-area: nav;
}

.main {
  grid-area: main;
}

.sidebar-top {
  grid-area: aside1;
}

.sidebar-bottom {
  grid-area: aside2;
}

```

Сопоставление каждой ячейки сетки с именованной областью сетки

Определение размеров полос сетки, как и прежде

Помещение каждого элемента сетки в именованную область сетки

ВНИМАНИЕ!

Каждая именованная область сетки должна формировать прямоугольник. Не получится создать более сложные формы, такие как *L* или *U*.

Вы также можете оставить ячейку пустой, задействуя точку в качестве имени. Например, следующий код определяет четыре области, окружающие пустую ячейку:

```

grid-template-areas: "top top right"
                    "left . right"
                    "left bottom bottom";

```

При создании сетки в зависимости от потребностей дизайна используйте наиболее удобный для вас синтаксис: пронумерованные линии, именованные линии или именованные области сетки. Последний способ, скорее всего, станет любимым для большого количества разработчиков. Он особенно эффективен, когда вы знаете, куда именно хотите поместить каждый элемент.

6.4. Явная и неявная сетка

В некоторых случаях вы не знаете, куда точно нужно поместить тот или иной элемент сетки. Возможно, вы работаете с большим количеством элементов и явная компоновка каждого из них — неподъемный труд. Или требуется неизвестное количество элементов, подгружаемых из базы данных. В этих случаях было бы разумнее определить сетку примерно, а затем позволить алгоритму размещения элементов сетки заполнить ее за вас.

Для этого вам потребуется положиться на *неявную сетку*. Используя свойства `grid-template-*` для определения полос, вы создаете *явную сетку*. Однако ее элементы по-прежнему могут располагаться вне явных полос, в этом случае будут автоматически сгенерированы неявные полосы, расширяющие сетку так, что она будет включать в себя и эти внешние элементы.

На рис. 6.11 показана сетка с единственной явно заданной полосой. При помещении элемента во вторую полосу (между линиями 2 и 3) добавляются дополнительные полосы, содержащие этот элемент.

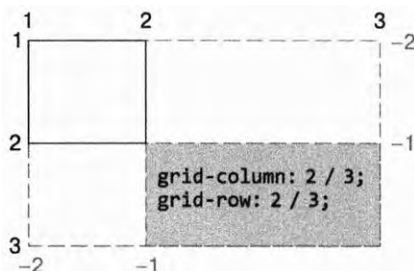


Рис. 6.11. Если элемент находится вне объявленных полос сетки, добавляются неявные полосы до тех пор, пока сетка не станет вмещать этот элемент

По умолчанию размер неявных полос сетки устанавливается как `auto`, а это значит, что они будут увеличиваться в размере по мере необходимости, чтобы уместить контент элемента сетки. Для указания других значений размерности всех неявных полос сетки могут использоваться свойства `grid-auto-columns` и `grid-auto-rows` (например, `grid-auto-columns: 1fr`).

ПРИМЕЧАНИЕ

Неявные полосы сетки не изменяют значения отрицательных номеров при ссылке на линии сетки. Отрицательная нумерация по-прежнему начинается с нижнего правого угла явной сетки.

Выполним разметку другой страницы, используя неявную сетку. Это будет портфолио с фотографиями (рис. 6.12). Определим полосы сетки для колонок, но строки будут неявными. Таким образом, страница не будет структурирована для конкретного количества элементов, ее можно будет адаптировать для любого их

числа. Каждый раз при необходимости переноса элементов на новую строку неявно будет добавляться еще одна строка.

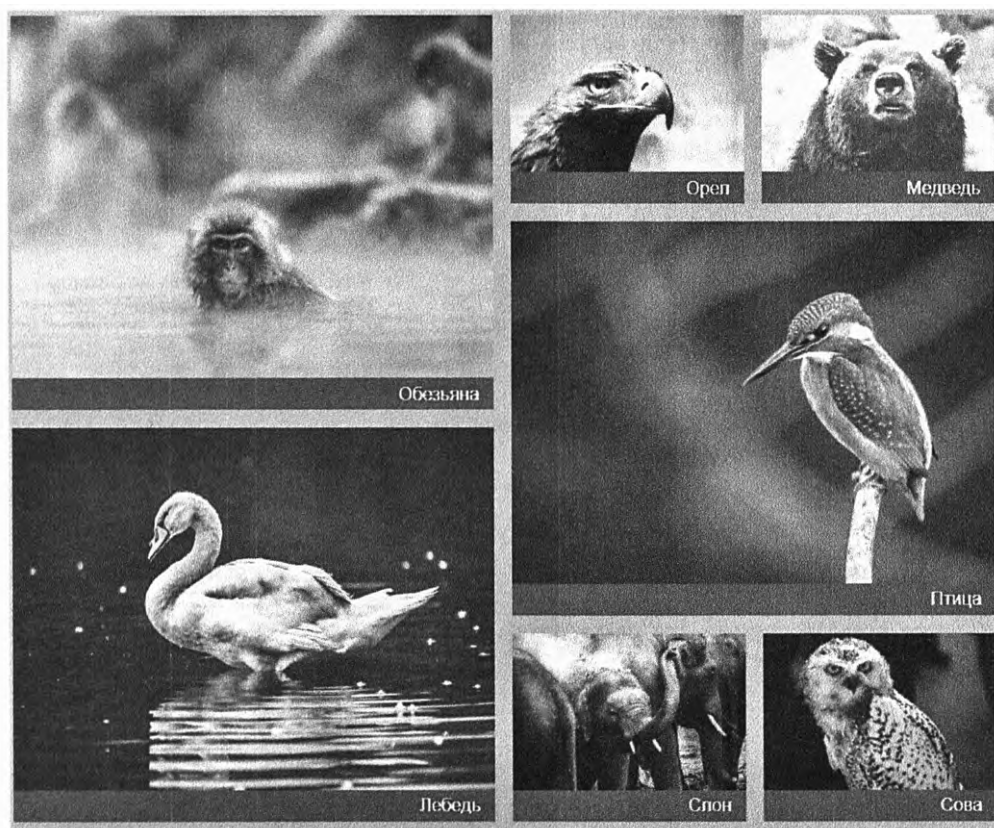


Рис. 6.12. Серия фотографий, выложенных в сетку с применением неявных строк

Эту забавную разметку было бы сложно создать с помощью плавающих элементов или flexbox-верстки. Она демонстрирует уникальные возможности CSS-сетки.

Чтобы выполнить разметку, потребуется новая страница. Создайте пустую страницу и новую таблицу стилей и свяжите их. Разметка страницы приведена в листинге 6.8. Добавьте ее на свою страницу.

Данная разметка — это элемент портфолио, который будет контейнером сетки, и набор фигур — элементов сетки. Каждая фигура содержит изображение и подпись к нему. Некоторым элементам я добавил класс `featured`, который увеличит размер изображений по сравнению с другими.

Я проведу вас по этому коду в несколько этапов. Сначала вы придадите форму полосам сетки и увидите изображения в базовом формате (рис. 6.13). После этого увеличите демонстрируемые изображения и внесете еще несколько деталей.

Листинг 6.8. Разметка для портфолио

```
<div class="portfolio">
  <figure class="featured">
    
    <figcaption>Обезьяна</figcaption>
  </figure>
  <figure>
    
    <figcaption>Орел</figcaption>
  </figure>
  <figure class="featured">
    
    <figcaption>Птица</figcaption>
  </figure>
  <figure>
    
    <figcaption>Медведь</figcaption>
  </figure>
  <figure class="featured">
    
    <figcaption>Лебедь</figcaption>
  </figure>
  <figure>
    
    <figcaption>Слон</figcaption>
  </figure>
  <figure>
    
    <figcaption>Сова</figcaption>
  </figure>
</div>
```

Каждый элемент figure станет элементом сетки

Заключение изображения и подписи к нему внутри элемента figure

Класс featured увеличит размер некоторых изображений

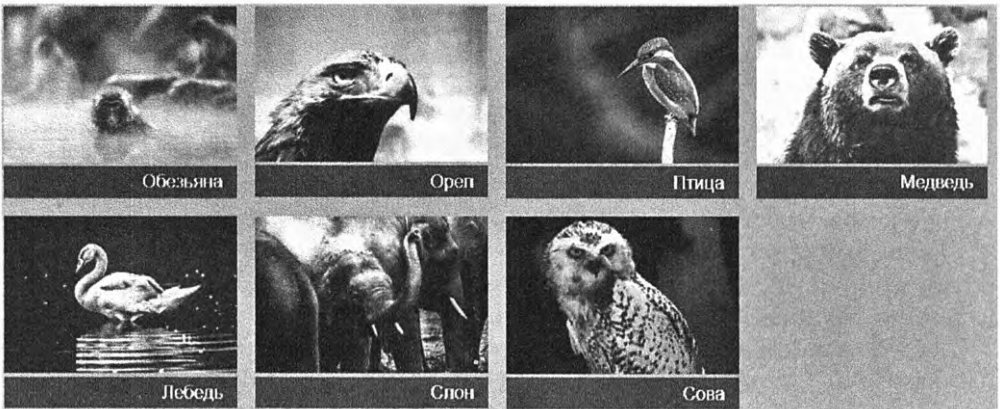


Рис. 6.13. Разметка изображений на базовой сетке

Стили для этой страницы приведены в листинге 6.9. В коде используется свойство `grid-auto-rows` для установки размера 1 fr для всех неявных строк сетки, таким образом, все они будут одинаковой высоты. Также впервые появляются две новые

концепции — `auto-fill` и функция `minmax()`, которые я совсем скоро объясню. Добавьте эти стили в свою таблицу стилей.

Листинг 6.9. Сетка с неявными строками

```
body {
  background-color: #709b90;
  font-family: Helvetica, Arial, sans-serif;
}

.portfolio {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
  grid-auto-rows: 1fr;
  grid-gap: 1em;
}

.portfolio > figure {
  margin: 0;
}

.portfolio img {
  max-width: 100%;
}

.portfolio figcaption {
  padding: 0.3em 0.8em;
  background-color: rgba(0, 0, 0, 0.5);
  color: #fff;
  text-align: right;
}
```

Установка минимальной ширины
колонок 200 пикселей
и автоматическое заполнение сетки

Установка высоты неявной
горизонтальной полосы
сетки размером 1 fr

Переопределение полей,
назначаемых браузером

Иногда вам не нужно будет фиксировать размер полосы сетки, но при этом следует ограничивать размер определенными минимальным и максимальным значениями. Здесь-то и вступает в игру функция `minmax()`, устанавливающая минимальный и максимальный размеры. (Если максимальный размер меньше минимального, он игнорируется.) Получив указание `minmax(200px, 1fr)`, браузер гарантирует, что ширина всех полос будет не менее 200 пикселей.

Ключевое слово `auto-fill` — это специальное значение, которое можно передать функции `repeat()`. С этой установкой браузер создаст столько колонок сетки, сколько помещается в доступном пространстве без нарушения установленных ограничений размера (значения функции `minmax()`).

Совместное использование `auto-fill` и `minmax(200px, 1fr)` означает, что в сетке будет столько колонок, сколько позволяет уместить доступное пространство, при этом каждая шириной не меньше 200 пикселей. Кроме того, поскольку ни одна из полос не может быть выше 1 fr или максимального значения, все они будут одинаковыми.

На рис. 6.13 область просмотра вмещает четыре колонки по 200 пикселей, поэтому добавляется именно это количество полос. Если бы область просмотра была шире, то в ней могло бы поместиться больше колонок, а при меньшем экране — меньше.

Обратите внимание на то, что использование ключевого слова `auto-fill` может привести к созданию нескольких пустых полос сетки, если нет достаточного количества

элементов для заполнения всех созданных полос. Если не хотите, чтобы создавались пустые полосы сетки, воспользуйтесь ключевым словом `auto-fit` вместо `auto-fill`. В результате непустые полосы расширятся и заполнят доступное пространство (см. сайт gridbyexample.com/examples/example37/, чтобы увидеть разницу на конкретных примерах).

Какое из ключевых слов, `auto-fill` или `auto-fit`, вы примените, зависит от того, хотите ли вы, чтобы получилась полоса ожидаемого размера или чтобы была заполнена определенная часть сетки. Сам я предпочитаю вариант `auto-fit`.

6.4.1. Внесем разнообразие

Теперь сделаем сетку визуально привлекательной, увеличив размер демонстрируемых изображений (в данном примере — фотографий безымянной птицы и лебедя). Каждый элемент сетки в настоящий момент занимает область сетки 1×1 . Увеличьте размер демонстрируемых изображений до заполнения области сетки 2×2 — с помощью класса `featured` заставьте их занять пространство двух полос сетки в обоих направлениях.

Однако тут возникает проблема. В зависимости от порядка следования элементов увеличение размера некоторых из них может привести к появлению промежутков в сетке (рис. 6.14). На этом рисунке фотография птицы — третий элемент в сетке. Так как этот снимок больше прочих, он не помещается в пространство справа от второго изображения — фотографии орла и сдвигается вниз, на новую полосу.

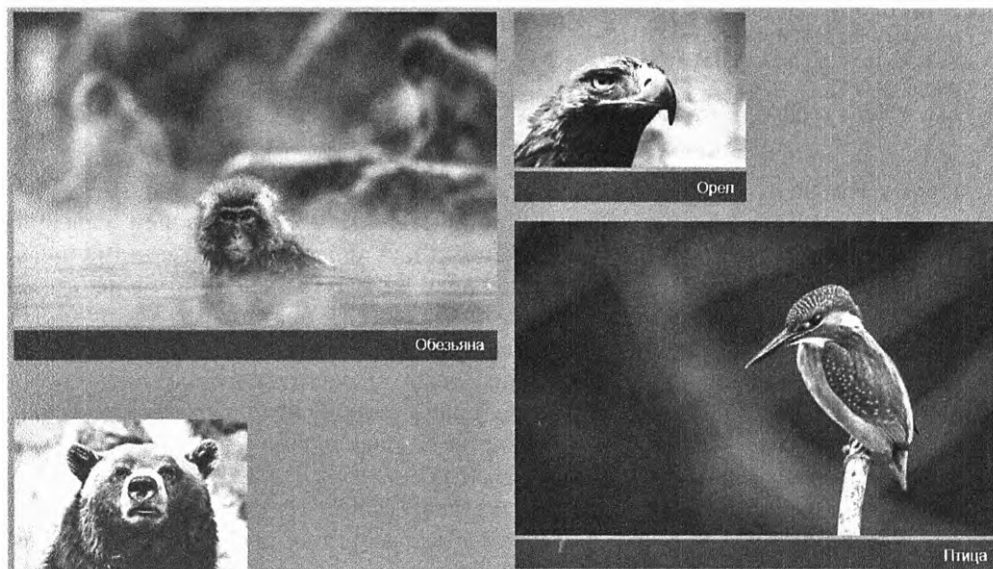


Рис. 6.14. Из-за увеличения размеров некоторых элементов в сетке появились промежутки — пространство, где увеличенные элементы не смогли поместиться

Если вы не позиционируете элементы на сетке вручную, это делает за вас алгоритм размещения элементов. По умолчанию он располагает их колонка за колонкой, строка за строкой в соответствии с порядком элементов в разметке. Когда элемент

не помещается в одной строке (то есть его ширина больше суммарной ширины свободных полос сетки), алгоритм в поисках достаточного количества места переносит его на новую строку. В данном случае фотография птицы была перенесена на вторую строку, под фотографию орла.

Модуль CSS3 Grid Layout предоставляет еще одно свойство — **grid-auto-flow**, которое может использоваться для управления поведением алгоритма размещения. Изначальное значение этого свойства — **row** — ведет себя так, как описано ранее. После присвоения значения **column** данное свойство сначала помещает элемент в колонку и перемещает его на следующую строку только тогда, когда свободного места в колонке недостаточно.

Можете добавить также ключевое слово **dense** (например, **grid-auto-flow: column dense**). В результате алгоритм попытается заполнить промежутки в сетке, даже если для этого потребуются изменить порядок отображения некоторых элементов. Если вы задействуете это свойство на странице, то более мелкие элементы сетки «подстрахуют» и заполнят промежутки, созданные более крупными. Результат показан на рис. 6.15.

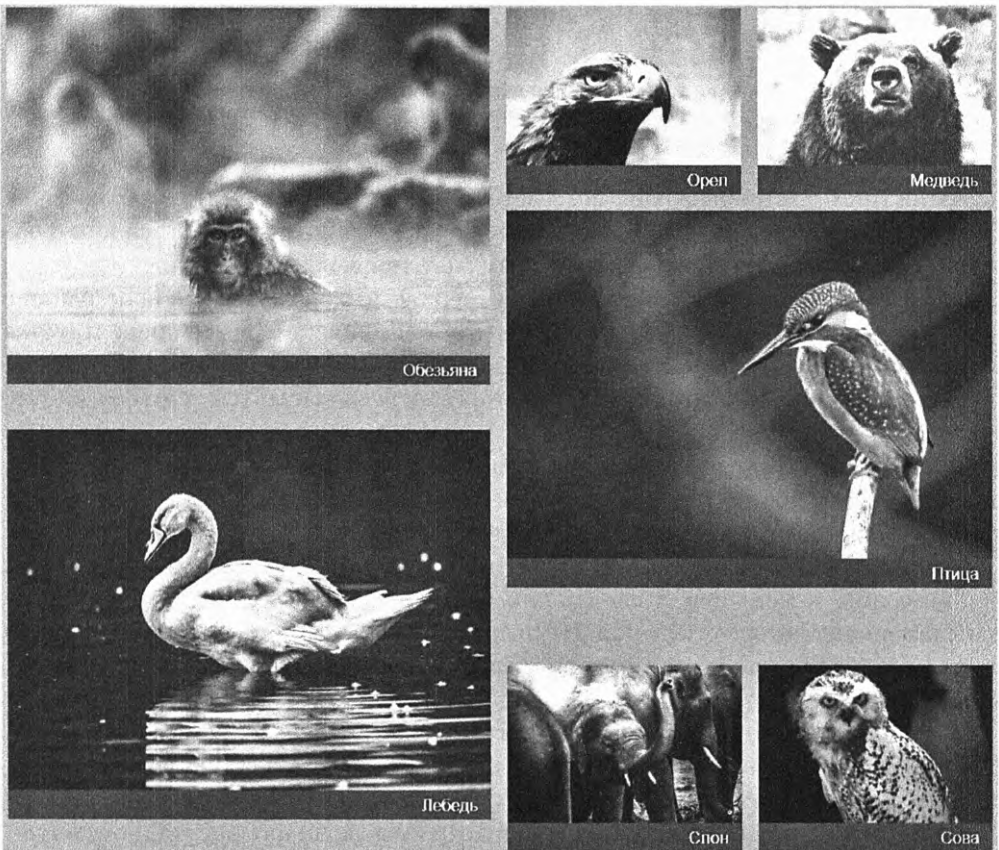


Рис. 6.15. Применение ключевого слова **dense** свойства **grid-auto-flow** позволяет более мелким элементам заполнить промежутки в сетке

Благодаря опции `dense` мелкие элементы заполняют промежутки, оставленные более крупными элементами сетки. Порядок в файле исходного кода по-прежнему таков: обезьяна, орел, птица, затем медведь, но фотография медведя перемещается в положение перед фотографией птицы, заполняя пустоту.

Добавьте в свою таблицу стилей следующий код (листинг 6.10). Он увеличивает размер демонстрируемых изображений так, что они заполняют две полосы в обоих направлениях. В коде также используется ключевое слово `dense` свойства `grid-auto-flow`.

Листинг 6.10. Увеличение демонстрируемых изображений

```
.portfolio {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
  grid-auto-rows: 1fr;
  grid-gap: 1em;
  grid-auto-flow: dense;
}

.portfolio .featured {
  grid-row: span 2;
  grid-column: span 2;
}
```

Задействование алгоритма
плотного размещения элементов

Увеличение демонстрируемых изображений так,
чтобы они занимали две полосы сетки в обоих направлениях

В листинге стоит объявление `grid-auto-flow: dense`, эквивалентное объявлению `grid-auto-flow: row dense`. (В первой части значения подразумевается ключевое слово `row`, так как оно является начальным значением.) Затем код нацеливается на два элемента `featured` и увеличивает их размер так, что они занимают пространство двух полос в обоих направлениях. Обратите внимание: в примере используется только ключевое слово `span`, при этом не задается размещение элементов сетки в конкретных полосах. Это позволяет алгоритму располагать элементы сетки там, куда они помещаются.

В зависимости от размера области просмотра то, что вы видите на экране, не обязательно точно совпадает с рис. 6.12. Это происходит потому, что использовано свойство `auto-fill` для определения количества вертикальных полос сетки. Экран с большим разрешением позволит создать большее количество полос, а с меньшим — меньшее. Я сделал снимок экрана шириной 1000 пикселей, на котором появляются четыре полосы сетки. Изменяйте ширину окна своего браузера, чтобы увидеть автоматический ответ сетки, заполняющей доступное пространство.

Используйте плотный автоматический поток (`grid-auto-flow: dense`) осторожно, так как элементы могут отображаться не в том порядке, в каком указаны в HTML-файле. Это может несколько сбить с толку пользователей, действующих для навигации клавиатуру (клавишу `Tab`), или программы экранного доступа — они применяют для навигации порядок из файла исходного кода, а не порядок фактического отображения на экране.

Подсетки

Одно из ограничений CSS-сетки — требование конкретной структуры DOM, а именно: все элементы сетки должны быть прямыми потомками контейнера сетки. Таким образом, становится невозможно выравнивать глубоко вложенные элементы сетки.

Вы можете поместить в сетке элемент `display: grid` для создания внутренней сетки. Однако элементы внутренней сетки не обязательно будут выравниваться с элементами внешней. Кроме того, размер элементов одной сетки не будет влиять на размер полос другой.

В дальнейшем будет реализовано решение в виде *подсеток*. После применения свойства элемента `display: subgrid` этот элемент становится контейнером внутренней сетки, причем ее полосы будут выравниваться с полосами внешней сетки. К сожалению, данная функция не была реализована ни в одном браузере до выпуска спецификации уровня 2.

Появления этой функции ожидают многие. Следите за новостями.

6.4.2. Подгонка элементов для заполнения полосы сетки

Сейчас разметка уже довольно сложна. Вам не пришлось долго работать, чтобы точно позиционировать элементы, вы позволили браузеру сделать это за вас.

Осталась одна небольшая проблема: более крупные изображения не целиком заполняют ячейки сетки, под ними остаются небольшие промежутки. В идеале верхние и нижние края изображений в одной полосе должны находиться на одном уровне. Как показано на рис. 6.16, верхние края изображений выровнены, а нижние — нет.

Давайте исправим этот недостаток. Как вы, наверное, помните, каждый элемент сетки — это элемент `figure`, содержащий два дочерних элемента — изображение и подпись:

```
<figure class="featured">
  
  <figcaption>Обезьяна</figcaption>
</figure>
```

По умолчанию элементы сетки растягиваются для заполнения всей ее области, но их дочерние элементы не увеличиваются, поэтому образуется неиспользуемая высота области сетки. Простой способ исправить это — воспользоваться *flexbox-версткой*. В листинге 6.11 вы сделаете каждый элемент `figure` *flex-контейнером* со значением направления `column`, чтобы элементы устанавливались вертикально один на другой. Затем сможете применить к изображению свойство `flex-grow`, которое растянет его до заполнения свободного пространства.

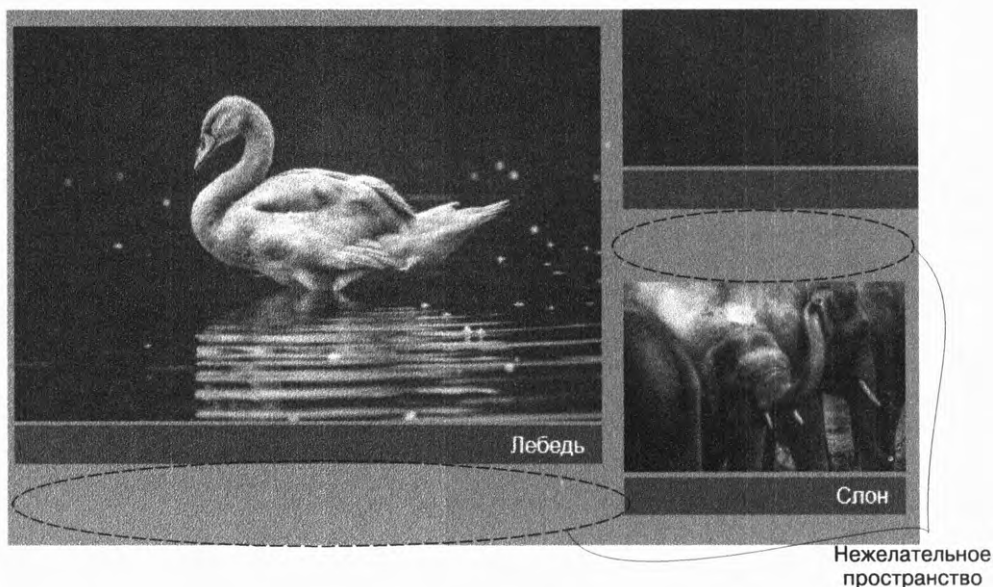


Рис. 6.16. Изображения не заполняют ячейки сетки, оставляя нежелательные промежутки

Однако растягивание изображения способно вызвать проблему. Это изменит соотношение ширины и высоты, искажая изображения. К счастью, в CSS есть специальное свойство **object-fit** для управления этим поведением. По умолчанию свойству **object-fit** элемента **img** присвоено значение **fill**, означающее, что размер изображения будет изменен для заполнения элемента **img**. Для того чтобы скорректировать такое поведение, можно установить другие значения.

Например, свойство **object-fit** принимает также значения **cover** и **contain** (рис. 6.17). Они дают браузеру задание изменить размер изображения внутри визуализированного блока без искажения отношения сторон.

- ❑ Чтобы расширить изображение до заполнения блока (в результате его часть будет обрезана), используйте значение **cover**.
- ❑ Для изменения размера изображения так, чтобы оно полностью помещалось в блоке (что приведет к появлению пустого пространства внутри блока), возьмите значение **contain**.

Здесь есть важное отличие: имеются блок, определяемый высотой и шириной элемента **img**, и обработанное изображение. По умолчанию размеры этих элементов совпадают. Свойство **object-fit** позволяет управлять размером обработанного изображения внутри блока, но размер самого блока остается неизменным.

Поскольку изображения растягиваются с помощью свойства **flex-grow**, следует воспользоваться также свойством **object-fit**: **cover**, чтобы предотвратить искажение изображения. Будут отсечены небольшие участки по краям — таков компромисс, на который придется пойти.

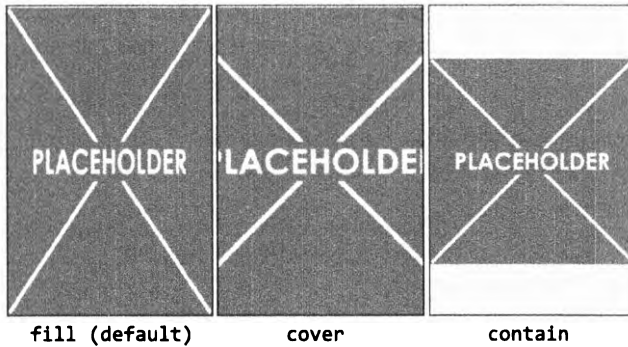


Рис. 6.17. Применение свойства `object-fit` для управления тем, как изображение будет визуализировано внутри соответствующего ему блока

Конечный результат показан на рис. 6.18. Для более детального рассмотрения этого свойства обратитесь к сайту css-tricks.com/on-object-fit-and-object-position/.

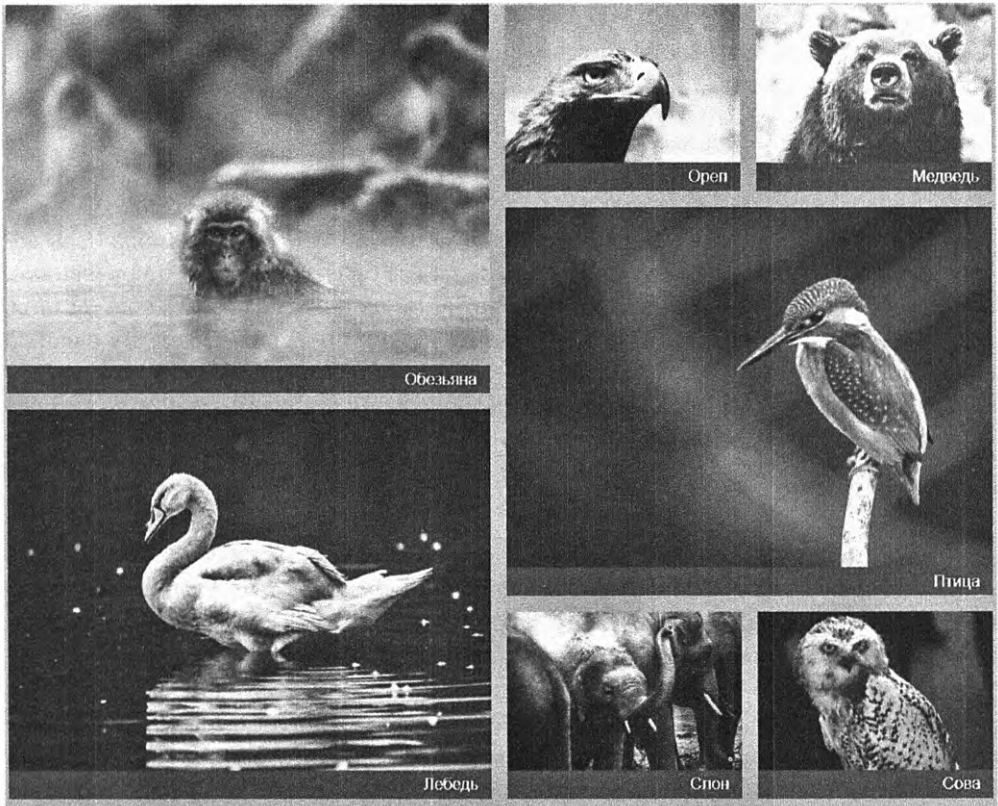


Рис. 6.18. Теперь все изображения строго выровнены и заполняют свои контейнеры

Верхние и нижние края всех изображений и их подписей выровнены в каждой полосе сетки. Код для этого приведен в листинге 6.11. Добавьте его в свою таблицу стилей.

Листинг 6.11. Использование вертикального flex-контейнера для растягивания изображения и заполнения области сетки

```
.portfolio > figure {
  display: flex;
  flex-direction: column;
  margin: 0;
}

.portfolio img {
  flex: 1;
  object-fit: cover;
  max-width: 100%;
}
```

Превращение всех элементов сетки
в вертикальный flex-контейнер

Применение свойства flex-grow,
чтобы изображение заполнило
все доступное пространство flex-контейнера

Позволяет изображению
заполнить блок без растягивания
(вместо этого обрезая его по краям)

Это завершает дизайн портфолио с фотографиями. Все выровнено в четкой сетке, браузер принимает за вас решение о количестве вертикальных полос и размере каждой из них. Использование плотного автоматического потока позволяет браузеру заполнить промежутки.

6.5. Запросы функций

Теперь, ознакомившись с основами верстки методом CSS-сетки, возможно, вы задаетесь вопросом: неужели нужно ждать, пока все браузеры станут поддерживать сетку, чтобы ею пользоваться? Нет. При желании начните делать это уже сегодня. Вам необходимо поразмыслить, как именно нужно поступать браузеру с вашей разметкой, если он не поддерживает CSS-сетку, и обеспечить эти запасные стили.

В CSS недавно было добавлено средство, называемое *запросом функций*, которое помогает решить эту задачу. Оно выглядит следующим образом:

```
@supports (display: grid) {
  ...
}
```

После правила `@supports` идет объявление в скобках. Если браузер понимает его (в данном случае поддерживает сетку), то применяет любые наборы правил, указанные между фигурными скобками. В противном случае — не применяет.

Это означает, что вы можете предоставить один набор стилей, используя старые технологии разметки, такие как плавающие элементы. Стили не обязательно должны быть идеальными (придется пойти на некоторые компромиссы), но сайт окажется работоспособным. Затем с помощью запроса функций можно создать полнофункциональную разметку уже с использованием CSS-сетки.

Воспользуемся этой возможностью для нашего портфолио. Для старых браузеров выполните более простую разметку с помощью строчно-блочных элементов, затем поместите весь код, связанный с CSS-сеткой, в блок запроса функций. Браузеры, не поддерживающие CSS-сетку, обработают страницу, как показано на рис. 6.19.



Рис. 6.19. Запасная разметка для браузеров, не поддерживающих сетку

При создании этой разметки мы пошли на пару компромиссов: демонстрируемые изображения не увеличатся в размере, а колонки будут иметь фиксированную ширину 300 пикселей вместо того, чтобы расширяться до заполнения доступной ширины области просмотра. Поскольку элементы `figure` отображаются методом `inline-block`, они будут переноситься на новую строку в нормальном режиме. Это позволит вместить в область просмотра большее количество изображений, если для них достаточно места.

Код для этого, включая запрос функций, приведен в листинге 6.12. Обновите свою таблицу стилей соответствующим образом.

Запасной и прочие базовые стили, такие как цвета, находятся вне блока запроса функций, поэтому они применяются всегда. Если вы откроете страницу в браузере, не поддерживающем CSS-сетку, то увидите запасную разметку (см. рис. 6.19). Все стили, связанные с CSS-сеткой, помещены в блок запроса функций, поэтому они будут задействованы только в том случае, если браузер поддерживает сетку.

Правило `@supports` можно задействовать для запроса любых функций CSS. Воспользуйтесь выражением `@supports (display: flex)` для запроса поддержки flexbox-верстки или `@supports (mix-blend-mode: overlay)` для запроса поддержки режима смешивания (см. главу 11 для получения дополнительной информации о режимах наложения).

Листинг 6.12. Запрос функций для прогрессивного улучшения

```
.portfolio > figure {
  display: inline-block;
  max-width: 300px;
  margin: 0;
}

.portfolio img {
  max-width: 100%;
  object-fit: cover;
}

.portfolio figcaption {
  padding: 0.3em 0.8em;
  background-color: rgba(0, 0, 0, 0.5);
  color: #fff;
  text-align: right;
}

@supports (display: grid) {
  .portfolio {
    display: grid;
    grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
    grid-auto-rows: 1fr;
    grid-gap: 1em;
    grid-auto-flow: dense;
  }

  .portfolio > figure {
    display: flex;
    flex-direction: column;
    max-width: initial;
  }

  .portfolio img {
    flex: 1;
  }

  .portfolio .featured {
```

Использование строчно-блочной компоновки
в качестве запасной разметки

Запрос функций
для поддержки CSS-сетки

Помещение всех стилей CSS-сетки
внутри блока запроса функций

Переопределение
запасных стилей

```
    grid-row: span 2;  
    grid-column: span 2;  
  }  
}
```

ВНИМАНИЕ!

Internet Explorer не поддерживает правило `@supports`. Этот браузер игнорирует все правила внутри блока запроса функций, даже если фактически они поддерживаются. Обычно это не проблема, так как вам нужно, чтобы устаревшие браузеры обрабатывали запасную разметку.

Запросы функций могут быть составлены и несколькими другими способами:

- ❑ `@supports not (<объявление>)` — применять правила в блоке запроса функций, только если запрашиваемое объявление не поддерживается;
- ❑ `@supports not (<объявление>) or (<объявление>)` — применять правила, если поддерживается хотя бы одно из запрашиваемых объявлений;
- ❑ `@supports not (<объявление>) and (<объявление>)` — применять правила, если поддерживаются оба запрашиваемых объявления.

Приведенные варианты можно комбинировать для запроса более сложных ситуаций. Ключевое слово `or` используется для запроса поддержки с применением свойств с префиксами:

```
@supports (display: grid) or (display: -ms-grid)
```

Это объявление будет нацелено на браузеры, которые поддерживают версию свойства со снятым префиксом, а также более ранние версии MS Edge, требующие наличия префикса `-ms-`. Предупреждаю: частичная поддержка сетки в более ранних версиях Edge не такая стабильная, как в современных браузерах. Велика вероятность того, что все затраченные усилия не заставят работать версию запроса `@supports` с префиксами и вы в конце концов все равно от нее откажетесь. Таким образом, более ранние версии Edge будут обрабатывать запасную разметку.

6.6. Выравнивание

Модуль CSS3 Grid Layout задействует свойства выравнивания, использующиеся в технологии flexbox, а также несколько новых. Я уже рассказывал о большинстве этих свойств в предыдущей главе, но давайте рассмотрим, как они применяются в CSS-сетке. Если вам нужен больший контроль за различными аспектами CSS-сетки, эти свойства могут стать полезными.

В CSS представлены три свойства распределения (выравнивания) по ширине: `justify-content`, `justify-items` и `justify-self`. Они управляют горизонтальным позиционированием. Я запоминаю их, ассоциируя с выравниванием текста по ширине в текстовом процессоре, которое распределяет текст по горизонтали.

Также имеется три свойства выравнивания: `align-content`, `align-items` и `align-self`. Они управляют позиционированием по вертикали. Я запоминаю их, ассоциируя со свойством табличной разметки `vertical-align`. Все эти свойства проиллюстрированы на рис. 6.20.

Можете использовать свойства `justify-content` и `align-content` для выполнения горизонтального и вертикального позиционирования полос сетки внутри ее контейнера. Это необходимо, если общий размер сетки меньше размера контейнера сетки. Рассмотрим следующий код:

```
.grid {  
  display: grid;  
  height: 1200px;  
  grid-template-rows: repeat(4, 200px);  
}
```

В нем явно устанавливается высота контейнера сетки 1200 пикселей, но при этом определяются только 800 пикселей для горизонтальных полос. Свойство `align-content` устанавливает, каким образом будут распределены оставшиеся 400 пикселей пространства. Поддерживаются следующие значения:

- ❑ `start` — помещает полосы сетки в верхний левый угол контейнера сетки (для flex-блока используйте свойство `flex-start`);
- ❑ `end` — помещает полосы сетки в нижний правый угол контейнера сетки (для flex-блока задействуйте свойство `flex-end`);
- ❑ `center` — помещает полосы сетки в центр контейнера сетки;
- ❑ `stretch` — изменяет размер полос до заполнения размера контейнера сетки;
- ❑ `space-between` — поровну распределяет свободное место между всеми полосами сетки (эффективно переопределяя любое значение `grid-gap`);
- ❑ `space-around` — распределяет свободное место между всеми полосами сетки так, что со всех сторон оказываются промежутки половинного размера;
- ❑ `space-evenly` — распределяет свободное место между всеми полосами сетки так, что размеры промежутков со всех сторон равны (не поддерживается flex-блоками).

Чтобы увидеть детальные примеры свойств распределения/выравнивания, посетите сайт gridbyexample.com. Это отличный ресурс, чья богатая коллекция примеров использования CSS-сетки собрана Рэйчел Эндрю — разработчиком и членом консорциума W3C.

CSS-сетка — очень обширная тема, и я познакомил вас только с наиболее важными концепциями, которые следует знать. Я бы порекомендовал продолжить экспериментировать с сетками. Существует гораздо большее количество вариантов смещения и сопоставления свойств, чем мы разобрали в этой главе, поэтому ставьте перед собой задачи пробовать что-то новое. Повстречавшись с интересной разметкой страницы в Интернете, попробуйте создать ее с помощью CSS-сетки.

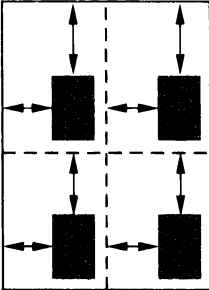
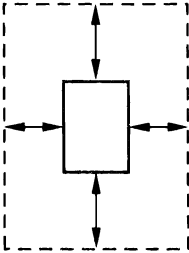
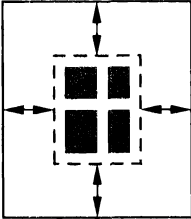
Свойства	Применимо к	Выравнивание
<code>justify-items</code> <code>align-items</code>	Контейнер сетки	<div>Элементы внутри областей сетки</div> 
<code>justify-self</code> <code>align-self</code>	Элемент сетки	<div>Элемент внутри области сетки</div> 
<code>justify-content</code> <code>align-content</code>	Контейнер сетки	<div>Полосы сетки внутри контейнера</div> 

Рис. 6.20. Свойства выравнивания для сетки

Итоги главы

- ❑ CSS-сетка — отличный инструмент для создания высокоуровневой разметки веб-страницы и не только.
- ❑ Для создания завершенной системы разметки можно использовать CSS-сетку в сочетании с flexbox-версткой.
- ❑ Применяйте тот синтаксис (пронумерованные линии сетки, именованные линии сетки или именованные области сетки), который интуитивно более понятен для вас в каждой конкретной ситуации.
- ❑ Используйте свойства `auto-fill`/`auto-fit` и неявные сетки для создания разметок с большим или неизвестным числом элементов.
- ❑ Применяйте запросы функций для прогрессивного усовершенствования разметки.

7

Контексты позиционирования и наложения

В этой главе

- Варианты позиционирования элементов: фиксированное, относительное и абсолютное.
- Создание модальных окон и раскрывающихся меню.
- Треугольники средствами CSS.
- Контексты наложения и z-index.
- Новый тип позиционирования — липкое.

Мы уже рассмотрели несколько способов управления макетом страницы, от табличного отображения до flexbox-верстки и плавающих элементов. В этой главе разберем один важный метод — свойство `position`, которое используется для создания раскрывающихся меню, модальных окон и других эффектов современных веб-приложений.

Позиционирование может стать сложным этапом в процессе разработки. Это область, в которой многие разработчики имеют лишь поверхностные знания. Не имея полного понимания позиционирования и последствий его применения, легко попасть в просак.

Взглянув на различные типы позиционирования, мы обязательно поймем, как они работают. Затем вы узнаете о контексте наложения — своего рода скрытом побочном эффекте позиционирования. Поняв, что это такое, вы застрахуетесь от неприятностей, если когда-нибудь заблудитесь при верстке макета страницы. У вас будут инструменты, необходимые для возвращения на верный путь.

Начальным значением свойства `position` является `static`. Все, что мы делали в предыдущих главах, относится к статическому позиционированию. Когда вы заметите значение этого свойства на какое-то иное, элемент станет *позиционированным*. Таким образом, элемент со статическим позиционированием, то есть со значением `static` свойства `position`, считается *непозиционированным*.

Методы верстки, рассмотренные в предыдущих главах, применяют разные инструменты для управления потоком документа. Позиционирование же полностью извлекает элементы из потока документа. Оно позволяет размещать их где угодно на экране, например поверх или позади друг друга с перекрытием.

7.1. Фиксированное позиционирование

Фиксированное позиционирование распространено не так широко, как некоторые другие типы позиционирования, но, возможно, понять его проще всего, поэтому с него и начнем. Применение свойства `position: fixed` к элементу позволяет произвольно позиционировать его в области просмотра. Это делается с помощью четырех свойств, составляющих комплект: `top`, `right`, `bottom` и `left`. Значения, которые вы им присваиваете, указывают, как далеко фиксированный элемент должен находиться от каждой из сторон области просмотра браузера. Например, свойство `top: 3em` означает, что верхний край фиксированного элемента будет на 3 em ниже верхнего края области просмотра.

Установив эти четыре значения, вы также неявно задаете ширину и высоту элемента. Например, указав свойства `left: 2em; right: 2em`, вы сделаете так, что левый край элемента будет отступать на 2 em от левого края области просмотра, а правый — на 2 em от правого края, таким образом, элемент будет на 4 em меньше общей ширины области просмотра. То же самое со свойствами `top` и `bottom` и высотой области просмотра.

7.1.1. Создание модального окна с фиксированным позиционированием

Применим эти свойства для создания модального окна (рис. 7.1). Оно появится поверх контента страницы, перекрывая его до закрытия окна.

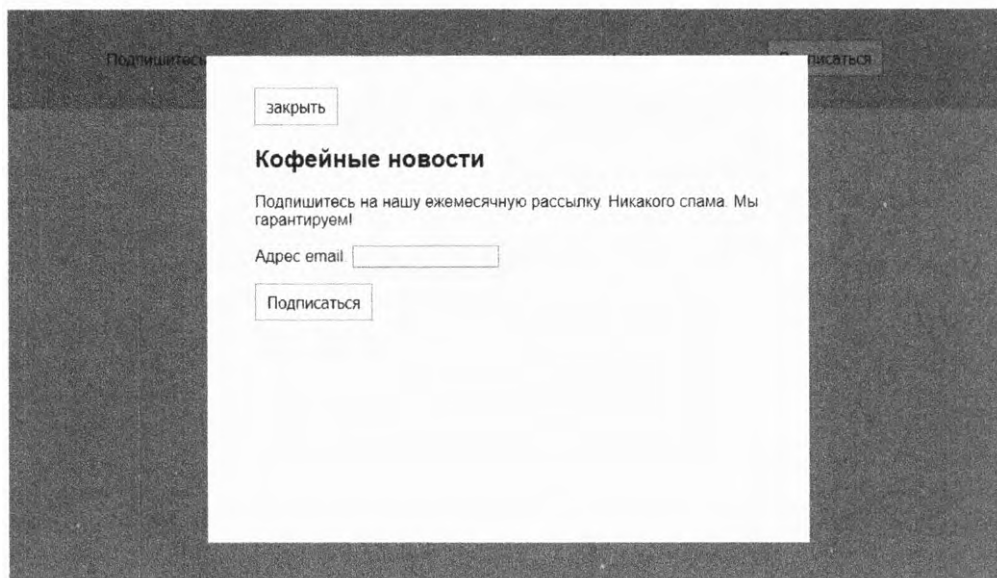


Рис. 7.1. Блок с модальным окном

Как правило, вы будете использовать модальное окно, чтобы пользователь что-либо прочитал или ввел какие-то данные перед тем, как продолжить работу. Например, наше окно содержит форму, в которой пользователь может подписаться на рассылку новостей. Сначала вы спрячете окно с помощью свойства `display: none`, а затем добавите небольшой JavaScript-сценарий, чтобы изменить значение свойства на `block` и открыть модальное окно.

Создайте новую страницу и добавьте приведенный в листинге 7.1 код в тело страницы (элемент `body`). Контент помещен внутрь двух элементов, также добавлен элемент `script` с JavaScript-кодом для обеспечения основной функциональности.

Листинг 7.1. Создание блока с модальным окном

```
<header class="top-banner">
  <div class="top-banner-inner">
    <p>Подпишитесь на нашу ежемесячную рассылку новостей
      из мира кофе. Нажмите кнопку:
      <button id="open">Подписаться</button>
    </p>
  </div>
</header>
<div class="modal" id="modal">
  <div class="modal-backdrop"></div>
  <div class="modal-body">
    <button class="modal-close" id="close">закрыть</button>
    <h2>Кофейные новости</h2>
    <p>Подпишитесь на нашу ежемесячную рассылку. Никакого спама.
      Мы гарантируем!</p>
    <form>
      <p>
        <label for="email">Адрес email:</label>
        <input type="text" name="email"/>
      </p>
      <p><button type="submit">Подписаться</button></p>
    </form>
  </div>
</div>

<script type="text/javascript">
var button = document.getElementById('open');
var close = document.getElementById('close');
var modal = document.getElementById('modal');

button.addEventListener('click', function(event) {
  event.preventDefault();
  modal.style.display = 'block';
});

close.addEventListener('click', function(event) {
  event.preventDefault();
  modal.style.display = 'none';
});
</script>
```

Кнопка открытия модального окна

Контейнер для модального окна

«Задник», скрывающий контент за модальным окном

Содержание модального окна

Открывает модальное окно, когда пользователь нажимает кнопку «Подписаться»

Закрывает модальное окно, когда пользователь нажимает кнопку «Закрыть»

Первый элемент в листинге — это баннер в верхней части страницы. На нем находится кнопка, запускающая модальное окно. Второй элемент — само модальное окно. Оно включает в себя пустой класс `modal-backdrop`, который использован, чтобы загородить остальную часть страницы, привлекая внимание пользователя к содержимому модального окна. Контент окна находится внутри элемента `modal-body`.

В листинге 7.2 приведен CSS-код для его реализации. Добавьте его в таблицу стилей. Он включает в себя базовое форматирование для верхнего баннера, а также стили для модального окна.

Листинг 7.2. Добавление стилей для модального окна

```
body {
  font-family: Helvetica, Arial, sans-serif;
  min-height: 200vh;
  margin: 0;
}

button {
  padding: .5em .7em;
  border: 1px solid #8d8d8d;
  background-color: white;
  font-size: 1em;
}

.top-banner {
  padding: 1em 0;
  background-color: #ffd698;
}

.top-banner-inner {
  width: 80%;
  max-width: 1000px;
  margin: 0 auto;
}

.modal {
  display: none;
}

.modal-backdrop {
  position: fixed;
  top: 0;
  right: 0;
  bottom: 0;
  left: 0;
  background-color: rgba(0, 0, 0, 0.5);
}

.modal-body {
  position: fixed;
  top: 3em;
  bottom: 3em;
  right: 20%;
  left: 20%;
}
```

Добавляет вертикальную полосу прокрутки по высоте страницы (только для демонстрационных целей)

Скрывает модальное окно по умолчанию; JavaScript-сценарий устанавливает свойство `display: block` при открытии модального окна

Используется полупрозрачный фон, затемняющий остальную часть страницы, пока модальное окно открыто

Позиционирует основную часть модального окна

```
padding: 2em 3em;  
background-color: white;  
overflow: auto;  
}
```

Позволяет прокручивать тело
модального окна, если необходимо

```
.modal-close {  
  cursor: pointer;  
}
```

В этом CSS-коде дважды используется фиксированное позиционирование. Сначала селектор `modal-backdrop` присваивает каждой из четырех сторон значение `0`. Таким образом, задник занимает все пространство области просмотра. Ему задан цвет фона `rgba(0, 0, 0, 0.5)`. Эта запись определяет значения красного, зеленого и синего цветов как `0`, что соответствует черному цвету. Четвертое значение — это альфа-канал, который определяет прозрачность цвета: значение `0` — полностью прозрачный, `1` — полностью непрозрачный. Значением `0.5` задана полупрозрачность. Благодаря этому будет затенен весь контент страницы позади элемента.

Вторая позиция, где применено фиксированное позиционирование, находится в селекторе `modal-body`. Вы позиционируете каждую из четырех сторон тела модального окна в области просмотра: по `3em` от верхнего и нижнего краев и по `20 %` от левого и правого. А также устанавливаете белый цвет фона. Так для модального окна создается белый блок, центрированный на экране. Можете прокручивать страницу, как хотите, но задник и тело модального окна останутся на своих местах.

Загрузив страницу, вы увидите в верхней части экрана бледно-желтый баннер с кнопкой. Нажмите ее, чтобы открыть позиционированное модальное окно. Из-за фиксированного позиционирования оно остается на месте, даже если вы прокручиваете страницу (большое значение свойства `min-height` элемента тела допускает прокрутку, чтобы проиллюстрировать это поведение).

Чтобы закрыть модальное окно, нажмите кнопку **Закрыть** в его верхней части. Сейчас она не на своем месте, но позже мы вернемся к ней и переместим.

7.1.2. Управление размером позиционированных элементов

При позиционировании элемента не нужно задавать значения для всех четырех сторон. Можно указать только необходимые стороны, а затем использовать свойства `width` и/или `height`, чтобы определить его размер.

Можете также разрешить элементу иметь естественный размер. Рассмотрим эти объявления:

```
position: fixed;  
top: 1em;  
right: 1em;  
width: 20%
```

С помощью этого кода элемент размещается на расстоянии `1em` от верхней и правой сторон области просмотра, его ширина составляет `20 %` от ширины области

просмотра. Так как свойства `bottom` и `height` элемента не указаны, высота элемента определяется его контентом. Этот прием можно использовать для прикрепления навигационной панели к экрану. Она остается на месте, даже если пользователь прокручивает контент на странице.

Поскольку фиксированный элемент удаляется из потока документов, он перестает влиять на положение других элементов на странице. Они будут двигаться в потоке документа так, как будто фиксированного элемента там нет, а это значит, что он станет перекрывать их. Обычно это допустимо, ведь вы хотите, чтобы фиксированный элемент располагался поверх прочего контента и был центрирован, пока пользователь не закроет его.

Если речь идет о постоянных объектах, например о навигационных панелях, следует позаботиться о том, чтобы они не перекрывали другой контент. Обычно проще всего сделать это, снабдив контент полями. Например, разместите весь контент в контейнере со свойством `right-margin: 20%`. Это поле будет двигаться вслед за фиксированным элементом, и контент не окажется перекрытым.

7.2. Абсолютное позиционирование

Фиксированное позиционирование, как вы только что видели, позволяет определять положение элемента в области просмотра, которая называется его *содержащим блоком*. Объявление `left: 2em`, к примеру, располагает левый край позиционированного элемента на расстоянии 2 em от левого края содержащего блока.

Абсолютное позиционирование работает таким же образом, за исключением того, что там используется другой содержащий блок. Его позиция основана не на области просмотра, а на ближайшем позиционированном родительском элементе. Как и в ходе работы с фиксированным элементом, свойства `top`, `right`, `bottom` и `left` определяют положение сторон элемента внутри его содержащего блока.

7.2.1. Абсолютное позиционирование кнопки Закрыть

Давайте попрактикуемся — переместим кнопку **Закрыть** в верхний правый угол модального окна. После этого оно будет выглядеть как на рис. 7.2.

Чтобы сделать это, вы абсолютно позиционируете кнопку **Закрыть**. Ее родительским элементом является `modal-body`, который позиционирован фиксированно, поэтому становится содержащим блоком для кнопки. Измените стили кнопки, чтобы они соответствовали листингу 7.3.

В этом листинге кнопка расположена на расстоянии 0,3 em от верхнего края и на расстоянии 0,3 em от правого края элемента `modal-body`. Обычно содержащий блок (containing block) является предком элемента. В тех случаях, когда родительский элемент не позиционирован, браузер будет просматривать иерархию DOM до тех пор, пока не найдет позиционированный элемент, который затем станет использоваться как содержащий блок.

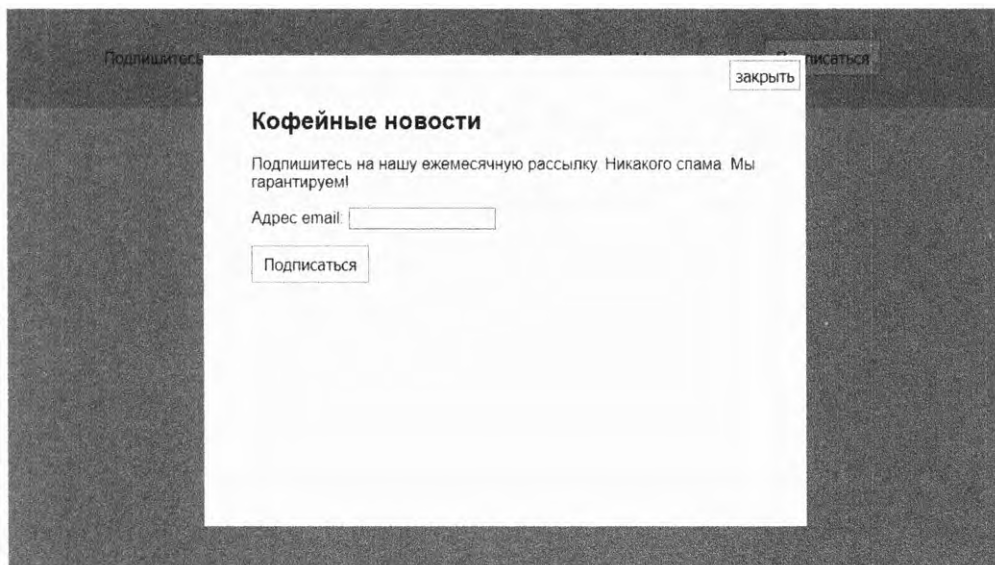


Рис. 7.2. Кнопка Закр^ыть расположена в верхнем правом углу модального окна

Листинг 7.3. Абсолютное позиционирование кнопки Закр^ыть

```
.modal-close {  
  position: absolute;  
  top: 0.3em;  
  right: 0.3em;  
  padding: 0.3em;  
  cursor: pointer;  
}
```

ПРИМЕЧАНИЕ

Если ни одному из предков элемента позиционирование не задано, то абсолютно позиционированный элемент будет размещаться на основе так называемого начального содержащего блока. Это область, по размерам равная области просмотра, привязанной к верхней части страницы.

7.2.2. Позиционирование псевдоэлементов

Вы разместили кнопку **Закр^ыть** в нужном месте, что довольно просто. В качестве кнопки **Закр^ыть** пользователи обычно ожидают увидеть определенный графический индикатор, такой как × (рис. 7.3).

Первое желание — удалить слово **Закр^ыть** из HTML-разметки и заменить его на ×, но это приведет к проблеме доступности: программы экранного доступа произносят название кнопки, поэтому текст должен осмысленно указывать на ее назначение. HTML-код должен иметь смысл сам по себе, до применения CSS.

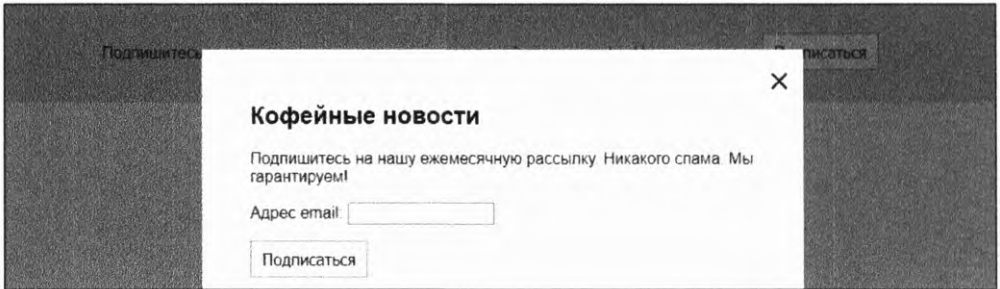


Рис. 7.3. Кнопка Закреть заменена символом ×

Вместо этого можно подключить CSS, чтобы убрать слово **Закреть** и отобразить ×. Вы сделаете это, выполнив два действия. Во-первых, вытолкнете текст кнопки за ее пределы и скроете наложение. Во-вторых, примените свойство `content`, чтобы добавить × к псевдоэлементу `::after` кнопки и, используя абсолютное позиционирование, центрировать его внутри кнопки. Обновите стили кнопок в соответствии с листингом 7.4.

СОВЕТ

Вместо × рекомендую взять символ Юникода для знака умножения. Он выглядит более симметрично и эстетично. HTML-символ `×` будет визуализироваться как нужный символ, но в свойстве `content` каскадной таблицы стилей вы должны использовать Юникод-значение сущности `\00D7`.

Листинг 7.4. Замена кнопки Закреть символом ×

```
.modal-close {
  position: absolute;
  top: 0.3em;
  right: 0.3em;
  padding: 0.3em;
  cursor: pointer;
  font-size: 2em;
  height: 1em;
  width: 1em;
  text-indent: 10em;
  overflow: hidden;
  border: 0;
}

.modal-close::after {
  position: absolute;
  line-height: 0.5;
  top: 0.2em;
  left: 0.1em;
  text-indent: 0;
  content: "\00D7";
}
```

Создается небольшая квадратная кнопка

Заставляет текст переполнять элемент и скрывает его

Добавляется символ U+00D7 (знак умножения)

Этот листинг явно устанавливает размер кнопки равным 1 ем. Затем свойство `text-indent` смещает текст вправо, за пределы элемента. Точное значение неважно, если оно больше ширины кнопки. Затем, поскольку свойство `text-indent` унаследованное, вы сбрасываете его на 0 в псевдоклассе, так что символ *x* будет без отступа.

Псевдокласс получил абсолютное позиционирование. Он ведет себя как дочерний элемент кнопки, поэтому позиционированная кнопка становится содержащим блоком для своего псевдоэлемента. Малое значение `line-height` ограничивает высоту псевдоэлемента, а свойства `top` и `left` размещают его в центре кнопки. Я подобрал точные значения после множества проб и ошибок. Рекомендую поэкспериментировать с этими значениями на панели инструментов разработчика браузера, чтобы увидеть, как они влияют на позиционирование.

Абсолютное позиционирование — сильный игрок среди типов позиционирования. Оно часто используется в сочетании с JavaScript-сценариями для всплывающих меню, подсказок и информационных окон. Мы воспользуемся его преимуществами при создании раскрывающегося меню, но сначала нужно взглянуть на его компаньона — относительное позиционирование.

7.3. Относительное позиционирование

Относительное позиционирование — вероятно, наименее понятный тип позиционирования. Когда вы применяете свойство `position: relative` к элементу, то обычно не видите никаких видимых изменений на странице. Относительно позиционированный элемент и все элементы вокруг него будут оставаться там же, где и были (правда, можно заметить некоторые изменения, произошедшие с элементами на переднем плане, но об этом чуть позже).

Свойства `top`, `right`, `bottom` и `left`, если они применяются, будут перемещать элемент из исходного положения, но он не изменит положение каких-либо элементов вокруг себя. Пример приведен на рис. 7.4. Здесь показаны четыре строчно-блочных элемента. Я придал второму элементу три дополнительных свойства: `position: relative`, `top: 1em`, `left: 2em`. Как видите, элемент переместился из исходного положения, но другие затронуты не были. Они все еще находятся в нормальном потоке документа относительно начального положения смещенного элемента.

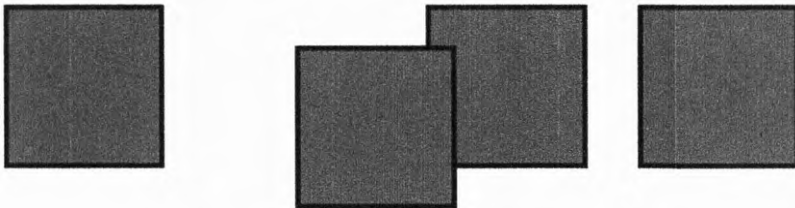


Рис. 7.4. Второй элемент смещен с использованием относительного позиционирования

Свойство `top: 1em` перемещает элемент на 1 em вниз от исходного верхнего края. А свойство `left: 2em` смещает элемент на 2 em вправо от исходного левого края. Эти смещения могут привести к тому, что элемент будет перекрывать элементы, расположенные ниже его или рядом с ним. При позиционировании используются и отрицательные значения, поэтому свойство `bottom: -1em` переместит элемент вниз на 1 em так же, как `top: 1em`.

ПРИМЕЧАНИЕ

В отличие от фиксированного и абсолютного позиционирования здесь вы не сможете использовать свойства `top`, `right`, `bottom` и `left`, чтобы изменить размер относительно позиционированного элемента. Эти значения будут смещать элемент вверх или вниз, влево или вправо. Применяйте свойство `top` или `bottom`, но не оба сразу (`bottom` будет проигнорировано). Аналогично используется свойство `left` или `right`, но не оба (`right` будет проигнорировано).

Иногда полезно задействовать эти свойства для коррекции относительной позиции элемента, чтобы подтолкнуть его на место, но, честно говоря, вы редко будете работать с относительным позиционированием. Гораздо чаще станете применять свойство `position: relative` к содержащему блоку для абсолютного позиционирования элемента внутри него.

7.3.1. Создание раскрывающегося меню

Используем относительное и абсолютное позиционирование для создания раскрывающегося меню. Выглядит оно как простой прямоугольник, но, когда пользователь наводит на него указатель мыши, открывается список ссылок (рис. 7.5).

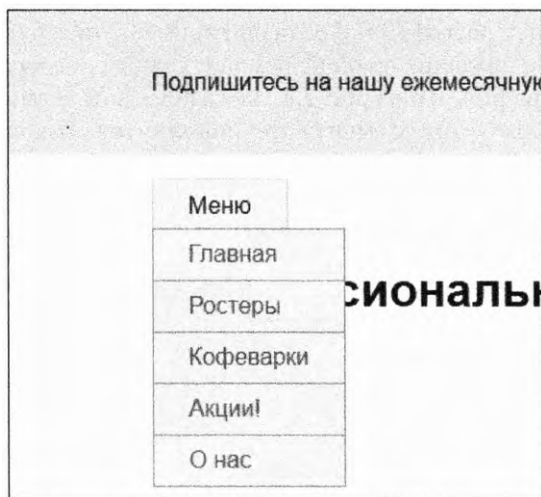


Рис. 7.5. Раскрывающееся меню

В листинге 7.5 показана разметка этого меню. Добавьте ее в свой HTML-код после закрывающего тега `</div>` элемента `div class = "modal"`. Этот код содержит контейнер, который вы примените для центрирования контента и его выравнивания с контентом баннера. Я также добавил заголовок `h1` под всплывающим окном, чтобы проиллюстрировать, как оно отображается поверх прочего контента на странице.

Листинг 7.5. HTML-код раскрывающегося меню

```
<div class="container">
  <nav>
    <div class="dropdown">
      <div class="dropdown-label">Меню</div>
      <div class="dropdown-menu">
        <ul class="submenu">
          <li><a href="/">Главная</a></li>
          <li><a href="/coffees">Ростеры</a></li>
          <li><a href="/brewers">Кофеварки</a></li>
          <li><a href="/specials">Акции!</a></li>
          <li><a href="/about">О нас</a></li>
        </ul>
      </div>
    </div>
  </nav>

  <h1>Профессиональная техника для обжарщиков</h1>
</div>
```

Контейнер для раскрывающегося меню

Метка отображается всегда

Контейнер div для отображения/скрытия контента, когда меню открыто или закрыто

Контейнер раскрывающегося меню включает в себя два потомка: метку — серый прямоугольник, который отображается всегда, и контент меню, который будет отображаться/скрываться при открытии/закрытии меню. Раскрывающееся меню будет позиционировано абсолютно, чтобы не могло перемещаться по макету страницы при наведении указателя мыши. Это означает, что оно отображается поверх прочего контента, когда раскрыто.

Затем примените относительное позиционирование к контейнеру раскрывающегося меню. Это устанавливает содержащий блок для абсолютно позиционированного меню. Добавьте в таблицу стилей следующие стили (листинг 7.6).

Листинг 7.6. Раскрытие меню при наведении указателя мыши

```
.container {
  width: 80%;
  max-width: 1000px;
  margin: 1em auto
}

.dropdown {
  display: inline-block;
  position: relative;
}

.dropdown-label {
```

Установка содержащего блока

```
padding: 0.5em 1.5em;
border: 1px solid #ccc;
background-color: #eee;
}

.dropdown-menu {
  display: none;
  position: absolute;
  left: 0;
  top: 2.1em;
  min-width: 100%;
  background-color: #eee;
}
.dropdown:hover .dropdown-menu {
  display: block;
}

.submenu {
  padding-left: 0;
  margin: 0;
  list-style-type: none;
  border: 1px solid #999;
}

.submenu > li + li {
  border-top: 1px solid #999;
}

.submenu > li > a {
  display: block;
  padding: 0.5em 1.5em;
  background-color: #eee;
  color: #369;
  text-decoration: none;
}

.submenu > li > a:hover {
  background-color: #fff;
}
```

Меню изначально скрыто

Позиционирование контента меню под раскрывающимся меню

Открывает меню при наведении указателя мыши

Теперь, когда вы наводите указатель мыши на метку **Меню**, ниже появляется раскрывающееся меню. Обратите внимание: вы задали состояние `:hover` для всего контейнера, чтобы открыть меню. Это означает, что до тех пор, пока указатель мыши остается поверх любой части его контента, будь то `dropdown-label` или `dropdown-menu`, меню остается открытым.

В абсолютно позиционированном элементе `dropdown-menu` вы использовали свойство `left: 0`, чтобы выровнять его левую сторону по левой стороне раскрывающегося меню. Затем применили свойство `top: 2.1em`, чтобы поместить его верхний край под меткой (высота метки вместе с отступом и границей составляет около 2,1 em). Свойство `min-width` со значением `100%` гарантирует, что метка будет по крайней

мере такой же широкой, как и контейнер меню (его ширина определяется элементом `dropdown-label`). Далее использовали класс `submenu` для форматирования меню внутри раскрывающегося меню. (Если в этот момент вы откроете модальное окно, то увидите, что оно очень странно отображается позади меню. Это нормально, мы скоро решим эту проблему.)

Важные замечания по поводу раскрывающихся меню

Для простоты в примере в листинге 7.6 используется псевдокласс `:hover`, отвечающий за открытие меню, когда пользователь наводит на него указатель мыши. Этот пример не завершен. Как правило, более надежный подход — применение JavaScript-сценария для добавления и удаления класса, который контролирует, будет ли открыто меню. Этот прием позволяет добавить небольшую задержку перед открытием/закрытием меню, чтобы предотвратить непреднамеренное срабатывание при быстром проходе указателя мыши.

Кроме того, хотя наш пример работает при использовании мыши, он не будет действовать на части устройств с сенсорным экраном (лишь некоторые устройства с сенсорным экраном иницируют состояние `:hover` при нажатии). Он также не учитывает проблемы доступности программ экранного доступа или навигации с помощью клавиатуры. Было бы разумно увеличить раскрывающееся меню, чтобы обеспечить работу сенсорных элементов управления, и оставлять меню открытым, когда пользователь нажимает клавишу `Tab` для навигации по ссылкам в нем.

Программирование на языке JavaScript выходит за рамки этой книги, но если вы хорошо разбираетесь в нем, то решите эти проблемы, самостоятельно написав нужный код. В качестве альтернативы используйте стороннюю библиотеку, которая предоставляет функции раскрывающихся меню, реализуя их за вас, а затем задействуйте CSS, чтобы настроить внешний вид по своему вкусу.

7.3.2. Создание треугольника CSS

Добавим последний штрих к раскрывающемуся меню. Оно работает уже сейчас, но вид метки **Меню** не говорит пользователю о том, что кнопка скрывает дополнительный контент. Добавим к метке небольшой направленный вниз треугольник, чтобы показать, что меню можно открыть.

Используйте небольшой трюк с границами, чтобы нарисовать треугольник. С помощью псевдоэлемента `::after` метки раскрывающегося меню нарисуйте треугольник, затем примените абсолютное позиционирование, чтобы поместить его с правой стороны метки.

В большинстве случаев, добавляя границы к элементу, вы делаете их тонкими, шириной 1 или 2 пиксела. Но понаблюдайте, что происходит, когда вы задаете

намного более толстую границу (рис. 7.6). Я обозначил все стороны границы разными цветами, чтобы было видно, где заканчивается один край и начинается другой.

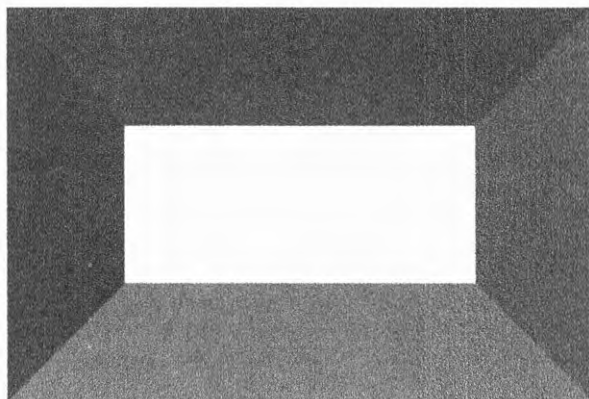


Рис. 7.6. Элемент с толстыми границами

Обратите внимание: в углах, где встречаются границы двух сторон, образуется диагональ. Теперь посмотрите, что произойдет, если уменьшить элемент до нулевых высоты и ширины (рис. 7.7). Границы объединятся и сойдутся к центру.

Граница каждой из сторон элемента образует треугольник. Верхняя граница указывает вниз, правая — влево и т. д. Можно окрасить одну из них, чтобы создать нужный треугольник, и установить для остальных прозрачность. Элемент с прозрачными левой и правой границами и видимой верхней будет выглядеть как обычный треугольник (рис. 7.8).

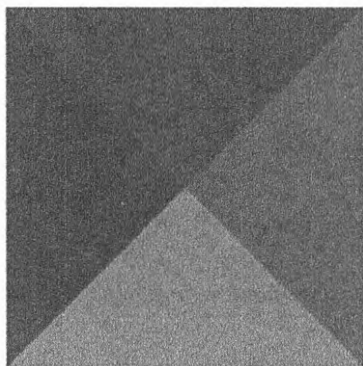


Рис. 7.7. Если элемент не имеет высоты или ширины, каждая граница образует треугольник

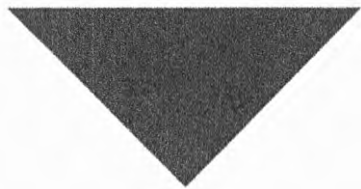


Рис. 7.8. Треугольник, сформированный из верхней границы элемента

Применим стили к методу `::after` псевдоэлемента, чтобы создать треугольник и разместить его, используя абсолютное позиционирование. Добавьте в таблицу стилей следующие стили (листинг 7.7).

Листинг 7.7. Абсолютное позиционирование треугольника на метке раскрывающегося меню

```
.dropdown-label {
  padding: 0.5em 2em 0.5em 1.5em;
  border: 1px solid #ccc;
  background-color: #eee;
}

.dropdown-label::after {
  content: "";
  position: absolute;
  right: 1em;
  top: 1em;
  border: 0.3em solid;
  border-color: black transparent transparent;
}

.dropdown:hover .dropdown-label::after {
  top: 0.7em;
  border-color: transparent transparent black;
}
```

Увеличение правого отступа, чтобы добавить место для треугольника

Позиционирование элемента справа от метки

Применение верхней границы для создания треугольника с углом вниз

При наведении указателя мыши изменяется на треугольник с углом вверх

У псевдоэлемента нет контента, поэтому он не имеет высоты или ширины. Затем используете свойство `border-color`, чтобы сделать верхнюю границу черной, а боковые и нижнюю границы прозрачными для создания треугольника углом вниз. Дополнительный отступ, применяемый к элементу `dropdown-label`, создает пространство справа, где можно разместить треугольник. Результат показан на рис. 7.9.

Когда меню открыто, треугольник меняется — теперь угол направлен вверх, показывая, что меню можно закрыть. Небольшое изменение значения свойства `top` (от 1 em до 0,7 em) помогает треугольнику с углом вверх визуально отображаться в той же позиции, что и стрелка вниз.

Как вариант можете создать подобный указатель меню с помощью рисунка или фонового изображения, но несколько строк CSS-кода избавляют пользователей от лишнего сетевого запроса. Результатом будет небольшой штрих, который способен принести много пользы вашему приложению или сайту.

Эту технику можно использовать для создания других сложных форм, например трапеций, шестиугольников и звезд. Внушительный список фигур, построенных с помощью CSS, находится на сайте css-tricks.com/examples/ShapesOfCSS/.

Меню ▾

Рис. 7.9. Метка меню с треугольником с углом вниз

7.4. Контексты наложения и z-индекса

Позиционирование полезно, но важно знать о последствиях его применения. Когда вы удаляете элемент из потока документа, вам приходится отвечать за все, чем поток документа обычно занимается самостоятельно.

Вы должны убедиться, что элемент не выходит за пределы области просмотра браузера, тем самым скрываясь от пользователя. И следует проверить, что он не перекрывает важный контент, который вам нужен.

В конце концов, вы будете сталкиваться с проблемой наложения. При размещении нескольких элементов на одной странице может возникнуть ситуация, когда два позиционированных элемента перекрывают друг друга. Иногда один объект неожиданно возникает поверх других. В данной главе я намеренно создал такой сценарий, чтобы проиллюстрировать сказанное.

Нажмите кнопку **Подписаться** в шапке созданной вами страницы, чтобы открыть модальное окно. Если вы разместили разметку раскрывающегося меню после кода модального окна в HTML-коде, результат будет выглядеть как на рис. 7.10. Обратите внимание: раскрывающееся меню, добавленное на страницу, отображается поверх модального окна.

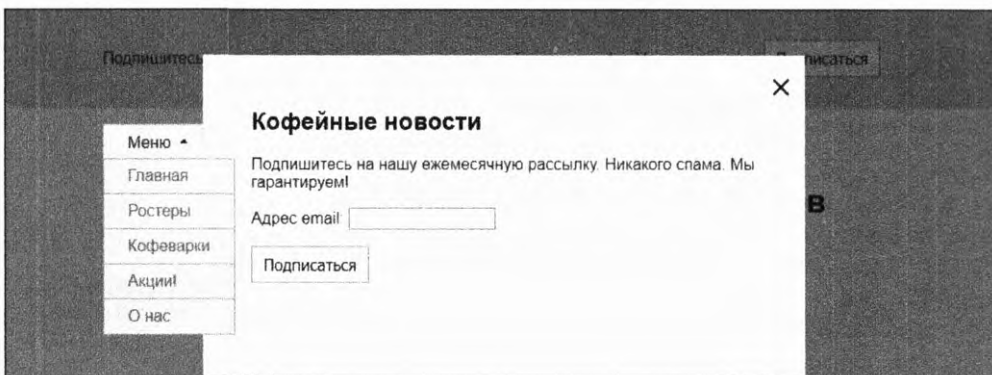


Рис. 7.10. Ошибка: модальное окно отображается позади раскрывающегося меню

Проблему можно решить несколькими способами. Прежде чем заниматься этим, важно понять, как браузер определяет порядок наложения объектов. Для этого нужно более внимательно изучить, как браузер визуализирует (или рендерит) страницу.

7.4.1. Процесс рендеринга и порядок наложения

Поскольку браузер анализирует HTML-код в DOM, он создает и другую древовидную структуру, называемую *деревом визуализации*. Оно показывает внешний вид и позицию каждого элемента, а также определяет порядок, в котором браузер будет *рисовать* элементы. Этот порядок важен, потому что элементы, нарисованные позже, появляются поверх элементов, нарисованных ранее, и перекрывают их.

При нормальных обстоятельствах, то есть до применения позиционирования, этот порядок определяется порядком указания элементов в HTML-коде. Рассмотрим три элемента разметки:

```
<div>первый</div>
<div>второй</div>
<div>третий</div>
```

Вариант их наложения показан на рис. 7.11. Я использовал определенные отрицательные значения полей, чтобы заставить их перекрываться, но не применял к ним никакого позиционирования. Элементы, указанные в разметке позже, нарисованы поверх предыдущих.

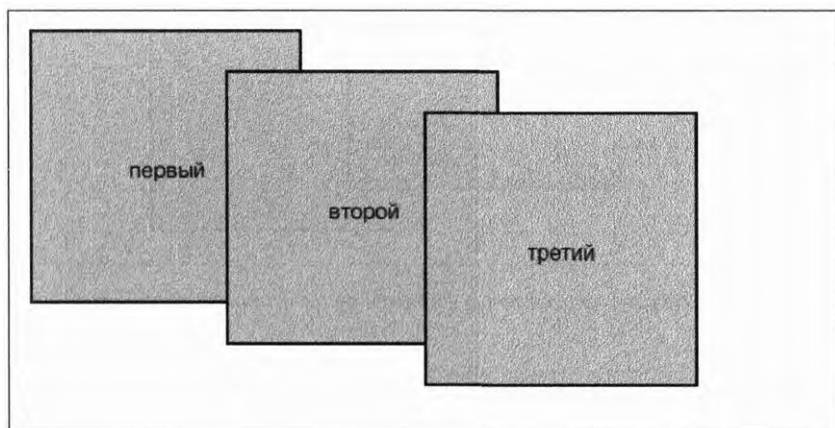


Рис. 7.11. Три элемента, накладываемые обычным образом. Упомянутые в разметке позже появляются поверх ранних

Это поведение меняется, когда вы начинаете позиционировать элементы. Браузер сначала рисует все непозиционированные элементы, а затем позиционированные. По умолчанию любой позиционированный элемент появляется поверх любых непозиционированных. На рис. 7.12 показан результат добавления свойства `position: relative` к первым двум элементам. Это выводит их на передний план, перекрывая статически расположенный третий элемент, хотя порядок элементов в HTML-коде не изменен.

Обратите внимание: среди позиционированных элементов второй элемент все еще появляется поверх первого. Позиционированные элементы выводятся на передний план, но исходная зависимость между ними остается неизменной.

В контексте нашего примера это означает, что и модальное окно, и раскрывающееся меню отображаются поверх статического контента (что, собственно, и нужно), но в зависимости от того, что указано первым в разметке, один объект отображается поверх другого. Один из способов исправить проблему с меню — переместить элемент `div class = "modal"` и все его содержимое куда-нибудь после раскрывающегося меню.

Как правило, модальные окна добавляются в конец страницы как последняя часть контента перед закрывающим тегом `</body>`. Большинство библиотек JavaScript для создания модальных окон делают это автоматически.

Поскольку позиция модального окна фиксирована, не имеет значения, где оно указано в разметке, — окно всегда будет располагаться в центре экрана.

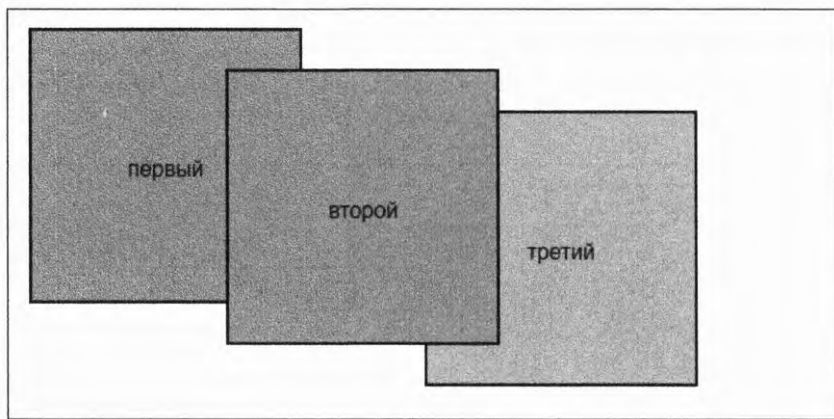


Рис. 7.12. Позиционированные элементы рисуются поверх статических

Перемещение элемента в другое место разметки не повлияло бы на фиксированное позиционирование, но это не то решение, которое можно использовать в случае относительного или абсолютного позиционирования. Относительное позиционирование зависит от документа, а абсолютное позиционирование — от его позиционированного элемента-предка. Нам нужен способ контролировать их наложение. Это делается с помощью свойства `z-index`.

7.4.2. Управление наложением с помощью свойства `z-index`

Свойству `z-index` может быть присвоено любое целое число, и положительное, и отрицательное. Буква *z* относится к уровню глубины в прямоугольной системе координат *XYZ*. Элементы с большим значением свойства `z-index` появляются поверх элементов с меньшим его значением. Элементы с отрицательным значением свойства `z-index` появляются позади статических элементов.

Свойство `z-index` — это второй подход, который можно использовать для устранения проблем наложения компонентов страницы. Он обеспечивает большую свободу структурирования HTML-кода. Применим свойство `z-index` со значением 1 к элементу `modal-backdrop` и со значением 2 — к элементу `modal-body` (таким образом, тело модального окна появится поверх задника). Обновите эту часть кода таблицы стилей, чтобы она соответствовала листингу 7.8.

Свойство `z-index` кажется простым, но из-за него возникают два сбоя. Во-первых, это свойство работает только с позиционированными элементами. Вы не можете управлять порядком наложения статических элементов. Во-вторых, применение `z-index` к позиционированному элементу формирует так называемый контекст наложения.

Листинг 7.8. Добавление свойства `z-index` к модальному окну, чтобы вывести его поверх раскрывающегося меню

```
.modal-backdrop {  
  position: fixed;  
  top: 0;  
  right: 0;  
  bottom: 0;  
  left: 0;  
  background-color: rgba(0, 0, 0, 0.5);  
  z-index: 1;  
}  
  
.modal-body {  
  position: fixed;  
  top: 3em;  
  bottom: 3em;  
  right: 20%;  
  left: 20%;  
  padding: 2em 3em;  
  background-color: white;  
  overflow: auto;  
  z-index: 2;  
}
```

Выводит задник поверх элементов без `z-индекса`

Выводит элемент `modal-body` поверх задника

7.4.3. Контексты наложения

Контекст наложения состоит из элемента или группы элементов, которые браузер рисует вместе. Один элемент — это корень контекста наложения, поэтому, когда вы добавляете свойство `z-index` к позиционированному элементу, он становится корнем нового контекста наложения. Все его дочерние элементы — часть этого контекста наложения.

Контексты наложения не следует путать с блочными контекстами форматирования (см. главу 4). Это разные понятия, хотя и не обязательно взаимоисключающие. Сложные контексты обуславливают, какие элементы находятся поверх других, блочные контексты форматирования связаны с потоком документа и определяют, будут ли элементы перекрываться.

То, что все элементы контекста наложения рисуются вместе, имеет важное последствие: ни один элемент вне контекста наложения не может быть размещен между любыми двумя элементами, находящимися внутри него. Иными словами, если элемент накладывается поверх контекста наложения, выше его невозможно поместить ни один элемент, находящийся внутри этого контекста. Аналогично, если элемент на странице помещается позади контекста наложения, ни один элемент внутри этого контекста нельзя поместить позади данного элемента.

Возможно, это трудно понять, поэтому проиллюстрирую сказанное примером. На новую HTML-страницу добавьте следующую разметку (листинг 7.9).

Листинг 7.9. Пример контекста укладки

```

<div class="box one positioned">
  первый
  <div class="absolute">вложенный</div>
</div>
<div class="box two positioned">второй</div>
<div class="box three">третий</div>

```

Эта разметка состоит из трех полей, два из которых будут позиционированы и отформатированы с помощью свойства `z-index`: 1. Элемент `absolute` внутри первого блока позиционируется и форматируется с помощью свойства `z-index: 100`. Несмотря на высокое значение свойства `z-index`, первый блок по-прежнему появляется позади второго, потому что его родительский элемент формирует контекст наложения позади второго блока (рис. 7.13).

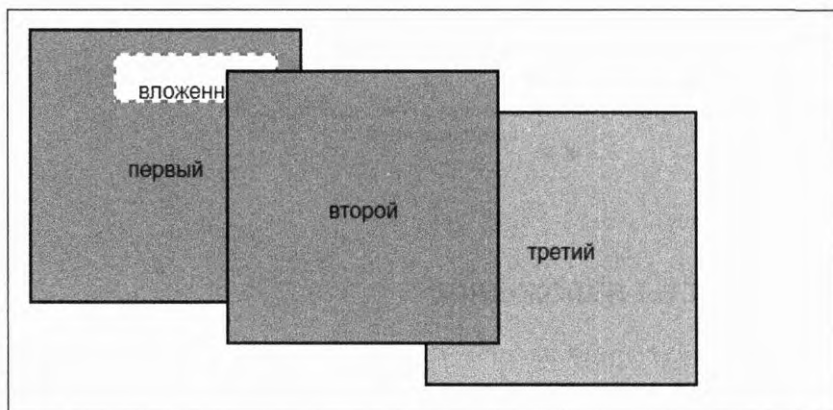


Рис. 7.13. Весь контекст наложения накладывается относительно прочих элементов на странице

Стили этого сценария приведены в листинге 7.10. Примените их на своей странице. Большинство стилей предназначены для настройки размеров и цвета, чтобы наглядно передать порядок наложения. Отрицательные значения полей заставляют элементы перекрываться. Единственные существенные объявления — свойства `position` и `z-index` различных элементов.

Листинг 7.10. Создание контекстов наложения

```

body {
  margin: 40px;
}

.box {
  display: inline-block;
  height: 200px;
  width: 200px;
  line-height: 200px;
  text-align: center;
}

```

```
border: 2px solid black;
background-color: #ea5;
margin-left: -60px;
vertical-align: top;
}

.one {
  margin-left: 0;
}

.two {
  margin-top: 30px;
}

.three {
  margin-top: 60px;
}

.positioned {
  position: relative;
  background-color: #5ae;
  z-index: 1;
}

.absolute {
  position: absolute;
  top: 1em;
  right: 1em;
  height: 2em;
  background-color: #fff;
  border: 2px dashed #888;
  z-index: 100;
  line-height: initial;
  padding: 1em;
}
```

Каждый позиционируемый блок устанавливает контекст наложения со свойством z-index: 1

Свойство z-index определяет позицию наложения элемента в его контексте наложения

Первый блок, размещенный позади второго, является корнем контекста наложения. Из-за этого абсолютно позиционированный элемент внутри него не может появиться поверх второго блока, даже с учетом высокого значения свойства `z-index`. Поэкспериментируйте с этим примером на панели инструментов разработчика браузера, чтобы разобраться в принципах работы. Управляйте значениями свойства `z-index` всех элементов, чтобы получить разные результаты.

ПРИМЕЧАНИЕ

Добавление свойства `z-index` позиционируемому элементу — наиболее важный способ создания контекста наложения, но некоторые другие свойства также могут создать его. Значение свойства `opacity` менее 1 также создает его, как и свойства `transform` и `filter`. Они фундаментально влияют на то, как элемент и его потомки визуализируются, поэтому все они рисуются вместе. Корень документа (элемент `html`) также создает контекст наложения верхнего уровня для всей страницы.

Все элементы в контексте наложения размещаются в следующем порядке от заднего плана к переднему.

1. Корневой элемент контекста наложения.
2. Позиционированные элементы с отрицательным значением свойства `z-index` (и их дочерние элементы).
3. Непозиционированные элементы.
4. Позиционированные элементы со свойством `z-index: auto` (и их дочерние элементы).
5. Позиционированные элементы с положительным значением свойства `z-index` (и их дочерние элементы).

Переменные для отслеживания позиций `z-index`

В таблице стилей легко напортачить с позициями `z-index`, если не понимаешь, каким должен быть порядок в приоритетах различных компонентов. Разработчики, верстающие стили, могут добавить, к примеру, модальное окно и, опасаясь, что оно скроется за каким-либо контентом страницы, присвоить ему высокое значение свойства `z-index`, скажем 999999. Если таких ситуаций будет несколько, останется только догадываться, каким окажется значение свойства `z-index` нового компонента.

Если вы используете препроцессор, такой как LESS или SASS (см. приложение Б), или работаете с браузерами, которые поддерживают пользовательские свойства (см. главу 2), обратите эти особенности в свою пользу. Поместите все значения свойства `z-index` в переменные в одной позиции таблицы стилей. Таким образом вы сразу поймете, какой объект появится поверх другого:

```
--z-loading-indicator: 100;  
--z-nav-menu:         200;  
--z-dropdown-menu:    300;  
--z-modal-backdrop:   400;  
--z-modal-body:       410;
```

Увеличивайте значения на десятки или сотни, чтобы в случае необходимости без проблем добавлять новые значения между имеющимися.

Если вы когда-либо обнаружите, что свойство `z-index` ведет себя не так, как ожидается, посмотрите дерево DOM предков элемента, чтобы определить корень контекста наложения. Затем настройте значения свойства `z-index` так, чтобы перенести весь контекст наложения вперед или назад. Будьте осторожны: можно обнаружить несколько контекстов наложения, вложенных друг в друга.

Если страница сложная, бывает трудно определить, какой именно контекст наложения отвечает за поведение, которое вы видите. По этой причине всегда будьте осторожны при создании нового контекста наложения. Не создавайте его, если нет конкретной причины. Это вдвойне верно для элементов, которые занимают круп-

ные части страницы. Когда возможно, перенесите автономные позиционированные элементы, такие как модальные окна, на верхний уровень DOM, прямо перед закрывающим тегом `</body>`, чтобы избавить их от внешних контекстов наложения.

Некоторые разработчики попадают в ловушку позиционирования большого количества элементов на всей странице. Вы должны бороться с этим желанием. Чем больше элементов вы позиционируете, тем сложнее становится страница и тем труднее ее отлаживать. Если вы поймали себя на том, что разместили десятки элементов, вернитесь назад и переосмыслите ситуацию. Это особенно важно, если вы столкнулись с проблемами компоновки макета. Когда макет может быть сверстан с использованием других методов, нужно так и сделать.

Если вы можете взвалить на плечи потока документа часть работы, а не явно позиционировать объекты, браузер позаботится об этом во множестве основополагающих случаев. Помните, что позиционирование извлекает элементы из потока документа. В целом вы должны вспоминать о позиционировании только тогда, когда нужно разместить одни элементы поверх других.

7.5. Липкое позиционирование

Продолжительное время использовались четыре основных типа позиционирования: статическое, фиксированное, абсолютное и относительное, но теперь в браузерах доступен новый тип — *липкое позиционирование*. Это своего рода гибрид между относительным и фиксированным позиционированием: элемент прокручивается со страницей до тех пор, пока не достигнет определенной позиции на экране, после чего «заблокируется» на месте, даже если пользователь продолжит прокрутку. Обычно таким поведением награждают навигационные панели.

Некоторое время липкое позиционирование поддерживал только браузер Firefox. Но на момент написания книги эта функция стала доступна в браузерах Chrome и Edge. Safari поддерживает этот тип позиционирования с вендорным префиксом (`position: -webkit-sticky`). Обязательно посетите сайт caniuse.com/#feat=css-sticky, чтобы получить актуальную информацию о поддержке липкого позиционирования браузерами. Как правило, в качестве резервного варианта для других браузеров используется фиксированное или абсолютное позиционирование.

Давайте обновим страницу с модальным окном и раскрывающимся меню. Вы измените ее макет на двухколоночный, добавив липкую боковую панель в качестве правой колонки. Страница примет вид, показанный на рис. 7.14.

Когда страница загружается, положение боковой панели не выглядит необычным. Панель будет прокручиваться в обычном режиме вместе со страницей до определенной позиции. Как только панель покинет область просмотра, она будет зафиксирована на месте.

Панель останется на месте как элемент с фиксированным позиционированием, пока остальная часть страницы продолжает прокручиваться. Если немного прокрутить страницу, она примет примерно такой вид, как на рис. 7.15.

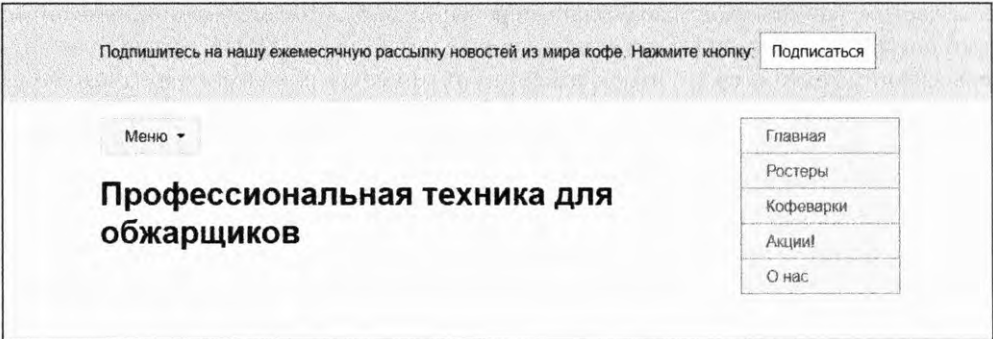


Рис. 7.14. Липкая боковая панель первоначально расположена как любая другая



Рис. 7.15. Боковая панель фиксируется на месте

Давайте немного изменим структуру страницы, чтобы определить две колонки. Измените контейнер в HTML-коде, чтобы он соответствовал листингу 7.11. Так вы поместите существующий контент (раскрывающееся меню и заголовки страницы) в левую колонку и добавите правую с прикрепленным меню.

Листинг 7.11. Переход к двухколоночному макету с боковой панелью

```
<div class="container">
  <main class="col-main">
    <nav>
      <div class="dropdown">
        <div class="dropdown-label">Меню</div>
        <div class="dropdown-menu">
          <ul class="submenu">
            <li><a href="/">Главная</a></li>
            <li><a href="/coffees">Ростеры</a></li>
            <li><a href="/brewers">Кофеварки</a></li>
            <li><a href="/specials">Акции!</a></li>
            <li><a href="/about">О нас</a></li>
          </ul>
        </div>
      </div>
    </nav>
```

Обтекание существующего контента
в элементе col-main основной колонки

```

<h1>Профессиональная техника для обжарщиков</h1>
</main>

<aside class="col-sidebar">
  <div class="affix">
    <ul class="submenu">
      <li><a href="/">Главная</a></li>
      <li><a href="/coffees">Ростеры</a></li>
      <li><a href="/brewers">Кофеварки</a></li>
      <li><a href="/specials">Акции!</a></li>
      <li><a href="/about">О нас</a></li>
    </ul>
  </div>
</aside>
</div>

```

Добавление второй колонки
с элементом affix внутри нее

Теперь обновите CSS-код, чтобы создать пластичный флекс-контейнер, в который уместятся две колонки. Для примера вы переназначите стили подменю от раскрывающегося меню, хотя и можете добавлять любые желаемые элементы и стили к боковой панели. Добавьте приведенные далее стили в свою таблицу стилей.

Листинг 7.12. Создание двухколоночного макета и меню с липким позиционированием

```

.container {
  display: flex;
  width: 80%;
  max-width: 1000px;
  margin: 1em auto;
  min-height: 100vh;
}

.col-main {
  flex: 1 80%;
}

.col-sidebar {
  flex: 20%;
}

.affix {
  position: sticky;
  top: 1em;
}

```

Создание флекс-контейнера
для двухколоночного макета

Приращение
высоты контейнера

Компоновка двух колонок

Применение липкого позиционирования
к боковому меню. Оно будет пристыковано
на расстоянии 1 em от верхнего края
области просмотра

Большинство из этих изменений настраивают двухколоночную компоновку. С учетом этого для позиционирования элемента `affix` требуется всего две декларации. Значение свойства `top` определяет позицию фиксации элемента: 1 em от верхнего края области просмотра.

Липкий элемент всегда будет находиться в пределах своего родительского элемента — в нашем случае `col-sidebar`. Когда вы прокручиваете страницу вниз, боковая панель `col-sidebar` всегда будет двигаться вместе с ней, но элемент `affix` будет

прокручиваться до тех пор, пока не достигнет позиции блокировки. Продолжайте прокручивать страницу, и он разблокируется и снова станет подвижным. Это происходит, когда нижний край родительского элемента достигает нижнего края липкого меню. Обратите внимание на то, что родительский элемент должен быть выше липкого элемента, чтобы он мог правильно расположиться, поэтому я искусственно увеличил его высоту, добавив значение свойства `min-height` в его `flex`-контейнер.

Итоги главы

- ❑ Используйте фиксированное позиционирование для модальных окон.
- ❑ Применяйте абсолютное позиционирование для раскрывающихся меню, всплывающих подсказок и прочих динамических взаимодействий.
- ❑ Помните о проблемах совместимости при использовании данных типов позиционирования.
- ❑ У свойства `z-index` есть две особенности: оно работает только с позиционируемыми элементами, с его помощью создается новый контекст наложения.
- ❑ Помните о потенциальных ошибках при создании нескольких контекстов наложения на странице.
- ❑ Отслеживайте реализацию в браузерах поддержки липкого позиционирования.

8

Адаптивный дизайн

В этой главе

- Верстка веб-страниц под различные устройства и размеры экрана.
- Использование медиазапросов для изменения дизайна в зависимости от размера области просмотра.
- Применение подхода Mobile First.
- Адаптивные изображения.

Всемирная паутина окружает нас повсюду. Мы используем ее на рабочем компьютере в офисе. Мы путешествуем по ней с помощью планшета, лежа в постели. Попасть в нее можно даже через экраны некоторых телевизоров. И мы всюду носим ее с собой в смартфонах. Веб-платформа HTML, CSS и JavaScript представляют собой универсальную экосистему, отличающуюся от всего, что было раньше.

Это ставит перед веб-разработчиками интересную задачу — создать сайт, который хорошо функционирует и выглядит вне зависимости от устройства, с которого пользователи на него вышли. Изначально многие разработчики решали эту задачу путем создания настольной и мобильной версий сайта. Затем сервер перенаправлял мобильные устройства с www.wombatcoffee.com на m.wombatcoffee.com. Мобильный сайт обычно отличался минимальной функциональностью и упрощенным дизайном, разработанным для небольших экранов.

Актуальность этого подхода уменьшалась по мере выхода на рынок все новых устройств. На какую версию сайта следует перенаправить планшет — мобильную или настольную? Как насчет большого смартфона? Или iPad Mini? Что, если пользователь мобильного устройства захочет выполнить действие, которое предусмотрено только в настольной версии сайта? В конце концов, вынужденная дихотомия между настольной и мобильной версиями создает больше проблем, чем решает. Кроме того, при таком подходе вы вынуждены поддерживать работу дополнительного сайта.

Гораздо лучше давать всем пользователям один и тот же HTML- и CSS-код. С помощью нескольких ключевых методов вы можете представлять свой контент по-разному в зависимости от размера области просмотра пользовательского браузера (а иногда — от разрешения экрана). В этом случае вам не понадобятся два разных

сайта. Вы создаете один сайт, который работает на смартфоне, планшете и любом другом устройстве. Этот подход, популяризированный веб-дизайнером Этаном Маркоттом (Ethan Marcotte), называется *адаптивным дизайном*.

Путешествуя по Всемирной паутине, обращайте внимание на сайты с адаптивным дизайном. Смотрите, как они реагируют на разную ширину окна браузера. Особенно интересными в этом плане являются сайты газет, поскольку их страницы вмещают очень большой объем контента. На момент написания этой книги сайт www.bostonglobe.com был отличным примером, предусматривающим одно-, двух- или трехколоночный макет в зависимости от ширины окна браузера. Суть адаптивного дизайна, как правило, заключается в том, что макет страницы мгновенно изменяется в ответ на изменение ширины окна браузера.

Тремя ключевыми принципами адаптивного дизайна являются следующие.

1. *Подход Mobile First*. Это означает, что мобильная версия сайта разрабатывается до создания настольной версии.
2. *Правило @media*. С его помощью вы можете адаптировать свои стили под разные размеры области просмотра. Этот синтаксис (часто называемый *медиазапросами*) позволяет писать стили, применяемые только при определенных условиях.
3. *Использование резиновых макетов*. Этот подход допускает масштабирование контейнеров в зависимости от ширины области просмотра.

В данной главе мы рассмотрим эти три принципа в процессе создания адаптивного макета страницы. После этого поговорим об изображениях, которые требуют особого внимания при верстке адаптивных сайтов.

8.1. Подход Mobile First

Первым принципом адаптивного дизайна является подход *Mobile First*. Как следует из названия, его суть заключается в том, что мобильная версия разрабатывается до создания настольной. Это лучший способ обеспечить функциональность обеих версий.

Разработка сайтов для мобильных устройств предполагает учет множества ограничений. Невелика площадь экрана. Интернет-соединение часто более медленное. Пользователь мобильного устройства задействует другой набор интерактивных элементов управления. Ввод текста возможен, но неудобен. Пользователь не может навести указатель мыши на элемент, чтобы получить тот или иной эффект. Если вы сначала создадите полнофункциональный интерактивный сайт, а затем попытаетесь адаптировать его с учетом этих ограничений, у вас, скорее всего, ничего не получится.

Подход же *Mobile First* предполагает, что вы с самого начала должны разрабатывать сайт с учетом этих ограничений. После создания рабочей мобильной версии (или хотя бы разработки ее плана) можете использовать прогрессивное улучшение для расширения функциональности версии, предназначенной для устройств с большим экраном.

На рис. 8.1 показана страница, которую вам предстоит создать. Понятно, что это макет страницы для мобильного устройства.



Рис. 8.1. Макет страницы для мобильного устройства

Эта страница состоит из трех основных компонентов: шапки, hero-изображения с наложенным поверх него небольшим абзацем текста и области с основным контентом. Можно также развернуть скрытое меню, щелкнув на значке в правом верхнем углу (рис. 8.2). Этот значок в виде трех горизонтальных линий часто называют *гамбургером*, поскольку он напоминает булочку с котлетой.

Мобильный макет обычно не предполагает никаких излишеств. Если не считать интерактивного меню, его фокусом является контент. В макетах страниц для больших экранов вы можете выделить много места под такие элементы, как заголовок, hero-изображение и меню. Однако пользователи мобильных устройств часто больше ориентированы на решение конкретных задач. Во время встречи с друзьями им может понадобиться быстро найти часы работы магазина или другую конкретную информацию, например цены или адрес.

Для мобильного дизайна главным является контент. Представьте себе дизайн настольной версии сайта: с одной стороны расположена статья, с другой — боковая

панель, содержащая ссылки и менее важные элементы. Вы хотите, чтобы на экране мобильного устройства сначала появлялась статья. Это означает, что в HTML-коде первым должен идти самый важный контент. Кроме того, это связано с проблемами доступности: программы экранного доступа сразу переходят к самому важному, а пользователь, перемещающийся по сайту с помощью клавиатуры, добирается до ссылок в статье прежде, чем до ссылок на боковой панели.

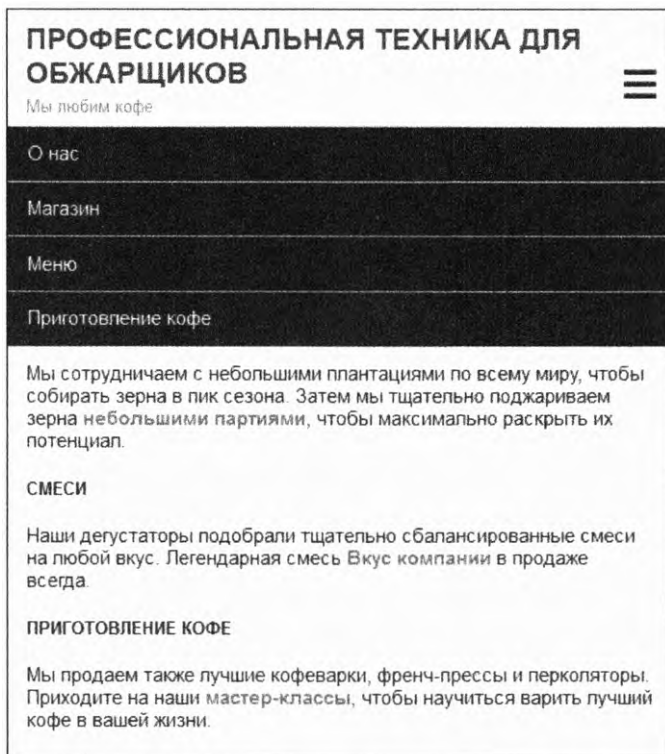


Рис. 8.2. Мобильная страница с меню, развернутым нажатием на значок-«гамбургер» или щелчком на нем

Это правило не жесткое. Вероятно, вы могли бы сказать, что hero-изображение не так важно, как расположенный под ним контент. Однако это яркая часть дизайна, поэтому я считаю, что в данном случае его стоит поместить в верхней части страницы. Кроме того, оно содержит минимум контента, так что навигация по нему не требует особых усилий.

ВАЖНО

При написании HTML-кода для адаптивного макета важно включить в него все необходимое для всех размеров экрана. Можете применить для каждого случая разные CSS, однако HTML-код должен быть одним и тем же.

Теперь рассмотрим макет для более крупных экранов. Сначала вы создадите мобильную версию, однако в процессе работы будет проще принимать решения, если вы станете учитывать общий дизайн. В этом упражнении вы добавите *контрольную точку* для переключения между макетами для среднего и большого экранов. На рис. 8.3 показан макет для экрана среднего размера.



Контрольная точка — это конкретная точка, в которой стили страницы изменяются, оптимизируя ее макет для конкретного размера экрана.



Рис. 8.3. Страница в области просмотра среднего размера

Такой размер области просмотра предоставляет вам больше места для работы. Можете задать больший отступ для шапки и hero-изображения, элементы меню

расположить рядом друг с другом на одной строке, так что их уже не обязательно скрывать. А поскольку меню не придется разворачивать, не нужен и значок «гамбургер». Теперь основной контент можно разделить на три колонки равной ширины. Большинство элементов заполняют область просмотра, не доходя 1 em до ее краев.

Макет страницы для большого экрана будет таким же, однако вы увеличите поля по бокам и сделаете него-изображение более крупным. Этот макет показан на рис. 8.4.



Рис. 8.4. Страница в области просмотра большого размера

Поскольку сначала вы создаете мобильную версию, важно представлять, как будут выглядеть макеты для более крупных экранов, ведь это может повлиять на структуру HTML-кода. Создайте новую веб-страницу и таблицу стилей. Подключите таблицу стилей и добавьте листинг 8.1 в HTML-элемент `body`. Разметка выглядит так же, как и при создании неадаптивного дизайна. Я учел несколько нюансов для мобильной версии, о которых расскажу далее.

Листинг 8.1. Разметка страницы для адаптивного дизайна

```
<header id="header" class="page-header">
  <div class="title">
    <h1>Профессиональная техника для обжарщиков</h1>
```

```
<div class="slogan">Мы любим кофе</div>
</div>
</header>
```

```
<nav class="menu" id="main-menu">
  <button class="menu-toggle" id="toggle-menu">
    свернуть меню
  </button>
  <div class="menu-dropdown">
    <ul class="nav-menu">
      <li><a href="/about.html">О нас</a></li>
      <li><a href="/shop.html">Магазин</a></li>
      <li><a href="/menu.html">Меню</a></li>
      <li><a href="/brew.html">Приготовление кофе</a></li>
    </ul>
  </div>
</nav>
```

Добавляет значок-«гамбургер»
для мобильного меню

Главное меню, которое
на мобильных устройствах
по умолчанию будет скрыто

```
<aside id="hero" class="hero">
  Добро пожаловать на сайт нашей компании! Всеми силами
  мы стараемся поддерживать репутацию лучшей компании
  для обжарщиков.
</aside>
```

```
<main id="main">
  <div class="row">
    <section class="column">
      <h2 class="subtitle">Моносорта</h2>
      <p>Мы сотрудничаем с небольшими плантациями
        по всему миру, чтобы собирать зерна в пик сезона.
        Затем мы тщательно поджариваем зерна
        <a href="/batch-size.html">небольшими партиями</a>,
        чтобы максимально раскрыть их потенциал.</p>
    </section>
    <section class="column">
      <h2 class="subtitle">Смеси</h2>
      <p>Наши дегустаторы подобрали тщательно сбалансированные
        смеси на любой вкус. Легендарная смесь
        <a href="/house-blend.html">Вкус компании</a>
        в продаже всегда.</p>
    </section>
    <section class="column">
      <h2 class="subtitle">Приготовление кофе</h2>
      <p>Мы продаем также лучшие кофеварки, френч-прессы
        и перколяторы. Приходите на наши
        <a href="/classes.html">мастер-классы</a>,
        чтобы научиться варить лучший кофе
        в вашей жизни.</p>
    </section>
  </div>
</main>
```

Добавляет
строку и колонки
для средней
и большой
областей просмотра

В этой разметке кнопка открытия и закрытия меню для мобильных экранов находится внутри элемента `nav`. Меню `nav-menu` расположено там, где оно может удовлетворить потребности как мобильной, так и настольной версии. Разметка включает также классы `row` и `column`, позволяющие в будущем создать макет для настольной версии. (Если все это вам неизвестно, ничего страшного.)

Приступим к стилевому форматированию страницы. Сначала вы добавите некоторые из самых простых стилей, задающих параметры шрифта, заголовков и цветов (рис. 8.5). Поскольку сейчас нас больше всего интересуют стили для мобильной версии, уменьшите ширину браузера, чтобы имитировать экран мобильного устройства. Благодаря этому вы получите представление о том, как выглядит страница на маленьком экране.



Рис. 8.5. Применение первого набора стилей

Эти стили приведены в листинге 8.2. Добавьте их в свою таблицу стилей, чтобы настроить размеры блока, шрифт и цвета ссылок. Здесь используется адаптивный размер шрифта, зависящий от размера области просмотра, о котором говорилось в главе 2 (подраздел 2.4.1). Далее определены стили для шапки и основного тела страницы.

Листинг 8.2. Добавление исходных стилей страницы

```
:root {  
    box-sizing: border-box;  
    font-size: calc(1vw + 0.6em);  
}  
  
*,  
*::before,  
*::after {  
    box-sizing: inherit;  
}  
  
body {  
    margin: 0;  
    font-family: Helvetica, Arial, sans-serif;  
}  
  
/*  
 * Ссылки  
 */  
a:link {  
    color: #1476b8;  
    font-weight: bold;  
    text-decoration: none;  
}  
a:visited {  
    color: #1430b8;  
}  
a:hover {  
    text-decoration: underline;  
}  
a:active {  
    color: #b81414;  
}  
  
/*  
 * Шапка  
 */  
.page-header {  
    padding: 0.4em 1em;  
    background-color: #fff;  
}  
  
.title > h1 {  
    color: #333;  
    text-transform: uppercase;  
    font-size: 1.5rem;  
    margin: .2em 0;  
}  
  
.slogan {  
    color: #888;
```

Базовый размер шрифта изменяется
вместе с размером области просмотра

Шапка и название
страницы

```

font-size: 0.875em;
margin: 0;
}

/*
 * Hero-изображение
 */
.hero {
padding: 2em 1em;
text-align: center;
background-image: url(coffee-beans.jpg);
background-size: 100%;
color: #fff;
text-shadow: 0.1em 0.1em 0.3em #000;
}

/*
 * Основной раздел
 */
main {
padding: 1em;
}

.subtitle {
margin-top: 1.5em;
margin-bottom: 1.5em;
font-size: 0.875rem;
text-transform: uppercase;
}

```

Добавление на страницу
hero-изображения

Тень от текста обеспечивает
читаемость светлого текста
на сложном фоне

Основной контент

Эти стили довольно просты. Они преобразуют текст заголовка и подзаголовков в теле страницы в верхний регистр, добавляют поля и отступы, а также настраивают размер шрифтов различных компонентов страницы.

Свойство `text-shadow`, относящееся к hero-изображению, может показаться вам чем-то новым. Оно состоит из нескольких значений, вместе определяющих тень, которую отбрасывает текст. Первые два значения — это декартовы координаты, определяющие сдвиг тени относительно положения текста. Значения `0.1em 0.1em` сдвигают ее чуть вправо и вниз. Третье значение (`0.3em`) определяет степень размытия тени. Наконец, значение `#000` определяет цвет тени.

8.1.1. Создание мобильного меню

Теперь нужно создать самую сложную часть страницы — меню. Сделаем это прямо сейчас. Готовое меню представлено на рис. 8.6.

Написание кода на любом языке часто представляет собой итеративный процесс, и CSS не исключение. При создании меню для этой страницы пришлось учесть некоторые нюансы. Сначала я попытался поместить элемент `pav` в элемент `header`, поскольку хотел, чтобы значок-«гамбургер» располагался именно там. Однако, начав работу с CSS, понял, что эти два элемента должны быть родственными, поскольку

это позволит им естественным образом расположиться в макете настольной версии. Иногда написание правильного HTML-кода требует неоднократного пересмотра некоторых из его разделов.

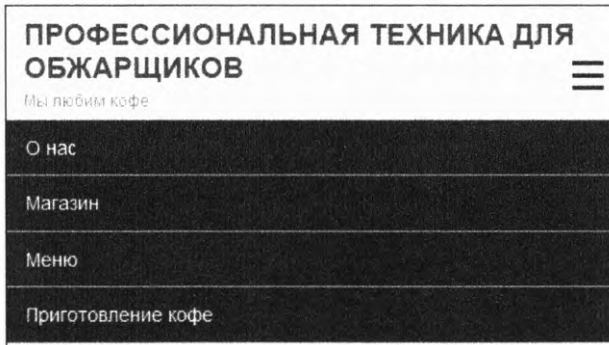


Рис. 8.6. Открытое навигационное меню на экране мобильного устройства

С точки зрения функциональности это меню очень похоже на раскрывающееся меню, созданное в предыдущей главе (см. листинг 7.6). Сначала вы скроете элемент `menu-dropdown`. Затем вместо использования `hover`-эффекта добавите функции JavaScript. Когда пользователь щелкнет на кнопке-переключателе `menu-toggle` или коснется ее, появится раскрывающееся меню. Очередное нажатие этой кнопки скроет меню.

СОВЕТ

Программы экранного доступа задевают в качестве ориентиров определенные элементы HTML5, такие как `form`, `main`, `nav` и `aside`. Это помогает плохо видящим пользователям быстро перемещаться по странице. Важно, чтобы вы поместили кнопку для раскрытия меню внутрь элемента `nav`, чтобы пользователь быстро ее обнаружил. В противном случае он перейдет к элементу `nav` и увидит, что тот пуст (программа экранного доступа игнорирует раскрывающееся меню при использовании свойства `display: none`).

Неоднозначное меню-«гамбургер»

В последние годы меню-«гамбургеры» стали очень популярными. Они позволяют уместить большее количество информации на маленьком экране, однако с ними связаны и некоторые издержки. Было установлено, что скрытие важных элементов, например основного навигационного меню, уменьшает взаимодействие пользователей с этими элементами.

Эти нюансы вам необходимо обсудить со своей командой или дизайнером. Иногда использование такого меню оправданно, а иногда — нет. В любом случае следует ознакомиться с методами создания меню-«гамбургера».

Рассматривая листинг 8.1, обратите внимание на то, что элемент `nav` располагается после элемента `header` в качестве родственного элемента. Это означает, что он займет пространство под шапкой в потоке документа. Теперь нужно сделать кое-что необычное: вы применяете абсолютное позиционирование для перемещения кнопки-переключателя меню вверх, чтобы она отображалась внутри элемента `header`. Добавьте код из листинга 8.3 в свою таблицу для стилизового форматирования меню.

Листинг 8.3. Стили мобильного меню

```
.menu {
  position: relative;
}

.menu-toggle {
  position: absolute;
  top: -1.2em;
  right: 0.1em;

  border: 0;
  background-color: transparent;

  font-size: 3em;
  width: 1em;
  height: 1em;
  line-height: 0.4;
  text-indent: 5em;
  white-space: nowrap;
  overflow: hidden;
}

.menu-toggle::after {
  position: absolute;
  top: 0.2em;
  left: 0.2em;
  display: block;
  content: "\2261";
  text-indent: 0;
}

.menu-dropdown {
  display: none;
  position: absolute;
  right: 0;
  left: 0;
  margin: 0;
}

.menu.is-open .menu-dropdown {
  display: block;
}
```

Создает блок-контейнер для обоих абсолютно позиционированных дочерних элементов

Отрицательное значение свойства `top` перемещает кнопку вверх, за пределы ее блока-контейнера

Переопределяет стили кнопок пользовательского агента

Скрывает текстовое содержимое кнопки и задает ее фиксированный размер

Накладывает поверх кнопки символ Юникода — значок «гамбургер»

Отображает раскрывающееся меню при добавлении в меню класса `is-open`

В этом фрагменте в основном используются уже знакомые вам методы. К меню применено относительное позиционирование с целью задания блока-контейнера для обоих его дочерних элементов: кнопки-переключателя и раскрывающегося меню. Кнопка-переключатель перемещается вверх за счет отрицательного значения свойства `top`, а значение свойства `right` позиционирует ее в правой части экрана. Благодаря этому она отображается в шапке справа от названия страницы.

Затем выполняете трюк с заменой кнопки: ограниченная ширина, большое значение свойства `text-indent` и свойство `overflow: hidden` скрывают текст кнопки-переключателя. Затем в качестве содержимого псевдоэлемента `:after` задаете символ Юникода (`\2261`). Это математический символ, состоящий из трех горизонтальных линий, — значок-«гамбургер». Вместо этого для дополнительной настройки значка примените фоновое изображение в псевдоэлементе.

Если вы не знаете, почему используется тот или иной стиль, преобразуйте его в комментарий и посмотрите, как это повлияет на страницу. Она будет выглядеть немного странно на большом экране, так что уменьшите ширину окна браузера для имитации экрана мобильного устройства.

Класс `is-open` — это еще один трюк. При наличии этого класса конечный селектор (`.menu.is-open .menu-dropdown`) нацелен на раскрывающееся меню. При его отсутствии селектор нацелен на него не будет. Это обеспечивает функциональность меню. На рис. 8.7 показано раскрывающееся меню до применения остальных стилей (обратите внимание на четыре ссылки перед `h2`-изображением слева).



Рис. 8.7. Кнопка-«гамбургер» в действии

Код JavaScript в листинге 8.4 добавляет и удаляет класс `is-open` при нажатии кнопки-переключателя. Добавьте его на свою страницу перед закрывающим тегом `</body>`.

Листинг 8.4. JavaScript для обеспечения функциональности раскрывающегося меню

```

<script type="text/javascript">
(function() {
  var button = document.getElementById('toggle-menu');
  button.addEventListener('click', function(event) {
    event.preventDefault();
    var menu = document.getElementById('main-menu');
    menu.classList.toggle('is-open');
  });
})();
</script>

```

Обработчик события щелчка (срабатывает также в результате прикосновения к сенсорному экрану)

Переключает класс is-open меню

Теперь щелчок на значке-«гамбургере» должен приводить к раскрытию меню. Вы увидите его текст поверх контента. Щелкните на значке-«гамбургере» еще раз, чтобы закрыть меню. Таким образом, код CSS будет показывать и скрывать правильные элементы, коду JavaScript достаточно изменить лишь одно имя класса.

Сейчас, когда раскрывающееся меню работает как надо, применим стили к элементу `nav-menu`. Добавьте код из листинга 8.5 в свою таблицу стилей.

Листинг 8.5. Стилизовое форматирование навигационного меню

```

.nav-menu {
  margin: 0;
  padding-left: 0;
  border: 1px solid #ccc;
  list-style: none;
  background-color: #000;
  color: #fff;
}

.nav-menu > li + li {
  border-top: 1px solid #ccc;
}

.nav-menu > li > a {
  display: block;
  padding: 0.8em 1em;
  color: #fff;
  font-weight: normal;
}

```

Добавление границы между пунктами меню

Добавление отступа, обеспечивающего достаточно места для совершения щелчка

И здесь нет ничего нового. Поскольку меню представляет собой список (``), вы переопределяете левый отступ пользовательского агента и удаляете маркеры списка. Комбинатор смежных элементов нацеливается на все пункты меню, кроме первого, добавляя границы между ними.

В данном случае важно обратить внимание на отступ между ссылками в меню. Вы разрабатываете дизайн для мобильных устройств, которые обычно имеют сен-

сорный экран. Ключевые чувствительные к прикосновению области должны быть довольно большими, чтобы их легко было коснуться пальцем.

СОВЕТ


При разработке дизайна для мобильных устройств с сенсорным экраном делайте все основные элементы управления настолько большими, чтобы их легко было коснуться пальцем. Не заставляйте пользователей увеличивать масштаб, чтобы точно нажать маленькую кнопку или ссылку.

8.1.2. Добавление метатега viewport

Мобильная версия макета готова, однако в ней отсутствует одна важная деталь — *метатега viewport*. Этот HTML-элемент сообщает мобильным устройствам о том, что вы разработали специальный макет для небольших экранов. Без него мобильный браузер предположит, что страница не адаптивная, и попытается имитировать работу настольного браузера. И все ваши усилия по разработке мобильного дизайна окажутся бессмысленными. Мы этого не хотим. Обновите элемент `head` в HTML-коде, добавив в него метатег, как показано в листинге 8.6.

Листинг 8.6. Добавление метатега viewport для обеспечения адаптивности

```
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
    content="width=device-width, initial-scale=1">
  <title>Профессиональная техника для обжарщиков</title>
  <link href="styles.css" />
</head>
```



Атрибут метатега `content` делает две вещи. Во-первых, он дает браузеру команду использовать ширину экрана устройства при интерпретации кода CSS вместо того, чтобы имитировать работу настольного браузера. Во-вторых, задействует свойство `initial-scale` для задания 100 % масштаба при загрузке страницы.

СОВЕТ

Инструменты разработчика (DevTools) современных браузеров позволяют эмулировать мобильный браузер, включая меньший размер области просмотра и поведение метатега viewport. Они очень полезны для тестирования адаптивного дизайна. Более подробную информацию вы найдете по адресу developers.google.com/web/tools/chrome-devtools/device-mode/ (Chrome) или developer.mozilla.org/en-US/docs/Tools/Responsive_Design_Mode (Firefox).

Существуют и другие параметры, однако вам, скорее всего, подойдут те, что приведены здесь. Например, вы можете явно задать значение `width=320`, чтобы

браузер использовал область просмотра шириной 320 пикселей. Однако этот подход не предпочтительный, поскольку у мобильных устройств широкий диапазон размеров экрана. Свойство `device-width` позволяет отображать контент в самом подходящем формате.

Третий параметр, который разработчики обычно добавляют в атрибут `content`, — это `user-scalable=no`. Он запрещает пользователю масштабировать изображение на экране мобильного устройства двумя пальцами. Его применение, как правило, считается плохой практикой, и я не рекомендую вам это делать. Если ссылка слишком мала, чтобы на ней можно было щелкнуть, или если пользователь захочет лучше рассмотреть изображение, этот параметр не позволит ему воспользоваться функцией масштабирования.

Дополнительную информацию о метатэге `viewport` вы найдете по адресу developer.mozilla.org/en-US/docs/Mozilla/Mobile/Viewport_meta_tag.

8.2. Медиазапросы

Вторым компонентом адаптивного дизайна является использование медиазапросов. *Медиазапросы* позволяют вам написать набор стилей, которые применяются к странице только при определенных условиях. Это позволяет по-разному настраивать стили, исходя из размера экрана. Вы можете определить один набор стилей для маленьких устройств, второй — для устройств среднего размера и третий — для устройств с большим экраном, чтобы по-разному представить те или иные части страницы.

Медиазапросы задействуют правило `@media` для нацеливания на устройства, обладающие определенными особенностями. Простейший медиазапрос выглядит следующим образом:

```
@media (min-width: 560px) {  
  .title > h1 {  
    font-size: 2.25rem;  
  }  
}
```

Любые наборы правил могут быть заключены в фигурные скобки. Правило `@media` — это проверка условия, которое должно оказаться истинным для того, чтобы к странице был применен любой из этих стилей. В данном случае браузер проверяет, выполняется ли условие `min-width: 560px`. Отступ будет задан для элемента `page-header` только в том случае, если ширина области просмотра устройства пользователя — 560 пикселей или больше. Если это не так, содержащиеся внутри правила будут проигнорированы.

Правила внутри медиазапроса подчиняются обычным правилам каскадности. Они могут переопределять правила, находящиеся за пределами медиазапроса (на основании специфичности селекторов или исходного порядка), или переопре-

делиться этими правилами. Сам медиазапрос не влияет на специфичность содержащихся внутри селекторов.

ВНИМАНИЕ!

Вы должны использовать единицы `em` для задания контрольных точек медиазапросов. Это единственная единица, которая обеспечивает одинаковый результат во всех основных браузерах, когда пользователь масштабирует страницу или изменяет размер шрифта, применяемый по умолчанию. В Safari контрольные точки, заданные с помощью единиц `rem` и пикселей, работают менее надежно. Кроме того, единицы `em` масштабируются вместе с изменением размера пользовательского шрифта по умолчанию, обычно это предпочтительно.

В данном примере я использовал пиксели, однако в медиазапросах лучше применять единицы `em`, исходя из размера шрифта браузера по умолчанию (обычно он равен 16 пикселям). Вместо 560 px следует задать значение 35 `em` ($560 / 16$).

Найдите стили `.title` в своей таблице стилей и добавьте медиазапрос из листинга 8.7, чтобы обеспечить адаптивность шапки страницы.

Листинг 8.7. Добавление контрольной точки в стили заголовка страницы

```
.title > h1 {  
  color: #333;  
  text-transform: uppercase;  
  font-size: 1.5rem;  
  margin: .2em 0;  
}  
  
@media (min-width: 35em) {  
  .title > h1 {  
    font-size: 2.25rem;  
  }  
}
```

Нацеливает контрольные точки на значение 35 `em` и выше

Переопределяет размер шрифта мобильного устройства (1,5 `rem`), увеличивая его

Теперь для заголовка заданы два размера шрифта, используемых в зависимости от размера области просмотра. Значение 1,5 `rem` будет применяться для области просмотра менее 35 `em`, а 2,25 `rem` — для более крупных экранов.

Можете протестировать эти стили, изменив ширину окна браузера. Сделайте его узким — и увидите более мелкий заголовок, предназначенный для мобильных устройств. Затем медленно расширьте окно браузера. Размер шрифта начнет плавно изменяться благодаря заданному для страницы адаптивному размеру шрифта (`calc()`) (см. листинг 8.2). Как только ширина экрана достигнет 35 `em` (или 560 px), размер шрифта заголовка переключится на версию 2,25 `rem`.

Точка, в которой ширина окна равна 560 пикселям, называется *контрольной точкой* (*breakpoint*). Чаще всего вы будете использовать одни и те же контрольные точки в разных медиазапросах в своей таблице стилей. О том, как выбирать эти точки, поговорим в следующей главе.

8.2.1. Типы медиазапросов

Можно дополнительно уточнить медиазапрос, объединив два условия с помощью ключевого слова `and`:

```
@media (min-width: 20em) and (max-width: 35em) { ... }
```

Этот комбинированный медиазапрос нацелен только на устройства, соответствующие обоим критериям. Если хотите, чтобы медиазапрос был нацелен на один из нескольких критериев, используйте запятую:

```
@media (max-width: 20em), (min-width: 35em) { ... }
```

Пример ориентирован на области просмотра как шириной 20 em и уже, так и размером 35 em и шире.

Медиафункции `min-width`, `max-width` и не только

В листингах вы использовали *медиафункцию* `min-width`, нацеливающуюся на устройства с областью просмотра, ширина которой превышает заданное значение, и медиафункцию `max-width` — для устройств с областью просмотра, ширина которой меньше его.

Медиафункции `min-width` и `max-width` вы будете применять чаще всего. Однако можете воспользоваться и другими типами медиафункций. Вот некоторые из них:

- ❑ `(min-height: 20em)` — нацеливается на области просмотра высотой 20 em и более;
- ❑ `(max-height: 20em)` — нацеливается на области просмотра высотой 20 em и менее;
- ❑ `(orientation: landscape)` — нацеливается на области просмотра, ширина которых превышает их высоту;
- ❑ `(orientation: portrait)` — нацеливается на области просмотра, высота которых превышает их ширину;
- ❑ `(min-resolution: 2dppx)` — нацеливается на устройства с разрешением экрана 2 точки на пиксел или выше, ориентируется на экраны высокой четкости (*retina display*);
- ❑ `(max-resolution: 2dppx)` — нацеливается на устройства с разрешением экрана до 2 точек на пиксел.

Полный список медиафункций вы найдете по адресу developer.mozilla.org/en-US/docs/Web/CSS/@media.

С медиазапросами, основанными на разрешении экрана, иногда возникают сложности, так как их поддержка внедрена в браузеры относительно недавно. Некоторые браузеры имеют ограниченную поддержку и/или требуют использования проприетарного синтаксиса. Например, IE9-11 и Opera Mini не поддерживают единицу `dppx`, поэтому вместо нее придется применять `dpi` (точки на дюйм) (например, 192 dpi вместо 2 dppx). Safari и iOS Safari поддерживают проприетарную медиафункцию `-webkit-min-device-pixel-ratio`. Короче говоря, лучший способ ориентации на экран высокой четкости — это следующая комбинация:

```
@media (-webkit-min-device-pixel-ratio: 2),
(min-resolution: 192dpi) { ... }
```

Данный подход работает во всех современных браузерах. Используйте его, когда есть смысл поместить на экраны изображения или значки с высоким разрешением. Так пользователи экранов с низким разрешением не будут расходовать трафик на загрузку более крупных изображений — разницу они все равно не смогут оценить. Об адаптивных изображениях более подробно поговорим далее в этой главе.

СОВЕТ

Вы можете поместить медиазапрос в элемент `link`. Добавление на страницу `<link rel="stylesheet" media="(min-width: 45em)" href="large-screen.css"/>` обеспечит применение содержимого файла `large-screen.css` к странице только в том случае, если условие медиазапроса `min-width` окажется истинным. Обратите внимание на то, что таблица стилей будет загружаться вне зависимости от ширины области просмотра, поэтому данная тактика предназначена лишь для организации кода, а не для экономии трафика.

Типы носителей

Последней особенностью медиазапросов является возможность нацелиться на *типы носителей*. Как правило, нужно будет думать о двух типах носителей — `screen` и `print`. Использование медиазапроса `print` позволяет контролировать то, как будет выглядеть страница, если пользователь решит ее распечатать. В этом случае можете удалить фоновые изображения (для экономии чернил), а также скрыть лишние навигационные элементы. Когда пользователь распечатывает страницу, ему обычно требуется только ее текстовое содержимое.

Чтобы написать стили, которые применяются только при печати, возьмите запрос `@media print`. Скобки не нужны, как и в случае с `min-width` и другими медиафункциями. Для ориентации только на отображение на экране используйте запрос `@media screen`.

Стили для печати

Когда речь идет о разработке CSS, о стилях для печати вспоминают в последнюю очередь, если вообще вспоминают. Однако не лишним будет обдумать вероятность того, что пользователи могут решить распечатать какие-то из ваших страниц. Для упрощения процесса печати существуют приемы, которыми следует воспользоваться. В большинстве случаев можно применить базовые стили для печати внутри медиазапроса `@media print { ... }`.

Используйте свойство `display: none`, чтобы скрыть такие несущественные части страницы, как навигационные меню и подвал. Если пользователь печатает страницу, его, скорее всего, интересует только основное ее содержимое.

Также можете глобально изменить цвета шрифтов на черный и удалить все фоновые изображения, расположенные позади блоков текста, и цвета подложек. Во многих случаях с этой задачей хорошо справляется универсальный селектор.

Здесь я использую `!important`, поэтому не стану беспокоиться о его переопределении в связи с каскадностью:

```
@media print {
  * {
    color: black !important;
    background: none !important;
  }
}
```

То, что вы потратите немного времени на работу со стилями для печати, может существенно помочь пользователям. Если вы работаете над сайтом, контент которого часто будет распечатываться, например над сайтом с кулинарными рецептами, убедитесь в том, что все печатается корректно.

8.2.2. Добавление контрольных точек на страницу

С практической точки зрения подход Mobile First предполагает, что чаще всего будет использоваться медиазапрос `min-width`. Сначала вы напишете свои мобильные стили вне каких-либо медиазапросов. Затем станете продвигаться к следующим контрольным точкам. Общая структура показана в листинге 8.8 (пока не нужно добавлять этот код на свою страницу).

Листинг 8.8. Общая структура адаптивного CSS-кода

```
.title {
  ...
}

@media (min-width: 35em) {
  .title {
    ...
  }
}

@media (min-width: 50em) {
  .title {
    ...
  }
}
```

← Мобильные стили. Применяются ко всем контрольным точкам

← Средняя контрольная точка. Переопределяет выбранные мобильные стили

← Большая контрольная точка. Переопределяет стили, выбранные для маленькой и средней контрольных точек

Мобильные стили идут первыми. Поскольку они находятся за пределами любых медиазапросов, то задаваемые ими правила применяются ко всем контрольным точкам. Затем следует медиазапрос для нацеливания на средние экраны с правилами, которые переопределяют и используют мобильные стили. Последним идет медиазапрос для нацеливания на большие экраны, в котором добавляется последний слой.

В зависимости от дизайна у вас имеются одна или несколько контрольных точек. Если на странице много элементов, может не понадобиться добавлять стили

для каждой контрольной точки: правила, предусмотренные для малых или средних контрольных точек, могут оказаться актуальными и для больших.

Иногда стили, относящиеся только к мобильным устройствам, оказываются сложными. Переопределение этих правил для большей контрольной точки может быть очень утомительным. В таком случае имеет смысл поместить эти стили внутрь медиазапроса `max-width`, чтобы они применялись только к меньшей контрольной точке. Однако слишком большое количество медиазапросов `max-width` способно сказать о том, что вы не придерживались принципов подхода Mobile First. Они должны быть исключением, а не правилом.

Добавим остальные стили для средней контрольной точки. На большом экране места для работы больше, поэтому можете увеличить интервалы. В листинге 8.9 вы зададите больший отступ между шапкой и основными элементами. Затем установите еще больший отступ для hero-изображения, чтобы сделать страницу визуально более интересной. Теперь не нужно прятать навигационное меню, поэтому вы обеспечите его постоянную видимость и скроете значок-«гамбургер» (листинг 8.10). Наконец, организуйте контент в три колонки (листинг 8.11). После этого страница будет выглядеть как на рис. 8.8.

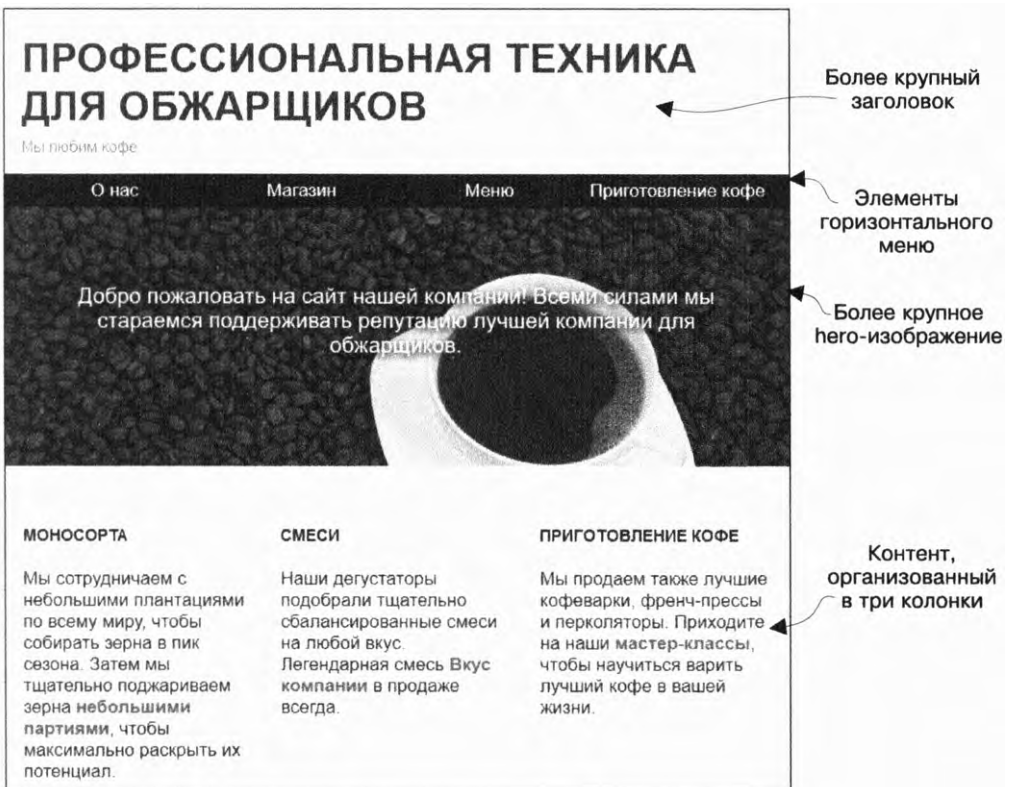


Рис. 8.8. Вид страницы, соответствующий средней контрольной точке

Некоторые из этих изменений, например, незначительное увеличение отступов или размера шрифта, довольно просты. Как правило, лучше всего добавлять очередное изменение сразу после правил для соответствующих селекторов. Для простоты я объединил их в листинг 8.9. Добавьте их в свою таблицу стилей.

Листинг 8.9. Корректировка отступа и шрифта для средней контрольной точки

```
.page-header {
  padding: 0.4em 1em;
  background-color: #fff;
}

@media (min-width: 35em) {
  .page-header {
    padding: 1em;
  }
}

.hero {
  padding: 2em 1em;
  text-align: center;
  background-image: url(coffee-beans.jpg);
  background-size: 100%;
  color: #fff;
  text-shadow: 0.1em 0.1em 0.3em #000;
}

@media (min-width: 35em) {
  .hero {
    padding: 5em 3em;
    font-size: 1.2rem;
  }
}

main {
  padding: 1em;
}

@media (min-width: 35em) {
  main {
    padding: 2em 1em;
  }
}
```

Увеличивает отступ шапки

Увеличивает отступ hero-изображения и размер шрифта

Увеличивает отступ основного контента

Всегда следите за тем, чтобы медиазапрос следовал за стилями, которые он переопределяет, чтобы стили внутри него имели приоритет. Увеличьте ширину окна браузера, чтобы увидеть, как изменится страница, когда она превысит 35 em.

Теперь поработаем над меню. Нужно, во-первых, удалить функции открытия и закрытия раскрывающегося меню, чтобы обеспечить его постоянную видимость,

во-вторых, изменить вертикальное расположение пунктов меню на горизонтальное. Все это мы сейчас сделаем. Добавьте следующие блоки медиазапросов в таблицу стилей после уже имеющихся стилей для `.menu` и `.nav-menu`.

Листинг 8.10. Реструктуризация навигационного меню для средней контрольной точки

```
@media (min-width: 35em) {  
  .menu-toggle {  
    display: none;  
  }  
  
  .menu-dropdown {  
    display: block;  
    position: static;  
  }  
}  
  
@media (min-width: 35em) {  
  .nav-menu {  
    display: flex;  
    border: 0;  
    padding: 0 1em;  
  }  
  
  .nav-menu > li {  
    flex: 1;  
  }  
  
  .nav-menu > li + li {  
    border: 0;  
  }  
  
  .nav-menu > li > a {  
    padding: 0.3em;  
    text-align: center;  
  }  
}
```

Скрывает кнопку-переключатель
и отображает содержимое
раскрывающегося меню

Переопределяет
абсолютное позиционирование

Преобразует меню во flex-контейнер
и позволяет элементам
увеличиваться в размере,
заполняя ширину экрана

Несмотря на то что меню имело множество сложных стилей, обеспечивавших работоспособность мобильного макета, не так уж трудно переопределить их и преобразовать макет обратно в статичный блочный вариант. Не нужно переопределять свойства `top`, `left` и `right` мобильных стилей, поскольку при статическом позиционировании они уже не играют роли.

В данном случае flex-контейнер отлично подходит для элементов меню: они будут увеличиваться, заполняя доступную ширину. Вы также скорректировали отступы для элементов меню, как и для других элементов, только на этот раз их уменьшили. Работая над дизайном для средней контрольной точки, вы можете предположить, что пользователь заходит на страницу не с маленького телефона, поэтому нет необходимости создавать крупные чувствительные к щелчкам кнопкой мыши области.

8.2.3. Добавление адаптивных колонок

Последнее изменение для средней контрольной точки предполагает создание нескольких колонок. Этот процесс аналогичен построению многоколоночных макетов, которым вы занимались в предыдущих главах. Нужно просто обернуть эти колонки в медиазапрос, чтобы они не применялись до определенной контрольной точки.

При написании разметки вы использовали классы `row` и `column`, подготавливая почву для трехколоночного макета. Теперь определим для них стили. Добавьте код из листинга 8.11 в таблицу стилей.

Листинг 8.11. Трехколоночный макет внутри медиазапроса

```
@media (min-width: 35em) {
  .row {
    display: flex;
    margin-left: -.75em;
    margin-right: -.75em;
  }
  .column {
    flex: 1;
    margin-right: 0.75em;
    margin-left: 0.75em;
  }
}
```

Использует flex-контейнер для создания колонок одинаковой ширины

Использует отрицательные значения полей для расширения строк и компенсации полей колонок (см. подраздел 4.5.2)

Задаёт величину зазора между колонками

Теперь измените размер окна браузера, чтобы увидеть, как появляются колонки при достижении контрольной точки. До этого к данным элементам не применяются никакие конкретные стили, поэтому их расположение подчиняется обычному потоку документа. После прохождения контрольной точки они превращаются во flex-контейнеры с flex-элементами.

По большей части адаптивный дизайн сводится к использованию следующего подхода: если макет предполагает расположение элементов рядом друг с другом, ставьте их бок о бок только на больших экранах. На меньших экранах позволяйте им занимать отдельную строку и заполнять всю ширину экрана. Этот метод может применяться к колонкам, медиаобъектам и другим элементам, которым бывает тесновато на узком экране. Вам интересно, как я пришел к контрольной точке 35 em в листинге 8.7? Я выбрал ее потому, что до ее достижения три колонки кажутся переполненными. В данном случае до точки 35 em колонки были слишком узкими.

Веб-дизайнер Брэд Фрост (Brad Frost) составил список адаптивных шаблонов, который находится по адресу bradfrost.github.io/this-is-responsive/patterns.html. Вы можете создать адаптивный дизайн с различными комбинациями колонок одинаковой или разной ширины. В итоге все сводится к тем или иным вариациям описанного подхода.

Иногда вам даже не понадобятся медиазапросы, так как обо всем позаботится обычное обертывание строк. Для этого используются макеты, основанные на flex-контейнерах со свойствами `flex-wrap: wrap` и `flex-basis`. Аналогично макет-сетка со свойствами `auto-fit` или `auto-fill` будет определять, сколько элементов должно уместиться в строке перед их переносом на новую. Вы также можете задействовать строчно-блочные элементы, хотя в этом случае они не будут увеличиваться, заполняя ширину контейнера.

Выбор контрольных точек

В начале главы я познакомил вас с наиболее простыми адаптивными элементами страницы, чтобы вы привыкли к использованию медиазапросов. В большинстве случаев первым делом вы будете устанавливать контрольные точки для частей макета, состоящих из нескольких колонок. Протестируйте несколько контрольных точек, пока не найдете наиболее подходящую. Убедитесь в том, что после контрольной точки колонки не слишком узкие.

Очень легко погрязнуть в размышлениях о конкретных устройствах: ширина экрана iPhone 7 составляет столько-то пикселей, ширина экрана некоего планшета — столько-то... Постарайтесь не беспокоиться об этом. Существуют сотни устройств с различным разрешением, вы никогда не учтете их все. Выбирайте контрольные точки, которые имеют смысл в вашем конкретном случае, тогда дизайн будет хорошо работать вне зависимости от устройства пользователя.

Разыскиваются: контейнерные запросы

Медиазапросы создают адаптивные макеты, основываясь на размере области просмотра, однако на протяжении нескольких лет разработчики и создатели браузеров пытались найти для этого лучший способ. Многие разработчики хотели бы иметь такую функцию, как *контейнерные запросы* (первоначально называемые *запросами элемента*).

Вместо того чтобы реагировать на размер области просмотра, такой запрос позволил бы стилям реагировать на размер контейнера элемента. Рассмотрим медиаобъект, созданный в главе 4.



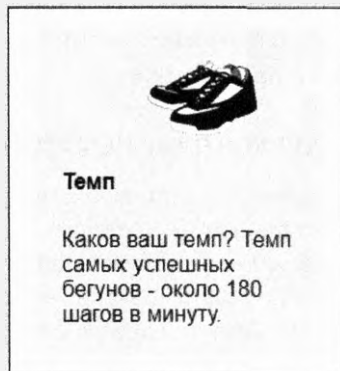
Темп

Каков ваш темп? Темп самых успешных бегунов - около 180 шагов в минуту

Медиаобъект с изображением и текстом, расположенными бок о бок

Это сработает на большом экране, позволяющем расположить изображение и блок текста рядом друг с другом. Однако в мобильном макете шаблон такого типа часто разделяется, в результате чего текст располагается под изображением.

Иногда вам нужно будет создать такой мобильный макет для большой контрольной точки. Подумайте, что произойдет, если поместить этот медиаобъект внутри узкой колонки (вроде тех, которые описаны в листинге 8.11): контейнер окажется слишком узким, чтобы элементы макета могли расположиться бок о бок, даже выше контрольной точки. Вместо этого более уместным было бы определить адаптивное поведение медиаобъекта, исходя из ширины не области просмотра, а его контейнера. К сожалению, это невозможно сделать напрямую. Единственный способ создания такого макета предполагает использование тщательно сконструированных селекторов потомков, которые учитывали бы такой сценарий (например, `.column .media > .media-image`), однако этот подход может оказаться не очень надежным.



Такая версия медиаобъекта может быть более уместной при ограниченной ширине экрана

Следите за появлением контейнерных запросов. Их трудно реализовать в браузере, поэтому такая функция еще не появилась, однако спрос на них очень велик. Будем надеяться, что в ближайшие годы в браузерах будут реализованы контейнерные запросы или что-то, что позволяет достичь тех же результатов.

8.3. Резиновые макеты

Третьим и последним принципом адаптивного дизайна является *резиновый макет*. Резиновый макет (иногда называемый также *жидким макетом*) предполагает использование контейнеров, размер которых изменяется в зависимости от ширины области просмотра. Такой подход противоположен созданию фиксированного макета, в котором размер колонок задается в пикселах или единицах em. Фиксированный контейнер (например, со свойством `width: 800px`) переполнит область просмотра на экранах более мелких устройств, создав необходимость в горизонтальной прокрутке. Резиновый контейнер в такой ситуации автоматически сжимается.

В резиновом макете для основного контейнера страницы ширина обычно не задается явно или определяется с использованием процентных значений. Тем не менее он может иметь левый и правый отступы или автоматически заданные левое и правое поля, создающие расстояние между его краями и краями области просмотра. Это означает, что он может быть несколько уже, чем область просмотра, но никогда не шире.

Внутри основного контейнера (-ов) любые колонки определяются с помощью процентных значений (например, ширина главной колонки равна 70 %, а боковой — 30 %). В таком случае контейнеры поместятся на экране вне зависимости от его ширины. Макет из flex-контейнеров тоже сработает при условии, что для flex-элементов будут заданы значения `flex-grow` и, что более важно, `flex-shrink`, которые позволяют элементам вписаться в экран вне зависимости от его ширины. Выработайте привычку размышлять о ширине контейнеров в процентных, а не в числовых значениях.

ПРИМЕЧАНИЕ

Веб-страница, описанная в данной главе, — это не первый пример использования резиновых контейнеров. На самом деле в этой книге я работал почти исключительно с резиновыми макетами. Сейчас вы уже должны иметь хорошее представление об этом подходе.

Веб-страница адаптивна по умолчанию. До применения CSS ширина блочных элементов не превышает ширину области просмотра, а строковые элементы переносятся на следующую строку, предотвращая переполнение горизонтального пространства. При добавлении стилей вы сами должны обеспечивать адаптивное поведение страницы. Иногда это легче сказать, чем сделать, однако мне помогает мысль о том, что я всегда начинаю с правильной отправной точки.

8.3.1. Добавление стилей для большой области просмотра

Теперь добавим еще несколько медиазапросов для следующей контрольной точки. Обратите внимание на то, что вы никогда не задаете фиксированную ширину контейнеров для контрольных точек, а позволяете им увеличиваться до 100%-ного размера (минус отступы и/или поля). А в случае с трехколоночными фрагментами страницы используете flex-контейнер, чтобы позволить ширине колонок достигать 1/3 ширины области просмотра.

Готовый макет страницы для большой области просмотра показан на рис. 8.9. Он похож на макет для средней области просмотра, однако предусматривает гораздо больше места для работы! Вы можете довольно свободно выбирать отступы и именно этим сейчас займетесь.

Отступы вдоль левой и правой сторон были увеличены с 1 до 4 em. Увеличены отступы со всех сторон текстового блока на hero-изображении, что также сделало изображение более крупным. Дополнительные стили показаны в листинге 8.12.

Добавьте все блоки медиазапросов (`min-width: 50em`) в свою таблицу стилей. Убедитесь в том, что они стоят после эквивалентных правил для меньших контрольных точек (для `.page-header`, `.hero` и `main`), чтобы стили в этих медиазапросах могли их переопределить.

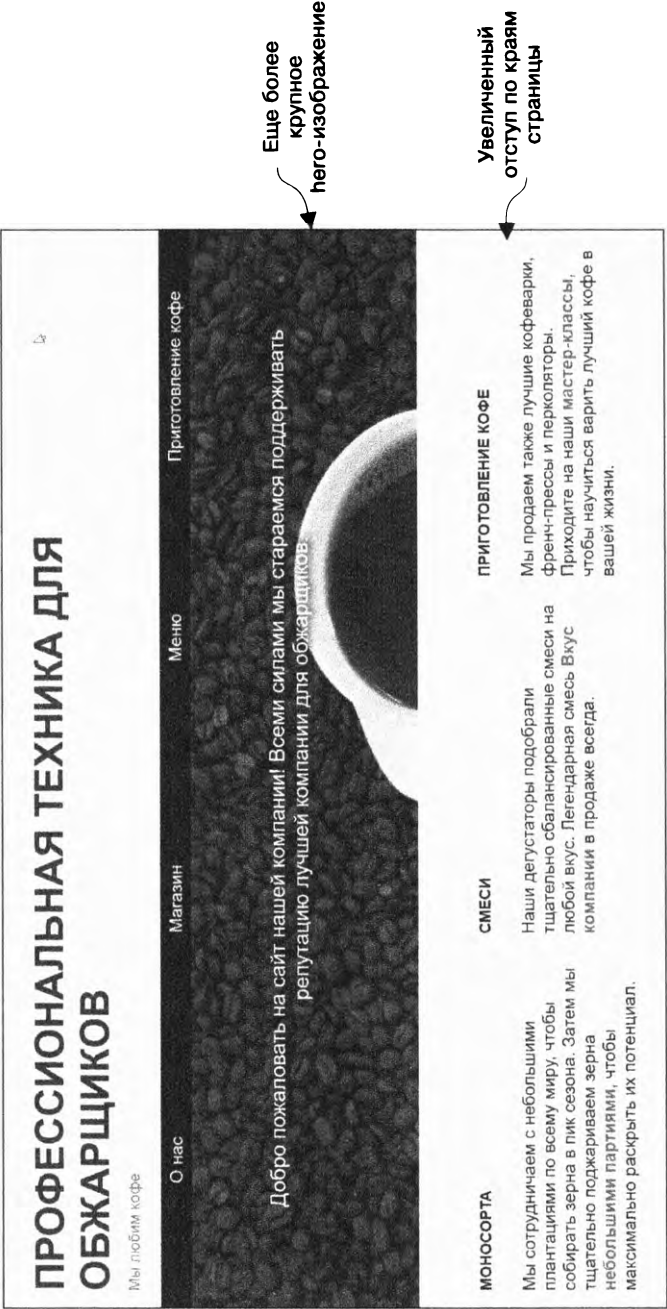


Рис. 8.9. Макет страницы для большой области проомотра

Листинг 8.12. Увеличение отступов для больших контрольных точек

```

@media (min-width: 50em) {
  .page-header {
    padding: 1em 4em;
  }
}

@media (min-width: 50em) {
  .hero {
    padding: 7em 6em;
  }
}

@media (min-width: 50em) {
  main {
    padding: 2em 4em;
  }
}

@media (min-width: 50em) {
  .nav-menu {
    padding: 0 4em;
  }
}

```

Увеличивает левый и правый отступы по краям страницы до 4 em

Увеличивает отступы со всех сторон текста на hero-изображении, что делает изображение более крупным

Осталось выполнить последнюю корректировку. Вы задали адаптивный размер шрифта в корневом элементе — `font-size: calc(1vw + 0.6em)`. На больших экранах он кажется слишком крупным. Чтобы исправить это, зафиксируйте максимальный размер шрифта в самой большой контрольной точке. Обновите таблицу стилей, добавив в нее код из листинга 8.13.

Листинг 8.13. Ограничение максимального размера адаптивного шрифта

```

:root {
  box-sizing: border-box;
  font-size: calc(1vw + 0.6em);
}

@media (min-width: 50em) {
  :root {
    font-size: 1.125em;
  }
}

```

Фиксирует размер шрифта в самой высокой контрольной точке

Теперь у вас есть адаптивная страница с тремя контрольными точками. Можете поэкспериментировать с ней. Измените ширину окна браузера и посмотрите, как это повлияет на внешний вид страницы.

8.3.2. Работа с таблицами

При создании резиновых макетов для экранов мобильных устройств наибольшие проблемы возникают с таблицами. Если число колонок в таблице больше определенного значения, ее ширина может легко превысить ширину экрана (рис. 8.10).

Страна	Регион/Плантация	Нотки	Це
Никарагуа	Матагальпа	Темный шоколад, миндаль	934,
Эфиопия	Йиргачефе	Холодный чай, черника	1068
Эфиопия	Нано Чалта	Мандарин, жасмин	1001

Рис. 8.10. На экране мобильного устройства правый край таблицы обрезан

Старайтесь найти другой способ организации данных для пользователей мобильных устройств. Например, можете перенести данные из каждой строки в отдельный блок, а затем позволить блокам выстроиться друг под другом. Или создать график либо диаграмму, которые хорошо вписываются в небольшие области просмотра. Тем не менее иногда без таблицы не обойтись.

Один из подходов заключается в том, чтобы заставить таблицу отображаться в виде обычных блочных элементов (рис. 8.11).

Никарагуа
Матагальпа
Темный шоколад, миндаль
934,00 ₺
Эфиопия
Йиргачефе
Холодный чай, черника
1068,00 ₺
Эфиопия
Нано Чалта
Мандарин, жасмин
1001,00 ₺

Рис. 8.11. Табличные данные с примененным ко всем строкам и ячейкам свойством display: block

Этот макет состоит из элементов `table`, `tr` и `td`, однако свойство `display: block` переопределяет их обычные значения `table`, `table-row` и `table-cell`. Можете использовать медиазапрос `max-width`, чтобы эти изменения касались только маленьких экранов. CSS-код для этой таблицы приведен в листинге 8.14 (примените его к любой табличной разметке, чтобы оценить результат).

Листинг 8.14. Обеспечение адаптивного поведения таблицы на экране мобильного устройства

```
table {  
    width: 100%;  
}  
  
@media (max-width: 30em) {  
    table, thead, tbody, tr, th, td {  
        display: block;  
    }  
  
    thead tr {  
        position: absolute;  
        top: -9999px;  
        left: -9999px;  
    }  
  
    tr {  
        margin-bottom: 1em;  
    }  
}
```

Превращает все элементы
таблицы в блоки

Скрывает строку заголовка,
сдвигая ее за край экрана

Добавляет небольшое пространство
между всеми наборами табличных данных

Этот код заставляет ячейки выстраиваться друг под другом, а затем добавляет расстояние между всеми элементами `tr`. При таком подходе строка `<thead>` больше не выравнивается в соответствии с расположенными под ней колонками, поэтому я использую абсолютное позиционирование, чтобы сдвинуть строку заголовка за край экрана. Я избегаю свойства `display: none` из соображений доступности: хочу, чтобы заголовки оставались доступными программам экранного доступа. Это, конечно же, не идеальное решение, но, когда все остальное оказывается неэффективным, оно может стать наилучшим вариантом.

8.4. Адаптивные изображения

При разработке адаптивного дизайна изображениям следует уделять особое внимание. Вам нужно не только уместить их на экране, но и учесть ограниченную пропускную способность мобильных устройств. Изображения, как правило, одни из самых долго загружаемых элементов страницы. Всегда следите за тем, чтобы изображения были хорошо сжаты. Используйте в графическом редакторе параметр **Сохранить для веб**, который позволяет уменьшить размер файла изображения, или другой инструмент для сжатия изображений, например tinypng.com.

Также убедитесь в том, что разрешение изображения не превышает оптимального значения. Это оптимальное значение зависит от размера области просмотра. Не стоит создавать большой файл для более мелких экранов, поскольку он все равно будет уменьшен.

8.4.1. Использование нескольких изображений для экранов разных размеров

Лучшей практикой считается создание нескольких копий изображения с разным разрешением. Если, исходя из медиазапросов, вы знаете, что экран определенного размера, то нет никакого смысла посылать чрезвычайно большое изображение: браузеру придется уменьшить его, чтобы оно вписалось в экран.

Задействуйте адаптивные методы для предоставления изображения в соответствии с размером экрана пользователя. В случае с hero-изображением на вашей странице это обеспечивается CSS-кодом, приведенным в листинге 8.15. Добавьте этот код в свою таблицу стилей.

Листинг 8.15. Добавление адаптивного фонового изображения

```
.hero {
  padding: 2em 1em;
  text-align: center;
  background-image: url(coffee-beans-small.jpg);
  background-size: 100%;
  color: #fff;
  text-shadow: 0.1em 0.1em 0.3em #000;
}

@media (min-width: 35em) {
  .hero {
    padding: 5em 3em;
    font-size: 1.2rem;
    background-image: url(coffee-beans-medium.jpg);
  }
}

@media (min-width: 50em) {
  .hero {
    padding: 7em 6em;
    background-image: url(coffee-beans.jpg);
  }
}
```

Помещает самое маленькое изображение на экран мобильного устройства

Помещает более крупное изображение на экран среднего размера

Помещает изображение с максимальным разрешением на крупный экран

Если вы просмотрите этот код в своем браузере, то не заметите никакой разницы. В этом-то и заключается вся суть. При небольшой контрольной точке ширина экрана в любом случае недостаточно велика для изображения с максимальным разрешением. Однако вы загрузили на десятки или даже сотни килобайт меньше, чем пришлось бы при загрузке изображения с полным разрешением. Если страница заполнена графикой, это может существенно сократить время ее загрузки.

8.4.2. Использование атрибута srcset для передачи нужного изображения

Медиазапросы решают проблему, когда изображение внедряется в страницу с помощью CSS-кода, но что, если изображения добавляются с помощью HTML-элемента `img`? Для встроенных изображений требуется другой подход, предполагающий работу с атрибутом `srcset` (сокращение от `source set`).

Этот атрибут был добавлен в HTML недавно. Он позволяет указать несколько URL-адресов изображений для одного элемента `img`, а также разрешение каждого из них. Затем браузер определит нужную версию изображения и загрузит ее (листинг 8.16).

Листинг 8.16. Адаптивное изображение `srcset`

```

```

URL-адрес каждого изображения и его ширина

Передает обычный атрибут `src` браузерам, которые не поддерживают атрибут `srcset` (например, Internet Explorer и Opera Mini)

Сейчас большинство браузеров поддерживают атрибут `srcset`, а те, которые этого не делают, будут использовать резервный атрибут `src`, загружая изображение, расположенное по указанному в нем URL-адресу. Это позволяет оптимизировать дизайн для разных размеров экрана. Еще лучше то, что браузер может внести корректировки для экранов с более высоким разрешением. Если экран устройства имеет повышенную плотность пикселей (2×), он способен загрузить изображения с более высоким разрешением.

Дополнительную информацию об адаптивных изображениях можно найти на странице jakearchibald.com/2015/anatomy-of-responsive-images/. В этой статье рассматриваются и другие полезные функции, например настройка размера экрана, исходя из загруженного изображения.

СОВЕТ

При разработке резинового макета всегда нужно следить за тем, чтобы ширина изображений не превышала ширину их контейнеров. Чтобы этого не происходило, добавляйте в таблицу стилей правило `img { max-width: 100 %; }`.

При разработке адаптивного дизайна структурировать различные области страницы можно множеством способов. Любой из них сводится к применению трех принципов — подхода *Mobile First*, медиазапросов и резинового макета.

Итоги главы

- ❑ Всегда начинайте разработку дизайна с создания макетов для мобильных устройств.
- ❑ Используйте медиазапросы с целью прогрессивного улучшения страницы для все более крупных областей просмотра.
- ❑ Применяйте резиновые макеты, которые вписываются в экран при любом размере окна браузера.
- ❑ Работайте с адаптивными изображениями, чтобы учесть ограничения пропускной способности мобильных устройств.
- ❑ Не забудьте включить свой метатег `viewport`.

Часть III

Масштабируемый CSS-код

Код представляет собой средство коммуникации не только с компьютером, но и с другими разработчиками, которые будут работать с данным кодом. То, как вы пишете и организуете CSS-код, так же важно, как и способ его отображения в браузере. В части III, в главах 9 и 10, я покажу, как структурировать CSS-код так, чтобы он был понятен и его было легко сопровождать в дальнейшем.

9

Модульный CSS

В этой главе

- Проблемы, возникающие по мере роста проекта.
- Организация CSS в модули.
- Предотвращение роста специфичности селекторов.
- Рассмотрение популярных методологий CSS.

В частях I и II мы взглянули на запутанность CSS и соответствующих инструментов при размещении элементов на странице. Поняли суть блочной модели, схлопывания полей, контекста наложения, плавающих элементов и flex-контейнеров. Эти навыки понадобятся, особенно когда вы начнете новый проект. Тем не менее мир разработки программного обеспечения устроен так, что вы проведете много времени не только за написанием нового кода, но и за обновлением и дополнением существующего. В работе с CSS это принесет вам абсолютно новую массу трудностей.

Изменения, вносимые в существующую таблицу стилей, могут затронуть произвольное количество элементов на произвольном количестве страниц вашего сайта. На эту тему есть старая шутка: два свойства CSS заходят в бар, в другом баре падает стул. Как обеспечить, чтобы изменения были выполнены там, где надо? И как вы узнаете, что они не затронули элементы, которые не должны изменяться?

В части III мы обсудим эти проблемы. Рассмотрим структуру CSS, уделяя меньше внимания объявлениям в таблице стилей и фокусируясь на выбираемых вами селекторах и HTML-разметке, к которой вы применяете стили. От того, как структурирован код, зависит, можно ли будет безопасно вносить изменения в дальнейшем без нежелательных побочных эффектов. Правильное структурирование начинается с понимания модульного CSS-кода, на котором будет сосредоточена данная глава.

Составлять *модульный CSS-код* означает разбивать страницу на составные части, которые должны быть пригодными для повторного применения в различных контекстах и не должны непосредственно зависеть друг от друга. Конечная цель: изменения в одной части CSS-кода не должны неожиданно влиять на другую.

Это сродни использованию модульной мебели, как в кухнях IKEA. Вместо того чтобы проектировать один гигантский кухонный шкаф, можно выбрать несколько отдельных элементов с одинаковым дизайном, благодаря чему они будут визуально совместимы. В условиях полной упорядоченности вы можете поместить каждый из элементов туда, куда захотите. С помощью модульного CSS-кода вместо того, чтобы проектировать одну гигантскую веб-страницу, вы проектируете все ее части независимыми от других частей, а затем соединяете их согласно тому порядку, который предпочтете.

Идея написания модульного кода не нова в программировании, но разработчики начали применять ее к CSS лишь в течение последних нескольких лет. Поскольку современные сайты и веб-приложения стали больше и сложнее, нужно было найти способ управлять растущей сложностью таблиц стилей.

В то время как в таблицах стилей любой селектор может сделать что угодно и где угодно на странице, модульные стили позволяют все упорядочить. Каждая часть таблицы стилей, которую мы станем называть *модулем*, будет нести ответственность за свои собственные стили, и ни один модуль не должен вмешиваться в стили другого. Это принцип инкапсуляции, примененный к CSS.



Инкапсуляция — это группировка взаимосвязанных функций и данных объекта. Прием используется для того, чтобы скрыть состояние или значение структурированного объекта с целью избежать внешнего воздействия на него.

В CSS нет понятий данных или традиционных функций, но есть селекторы и элементы, к которым они применяются. В целях инкапсуляции они будут частями, из которых состоит модуль, и каждый модуль будет ответственен за стилевое форматирование небольшого количества элементов DOM.

Учитывая инкапсуляцию, вы определите модуль для каждого дискретного компонента страницы: навигационных меню, диалоговых окон, индикаторов выполнения и изображения миниатюр. Каждый модуль должен быть определен уникальным именем класса, который задан элементу DOM. Каждый модуль также будет иметь собственный паттерн дочерних элементов для конструирования модуля на странице. Создавая ее, вы будете вкладывать одни модули в другие.

9.1. Базовые стили: закладываем основы

Прежде чем окунуться в создание модульных стилей, необходимо определить среду. Каждая таблица стилей начинается с набора универсальных правил, применяемых ко всей странице, это по-прежнему необходимо в модульном CSS-коде. Такие правила зачастую называются *базовыми правилами*, поскольку они закладывают

фундамент, на котором будут базироваться остальные стили. Данная часть таблицы стилей, по сути, не будет модульной, но она заложит основу для модульных стилей.

Создайте новую страницу и таблицу стилей и примените базовые стили из листинга 9.1 в CSS. Стили, приведенные в листинге, — всего лишь пример некоторых базовых стилей.

Листинг 9.1. Добавление базовых стилей

```
:root {
  box-sizing: border-box;
}

*,
*::before,
*::after {
  box-sizing: inherit;
}

body {
  font-family: Helvetica, Arial, sans-serif;
  margin: 0;
}
```

Изменение размеров блока (см. главу 3)

Размер шрифта по умолчанию на странице

Другие базовые стили обычно включают цвет ссылок, стили заголовков и поля. По умолчанию элемент `body` имеет небольшие поля, которые вы, вероятно, захотите ликвидировать. В зависимости от проекта может понадобиться применить стили к полям форм, таблицам и спискам.

СОВЕТ

Я рекомендую к использованию библиотеку `normalize.css`. Это небольшая таблица стилей, которая помогает выровнять расхождения в таблицах стилей агентов пользователей различных браузеров. Можете загрузить ее по адресу necolas.github.io/normalize.css/. Поставьте ее перед своей таблицей стилей как часть базовых стилей.

Базовые стили должны быть довольно универсальными. Добавляйте только те из них, которые должны быть применены к большей части страницы или к ней целиком. В селекторах не должно быть никаких имен классов или идентификаторов, выделяйте только элементы по типу тегов или в редких случаях по псевдоклассам. Смысл в том, что данные стили обеспечивают глобальный внешний вид, но легко могут быть изменены позднее.

Как только базовые стили проекта будут заданы, их редко станут изменять. Они обеспечивают стабильную основу, на которой можно спроектировать модульный CSS-код. После базовых стилей остальная часть таблицы стилей будет состоять в основном из модулей.

9.2. Простой модуль

Создадим простой модуль для коротких информационных сообщений. Поскольку каждый модуль должен иметь уникальное имя, назовите его `Message`. Задайте ему цвет и границу, чтобы привлечь внимание пользователя (рис. 9.1).

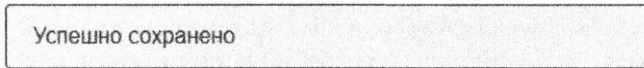


Рис. 9.1. Модуль `Message`

Разметка данного модуля состоит из одного элемента `div` с классом `message`. Добавьте разметку (листинг 9.2) на страницу.

Листинг 9.2. Разметка модуля `Message`

```
<div class="message">
  Успешно сохранено
</div>
```

CSS-код представляет собой единственный набор правил. Он обращается к модулю по имени класса. Затем определяет отступы, границы, скругление границ и цвета. Чтобы применить эти правила к модулю `Message`, добавьте код из листинга 9.3 в таблицу стилей после базовых стилей.

Листинг 9.3. Реализация модуля `Message`

```
.message {
  padding: 0.8em 1.2em;
  border-radius: 0.2em;
  border: 1px solid #265559;
  color: #265559;
  background-color: #e0f0f2;
}
```

← Обращается к модулю `Message` по имени класса

Вам должны быть знакомы все эти свойства, поэтому на данном этапе все кажется понятным. Данный CSS-код выглядит так же, как в остальной части книги. И действительно, большая часть написанного до сих пор кода соответствует принципам модульного подхода. Я лишь не заострял на этом внимание до текущего момента. Давайте определим, что делает этот CSS-код модульным.

Важно, что селектор данного модуля состоит из единственного имени класса. Больше ничто в нем не ограничивает расположение стилей на странице. Если бы мы взяли селектор `#sidebar .message`, это означало бы, что модуль может быть использован только внутри элемента с идентификатором `#sidebar`. Без такого ограничения он пригоден для повторного применения в любом контексте.

Присвоив класс элементу, можете повторно задействовать данные стили для того, чтобы обеспечивать отклик на ввод данных в формы, создавать заметный

вспомогательный текст или привлекать внимание к заявлению об отказе от ответственности. Используя повторно один и тот же модуль, вы формируете последовательный пользовательский интерфейс. Везде, где применяется, он будет выглядеть одинаково. У вас не получатся немного разный зеленовато-голубой цвет в одних местах или немного бóльшие отступы в других.

Я работал над проектами, где CSS-код не был модульным. В одном из них серия похожих кнопок использовалась в различных частях приложения. Там были кнопки `.save-form`, `.login-form`, `.toolbar` и `.options`. Один и тот же код в таблице стилей повторялся множество раз, но кнопки не были идентичными друг другу. Когда требовалось изменить оформление всех этих кнопок, приходилось последовательно вносить изменения, которые применялись к каждому экземпляру конкретной кнопки. Из-за этого возникали ошибки и упущения. Поэтому некоторые кнопки имели небольшие различия в отступах или оттенках красного.

Решено было превратить множество кнопок в единственный пригодный для многократного применения модуль, который работает независимо от расположения на странице. Создание модуля обеспечивало не только более простой код (меньше повторов), но и визуальную стабильность. Такое оформление выглядело более профессионально и меньше походило на поспешное нагромождение элементов. На подсознательном уровне это помогает пользователям больше доверять вашему приложению.

9.2.1. Вариации модуля

Последовательность — это хорошо, но иногда нужно намеренно отклониться от нее. Модуль `Message` неплох, но, возможно, потребуется, чтобы при определенных обстоятельствах он выглядел иначе. Например, если необходимо отобразить сообщение об ошибке, лучше, чтобы оно было красного цвета, а не зеленовато-голубого. Или если вы хотите различать чисто информационные сообщения и те, которые свидетельствуют об успешном действии, например сохранении. Можете сделать это, определив *модификаторы*.

Создайте модификатор, определяя новый класс, имя которого начинается с имени модуля. Например, модификатор `error` для модуля `Message` может выглядеть следующим образом: `message-error`. Включив имя модуля в название модификатора, вы точно определите, что данный класс принадлежит модулю `Message`.

ПРИМЕЧАНИЕ

Существует популярное соглашение — использовать два дефиса, чтобы указать на модификатор, например: `message--error`.

Создадим три модификатора модуля: `success`, `warning` и `error`. Добавьте приведенный в листинге 9.4 код в таблицу стилей.

Листинг 9.4. Модуль Message с классами модификаторов

```
.message {  
  padding: 0.8em 1.2em;  
  border-radius: 0.2em;  
  border: 1px solid #265559;  
  color: #265559;  
  background-color: #e0f0f2;  
}  
  
.message--success {  
  color: #2f5926;  
  border-color: #2f5926;  
  background-color: #cfe8c9;  
}  
  
.message--warning {  
  color: #594826;  
  border-color: #594826;  
  background-color: #e8dec9;  
}  
  
.message--error {  
  color: #59262f;  
  border-color: #59262f;  
  background-color: #e8c9cf;  
}
```

← Базовый модуль Message

← Модификатор success, изменяющий цвет сообщения на зеленый

← Модификатор warning, изменяющий цвет сообщения на желтый

← Модификатор error, изменяющий цвет сообщения на красный

Стили модификаторов не требуют переопределения всего модуля. Необходимо только перезаписать те части, которые должны быть изменены. В данном случае следует изменить цвет текста, границ и фона.

Чтобы использовать модификатор, присвойте классы модуля и модификатора элементу, как показано в листинге 9.5. Так будут применены стили модуля, а модификатору позволено при необходимости менять некоторые из них.

Листинг 9.5. Экземпляр модуля Message с модификатором error

```
<div class="message message--error">  
  Неверный пароль  
</div>
```

← Оба класса присваиваются элементу

Аналогично добавьте модификаторы `success` или `warning`, когда необходимо использовать эти версии. Они меняют цвета модуля, а другие могут воздействовать на его размер или даже расположение.

Варианты модуля Button

Создадим другой модуль с несколькими вариантами. Это будет модуль `Button` с вариантами большого и малого размера, а также различной расцветки (рис. 9.2). Таким образом вы можете использовать цвет, чтобы придать кнопкам визуальный

смысл. Зеленый цвет укажет на позитивное действие, такое как сохранение или отправка формы. Красный будет означать предупреждение, призванное предотвратить случайное нажатие пользователем кнопки отмены.

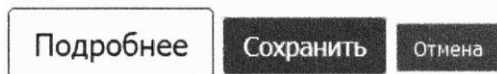


Рис. 9.2. Кнопки с применением модификаторов, меняющих размеры и цвета

Стили для этих кнопок приведены в листинге 9.6. Он включает главный модуль Button, а также четыре класса модификаторов: по два для размера и цвета. Добавьте этот код в таблицу стилей.

Листинг 9.6. Модуль Button и его модификаторы

```
.button {
  padding: 0.5em 0.8em;
  border: 1px solid #265559;
  border-radius: 0.2em;
  background-color: transparent;
  font-size: 1rem;
}

.button--success {
  border-color: #cfe8c9;
  color: #fff;
  background-color: #2f5926;
}

.button--danger {
  border-color: #e8c9c9;
  color: #fff;
  background-color: #a92323;
}

.button--small {
  font-size: 0.8rem;
}

.button--large {
  font-size: 1.2rem;
}
```

← Базовые стили кнопки

← Зеленый цветовой вариант success

← Красный цветовой вариант danger

← Вариант small

← Вариант large

Модификаторы размеров настраиваются с помощью изменения размеров шрифта на больший или меньший. Вы использовали данный метод в главе 2. При изменении размеров шрифта корректируется размер элемента в единицах em, который, в свою очередь, изменяет размеры отступов и скругление границ, не требуя перезаписи их объявленных значений.

СОВЕТ

Всегда храните код модуля в одном месте. Тогда ваша таблица стилей будет состоять из последовательных серий модулей.

Когда все модификаторы собраны, у HTML-верстальщика есть варианты выбора. Можно добавить классы модификаторов, чтобы менять размер кнопок в зависимости от их важности. А также выбрать цвет, доносящий до пользователя определенное сообщение.

В листинге 9.7 приведен HTML-код с применением модификаторов для создания различных кнопок. Добавьте его на свою страницу, чтобы увидеть, как он работает.

Листинг 9.7. Использование модификаторов для создания различных типов кнопок

```

<button class="button button--large">Подробнее</button>
<button class="button button--success">Сохранить</button>
<button class="button button--danger button--small">Отмена</button>
  
```

Первая кнопка большого размера. Вторая окрашена в зеленый цвет и говорит об успешности операции. Третья использует два модификатора: цвета (*danger*) и размера (*small*), как и кнопки на рис. 9.2.

Синтаксис с применением двойного дефиса может казаться немного странным. Его польза становится более очевидной, когда вы начинаете создавать модули для элементов с длинными именами, таких как навигационное меню или врезки. Добавление модификатора к модулям, имена которых состоят из двух и более слов, может дать такие имена классов, как *nav-menu--horizontal* или *pull-quote--dark*.

Синтаксис с двойным дефисом сразу показывает, какая часть класса — имя модуля, а какая — модификатор. Класс *nav-menu--horizontal* — не то же самое, что класс *nav--menu-horizontal*. Это вносит ясность в работу с проектами, содержащими большое количество модулей со схожими именами.

ПРИМЕЧАНИЕ

Нотация двойного дефиса была популяризована методологией БЭМ. Ее и некоторые другие методологии представлю ближе к концу главы.

Не создавайте зависящие от контекста селекторы

Представьте, что вы поддерживаете сайт, имеющий светлое раскрывающееся меню в заголовке. В какой-то момент руководство сообщает вам, что это меню необходимо инвертировать, сделав его темным с белым текстом.

Решив не задействовать модульный подход, вы можете присвоить раскрывающемуся списку селектор `.page-header .dropdown`. В этом случае будут перезаписаны изначально установленные цвета класса `dropdown`. Модульный подход строго запрещает применение этого селектора. Несмотря на то что использование селектора-потомка сейчас сработает, в будущем возникнет множество проблем. Рассмотрим возможные варианты.

Во-первых, необходимо решить, к чему этот код относится — к стилям заголовка страницы или только к раскрывающемуся меню? После добавления множества узконаправленных правил таблица стилей превратится в бессистемный список несвязанных стилей. И если вам понадобится изменить некоторые стили позднее, вспомните ли вы, где они находятся?

Во-вторых, данный подход увеличивает специфичность селекторов. Если вы решите вносить дальнейшие изменения, то столкнетесь с этой специфичностью и, вероятно, увеличите ее.

В-третьих, возможно, позже темный раскрывающийся список нужно будет применить где-то еще. Тот, который вы создали, связан с шапкой страницы. Если понадобится темный раскрывающийся список на боковой панели, то придется добавить в набор правил новые селекторы, соответствующие обоим сценариям, или продублировать стили полностью.

Наконец, если так будет продолжаться долго, то придется создавать все более длинные селекторы, которые связывают CSS с конкретной структурой HTML. Например, если у вас есть селектор `#products-page .sidebar .social-media div:first-child h3`, то набор правил тесно связан с определенным местом на определенной странице.

Все эти проблемы лежат в корне множества разочарований, с которыми сталкиваются разработчики, применяя CSS. Чем дольше таблица стилей используется и поддерживается, тем хуже становится ситуация. Специфичность селекторов постепенно растет по мере того, как новые стили перекрывают старые. Прежде чем понять это, вы обнаружите, что пишете селектор с двумя идентификаторами и пятью классами, только чтобы адресовать его флажку.

Станет трудно находить правила, поскольку отдельным элементам адресованы части кода в множестве разрозненных частей таблицы стилей. Все труднее станет понимать организацию таблицы стилей и ее воздействие на страницу. Если в коде трудно разобраться, ошибки станут появляться все чаще. Даже крошечные изменения на странице могут испортить значительную часть ее стилевого форматирования. Удаление старого кода станет небезопасным, поскольку никто не знает, что он делает и важен ли он по-прежнему. Таблица стилей разрастается, и проблемы становятся еще сложнее. Их и стремится предотвратить модульный подход.

Если нужно, чтобы модуль выглядел или вел себя по-другому, создайте класс модификатора, который может быть применен непосредственно к определенному элементу. Например, вместо того, чтобы создавать класс `.page-header .dropdown`, создайте класс `.dropdown--dark`. Таким образом, модуль и только модуль будет

ответственен за свой внешний вид. Другие модули не смогут повлиять на его изменение. Темный раскрывающийся список не связан ни с одной глубоко вложенной структурой HTML, поэтому можете поместить его в любое место на странице.

Никогда не используйте селекторы-потомки, чтобы изменить модуль на основе его местоположения на странице. Следуя этому правилу, вы убережете таблицы стилей от падения в пропасть нечитаемого кода.

9.2.2. Модули с множеством элементов

Модули `Message` и `Button`, которые вы недавно создали, просты и удобны, они состоят лишь из одного элемента. Но многие модули, которые вы будете разрабатывать, нуждаются в нескольких элементах. Не получится создать раскрывающееся меню или модальное окно из одного элемента.

Выполним более сложный модуль. Это будет медиаобъект (рис. 9.3), как тот, с которым вы работали в главе 4 (см. подраздел 4.5.1).

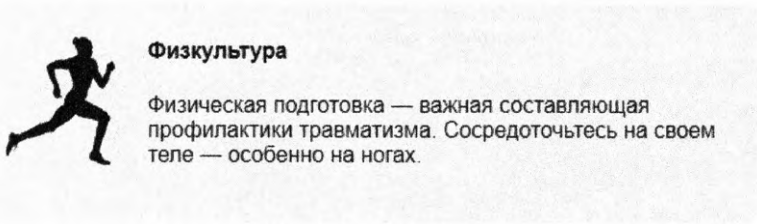


Рис. 9.3. Модуль `Media`, состоящий из четырех элементов

Данный модуль состоит из четырех элементов: контейнера `div`, содержащего изображение и основную часть, которая, в свою очередь, имеет заголовок. Как и в ходе работы с другими модулями, вы присвоите главному контейнеру класс `media`, который будет соответствовать имени модуля. Для изображения и основной части примените классы `media__image` и `media__body`. Их имена начинаются с названия модуля, затем ставится двойное подчеркивание, далее идут имена дочерних элементов (это еще одно правило методологии БЭМ). Как и двойной дефис в модификаторах, такие имена классов сразу дают понять, какую роль играет элемент и какому модулю он принадлежит.

Правила модуля `Media` приведены в листинге 9.8. Добавьте их в таблицу стилей.

Вы заметите, что не должны задействовать много селекторов-потомков. Изображение — это дочерний элемент модуля `Media`, поэтому можно использовать селектор `.media > .media__image`, но в этом нет необходимости. Поскольку имя модуля содержится в имени класса `media__image`, оно и так уникально.

Я задействовал для заголовка комбинатор прямых потомков. Мог взять класс `media__title` (или даже `media__body__title`, чтобы полностью отразить его позицию в иерархии), но пришел к выводу, что это не всегда необходимо. В данном случае я решил, что элемент `h4` достаточно описателен, чтобы показать: он является

заголовком модуля Media. Это препятствует применению различных заголовков (h3 или h5). Если вам не нравится такое ограничение, используйте в своих модулях имя класса для обращения к элементу.

Листинг 9.8. Модуль Media с дочерними элементами

```
.media {
  padding: 1.5em;
  background-color: #eee;
  border-radius: 0.5em;
}
.media::after {
  content: "";
  display: block;
  clear: both;
}
.media_image {
  float: left;
  margin-right: 1.5em;
}
.media_body {
  overflow: auto;
  margin-top: 0;
}
.media_body > h4 {
  margin-top: 0;
}
```

← Главный контейнер

← Clearfix

← Дочерние элементы изображения и основной части

← Заголовок внутри основной части

Добавьте разметку для данного модуля (листинг 9.9) на страницу.

Листинг 9.9. Разметка для модуля Media

```
<div class="media">
  
  <div class="media_body">
    <h4>Физкультура</h4>
    <p>
      Физическая подготовка &mdash; важная составляющая
      профилактики травматизма. Сосредоточьтесь
      на своем теле &mdash; особенно на ногах.
    </p>
  </div>
</div>
```

← Дочерний элемент изображения

← Дочерний элемент основной части

← Дочерний элемент заголовка

Данный модуль универсален. Он работает в контейнерах разных размеров, адаптируясь под их ширину. Вы можете помещать абзацы текста в основную часть. И использовать его с меньшими или бо́льшими изображениями (при этом имеет смысл задать изображению свойство `max-width`, чтобы оно не сжимало основную часть).

Применяйте варианты и дочерние элементы совместно

Можно разработать также вариации модуля. Теперь несложно создать версию, в которой изображение находится справа, а не слева (рис. 9.4).

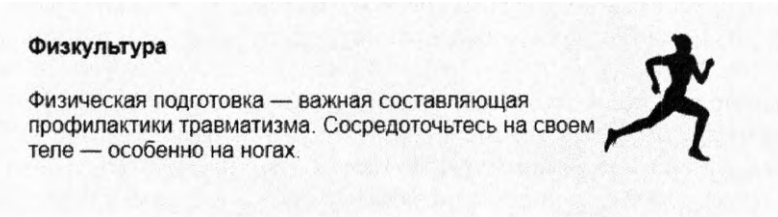


Рис. 9.4. Модуль Media, правосторонний вариант

Вариант `media--right` выполнит эту задачу. Вы можете добавить класс варианта в основной элемент `div` модуля (он будет выглядеть так: `<div class="media media--right">`). Затем используйте его, чтобы сделать изображение плавающим справа.

Присвойте класс модификатора элементу в своей HTML-разметке. Добавьте код из листинга 9.10 в таблицу стилей, чтобы увидеть эту версию.

Листинг 9.10. Определение правого варианта модуля Media

```
.media--right > .media__image {  
  float: right;  
}
```

Обращается к дочернему элементу изображения, но только тогда, когда присутствует модификатор `right`

Данное правило перекрывает изначальное правило изображения модуля Media `float: left`. Благодаря особенностям свойства `float` не нужно изменять порядок элементов в HTML-разметке.

Избегайте основных названий элементов в селекторах модулей

В модуле Media я задействовал селектор `.media body > h4`, чтобы выделить элемент заголовка. Я был готов сделать это, поскольку элемент `h4` представляет собой второстепенный заголовок. Данный метод применяю также для модулей, содержащих списки. Считаю, что проще выделять элементы меню селектором `.menu > li`, чем добавлять класс `menu__item` к каждому элементу списка, хотя мнения на этот счет расходятся.

Избегайте обращений, базирующихся на основных типах элементов, таких как `div` и `span`. Селекторы, подобные `.page-header > span`, слишком общие. Когда вы только создавали модуль, то могли использовать в нем лишь один элемент `span` для одной цели, но существует вероятность, что позже понадобится еще один элемент `span` для другой цели. Присвоение классов элементам `span` в дальнейшем осложняется тем, что потребуется отслеживать каждое применение модуля в разметке и вносить в него изменения.

9.3. Составление крупных структур из модулей

В своей книге «Чистый код: создание, анализ и рефакторинг» Роберт Мартин (Robert C. Martin) написал: «Первое правило: классы должны быть компактными. Второе правило: классы должны быть еще компактнее». Он говорил о классах в объектно-ориентированном программировании, но этот же принцип применим и для CSS-модулей.

Каждый модуль должен отвечать за что-то одно. Модуль Message отвечал за то, чтобы сообщения были заметными. Модуль Media — за расположение изображения по отношению к определенному тексту. Вам необходимо кратко сформулировать его цель. Некоторые модули станут отвечать за расположение, другие — за стили. Если у модуля больше одной цели, нужно подумать о его разбивке на меньшие модули.

В качестве примера создадим раскрывающееся меню (рис. 9.5). Оно будет выглядеть как то, которое разрабатывалось в главе 7 (см. подраздел 7.3.1).

Начиная работать над модулем, спросите себя: «За что в конечном итоге отвечает данный модуль?» В нашем случае первый ответ может быть таким: «За кнопку, которая разворачивает раскрывающееся меню и представляет пункты этого меню расположенными вертикально друг над другом».

Это подходящее описание сценария, но я в работе придерживаюсь практического правила: «Если при описании сферы ответственности модуля нужно использовать союз «и», то можно описать несколько сфер ответственности». Будет ли это разворачиванием меню, или это задача представления расположения пунктов меню?

Когда для описания области ответственности модуля требуется союз «и», убедитесь, что вы не описываете две или более зоны ответственности. Может быть, она все же одна, это не строгое правило. Но если их все-таки несколько, нужно определить отдельные модули под каждую задачу. В этом заключается важный принцип инкапсуляции, который называется *принципом единственной ответственности*. Когда возможно, распределите сферы ответственности по отдельным модулям. Это поможет каждому модулю оставаться компактным, сфокусированным и простым для понимания.

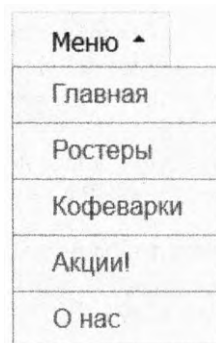


Рис. 9.5. Раскрывающееся меню

9.3.1. Разделение ответственности между модулями

Создадим раскрывающееся меню с двумя разными модулями. Первый я назову *Dropdown*, в нем будет кнопка, отвечающая за контроль видимости контейнера. Вы можете сказать, что она отвечает за отображение и скрытие контейнера. А также описать внешний вид кнопки и небольшой треугольник, указывающий на действие.

Описание модуля в таких деталях требует использования союза «и», но все эти пункты относятся к основной зоне ответственности, поэтому, я считаю, краткого описания достаточно.

Второй модуль, назову его *Menu*, будет представлять собой список ссылок. Затем вы создадите полный интерфейс, помещая экземпляр модуля *Menu* внутрь контейнера модуля *Dropdown*.

Вначале добавьте на страницу разметку из листинга 9.11. Она содержит модуль *Dropdown*, включающий в себя модуль *Menu*. В ней есть также минимальное количество кода JavaScript, чтобы добавить функционал, когда *toggle*-кнопка нажата.

Листинг 9.11. Конструирование раскрывающегося меню из двух модулей

```
<div class="dropdown">
  <button class="dropdown_toggle">Меню</button>
  <div class="dropdown_drawer">
    <ul class="menu">
      <li><a href="/">Главная</a></li>
      <li><a href="/coffees">Ростеры</a></li>
      <li><a href="/brewers">Кофеварки</a></li>
      <li><a href="/specials">Акции!</a></li>
      <li><a href="/about">О нас</a></li>
    </ul>
  </div>
</div>
```

Toggle-кнопка для раскрывающегося меню

Дочерний элемент drawer, служащий контейнером для меню

Модуль Menu, вложенный в элемент drawer

```
<script type="text/javascript">
  (function () {
    var toggle = document.querySelector('.dropdown_toggle');
    toggle.addEventListener('click', function (event) {
      event.preventDefault();
      var dropdown = event.target.parentNode;
      dropdown.classList.toggle('is-open');
    });
  })();
</script>
```

Разворачивает класс is-open, когда нажата toggle-кнопка

Я использовал нотацию двойного подчеркивания, чтобы указать, что *toggle*-элемент и *drawer*-элемент — это дочерние элементы модуля *Dropdown*. При нажатии *toggle*-кнопки будет отображен или скрыт *drawer*-элемент. Код JavaScript выполняет эту задачу, добавляя класс *is-open* в основной элемент раскрывающегося меню или удаляя его оттуда.

Стили для меню показаны в листинге 9.12. Добавьте их в свою таблицу стилей. Они аналогичны рассмотренным в главе 7. Я обновил имена классов, чтобы они соответствовали нотации двойного подчеркивания. Вы получаете функционирующий раскрывающийся список, хотя меню в нем остается не отформатированным с помощью стилей.

Листинг 9.12. Определение модуля Dropdown

```

.dropdown {
  display: inline-block;
  position: relative;
}

.dropdown__toggle {
  padding: 0.5em 2em 0.5em 1.5em;
  border: 1px solid #ccc;
  font-size: 1rem;
  background-color: #eee;
}

.dropdown__toggle::after {
  content: "";
  position: absolute;
  right: 1em;
  top: 1em;
  border: 0.3em solid;
  border-color: black transparent transparent;
}

.dropdown__drawer {
  display: none;
  position: absolute;
  left: 0;
  top: 2.1em;
  min-width: 100%;
  background-color: #eee;
}

.dropdown.is-open .dropdown__toggle::after {
  top: 0.7em;
  border-color: transparent transparent black;
}

.dropdown.is-open .dropdown__drawer {
  display: block;
}

```

Устанавливает внешний блок для абсолютного позиционирования drawer-элемента

Рисует треугольник с помощью границ (см. главу 7)

Изначально скрывает класс, затем отображает его, когда присутствует класс is-open

Переворачивает треугольник, когда список раскрыт

В данном листинге использовано относительное позиционирование в основном элементе, чтобы установить внешний блок для drawer-элемента, который, в свою очередь, позиционирован абсолютно. Это обеспечивает некоторые стили для toggle-кнопки, включая треугольник в псевдоэлементе `::after`. Наконец, отображается drawer-элемент, и треугольник переворачивается, когда появляется класс `is-open`.

Все это занимает около 35 строк кода. Событий здесь не так уж много, чтобы не удерживать их в голове во время работы над модулем. Когда в будущем вы вернетесь к модулю, чтобы внести в него изменения, вы захотите, чтобы он был компактным и в нем легко было разобраться.

Позиционирование в модуле

При создании данного модуля вы впервые использовали позиционирование. Обратите внимание на то, что оно устанавливает собственный внешний блок (свойство `position: relative` в основном элементе). Абсолютно позиционированные элементы (`drawer`-элемент и псевдоэлемент `::after`) основываются на позициях, определенных в том же модуле.

По возможности старайтесь сохранять позиционированные элементы, связанные друг с другом, в одном и том же модуле. Так модуль не будет вести себя странно, если поместить его в другой позиционированный контейнер.

Классы состояния

У класса `is-open` есть специальное назначение в модуле `Dropdown`. Он предназначен быть добавленным или удаленным из модуля динамично благодаря использованию JavaScript. Он является также примером *класса состояния*, поскольку указывает на текущее состояние модуля.

Есть распространенное соглашение — давать всем классам состояния имена с префиксами `is-` или `has-`. Так их назначение становится очевидным: они указывают на текущее состояние модуля или говорят о его ожидаемом изменении. Другими примерами классов состояния можно назвать `is-expanded`, `is-loading` или `has-error`. Точное определение того, что они делают, зависит от модуля, в котором они используются.

ВАЖНО

Группируйте код классов состояния вместе с остальной частью кода модуля. Затем каждый раз, когда понадобится применять JavaScript для динамического изменения отображения модуля, задействуйте классы состояния, чтобы вызвать изменения.

Препроцессоры и модульный CSS-код

Все препроцессоры, такие как Sass или Less, предоставляют возможность слияния CSS-файлов в один, что позволяет организовывать стили в нескольких файлах и папках, но обслуживать их через браузер как один файл. Это сокращает количество сетевых запросов, необходимых браузеру, позволяя разбить код на части управляемых размеров. По моему мнению, это одна из самых полезных возможностей, обеспечиваемых препроцессорами.

Если вы, используя процессор, будете помещать каждый CSS-модуль в собственный соответствующим образом названный файл, это будет очень здорово. Организуйте их в папки по мере необходимости. Затем создайте основную таблицу стилей, которая импортирует все модули. Таким образом, вам не понадобится искать каждый модуль в большой таблице стилей, чтобы внести изменения. Вы будете точно знать, где его найти.

Создайте файл `main.scss`, который содержит только операторы `@import`. Это будет выглядеть примерно так:

```
@import 'base';
@import 'message';
@import 'button';
@import 'media';
@import 'dropdown';
```

Тогда препроцессор скопирует базовые стили из файла `base.scss` и стили модулей из соответствующих файлов, создав единую таблицу стилей, содержащую их все. Таким образом, у каждого модуля будет собственный файл, благодаря чему редактирование заметно упростится.

Посмотрите приложение В, чтобы получить более подробную информацию о препроцессорах, или изучите документацию своего препроцессора, чтобы узнать больше о его директиве импорта.

Модуль Menu

Теперь, когда модуль `Dropdown` работает, можно переключиться на модуль `Menu`. Больше не беспокойтесь об открытии и закрытии раскрывающегося меню, поскольку об этом позаботится модуль `Dropdown`. Модуль `Menu` применит необходимые стили к списку ссылок.

Данные стили приведены в листинге 9.13. Добавьте их в таблицу стилей.

Листинг 9.13. Стили модуля `Menu`

```
.menu {
  padding-left: 0;
  margin: 0;
  list-style-type: none;
  border: 1px solid #999;
}

.menu > li + li {
  border-top: 1px solid #999;
}

.menu > li > a {
  display: block;
  padding: 0.5em 1.5em;
  background-color: #eee;
  color: #369;
  text-decoration: none;
}

.menu > li > a:hover {
  background-color: #fff;
}
```

← Перекрывает стили пользовательского агента, чтобы удалить маркеры списка

← Добавляет границы между ссылками

← Форматирует с помощью стилей большие чувствительные к щелчку ссылки

← Добавляет подсветку при наведении указателя мыши

Это те же объявления, которые вы использовали в раскрывающемся списке в главе 7. Каждый элемент `li` — это дочерний элемент модуля, поэтому я не считаю нужным добавлять двойное подчеркивание в каждый из них. Селектор прямого потомка `.menu > li` достаточно специфичен.

Данный модуль абсолютно автономен. Он не зависит от модуля Dropdown. Благодаря этому сохраняется простота кода, потому что вам не нужно понимать одно, чтобы разобраться в другом. Также становится реальным более гибкое повторное применение данных модулей.

Можно создать другой стиль меню вне зависимости от того, вариант это или отдельный модуль, и при необходимости использовать его в раскрывающемся списке. И задействовать это меню где угодно за пределами раскрывающегося списка. Вы не сможете предсказать, какой стиль потребуется страницам в будущем, но пригодные для повторного применения модули позволяют смешивать и согласовывать новые возможности со знакомым последовательным стилем.

9.3.2. Именованние модулей

Выбор подходящих имен модулей требует определенных раздумий. В ходе разработки можно дать модулю временное имя, но прежде, чем посчитать его готовым, нужно убедиться, что имени уделено некоторое внимание. Правильное именованние модулей — это, вероятно, самое сложное в создании модульного CSS-кода.

Рассмотрите модуль Media из начала главы. Он использовался для вывода изображения бегуна и текста совета (рис. 9.6).

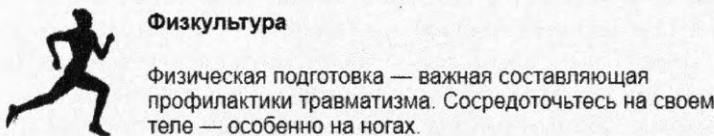


Рис. 9.6. Модуль Media с изображением и советом

Представьте, что вы еще не дали модулю имени, но знаете, что это нужно. Можете назвать его «бегущий совет». Это название точно отражает его суть и кажется подходящим. Но подумайте о других объектах, для которых применимы стили данного модуля. Что если вы захотите задействовать такой же элемент пользовательского интерфейса для чего-то другого? Возможно, в соответствии с темой сайта нужно создать серию таких элементов, чтобы перечислить предстоящие соревнования. Если вы сделаете это, имя «бегущий совет» станет неуместным.

Необходимо дать модулю имя, емкое независимо от контекста, в котором его планируется использовать. Также стоит избегать имен, просто описывающих внешний вид. Название «серый блок с изображением» кажется более универсальным, но вдруг вы решите в дальнейшем изменить цвет фона на светло-голубой? Или будете

выполнять полный редизайн сайта? Такое имя окажется неприемлемым, и вам придется переименовать модуль везде, где он применялся в HTML-разметке.

Вместо этого вы должны спросить себя, что указанный модуль представляет собой концептуально. Определить это — не всегда легкая задача. Имя *Media* на данном этапе хорошо подходит, модуль поддерживает некоторое изображение и текст. Имя производит сильное впечатление и не привязывает модуль к какому-то конкретному применению или визуальной реализации.

Модуль должен быть универсальным. Его имя в идеале должно быть простым и запоминающимся. Работая над дизайном множества страниц сайта, можете повторно использовать модуль для большого количества задач. Вы и другие члены вашей команды можете включать эти имена в свой лексикон и задействовать, когда говорите о сайте: «Давай применим здесь *медиа*» или «Эти *тайлы* расположены слишком близко».

Пока вы создали только модули *Message*, *Media*, *Dropdown* и *Menu*. Другие примеры хороших имен модулей: *Panel*, *Alert*, *Collapsible-section* и *Form-control*. Это поможет, если вы с самого начала имеете общее представление о дизайне сайта. Допустим, известно, что существует два несвязанных элемента пользовательского интерфейса, которые могут быть названы *Tile*, в таком случае вы должны дать им более точные имена, такие как *Media-tile* и *Headline-tile*.

Некоторые разработчики используют в именах модулей по два слова, чтобы они не были слишком общими: невозможно знать, когда потребуется абсолютно новый модуль *Tile*. Если существующий модуль *Tile* назван более точно, вы будете более свободны в том, как назвать новый модуль.

Давая имена классам вариантов вашего модуля, вы должны следовать тем же принципам. Например, если у вас есть модуль *Button*, не называйте дочерние классы красного и голубого вариантов *button--red* и *button--blue*. В будущем дизайн сайта может измениться, и вы не знаете наверняка, что эти цвета останутся. Попытайтесь найти значимые имена, такие как *button--danger* и *button--success*.

Модификаторы с более относительными значениями, такими как *Large* или *Small*, тоже неидеальны, но могут подойти. Тем не менее избегайте точных имен модификаторов, таких как *button--20px*.

9.4. Вспомогательные классы

Иногда вам нужен будет класс для того, чтобы сделать с элементом что-то простое, но нестандартное, например выровнять текст по центру, сделать элемент плавающим слева или добавить очистку. Такие классы называются *вспомогательными*.

В каком-то смысле вспомогательные классы похожи на компактные модули. Но такие классы должны быть узкоспециализированными. Я предпочитаю, чтобы они стояли ближе к концу таблицы стилей, после всех модулей.

В листинге 9.14 показаны четыре вспомогательных класса. Каждый из них выполняет определенное действие: выравнивает текст по центру, делает элемент плавающим слева, выполняет очистку (включая свойство *float*) и скрывает элемент.

Листинг 9.14. Примеры вспомогательных классов

```

.text-center {
  text-align: center !important;
}

.float-left {
  float: left;
}

.clearfix::before,
.clearfix::after {
  content: " ";
  display: table;
}

.clearfix::after {
  clear: both;
}

.hidden {
  display: none !important;
}

```

← Выравнивает текст по центру
внутри контейнера

← Делает элемент
плавающим слева

← Выполняет очистку

← Скрывает элемент
на странице

Да, я использовал объявление `!important`. Дважды. Вспомогательные классы — единственные, в которых можно задействовать данную аннотацию. В некоторых случаях это даже предпочтительно. Вне зависимости от того, где использован вспомогательный класс, он будет работать. Я гарантирую, что, в какой бы элемент вы ни добавляли класс `text-center`, вы так поступаете, потому что хотите выровнять текст по центру, не перезаписывая другие стили. Применение аннотации `!important` позволяет это сделать.

Чтобы увидеть эти классы в действии, добавьте их в элемент на странице. Запись `<div class="text-center">` выравнивает любой текст по центру внутри данного элемента. Добавьте класс `float-right` в тег ``, чтобы сделать элемент плавающим справа, и класс `clearfix` в контейнер элемента `img`, чтобы он поддерживал свойство `float`.

Вспомогательные классы призваны служить скорой помощью. Вам не нужен целый модуль, если необходимо сделать одно простое изменение на странице. При возникновении такой ситуации используйте вспомогательный класс. Но все же не увлекайтесь. В большинстве сайтов вам, скорее всего, не понадобится более дюжины таких классов.

9.5. Методологии CSS

Понятие модульного CSS-кода появилось несколько лет назад. Разработчики, испытывавшие проблемы увеличения масштаба CSS-кода в больших проектах, начали разрабатывать методологии, позволяющие повторно использовать код и сократить число ошибок. С тех пор новые методологии опираются на эти идеи. Они не сопровождаются

какими бы то ни было библиотеками и технологиями, но предоставляют ряд рекомендаций, призванных помочь организовать CSS-код.

Многими знаниями, озвученными в данной главе, я обязан людям, шедшим по этому пути до меня. Если вы примените изложенные здесь советы, значит, последуете большинству этих методологий.

Некоторые из методологий просты и предлагают всего несколько рекомендаций. Другие более жесткие и обеспечивают строгую организацию ваших стилей. Все они имеют собственную терминологию и соглашения в области именования. Но в целом все возвращаются к модульному подходу в CSS:

- ❑ OOCSS — Object-oriented CSS, создана Николь Салливан (Nicole Sullivan): github.com/stubbornella/oocss/wiki;
- ❑ SMACSS — Scalable and Modular Architecture for CSS, создана Джонатаном Снуком (Jonathan Snook): smacss.com;
- ❑ БЭМ — «Блок, элемент, модификатор», разработана компанией «Яндекс»: ru.bem.info/methodology/;
- ❑ ITCSS — Inverted Triangle CSS, создана Гарри Робертсом (Harry Roberts): www.creativebloq.com/web-design/manage-large-css-projects-itcss-101517528.

Порядок данного списка более или менее хронологичен. Он также соответствует растущему объему структуры. Методология OOCSS базируется всего на нескольких принципах, в то время как ITCSS имеет специфические правила именования классов и категоризации стилей, SMACSS и БЭМ — что-то среднее.

В данной главе я рассказал вам о трех главных разделах таблицы стилей: базовых правилах, правилах модулей и вспомогательных классах. Методология SMACSS добавляет раздел правил верстки страницы (боковая панель, подвал и, возможно, CSS-сетка). ITCSS делит категории на семь разделов.

Если вам интересны эти методологии, изучите их. Их терминология различается, но во многих отношениях они дополняют друг друга. Выберите понравившуюся методологию или разработайте собственный подход к модульному CSS-коду. Если работаете в команде, найдите компромисс. Если вам не понравился показанный мной синтаксис присвоения имен, рекомендуемый БЭМ, найдите другой или придумайте новый, который удовлетворит ваши потребности.

Альтернативные подходы в JavaScript

Создание модульных стилей в больших командах требует определенной твердости характера, так как нужно обеспечить, чтобы все члены команды соблюдали одни и те же соглашения. К тому же необходимо, чтобы новые имена модулей не конфликтовали с другими. Решая эти проблемы, некоторые сообщества разработчиков начали экспериментировать с альтернативными подходами к модульному CSS-коду. В ходе этого они обратились к JavaScript и разработали подход, известный как *встроенные стили*, или *CSS в JS*.

Вместо того чтобы полагаться на соглашения о присвоении классам имен, данный подход требует, чтобы код JavaScript создавал имена классов и обеспечивал уникальность или применял все стили на странице с помощью HTML-атрибута `style`. Для выполнения этой задачи появилось множество библиотек JavaScript, самые популярные из которых — Aphrodite, Styled Components и CSS Modules. Большинство из них связаны с конкретными JavaScript-фреймворками или наборами инструментов, такими как WebPack.

Данный подход по-прежнему экспериментальный и немного спорный, но о нем стоит знать, особенно если вы разрабатываете одностраничные приложения (single page applications (SPA)). Он работает только в приложениях, полностью визуализируемых с помощью JavaScript-фреймворков, таких как ReactJS. Следование данному пути требует некоторых компромиссов и ограничивает вас необходимостью выбора конкретного инструментария. Это неидеальное решение, но для некоторых оно оказалось удачным.

Итоги главы

- ❑ Разбивайте CSS-код на компактные, пригодные для повторного использования модули.
- ❑ Никогда не пишите стили, которые влияли бы на другие модули и меняли их внешний вид.
- ❑ Применяйте классы вариантов, чтобы создать множество версий одного и того же модуля.
- ❑ Разделите большие конструкции на меньшие модули, проектируйте страницы, соединяя множество модулей.
- ❑ Группируйте все правила модулей в таблице стилей.
- ❑ Используйте соглашения о присвоении имен, такие как двойные дефисы и двойные подчеркивания, чтобы сделать структуру модулей более простой для понимания.

10

Библиотеки компонентов

В этой главе

- Составление библиотеки компонентов для документирования модулей.
- Вовлечение библиотеки компонентов в процесс разработки.
- Подход CSS First для написания стилей.
- Безопасное редактирование и удаление CSS.
- Применение CSS-фреймворков, таких как Bootstrap.

После того как вы начнете писать CSS-код с применением модулей, станет меняться подход к созданию веб-страниц и веб-приложений. Поначалу страницы могут показаться одинаковыми. Но в какой-то момент, верстая какую-нибудь из них, вы поймете, что уже располагаете необходимыми для нее модулями. К примеру, если вам требуется медиаобъект, раскрывающееся или навигационное меню, а вы его создали ранее, значит, уже есть готовые для этого стили. Нужно будет только добавить корректное имя класса для некоторых правильно упорядоченных элементов.

Поскольку модули используются повторно, вы сможете компоновать эти части страницы без добавления нового кода в файл CSS-стилей. И увидите, что вместо написания HTML-страницы и применения к ней стилей можно брать уже существующие модули и использовать их на новой странице. По мере развития проекта все меньше нового CSS-кода придется писать. Вместо создания стилей заново вы сможете оперировать арсеналом модулей, готовых для применения в таблице стилей.

В больших проектах широко распространена практика объединения наборов документации для обеспечения таким арсеналом. Подобный набор документации называется *библиотекой компонентов* или *руководством по стилям*. Это не часть сайта или приложения, а отдельный набор HTML-страниц, демонстрирующих каждый CSS-модуль. Это инструмент разработки, который вы и ваша команда будете использовать для создания сайта.

В данной главе я покажу, как составлять библиотеку компонентов. Существует множество инструментов, которые могут в этом помочь (хотя это можно сделать и без их привлечения, если проявить инициативу). Расскажу об одном из них, называемом KSS. Но основная тема главы — принципы, применимые независимо от рабочего инструмента.

После того как вы запустите свою библиотеку, я опишу ее ключевые преимущества и то, как она способна усовершенствовать процесс разработки, особенно больших проектов. Эта глава — продолжение главы 9, поэтому, если вы ее пропустили, рекомендую все же обратиться к ней перед началом чтения главы 10.

Библиотека компонентов или руководство по стилям?

Некоторые библиотеки компонентов называют руководствами по стилям (стайлгайдами). Но между ними существуют некоторые различия.

Руководство по стилям охватывает не только технические инструкции по применению модулей, но и субъективные рекомендации о том, когда и для чего вы должны или не должны использовать их. Эти рекомендации обычно направляют работу разработчика для соблюдения в продукте требований торговой марки.

Если инструкции брендинга соотносятся с проектом, можете добавить их в свою библиотеку. Но это углубление уже в сферу маркетинга, а не разработки. И так как в этой главе я намерен раскрывать технические аспекты документа, то буду пользоваться термином «*библиотека компонентов*».

10.1. Введение в KSS

Я уже упоминал, что не хочу слишком долго обсуждать инструменты. Самые важные принципы CSS применимы независимо от инструментария, и я сосредоточусь на них, а не на ваших препроцессорах или системах сборки.

Составление библиотеки компонентов возможно и без вовлечения инструментов, однако значительно упрощается с небольшой их помощью. Некоторые библиотеки могут в этом поспособствовать: введите в строке поиска *style guide generator* (генератор руководства по стилям) — и найдете предостаточно материала. Среди генераторов не существует явного лидера, но один из них уверенно занимает первые позиции — это KSS. Это расшифровывается как *Knyle Style Sheets* (где *Knyle* — это отсылка к имени автора, Кайла Нита (Kyle Neath)).

Я покажу, как установить и запустить KSS. Как только вы его настроите, он начнет сканировать файлы стилей на наличие блоков комментариев с особой аннотацией *Styleguide*. Эти аннотации понадобятся для описания цели и применения всех модулей, KSS использует его для составления HTML-документации. Комментарии

могут содержать также фрагменты HTML-кода, демонстрирующие необходимую разметку для визуализации модуля. KSS это нужно для демонстрации модуля в документации (рис. 10.1).



Рис. 10.1. Визуализация KSS-документации для модуля Dropdown

На этом рисунке слева вы видите меню, содержащее разделы библиотеки компонентов. Справа — документация для модуля Dropdown, подобная той, которую вы составляли в главе 9. Здесь показаны визуализация раскрывающегося меню и HTML-код для его составления. С таким руководством те, кто имеет опыт работы с HTML, легко воспроизведут эту разметку на странице, и ваш файл стилей можно будет применить для нее.

10.1.1. Установка KSS

Изначально KSS был написан как Ruby-приложение. Но так как мы находимся в пространстве фронтенд-разработки, вероятно, вы лучше знакомы с JavaScript, поэтому я проведу вас через установку Node.js реализации KSS.

Если у вас не установлен Node.js, можете найти его здесь: nodejs.org. Скачайте и установите его, следуя инструкции. Node поставляется с менеджером пакетов `npm`, который используется для установки KSS. Я покажу необходимые для этого команды, но если хотите больше узнать про `npm` или нужна помощь в устранении проблем, посетите страницу [docs.npmjs.com/getting-started/Инициализация проекта](https://docs.npmjs.com/getting-started/Инициализация_проекта).

Как только Node и `npm` будут установлены, создайте в файловой системе каталог `pattern-library` для своего проекта. Перейдите к нему в оболочке командной строки. Выполните команду `npm init -y`, чтобы инициализировать новый проект. Флаг `-y` автоматически настраивает стандартную конфигурацию имени, прав и других

значений для проекта. (Если вы не укажете флаг `-u`, то менеджер `npm` запросит ввод этих данных.)

Во время инициализации проекта менеджер `npm` создает файл с именем `package.json` (листинг 10.1). В нем хранятся `npm`-метаданные для проекта в формате JSON.

Листинг 10.1. Сгенерированный файл `package.json`

```
{
  "name": "pattern-library",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Номер версии →

Краткое название вашего `npm`-проекта

Длинное описание проекта можно поместить здесь

Когда проект будет инициализирован, вы сможете установить KSS в качестве *зависимости*. Выполните команду `npm install --save-dev kss` в оболочке командной строки. В результате этого будут выполнены два действия: создан каталог `node_modules` в проекте, где устанавливаются KSS и его зависимости, и добавлено `kss` в список зависимостей разработки (`devDependencies`) в файле `package.json`.

Добавление конфигурации KSS

Для KSS необходим конфигурационный файл. Он будет передавать KSS пути к некоторым каталогам и файлам, которые требуются для составления впоследствии библиотеки компонентов. Создайте файл в каталоге своего проекта и присвойте ему имя `kss-config.json`. Перенесите в него содержимое листинга 10.2.

Листинг 10.2. Конфигурационный файл для KSS (`kss-config.json`)

```
{
  "title": "My pattern library",
  "source": [
    "./css"
  ],
  "destination": "docs/",
  "css": [
    "../css/styles.css"
  ],
  "js": [
    "../js/docs.js"
  ]
}
```

Путь к каталогу исходных CSS-файлов, который KSS будет сканировать

Путь к расположению будущей библиотеки компонентов

Путь к файлу стилей (относительно каталога назначения)

Путь к некоторому JavaScript-файлу (относительно каталога назначения)

Путь к каталогу исходных файлов показывает KSS, где искать CSS-файлы, которые он сканирует на наличие комментариев для документации. Затем он использует эти комментарии для создания страниц библиотеки компонентов в каталоге назначения.

Файлы, перечисленные под ключами `css` и `js`, будут добавлены к страницам библиотеки компонентов. Я выполнил их конфигурацию для каталогов `css` и `js` соответственно. Поэтому создадим эти каталоги и там же разместим исходные файлы `css/styles.css` и `js/docks.js`. Оставьте пока файлы пустыми, напишете в них код позднее.

ПРИМЕЧАНИЕ

В нашем случае файл стилей, указанный под ключом `css`, находится в том же каталоге, что и каталог назначения. Когда будете использовать препроцессор, такой как `Sass` или `Less`, каталог исходных файлов должен будет указывать на `Sass`- или `Less`-файлы, а ключ `css` — ссылаться на соответствующий CSS-файл.

В завершение настройки добавьте в файл `package.json` команду, которая инструктирует KSS создать библиотеку компонентов. Добавьте новый элемент в раздел `scripts` файла `package.json` с кодом, соответствующим листингу 10.3.

Листинг 10.3. Добавление скрипта для создания библиотеки в файл `package.json`

```
"scripts": {
  "build": "kss --config kss-config.json",
  "test": "echo \"Error: no test specified\" && exit 1"
},
```

Определение команды
создания библиотеки

Так в пакет добавляется команда создания библиотеки. Теперь при выполнении команды `npm run build` в оболочке NPM запустится KSS (из каталога `node_modules`), которому будет передан путь к созданному вами конфигурационному файлу KSS. Выполните команду `npm run build` еще раз, и вы увидите: `Error: No KSS documentation discovered in source files` (Ошибка: KSS не обнаружил нужную документацию). Передадим ее.

10.1.2. Написание KSS-документации

Включите в свою библиотеку компонентов некоторые модули из главы 9. Первым из них будет `Media` (рис. 10.2). Когда KSS составит эту страницу, он добавит пункт `Media` в расположенное слева оглавление и отобразит документацию справа.

KSS ищет в вашем файле стилей комментарии, соответствующие определенной структуре. Она предполагает наличие названия (обычно это имя модуля), текста

описания, примера HTML-кода и аннотации **Styleguide**, показывающей, где модуль располагается в оглавлении. Все эти элементы для удобства должны разделяться пустыми строками. В общем-то финальная KSS-аннотация — это все, что нужно для KSS, но лучше указывать и остальные данные.

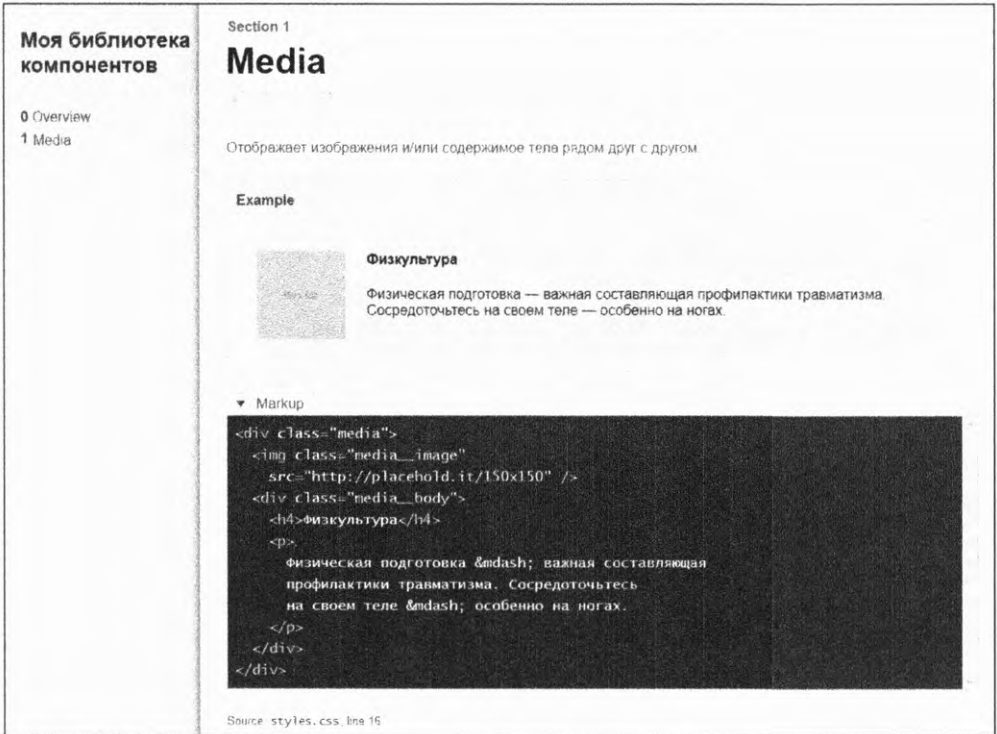


Рис. 10.2. Документация для модуля Media

Добавьте в файл стилей в `css/styles.css` код из листинга 10.4. В нем содержатся некоторые базовые стили и модуль Media. Над стилями модуля находится блок комментариев для KSS.

Листинг 10.4. Модуль Media с комментарием для KSS-документации

```
:root {
  box-sizing: border-box;
}

*,
*::before,
*::after {
  box-sizing: inherit;
}
```



```
body {
  font-family: Helvetica, Arial, sans-serif;
}
```

```
/*
Media
```

← Название (название модуля или раздела документации)

Отображает изображение и/или содержимое тела рядом друг с другом.

← Описание модуля и его применение

Markup:

← HTML-пример, демонстрирующий применение модуля

```
<div class="media">
  
  <div class="media__body">
    <h4>Физкультура</h4>
    <p>
      Физическая подготовка &mdash; важная составляющая
      профилактики травматизма. Сосредоточьтесь
      на своем теле &mdash; особенно на ногах.
    </p>
  </div>
</div>
```

Styleguide Media

← Styleguide-аннотация, которая добавляет все это в оглавление как пункт Media

```
*/
.media {
  padding: 1.5em;
  background-color: #eee;
  border-radius: 0.5em;
}
.media::after {
  content: "";
  display: block;
  clear: both;
}

.media__image {
  float: left;
  margin-right: 1.5em;
}

.media__body {
  overflow: auto;
  margin-top: 0;
}

.media__body > h4 {
  margin-top: 0;
}
```

Теперь выполните команду `npm run build` в оболочке командной строки. KSS создаст каталог `docs`, содержащий файл `section-media.html`. Откройте эту страницу в браузере, чтобы увидеть библиотеку компонентов. KSS также выведет предупреждение: `No homepage content found in homepage.md` (В `homepage.md` не обнаружен контент главной страницы). Я покажу, как решить эту проблему. Но сейчас рассмотрим комментарии для документации. Первые строки выглядят так:

```
/*
Media
```

Отображает изображение и/или содержимое тела рядом друг с другом.

Первая строка комментария определяет название (название модуля) раздела документации, а затем некий текст описывает назначение модуля. Это описание можно выполнить в формате *markdown*, чтобы добавить необходимое форматирование. Описание может состоять из нескольких абзацев.



markdown — распространенный текстовый формат, который поддерживает аннотации для базового форматирования. Окружите текст символами звездочки — и он будет выполнен курсивом, окружите его обратными кавычками (`), чтобы отформатировать как код. Обратитесь к странице github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet, чтобы прочитать полное руководство.

Когда создаете модуль, используйте описание, чтобы передать другим разработчикам все, что им нужно знать, чтобы его применить. Иногда достаточно одного предложения. Но, бывает, нужно указать, что модуль задействует JavaScript или задуман для применения вместе с другим модулем. Это ваша документация для работы с файлами стилей.

После описания идет аннотация `Markup:`, а вслед за ней — блок HTML-кода, демонстрирующего применение модуля. KSS визуализирует этот код для библиотеки компонентов, чтобы читатель мог его просмотреть. Затем выводит HTML-код в текстовом формате, чтобы его можно было скопировать:

Markup:

```
<div class="media">
  
  <div class="media_body">
    <h4>Физкультура</h4>
    <p>
      Физическая подготовка &mdash; важная составляющая
      профилактики травматизма. Сосредоточьтесь
      на своем теле &mdash; особенно на ногах.
    </p>
  </div>
</div>
```

Точность текста и изображения, применяемых в этом примере, неважны: они просто показывают разработчику, как он работает. В данном случае я использовал демонстрационное изображение с сайта placeholder.it. С помощью этого модуля разработчик может добавлять любой контент.

ВНИМАНИЕ!

Важно не оставлять пустых строк в середине HTML-кода, так как для KSS они означают завершение раздела разметки.

Последняя строка KSS-комментария должна содержать аннотацию Styleguide, за которой идет метка (латинскими буквами) для оглавления (здесь — **Media**):

```
Styleguide Media
*/
```

Это должна быть последняя строка комментария. Без нее KSS проигнорирует весь блок комментариев.

Когда обновляете файл стилей, обновите и документацию к нему. Наличие документации прямо в исходном коде значительно это упрощает. Добавляя новый модуль, напишите и блок документации к нему. После внесения изменений выполните команду `npm run build`, чтобы сгенерировать свежую версию библиотеки компонентов.

ВНИМАНИЕ!

KSS не удаляет старые страницы при генерации новых. Если вы переименуете или переместите часть документации в исходном коде, то соответствующий файл в каталоге останется на своем месте наряду с новым. После обновления страницы в браузере убедитесь, что вы не перезагружаете старый файл.

По причине того, что библиотека компонентов существует вместе со стилями, которые она документирует, любой разработчик, имеющий доступ к файлу стилей, получит доступ и к документации. Возможно, вы захотите разместить библиотеку компонентов также где-нибудь в Интернете, чтобы команда разработчиков могла работать с ней.

10.1.3. Документирование вариаций модуля

ЗадOCUMENTИРУЕМ другой модуль (листинг 10.5) — **Buttons** из главы 9. Он предлагает несколько вариантов: два цвета и два размера. KSS позволяет указывать множество вариантов, отображая каждый из них в библиотеке компонентов. Это будет выглядеть примерно так, как показано на рис. 10.3.



Рис. 10.3. Модуль Buttons с вариантами

Комментарий для документации к этому модулю будет подобен предыдущему, но здесь вы после разметки добавите раздел, чтобы указать каждый из модификаторов (см. листинг 10.5). Он представляет собой список классов модификаторов, каждый из которых сопровождается дефисом и описанием. Также вы добавите к примеру разметки аннотацию `{{modifier_class}}`, указывающую положение классов модификаторов.

KSS сканирует список определенных вами классов модификаторов и отображает их в библиотеке компонентов. `{{modifier_class}}` показывает, где нужно разместить эти классы. (Если вы знакомы с шаблонизатором `handlebars`, то, возможно, синтаксис покажется знакомым. Это то, что KSS использует за кадром при визуализации

модуля.) Выполните команду `npm run build`, чтобы заново собрать библиотеку компонентов и просмотреть документацию в браузере.

Листинг 10.5. Модуль Buttons и документация

```
/*
Buttons

Кнопки бывают разного размера и цвета.
Вы можете использовать кнопки разных
видов на одной странице.

Markup:
<button class="button {{modifier_class}}"> ← Указывает применение
    нажмите меня                             классов модификаторов
</button>
```

```
.button--success — зеленая кнопка оповещения
.button--danger — красная предупреждающая кнопка
.button--small — маленькая кнопка
.button--large — большая кнопка
```

Перечисляет все доступные
классы модификаторов

Styleguide Buttons

```
*/
.button {
  padding: 1em 1.25em;
  border: 1px solid #265559;
  border-radius: 0.2em;
  background-color: transparent;
  font-size: 0.8rem;
  color: #333;
  font-weight: bold;
}

.button--success {
  border-color: #cfe8c9;
  color: #fff;
  background-color: #2f5926;
}

.button--danger {
  border-color: #e8c9c9;
  color: #fff;
  background-color: #a92323;
}

.button--small {
  font-size: 0.8rem;
}

.button--large {
  font-size: 1.2rem;
}
```

СОВЕТ

Перезапускать KSS каждый раз после внесения изменений может оказаться утомительным. Если вы используете менеджер задач, такой как Gulp, для своих проектов, предлагаю настроить задачу, которая следит за изменениями и автоматически обновляет KSS. Большинство менеджеров задач оснащены плагином или другим механизмом, способным на это.

Теперь в оглавлении библиотеки компонентов (`docs/index.html`) должно оказаться три пункта: Overview, Buttons и Media. Второй и третий перенаправляют на те части документации, которые вы написали. Ссылка Overview недействительна, так как начальная страница еще не создана. Это и есть причина появления предупреждения «Отсутствует содержимое начальной страницы».

10.1.4. Создание начальной страницы

Добавим начальную страницу в библиотеку компонентов. В каталоге `css` создайте файл `homepage.md`. Это будет файл с разметкой, который послужит введением для библиотеки компонентов. Скопируйте в него код из листинга 10.6.

Листинг 10.6. Разметка в начальной странице

Библиотека компонентов  Заголовок страницы

Это коллекция всех модулей в нашей библиотеке таблиц стилей.
Можете использовать любые модули при оформлении страницы.

Теперь выполните команду `npm run build`, и предупреждение об отсутствии содержимого начальной страницы должно исчезнуть. Когда вы откроете файл `docs/index.html` в своем браузере, то увидите содержимое библиотеки.

В работе над проектами используйте такую страницу для введения в библиотеку компонентов. Вы можете составить инструкции о том, как перенести на страницу стили, отобразить корректные веб-шрифты (см. главу 13) или сделать что-то еще, что позволит разработчикам ознакомиться с тем, как применяются файлы стилей.

Поскольку вы открываете библиотеку компонентов прямо с диска, то могли заметить, что ссылка Overview в содержании все еще не работает. Так происходит, потому что KSS перенаправляет ее на адрес `./`, а не на файл `index.html`. Исправить ситуацию можно, запуская библиотеку на сервере, чтобы адрес `./` перенаправлял на файл `index.html` в браузере. Выполните это самостоятельно, работая с привычным инструментарием. Если не знаете, с чего начать, попробуйте найти ответ по адресу: www.npmjs.com/package/http-server.

10.1.5. Документирование модулей, которым требуется JavaScript

Некоторые модули предполагают работу с поддержкой JavaScript-сценариев. В таких случаях зачастую полезно будет поместить немного простого JavaScript-кода на страницу, чтобы продемонстрировать поведение модуля. Для этого не обязательно

включать полнофункциональную JavaScript-библиотеку в библиотеку компонентов. В большинстве случаев достаточно переключать различные классы состояний. Вы уже добавили в файл `kss-config.json` конфигурацию, которая размещает JavaScript-файл на странице:

```
"js": [
  "../js/docs.js"
]
```

KSS добавит на страницу указанные в этом массиве `js`-скрипты. Можете прописать для них код, который обеспечивает минимальную функциональность модулей. Чтобы продемонстрировать это, нужно добавить к файлу стилей модуль `Dropdown` (см. главу 9) и некоторую документацию к нему (листинг 10.7). Также следует добавить JavaScript-код, чтобы при щелчке на переключателе список пунктов меню открывался и закрывался. Модуль будет работать в библиотеке компонентов, чтобы продемонстрировать свою предполагаемую функциональность (рис. 10.4).

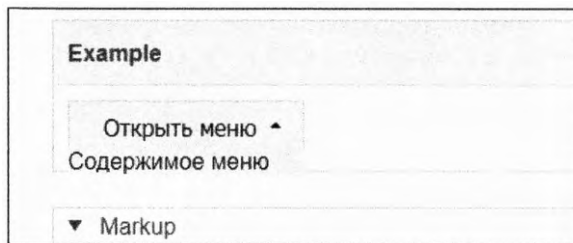


Рис. 10.4. Работающее `Dropdown`-меню в библиотеке компонентов (обратите внимание на то, что содержимое меню не отформатировано, потому что, как предполагается, это задача другого модуля)

Начните с добавления стилей и документации, показанной в листинге 10.7, в свой файл стилей. Важно задать некое руководство, посвященное тому, как JavaScript должен работать. Разработчики будут использовать эту информацию для сборки целого сайта или веб-приложения. Им нужно довольно много информации, чтобы сделать это корректно. Добавьте приведенный далее код в CSS-файл.

Листинг 10.7. Модуль `Dropdown` и документация к нему

```
/*
Dropdown

Меню. Нажмите кнопку, чтобы открыть/закрыть его.

Используйте JavaScript-код для класса is-open, чтобы открывать/закрывать меню.

Markup:
<div class="dropdown">
  <button class="dropdown__toggle">Открыть меню</button>
  <div class="dropdown__drawer">
```

Предоставляет инструкции о том, как разработчику следует использовать JavaScript-код для этого модуля

Пример разметки

```

    Содержимое меню
  </div>
</div>

Styleguide Dropdown
*/
.dropdown {
  display: inline-block;
  position: relative;
}

.dropdown__toggle {
  padding: 0.5em 2em 0.5em 1.5em;
  border: 1px solid #ccc;
  font-size: 1rem;
  background-color: #eee;
}

.dropdown__toggle::after {
  content: "";
  position: absolute;
  right: 1em;
  top: 1em;
  border: 0.3em solid;
  border-color: black transparent transparent;
}

.dropdown__drawer {
  display: none;
  position: absolute;
  left: 0;
  top: 2.1em;
  min-width: 100%;
  background-color: #eee;
}

.dropdown.is-open .dropdown__toggle::after {
  top: 0.7em;
  border-color: transparent transparent black;
}

.dropdown.is-open .dropdown__drawer {
  display: block;
}

```

Правила модуля
Dropdown (из главы 9)

Выполнение команды `npm run build` соберет эту документацию, но на данный момент она не интерактивна. Чтобы оживить ее, добавим JavaScript-код в `js/docs.js`. Поместите в этот файл код из листинга 10.8.

Указанный скрипт переключает класс `is-open` в списке при каждом нажатии кнопки переключателя. Полноценная реализация на сайте потребует больше кода для работы без каких-либо временных задержек или для закрытия меню при щелчке кнопкой мыши за его пределами. Хотя в библиотеке компонентов можно хранить и минимальный набор кода, постарайтесь корректно оформить открытое и закрытое

состояния. Справившись с этим, сможете сконцентрироваться на проблеме тонкостей правильной работы JavaScript извне библиотеки компонентов.

Листинг 10.8. Минимальный JavaScript-код для демонстрации модуля

```
(function () {  
  var dropdowns = document.querySelectorAll('.dropdown_toggle');  
  Array.prototype.forEach.call(dropdowns, function(dropdown) {  
    dropdown.addEventListener('click', function (event) {  
      event.target.parentNode.classList.toggle('is-open');  
    });  
  });  
})();
```

Определение всех экземпляров кнопки dropdown_toggle

Добавление слушателя события щелчка кнопкой мыши каждому экземпляру

Переключение класса is-open на элементе dropdown

10.1.6. Упорядочение контента библиотеки компонентов по разделам

Можете продолжать добавлять модули из главы 9 в файл стилей, снабжая их соответствующей документацией. Я не буду объяснять весь процесс, так как вы в основном его уже понимаете.

Последнее, что нужно сделать, — упорядочить библиотеку компонентов. Меню на рис. 10.2 выглядит прекрасно, но лишь до тех пор, пока в нем всего несколько элементов. Как только проекты начнут расти, понадобится рассортировать модули по категориям, чтобы в них было легко ориентироваться.

Добавим документацию для вспомогательных классов. Каждый из них должен быть раскрыт и показан отдельно, поэтому их нужно сгруппировать. В листинге 10.9 показано, что создается новый раздел **Utilities**, каждый его класс добавляется в подразделы, чтобы меню выглядело так, как показано на рис. 10.5.

Чтобы создать подразделы, используйте точку в styleguide-аннотации. Вы будете применять эти аннотации следующим образом: `Styleguide Utilities.clearfix`. В результате блок документации будет помещен в подраздел класса `clearfix` раздела `Utilities`.



Рис. 10.5. Три подраздела в разделе Utilities

ПРИМЕЧАНИЕ

KSS поддерживает до трех уровней вложенности (к примеру, `Utilities.alignment.text-center`).

Включите код из листинга 10.9 в свой файл стилей. С его помощью вы добавите три класса модуля `Utilities` (`text-center`, `float-left` и `clearfix`) и их комментарии для документации. Я также указал аннотацию `weight`, которая отвечает за порядок отображения подразделов.

Листинг 10.9. Группировка документации в одну категорию

/*
Центрирование текста

Выравнивание текста по центру блока с помощью свойства text-center

Markup:
<p class="text-center">Центрированный текст</p>

Weight: 1

```
Styleguide Utilities.text-center
*/
.text-center {
  text-align: center !important;
}
```

/*
Выравнивание по левому краю

Для выравнивания контента по левому краю используйте свойство float-left

Weight: 3

```
Styleguide Utilities.float-left
*/
.float-left {
  float: left;
}
```

/*
Класс clearfix

Назначьте элементу класс clearfix для сброса обтекания плавающего контента

Markup:
<div class="clearfix">
 floated
</div>

Weight: 2

```
Styleguide Utilities.clearfix
*/
.clearfix::before,
.clearfix::after {
  content: " ";
  display: table;
}
```

```
.clearfix::after {
  clear: both;
}
```

Точечная нотация нужна, чтобы поместить каждый из блоков документации в одну группу

Аннотация Weight необходима для управления порядком отображения подразделов

Точечная нотация нужна, чтобы поместить каждый из блоков документации в одну группу

Назначая каждому из вспомогательных классов расположение в одной категории, вы группируете их. Теперь обновите библиотеку компонентов, чтобы отобразить элемент **Utilities** в оглавлении. Выберите этот пункт, чтобы просмотреть все созданные подразделы.

ВНИМАНИЕ!

Styleguide-аннотация чувствительна к регистру. Когда помещаете несколько подразделов в одну категорию, убедитесь, что буквы названия категории набраны в правильном регистре. Иначе KSS создаст отдельные разделы, например, **Utilities** и **utilities**.

По умолчанию разделы и подразделы в библиотеке компонентов KSS упорядочиваются по алфавиту. Можете изменить это поведение с помощью аннотации **weight**. KSS выстраивает разделы согласно их весу (**weight**): чем он выше, тем ниже позиция в списке. Указывайте веса для того, чтобы обозначить важные разделы и задать их положение среди других важных разделов или, к примеру, управлять порядком перечисления подразделов в пределах родительского раздела.

Теперь вы знакомы с ключевыми функциями KSS. Если хотите расширить свои познания, научитесь точнее контролировать то, как выглядит библиотека компонентов. Скорректируйте внутренний файл стилей или шаблон, используемый для верстки страниц библиотеки компонентов. Более подробную информацию найдете в документации на сайте github.com/kss-node/kss-node.

10.2. Инновационный способ верстки CSS

Библиотеки компонентов не требуются для малых проектов, но их эффективность для крупных бесспорна. Если вы разрабатываете сайт в сотни тысяч страниц, то, вероятно, не сможете отформатировать их с помощью стилей все одновременно. Но, составляя многократно модули и документируя их в одном и том же месте, получите инструментарий, с помощью которого реально конструировать тысячи страниц.

Если вы работаете над большим веб-приложением вместе с группой других разработчиков, то, вероятно, при стилевом форматировании своих компонентов не избежите конфликтующих имен классов и множества дублированных реализаций одних и тех же пользовательских интерфейсов. Но с библиотекой компонентов разработчики смогут находить все стили, воспроизводить их и систематизированно именовать модули, чтобы имена классов не конфликтовали.

Контент-редакторы и разработчики, которые будут использовать вашу библиотеку компонентов, даже не обязаны знать язык CSS — они должны лишь понимать основы HTML. Они могут копировать задокументированные шаблоны и помещать их туда, куда захотят. Модульный CSS — ключ к масштабированию вашего CSS, а библиотека компонентов — это средство их упорядоченного хранения.

10.2.1. Метод CSS First

Библиотека компонентов — это трансформация концепции типичных подходов к CSS. Вместо того чтобы просто брать HTML-страницу и форматировать ее, вы составляете модульные стили и затем с их помощью собираете страницу. Этот подход я называю *разработкой CSS First*. Вместо написания сначала HTML-кода пишете CSS. Вы можете и должны разрабатывать каскадные таблицы стилей в пределах библиотеки компонентов перед применением этих стилей в своем проекте так, чтобы процесс разработки выглядел следующим образом.

1. При создании страницы составьте набросок или макет некоторых общих идей о том, как страница должна выглядеть.
2. Обратитесь к библиотеке компонентов за модулями, которые могут обеспечить то, что нужно для страницы, и используйте их. Начните с поверхности (разметки главной страницы и контейнеров) и углубляйтесь по ходу работы. Если вы в состоянии скомпоновать целую страницу из существующих модулей, сделайте это. Тогда не придется писать новый CSS-код.
3. Иногда случается, что вам не хватает того, чем библиотека компонентов не располагает. Вы будете чаще сталкиваться с этим в начале проекта и намного реже — позднее. Иногда придется составлять новые модули из имеющихся или создавать новый вариант существующего модуля. Отложите страницу, над которой работаете, и внесите разработку в библиотеку компонентов. Задокументируйте ее и убедитесь, что все работает так, как вы задумали.
4. Вернитесь к странице и примените новый файл стилей, добавьте на страницу новый модуль.

Такой подход обладает рядом преимуществ перед прочими. Во-первых, он обеспечивает для вашего сайта более последовательный интерфейс. И способствует многократному использованию существующих стилей вместо того, чтобы создавать их заново. К примеру, вместо разработки десяти различных страниц с десятью различными наборами стилей вы можете многократно пользоваться одними и теми же определенными типами наборов стилей. Каждый раз нужно будет остановиться и задуматься о том, нужен здесь новый стиль или у вас уже есть подходящий.

Во-вторых, разрабатывая модуль в пределах своей библиотеки компонентов, можно сфокусироваться на изолированной проблеме. В этом случае вы абстрагируетесь от большой картины конкретной страницы и сосредоточитесь на единичной задаче стилового форматирования модуля. Вместо разрешения одной проблемы единственной страницей проще думать о том, где еще пригодится новый модуль. Вы создадите общее многоцветное решение.

В-третьих, с помощью этого подхода члены вашей команды лучше освоят CSS. Разработчик, менее компетентный в области модулей, может передать часть работы более опытному. По завершении работы над модулем компетентный в CSS разработчик может передать другому разработчику ссылку, указывающую на модуль в библиотеке компонентов.

И в-четвертых, этот подход подтверждает актуальность созданной вами документации. Страницы библиотеки компонентов находятся там, где вы проверяете изменения в CSS, что означает: они всегда будут отображать настоящее, корректное поведение. Когда вы редактируете CSS, за документацией не нужно далеко ходить — она совсем рядом, в блоке комментариев. Это упрощает поддержку актуальности документации по мере внесения изменений. (Чуть позже я еще немного расскажу о редактировании модулей.)

Разработчики зачастую интересуются: как писать HTML-код, который легко форматировать? Считаю, что это неправильный вопрос. Лучше спросить: как писать стили, которые можно многократно использовать на любом количестве страниц? Сначала мы пишем CSS, а хорошо структурированный HTML-код придет позже.

10.2.2. Библиотека компонентов в качестве API

Применяя библиотеку компонентов, вы документируете *API* для взаимодействия с CSS. Каждый модуль сопровождается какими-то именами классов и крошечной DOM-структурой. Если определенный раздел HTML будет соответствовать этой структуре, то файл стилей корректно форматировует его (рис. 10.6).

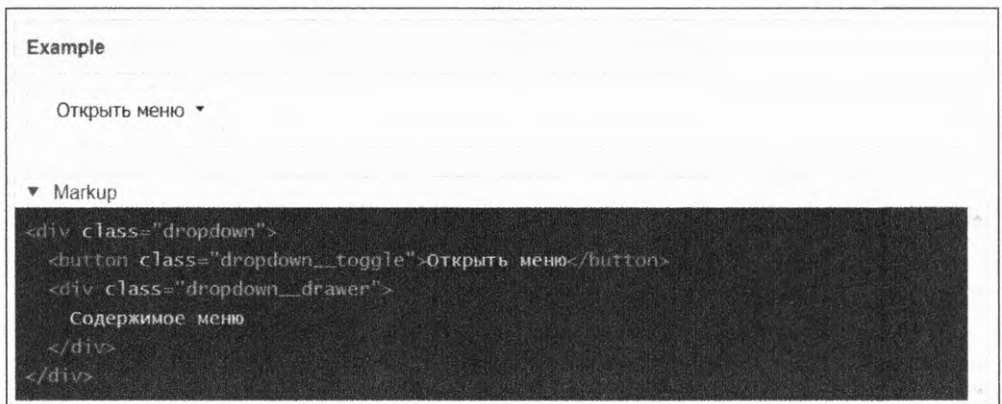


Рис. 10.6. Имена классов и структура HTML являются API



API (Application Programming Interface) — интерфейс программирования приложений. Это набор программных определений, которые описывают, как применять систему или взаимодействовать с ней. Традиционно он включал имена методов и параметров, если речь идет о языке программирования, или URL и параметры запросов — если об HTTP API. Я использую это выражение применительно к CSS для демонстрации того, что имена классов и элементы HTML могут выступать способами выстраивания интерфейса HTML со стилями.

Пример разметки в каждом модуле отображает тип связи между CSS и HTML. Он показывает, как HTML должен выстраивать интерфейс с CSS.

Когда вы составляете модули, этот API наиболее важен, потому что его труднее всего изменить в дальнейшем. HTML преобразовывать легко: можно менять контент внутри элемента. В некоторых случаях реально добавлять и удалять его или даже менять порядок элементов DOM в пределах модуля (убедитесь, что вы четко указали в документации, что элементы необязательны или их порядок может быть изменен). Также HTML может прекратить использовать модуль, сменив его на другой.

Таким же образом возможно изменять и CSS, пока эти изменения согласуются с API. Вы можете добавлять маленькие инструменты, такие как увеличение отступа или настройка цвета, или исправлять появляющиеся баги. Или вносить большие изменения, например переработать медиаобъект так, чтобы он задействовал flex-контейнеры вместо плавающих элементов, или перепроектировать модуль, чтобы он располагался не по горизонтали, а последовательно по вертикали. Пока ключевые элементы API (имена классов и DOM-структура) остаются неизменными, можно как угодно редактировать CSS.

Учитывайте, что вносимые изменения могут повлиять на многие компоненты сайта. Но пока HTML следует API-инструкциям, они будут запланированными. Вы хотите изменить внешний вид всех раскрывающихся меню по всему сайту? Это реально. Поскольку все раскрывающиеся меню на сайте используют один и тот же модуль (и один и тот же API), изменения будут единообразными.

Редактирование существующего модуля

Для наглядности представим гипотетический сценарий, согласно которому вы хотите изменить образ работы модуля. Вместо одной картинки нужно поддерживать две, расположенные по обеим сторонам контента (рис. 10.7).

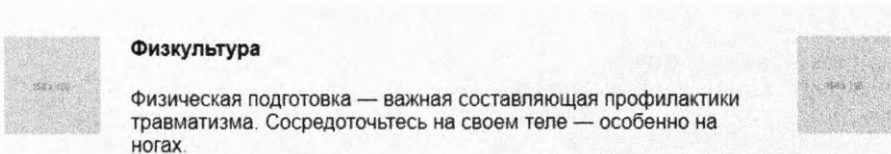


Рис. 10.7. Гипотетический медиаобъект с двумя изображениями

Для этого понадобится внести в CSS некоторые изменения. Пока они согласуются с API (это значит, что медиаобъекты продолжают работать, как предполагалось, с одной картинкой), можете спокойно менять стиль. Вы осуществите эту задумку, переработав модуль для работы с flex-контейнерами. Приступим.

Сначала необходимо добавить пример разметки в блок комментария. Оставьте там же предыдущий пример, чтобы проверить, что после внесения изменений он будет работать как прежде. Добавьте второй пример разметки, чтобы проверить

и новое поведение. Обновите комментарий для документации соответственно листингу 10.10.

Листинг 10.10. Добавление нового примера графики в документацию

```
/*
Media

Отображает изображения и/или содержимое тела
рядом друг с другом.
```

Обновление описания с указанием поддержки нескольких изображений

```
Markup:
<div class="media">
  
  <div class="media__body">
    <h4>Физкультура</h4>
    <p>
      Физическая подготовка &mdash; важная составляющая
      профилактики травматизма. Сосредоточьтесь
      на своем теле &mdash; особенно на ногах.
    </p>
  </div>
</div>
<div class="media">
  
  <div class="media__body">
    <h4>Физкультура</h4>
    <p>
      Физическая подготовка &mdash; важная составляющая
      профилактики травматизма. Сосредоточьтесь
      на своем теле &mdash; особенно на ногах.
    </p>
  </div>
  
</div>
```

Оригинальная разметка остается на своем месте

Добавление нового примера для двух изображений

Styleguide Media

Этот листинг позволит создать два экземпляра модуля в библиотеке компонентов. Пересоберите библиотеку компонентов, чтобы увидеть, как она выглядит. Прежде чем вносить изменения в CSS, вы увидите, что один из них работает, а другой — нет. Затем выполняйте изменения (листинг 10.11), пока оба не заработают (рис. 10.8).

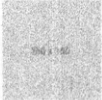
Теперь библиотека компонентов служит вам подстраховкой. Она покажет, выведут ли сделанные изменения существующие на сайте медиаобъекты из строя, и выступит средством проверки валидности кода.

Затем измените CSS-код, чтобы он учитывал новый сценарий. Примените изменения к файлу стилей, чтобы задействовать новый пример, убеждаясь в корректной работе первого.

Example

**Физкультура**

Физическая подготовка — важная составляющая профилактики травматизма. Сосредоточьтесь на своем теле — особенно на ногах.

**Физкультура**

Физическая подготовка — важная составляющая профилактики травматизма. Сосредоточьтесь на своем теле — особенно на ногах.



Рис. 10.8. Два варианта медиаобъектов, показанные в библиотеке компонентов

Листинг 10.11. Модуль Media, переработанный для работы с flex-контейнерами

```
.media {
  display: flex;
  align-items: flex-start;
  padding: 1.5em;
  background-color: #eee;
  border-radius: 0.5em;
}
```

← Смена контейнера на flex-контейнер: media_image и media_body становятся flex-элементами

← Выравнивание элементов по верхнему краю вместо того, чтобы применять заполнение, что предупреждает искажение изображения

```
.media > * + * {
  margin-left: 1.5em;
}
```

← Удаление правого поля изображения и замена его общим полем между всеми flex-элементами

```
.media_body {
  margin-top: 0;
}
```

```
.media_body > h4 {
  margin-top: 0;
}
```

Выполните команду `npm run build` и откройте страницу в библиотеке компонентов. Вы увидите, что изменения успешно внесены. Медиаобъект теперь включает два варианта. Поскольку изменения согласованы с API модуля, будьте уверены, что они ничего не испортили на сайте.

Без модульных стилей и библиотеки компонентов редактирование CSS может превратиться в полный хаос, так как вы не знаете точно, что HTML корректно структурирован по всему сайту и селекторы применены к правильным элементам. С аккуратным, документированным API редактировать можно безболезненно и даже с удовольствием.

Применение *semver* и рефакторинг кода

Иногда не будет возможности внести изменения без модификации API. Это нормально. Такая работа потребует чуть больше ресурсов, но все равно выполнима. Вы могли бы внести нужные изменения, пробежавшись по всему сайту или приложению и работая с каждым экземпляром HTML так, чтобы он соответствовал новому API. Но зачастую стоит объявить модуль устаревшим (следует указать это в документации) и создать новый модуль с нужной функциональностью. Старый модуль продолжит работать там, где он необходим, и начнется миграция на новый модуль, пока оба поддерживаются.

Для решения этой задачи я помечаю версии CSS-кода с помощью семантического менеджера версий *semver*. Изменение номера версии сообщает разработчикам приходу этих изменений.



semver — сокращение от Semantic Versioning (семантический версионер) — система программ версионирования, применяющая числа, разделенные точкой (к примеру, 1.4.2). Числа обозначают основную, вспомогательную и патч-версии в дереве версий. Обратитесь на сайт semver.org.

Когда я выполняю мелкие настройки, такие как исправление багов, я увеличиваю номер патч-версии (к примеру, с 1.4.2 до 1.4.3). Добавляя новый модуль или новую функциональность, согласующуюся с API, или помечая модуль как устаревший, я увеличиваю число вспомогательной версии и сбрасываю значение патч-версии до 0 (к примеру, с 1.4.2 до 1.5.0). Изредка я проверяю файл стилей и удаляю устаревшие модули, переходя к следующей основной версии (к примеру, от 1.4.2 к 2.0.0). Также создаю релиз основной версии, когда вношу значительные проектные изменения, такие как изменение дизайна сайта, даже если API остается прежним.

Существует множество способов использования версионирования. Все зависит от природы проекта, в котором применяются стили. Если вы храните CSS в модуле Node.js или Ruby Gem, возьмите их метод версионирования. А если CSS-код хранится статически на сервере, укажите версию в URL-адресе (example.com/css/1.4.2/styles.css) и храните множество версий одновременно.

Таким образом проект реально настроить для использования любой версии CSS. Вы можете выпустить версию 3.0.0 со значительными изменениями, но веб-приложение продолжит использовать старую версию до тех пор, пока разработчики не смогут проработать и обновить HTML, где действуют старые модули. Выполненные вами изменения в CSS не выведут приложение из строя, пока оно предусмотрено обновляется с новыми версиями файла стилей.

Библиотека компонентов документирует применение файла стилей, но авторы HTML могут оповещать, с какой версией стилей они работают. HTML- и CSS-код разделены. CSS следует разработать в самом начале, перед тем как его можно будет использовать с HTML, но HTML способен управлять переходом к новому файлу стилей. В этом и заключается преимущество разработки методом CSS First.

Такие решения не принимаются изолированно. Нужно обращаться к другим разработчикам команды, когда вы хотите обозначить как устаревшие или удалить те или иные модули. Вам необходим их отзыв о том, какие модули еще жизнеспособны, а какие больше не нужны.

Bootstrap, Foundation и другие фреймворки

Возможно, вы знакомы с некоторыми CSS-фреймворками, предоставляющими подготовленные наборы стилей. Обычно в них входят стили кнопок, форм, меню и какая-нибудь CSS-сетка. Популярные фреймворки — это Bootstrap (getbootstrap.com), Foundation (foundation.zurb.com) и Pure (purecss.io). Существует также бесчисленное множество прочих. Одни являются мощными библиотеками с десятками модулей, а другие компактны и предоставляют только самое необходимое.

Составляя библиотеку компонентов, вы можете почувствовать, что собираете свой фреймворк. И это правда! Каждый фреймворк — это библиотека компонентов, именно поэтому они настолько успешны. CSS в них выполнен с учетом многоцветового использования в разных обстоятельствах. Некоторые придерживаются принципов модульного CSS строже, чем другие, но все они в некоторой степени следуют им. И они всегда версионированы.

Различие между этими фреймворками и вашей библиотекой в том, что фреймворки задуманы универсальными. В библиотеке вы сможете привязать разработку модулей к проекту и обеспечить точное соответствие виду и бренду. А также, если нужно, создать два разных модуля Tile и быстро их адаптировать.

Разработчики часто спрашивают меня, стоит ли им применять фреймворк вроде Bootstrap. Я отвечаю: и да и нет.

Фреймворки полезны для быстрого построения проекта. Не затрачивая много сил, вы можете отформатировать с помощью стилей кнопки, блоки и раскрывающиеся меню. Но, судя по моему опыту, они никогда не предоставят все нужные модули. Вам всегда, за исключением малых проектов, необходимо будет добавлять собственные модули. К тому же с ними зачастую поставляются модули, которые, вероятнее всего, вам не пригодятся.

Если вы хотите использовать знакомый фреймворк, предлагаю взять только нужные части и удалить остальные. Не копируйте на страницу файл `bootstrap.css` как есть. Возьмите только те модули, которые хотите видеть в своем файле стилей (предполагается, что вы не нарушаете ничьих авторских прав). Сделайте эти фрагменты CSS-кода своими собственными.

Добавляя фреймворк на страницу перед файлом стиля, вы увидите, что приходится переписывать множество стилей и расширять фреймворк. Если вместо этого вы просто перенесете необходимое из фреймворка в свой файл стилей, то сможете редактировать стили непосредственно. Это поможет сделать CSS-код страницы более лаконичным, и за ними будет легче следить.

Не используйте фреймворк бездумно, поймите его концепцию. Представьте, что ваша библиотека компонентов — это универсальная библиотека, предназначенная для применения сторонними пользователями. Это поможет содержать стили в состоянии, пригодном для повторного использования, и позволит в будущем вносить изменения без серьезных нарушений работы на странице.

Зачастую CSS выступает только аддитивным языком. Разработчики боятся редактировать или удалять существующие стили, поскольку не знают наверняка, на что могут повлиять эти изменения опосредованно. В основном они модифицируют CSS, добавляя код в конец файла стилей, переопределяя более ранние правила и усложняя специфичность файла, пока он не станет неподдерживаемым.

Но если позаботиться об организации CSS — сделать так, чтобы он хранился модульно и сопровождался библиотекой компонентов, — то вы не попадете в эту ловушку. Вы всегда будете знать, где находятся стили для модуля. Каждый модуль отвечает за что-то одно. А библиотека компонентов помогает разработчикам быть в курсе того, что происходит в файле стилей.

Итоги главы

- ❑ Пользуйтесь таким инструментом, как KSS, для документирования и инвентаризации модулей.
- ❑ Применяйте библиотеку для документирования примеров разметки, вариантов модулей и JavaScript-кода для модулей.
- ❑ Разрабатывайте модули, задействуя подход CSS First.
- ❑ Подумайте об API, который определяет CSS, чтобы нечаянно не спровоцировать какое-либо противоречие с ним.
- ❑ Версионизируйте CSS с помощью инструмента `semver`.
- ❑ Не копируйте бездумно CSS-фреймворк на страницу, берите только необходимое.

Часть IV

Темы повышенной сложности

Важно, чтобы пользовательский интерфейс был отполирован до блеска. Пользователи склонны доверять профессионально выглядящему приложению и проводить больше времени на эстетичном сайте. В заключительных шести главах рассмотрим важные концепции, касающиеся проектирования. Вы узнаете, как, внося небольшие изменения, можно серьезно повлиять на внешний вид сайта.

11

Фоны, тени и режимы смешивания

В этой главе

- Линейный и радиальный градиенты.
- Тени блоков и текста.
- Размер и позиционирование фоновых изображений.
- Использование режимов смешивания для комбинирования фона и контента.

К этому моменту вы уже освоили базовые темы. У вас сформировалось понимание работы с каскадными таблицами стилей. Вы изучили множество приемов разметки. Кроме того, много времени было посвящено организации кода, что упрощает дальнейшую поддержку проектов. Изученных основ достаточно для того, чтобы создать сайт с нуля. Вы могли бы использовать эти знания в своих проектах, и результат был бы вполне приемлемым. Но не стоит на этом останавливаться.

Разница между хорошим и великолепным сайтами кроется в деталях. После того как вы позиционируете элемент и настроите его внешний вид с помощью стилей, попытайтесь взглянуть на него критически. Не будет ли он выглядеть лучше, если его увеличить, уменьшить, изменить расстояние от него до других объектов? Поиграйте немного с цветами. Не лучше ли затемнить? Или сделать светлее? Немного убрать яркость? А если вы работаете по созданному дизайнером макету, соответствует ли ваш проект ему максимально точно? Специалист ведь потратил много времени на проработку деталей. Убедитесь, что не нарушаете законы дизайна.

CSS — это область, где можно проявить свою творческую натуру, особенно в деталях, даже если вы не похожи на многих разработчиков и не считаете себя дизайнером или художником. И пусть вы не работаете с CSS постоянно, время от времени делать это придется. Вот этому и посвящена часть IV книги.

В последних главах этой книги рассматриваются детали, способные превратить сайт в нечто особенное. Я научу вас некоторым дизайнерским приемам, но не волнуйтесь, это не сделает ваше творчество шаблонным. Я сконцентрируюсь на общих правилах. Покажу, как использовать цвета, пространство и анимацию. Если вы хо-

тите, чтобы сайт не только был функциональным, но и выглядел эстетично, тогда вам сюда.

В этой главе я покажу техники, которые улучшат визуальную составляющую страницы. Посмотрите на кнопку, показанную на рис. 11.1. На нее наложены два эффекта: градиентный фон и тень. Они создают иллюзию глубины. Фон плавно переходит от голубого (цвет #57b) сверху к темно-синему внизу. Вы можете этого не осознавать, но комбинация цветов вместе с тенью, отбрасываемой вниз и вправо, создает иллюзию объема.



Рис. 11.1. Кнопка с градиентным фоном и тенью

В данной главе рассказывается о создании градиентных фонов и теней, а также приводятся примеры использования этих приемов на практике. Далее рассматривается интересный эффект — режимы смешивания, которые применяются, чтобы смешивать изображения и цвета, как захочется.

Не всегда стоит задействовать все эти трюки одновременно на одной странице. Поэтому рассмотрим несколько небольших отдельных примеров. Это даст вам набор инструментов, с помощью которых вы сможете в дальнейшем работать над проектами.

11.1. Градиенты

Как вы, наверное, заметили, в предыдущих главах мы использовали одноцветный фон и простые изображения. Сейчас предстоит сделать целое открытие касательно свойства `background`. По сути, это сокращение для восьми свойств:

- ❑ `background-image` — устанавливает в качестве фона изображение из файла или градиентный фон;
- ❑ `background-position` — определяет начальное положение фонового изображения;
- ❑ `background-size` — определяет, насколько большим будет изображение внутри элемента;
- ❑ `background-repeat` — определяет, нужно ли повторять изображение, чтобы полностью заполнить элемент;
- ❑ `background-origin` — определяет положение изображения относительно границ блока элемента, поля блока элемента (по умолчанию) или же контента блока элемента;
- ❑ `background-clip` — определяет, как изображение должно заполнять границы блока элемента (по умолчанию), поля блока элемента или же контент блока элемента;
- ❑ `background-attachment` — определяет, будет ли фоновое изображение перемещаться вместе с элементом (по умолчанию) или же останется неподвижным,

сохранит свое положение в области просмотра. Обратите внимание на то, что значение `fixed` может ухудшить внешний вид страницы;

- ❑ `background-color` — определяет одноцветный фон, который визуализируется позади фонового изображения.

В данной главе изучим все эти свойства. Для начала следует запомнить, что можно использовать просто `background` и определять лишь некоторые из этих свойств, остальные примут значения по умолчанию. Сам я предпочитаю указывать все, если проект требует применения более чем 2–3 свойств.

Свойство `background-image` особенно интересно. Вы уже знаете, что в качестве аргумента оно может принимать путь к изображению или URL-адрес (`background-image: url(coffee-beans.jpg)` — см. главу 8), но не только их, еще и функцию градиента. Например, определите градиент, который из белого плавно перетекает в синий (рис. 11.2).

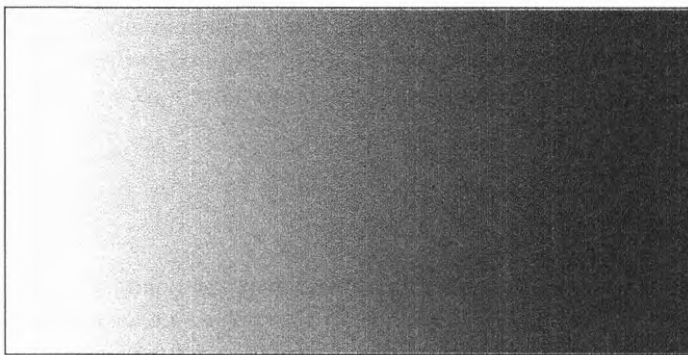


Рис. 11.2. Линейный градиент от белого к синему

Градиент — очень полезный эффект. Предлагаю немного изучить теорию, прежде чем переходить к практике. Создайте новую страницу и таблицу стилей. Добавьте в CSS-файл следующий код (листинг 11.1). Здесь используется функция `linear-gradient()`, которая и определяет градиент.

Листинг 11.1. Простейший линейный градиент

```
.fade {
  height: 200px;
  width: 400px;
  background-image: linear-gradient(to right, white, blue);
}
```

Плавное перетекает из белого в синий справа налево

Градиент по сути своей — это фоновое изображение, которое никак не влияет на размер элемента. В качестве примера я указал явные высоту и ширину элемента. Элемент пустой, поэтому необходимо явно указать высоту, чтобы увидеть градиент.

Функция `linear-gradient` принимает три параметра, определяющих ее поведение: угол, начальный и конечный цвета. В данном примере значение угла — `to right`,

оно обозначает, что градиент начинается у левого края элемента (белый цвет) и плавно переходит к правому краю (синий цвет). Для указания цветов используется и другой синтаксис, например восьмеричные значения (`#0000ff`), значения RGB (`rgb(0, 0, 255)`) или ключевое слово `transparent`. Добавьте на страницу элемент из листинга 11.2, чтобы увидеть градиент.

Листинг 11.2. Элемент с градиентным фоном

```
<div class="fade"></div>
```

Угол градиента задается несколькими способами. В данном примере использовано значение `to right`, но точно так же можно взять `to top` или `to bottom`. Или даже задать наклон, например `to bottom right`. Это обозначает, что градиент начнется в левом верхнем углу элемента и плавно перетечет в правый нижний.

Еще более полный контроль над углом градиента обеспечивают конкретные значения в градусах. Так, `0deg` эквивалентно `to top`. Увеличивая значение, можно управлять углом наклона, как стрелкой на часах: `90deg` указывает направо, `180deg` — вниз, а `360deg` — снова вверх. То есть пример из листинга 11.3 эквивалентен предыдущему.

Листинг 11.3. Задание градиента с помощью градусов

```
.fade {  
  height: 200px;  
  width: 400px;  
  background-image: linear-gradient(90deg, white, blue);  
}
```

Значение `90deg` идентично `to right`

Градусы — самый распространенный способ задания угла, но есть и еще несколько:

- ❑ `rad` — обозначает радианы. Полный круг равняется 2π , или примерно 6,2832 рад;
- ❑ `turn` — определяет количество полных оборотов вокруг своей оси. Один поворот равен 360° (`360deg`). Используйте десятичные дроби, чтобы задать неполный оборот: `0.25turn` равно `90deg`;
- ❑ `grad` — метрические минуты (грады). Полный круг составляет 400 град (`400grad`), а `100grad` равняются `90deg`.

Теперь немного поэкспериментируем, чтобы понять, как изменение значений влияет на градиент.

11.1.1. Использование нескольких цветовых узлов

В большинстве случаев вы будете применять двуцветные градиенты, преобразующие один цвет в другой. Однако существует возможность выполнять градиенты, состоящие из большего количества цветов, каждый из которых называется *цветовым узлом*. На рис. 11.3 показан градиент с тремя цветовыми узлами — красным, белым и синим.

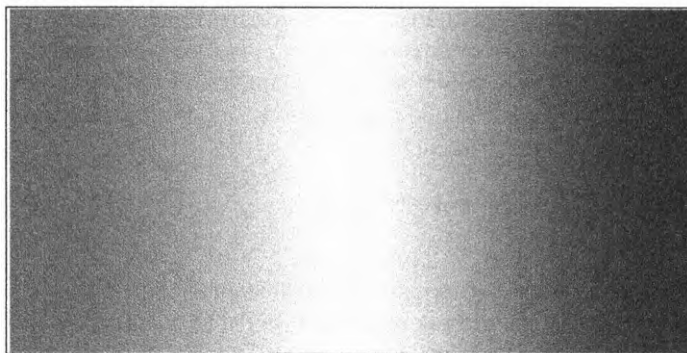


Рис. 11.3. Градиент с тремя цветовыми узлами — красным, белым и синим

Можете добавить и больше цветов в функцию `linear-gradient()`. Чтобы увидеть, как это выглядит, внесите в код изменения, показанные в листинге 11.4.

Листинг 11.4. Линейный градиент с множеством цветовых узлов

```
.fade {
  height: 200px;
  width: 400px;
  background-image: linear-gradient(90deg, red, white, blue);
}
```

Определяет
множество
цветовых узлов

В качестве аргумента функция градиента может принимать любое количество цветов, разделенных запятой. Цвета автоматически распределяются по элементу равномерно. В данном примере красный начинается у левого края (0 %), плавно перетекает в белый в центре (50 %), а затем в синий (100 %). Можно также явно задать положение цветовых узлов в функции градиента. Градиент, описанный в листинге 11.4, полностью совпадает со следующим:

```
linear-gradient(90deg, red 0%, white 50%, blue 100%)
```

Как можно догадаться, положение цветов настраивается любое, какое вам только захочется, вовсе не обязательно, чтобы они распределялись равномерно. Также не обязательно задавать положение цветовых узлов в процентах. Пиксели, `em` и любые другие единицы измерения отлично подойдут.

Полосы

Если вы расположите два цветовых узла в одной позиции, функция градиента немедленно сменит один цвет на другой, то есть плавного перехода между ними не будет. На рис. 11.4 представлен градиент, на котором красный мгновенно переходит в белый, а белый — в синий. Это создает иллюзию полос.

Код для создания данного градиента представлен в листинге 11.5. Обратите внимание на то, что градиент имеет четыре цветовых узла, два из которых белые.

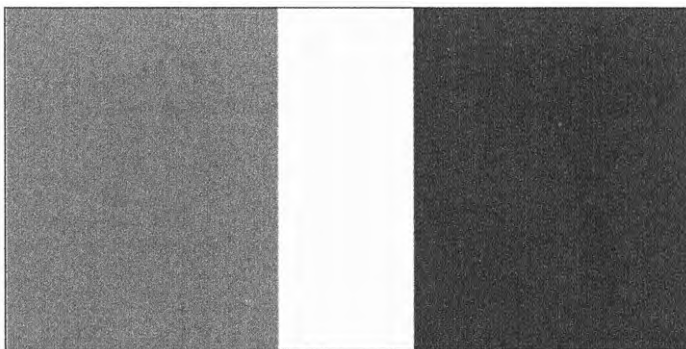


Рис. 11.4. Создание полос с помощью градиента: два цветовых узла находятся в одном месте

Листинг 11.5. Создание полос путем размещения двух цветовых узлов в одной позиции

```
.fade {  
  height: 200px;  
  width: 400px;  
  background-image: linear-gradient(90deg,  
    red 40%, white 40%,  
    white 60%, blue 60%);  
}
```

Цветовые узлы,
расположенные на одной позиции

Первый цветовой узел — красный в позиции 40 %, поэтому градиент представляет собой ровный красный цвет от левого края до точки 40 %. Следующий узел — белый, также в позиции 40 %, что создает резкий переход. Затем еще один белый цветовой узел в позиции 60 %, то есть с 40 до 60 % белый плавно перетекает в белый. И наконец, последний цветовой узел — синий в позиции 60 %, здесь происходит резкий переход к синему, который заполняет все пространство до правого края.

Повторяющийся градиент

Несмотря на то что предыдущий пример немного искусственный, с помощью данной техники реализуются очень интересные эффекты. В частности, ее можно использовать в немного другой, но очень похожей функции `repeating-linear-gradient()`. Она работает почти так же, как и `linear-gradient`, только шаблон повторяется. Этот прием создает полосы, в точности как на традиционных вывесках цирюльников, что очень хорошо смотрится, например, на индикаторах загрузки (рис. 11.5).



Рис. 11.5. Повторяющийся линейный градиент на индикаторе загрузки

В повторяющемся линейном градиенте лучше использовать фиксированную длину, нежели проценты, так как фиксированная величина определяет размер

узора, который будет повторяться. Код для создания индикатора загрузки показан в листинге 11.6. Добавьте его в свой проект.

Листинг 11.6. Создание индикатора загрузки с диагональными полосами

```
.fade {
  height: 1em;
  width: 400px;
  background-image: repeating-linear-gradient(-45deg,
    #57b, #57b 10px, #148 10px, #148 20px);
  border-radius: 0.3em;
}
```

Чередующиеся голубые и темно-синие полосы

Иногда вместо того, чтобы создавать градиент с нуля, я подбираю готовый и дорабатываю его под свои потребности. Примеры найдете по следующей ссылке: css-tricks.com/stripes-css/.

11.1.2. Использование радиального градиента

Другой распространенный градиент — радиальный. В данном случае цвет не пере-текает плавно от одного края элемента к другому, а начинается в центре и распро-страняется во все стороны. Пример простейшего радиального градиента представлен на рис. 11.6.

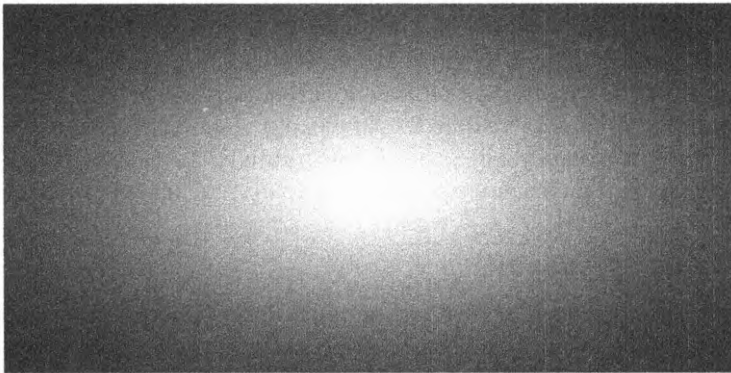


Рис. 11.6. Радиальный градиент от белого к синему

Добавьте следующий код (листинг 11.7) в свой проект, чтобы увидеть радиаль-ный градиент.

Листинг 11.7. Простейший радиальный градиент

```
.fade {
  height: 200px;
  width: 400px;
  background-image: radial-gradient(white, blue);
}
```

Плавно переходит от белого центра к синим краям

По умолчанию градиент располагается в центре элемента и равномерно меняет цвет к краям. Он эллиптический и повторяет пропорции элемента, то есть становится шире для широких элементов и удлинняется — для высоких.

В радиальных градиентах, как и в линейных, используются цветовые узлы. Можно задать несколько цветов и явно определить их положение с помощью процентов или единиц измерения длины. А также сделать градиент круглым, а не в форме эллипса или задать точку центра градиента. Функция `repeating-radialgradient()` позволяет повторить узор несколько раз.

Лучше это все объяснить на примере. На рис. 11.7 приведены несколько изображений градиента с соответствующим кодом. Советую попробовать добавить его на свою страницу и немного поэкспериментировать.




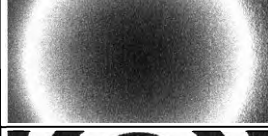

Значение	Результат
<code>radial-gradient(white, midnightblue)</code> Стандартный градиент (эллипс)	
<code>radial-gradient(circle, white, midnightblue)</code> Круговой градиент	
<code>radial-gradient(3 em at 25% 25%, white, midnightblue)</code> Градиент, расположенный на расстоянии 25 % от верхнего и левого краев, размером 3em	
<code>radial-gradient(circle, midnightblue 0%, white 75%, red 100%)</code> Радиальный градиент с явно заданными цветовыми узлами	
<code>repeating-radial-gradient(circle, midnightblue 0%, midnightblue 1em, white 1em, white 2em)</code> Повторяющийся радиальный градиент с эффектом полос	

Рис. 11.7. Примеры радиального градиента

В жизни я редко вижу необходимость использовать сложные радиальные градиенты. Большинство простых градиентов вполне удовлетворяют мои потребности. Если хотите глубже изучить эту тему, советую обратиться к документации MDN: developer.mozilla.org/ru/docs/Web/CSS/radial-gradient.

В приведенных примерах намеренно подобраны контрастные цвета. Это сделано, чтобы лучше продемонстрировать, как ведут себя градиенты. В настоящих проектах лучше использовать менее контрастные цвета.

Вместо того чтобы создавать переход от белого к черному, лучше применить сочетание белого со светло-серым или выполнить переход между двумя похожими оттенками синего. Это визуально более приятно для глаз и будет лучше воспринято пользователями. В некоторых случаях градиент будет даже незаметен, но он придаст странице определенную глубину, объем. Скоро я приведу пример, но для начала нужно рассмотреть тени.

11.2. Тени

Еще один эффект, который может углубить восприятие страницы, — это тень. Свойства, которые ее создают: `box-shadow` — образует тень от блока элемента, `text-shadow` — отбрасывает тень от визуализированного текста. Мы пару раз использовали `box-shadow` в предыдущих главах, но сейчас рассмотрим это свойство подробнее.

Значение `box-shadow: 1em 1em black` создает тень (рис. 11.8). Значения `1em` — смещения, они определяют, как далеко тень будет смещена относительно элемента по горизонтали или вертикали. Если это значение равно `0`, тень расположится прямо за элементом. А значение `black` определяет цвет отбрасываемой тени.

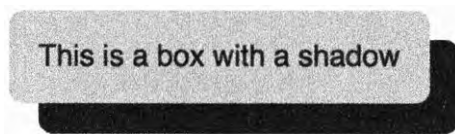


Рис. 11.8. Простейшая тень элемента

По умолчанию тень ровно такого же размера, как и элемент. Также она имеет закругленные края, радиус закругления которых соответствует радиусу закругления углов элемента. Значения горизонтального (по оси *X*) и вертикального (по оси *Y*) смещений и цвета — обязательные параметры. Существует еще два параметра, которые можно использовать по желанию: радиус размытия и радиус распространения. Полный синтаксис представлен на рис. 11.9.

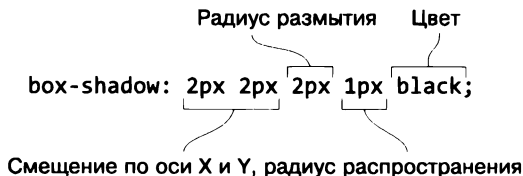


Рис. 11.9. Синтаксис кода тени элемента

Радиус размытия определяет, насколько сильно будут размыты края тени. Увеличение этого значения сделает их более мягкими и слегка прозрачными. Радиус распространения определяет размер тени. Положительное значение увеличивает размер отбрасываемой тени во всех направлениях, отрицательное — уменьшает ее.

11.2.1. Создание объема с помощью градиентов и теней

Попытаемся с помощью теней и градиентов создать кнопку, показанную на рис. 11.10. Вертикальный градиент, распространяющийся сверху вниз, создает трехмерный эффект. Тень его усиливает. В этом примере я также использую псевдокласс `:active`, чтобы задать другие свойства, когда кнопка нажата.



Рис. 11.10. Кнопка с эффектами градиента и тени. Стиль при нажатии кнопки меняется и показан справа

Градиент в данном случае едва различимый. Вы даже не сразу его заметите, но он придает кнопке слегка выпуклую форму. Тень слегка размыта, что делает ее более естественной. При нажатии кнопки тень исчезает, вместо этого появляется внутренняя тень. Это создает такой эффект, будто пользователь физически нажал кнопку. Если отпустить кнопку мыши, то наша кнопка вернется в начальное состояние. Это достигается с помощью псевдокласса `:active`.

Создайте новую страницу и новую таблицу стилей. Добавьте кнопку на страницу (листинг 11.8).

Листинг 11.8. Код для размещения кнопки

```
<button class="button">Регистрация</button>
```

Затем добавьте в таблицу стилей код из листинга 11.9. Он изменит стандартные, используемые браузером шрифты и границы, а также определит размер, тень и градиент кнопки.

Листинг 11.9. Стиль кнопки с тенью и градиентом

```
.button {  
  padding: 1em;  
  border: 0;  
  font-size: 0.8rem;  
  color: white;  
  border-radius: 0.5em;  
  background-image: linear-gradient(to bottom, #57b, #148);  
  box-shadow: 0.1em 0.1em 0.5em #124;  
}  
  
.button:active {  
  box-shadow: inset 0 0 0.5em #124,  
             inset 0 0.5em 1em rgba(0,0,0,0.4);  
}
```

Градиент
от светло-синего
к синему

Темно-синяя тень
с размытием 0,5 em

Две внутренние
тени блока

Фоновое изображение (`background-image`) в данном случае — градиент между двумя близкими оттенками синего. Тень блока смещена совсем немного, всего

на 0,1 em вниз и вправо, размытие средней интенсивности — 0,5 em. Увеличение смещения тени блока создаст эффект еще большего отдаления кнопки от страницы, как бы немного приподнимет ее. При нажатии стиль кнопки меняет свои свойства.

Тут я проделал две вещи. Вместо обычной тени блока использовал ключевое слово `inset`. Оно делает тень внутренней, то есть теперь она отображается внутри элемента. А также добавил еще одну тень, отделив ее запятой от предыдущей. Таким образом может быть добавлено сколько угодно теней.

Первая внутренняя тень (`inset 0 0 0.5em #124`) имеет смещение 0 и легкое размытие. Она добавляет легкое затемнение по периметру элемента. У второй тени (`inset 0 0.5em 1em rgba(0,0,0,0.4)`) есть легкое вертикальное смещение, что создает дополнительный эффект. Цвет RGBA определяет полупрозрачный черный. Поэкспериментируйте с этими значениями и посмотрите, как они влияют на конечное изображение.

ПРИМЕЧАНИЕ

При нажатии кнопки в браузере Chrome вы можете заметить легкое голубое свечение вокруг нее. Это свойство автоматически определяется браузером для кнопок в состоянии `:focus`. Эффект отключается следующим образом: `.button:focus { outline: none; }`. Если вы сделаете это, рекомендую заменить его чем-то другим, чтобы пользователи, которые для навигации применяют исключительно клавиатуру, знали, какой элемент сейчас активен.

Данный стиль применялся повсеместно на протяжении многих лет: элементы на экране напоминают реальные аналоги (этот подход известен как *скевоморфизм*). В реальном мире объекты не имеют идеально ровных цветов. Даже от объекта с гладкими поверхностями свет отражается по-разному, создавая блики и тени.

Придавая кнопке скругленную форму и тень, дизайнер делает ее похожей на реальный аналог. К другим скевоморфическим элементам относятся наложенные границы и изображения с наложенной текстурой типа кожи. В 2010–2013 годах этот вид дизайна уступил место *плоскому*.

11.2.2. Элементы с плоским дизайном

Если скевоморфический дизайн создает объекты, похожие на реальные предметы, то плоский дизайн отражает цифровую природу современного мира. Он делает акцент на ярких однородных цветах и упрощенном внешнем виде. Это означает использование меньшего количества градиентов, теней и закругленных углов. По иронии судьбы данная тенденция возникла только после того, как эти долгожданные эффекты появились в CSS (до этого тени и градиенты создавались с помощью изображений).

Плоский дизайн не обязательно означает, что ни один из рассматриваемых эффектов не применяется. Нет, они используются, но более тонко. Например, вместо градиента от светло-синего к темному создается градиент от одного оттенка синего к другому, что почти незаметно для глаз. Или, возможно, у элемента есть очень слабая тень. А может, у него и нет ни одного из этих свойств.

Преобразуем кнопку в такую же, но с плоским дизайном (рис. 11.11). Она не похожа на реальную, хотя и имеет небольшую тень.

Далее (листинг 11.10) представлены стили, создающие эффект наведения на эту кнопку указателя мыши, а также ее нажатия. Добавьте этот код в таблицу стилей.

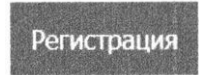


Рис. 11.11. Кнопка с плоским дизайном

Листинг 11.10. Кнопка с эффектами наведения указателя мыши и нажатия

```
.button {  
  padding: 1em;  
  border: 0;  
  color: white;  
  background-color: #57b; ← Одноцветный фон без градиента  
  font-size: 1rem;  
  padding: 0.8em;  
  box-shadow: 0 0.2em 0.2em rgba(0, 0, 0, 0.15); ← Едва различимая тень  
}  
.button:hover {  
  background-color: #456ab6; ← Легкое затемнение при наведении указателя и щелчке кнопкой мыши  
}  
.button:active {  
  background-color: #148;  
}
```

В тень блока было внесено несколько изменений. Она имеет только вертикальное смещение, поэтому направлена прямо вниз. Такой угол не слишком естественен. Я также использовал RGBA-цвета со значениями 0 для красного, зеленого и синего (черный) и альфа-значение 0,15 (почти прозрачный). Эффекты наведения и нажатия тоже «плоские». Они слегка меняют оттенок цвета фона на более темный. Размер шрифта увеличен, что также характерно для плоского дизайна.

11.2.3. Создание кнопок с более современным дизайном

Плоский дизайн все еще популярен, но и он не остается неизменным. Один из новых подходов состоит в сочетании плоского дизайна и скевоморфизма. Давайте последний раз изменим кнопку так, чтобы она выглядела как на рис. 11.12. В данном дизайне использованы элементы обоих стилей.



Рис. 11.12. Еще один стиль плоского дизайна (справа — вид нажатой кнопки)

Это по-прежнему минималистичный дизайн, но элемент имеет толстую нижнюю границу, что создает такой же эффект, как на лицевой стороне 3D-куба. Темная линия, по сути, вовсе не граница (*border*), а тень блока (*box-shadow*) без размытия, что позволяет ей в точности повторять изогнутый край.

При нажатии кнопка смещается на несколько пикселей ниже, что создает эффект приближения к странице. Дополните свой код (листинг 11.11), чтобы добиться такого же эффекта.

Листинг 11.11. Кнопка в современном стиле

```
.button {
  padding: 0.8em;
  border: 0;
  font-size: 1em;
  color: white;
  border-radius: 0.5em;
  background-color: #57b;
  box-shadow: 0 0.4em #148;
  text-shadow: 1px 1px #148;
}

.button:active {
  background-color: #456ab5;
  transform: translateY(0.1em);
  box-shadow: 0 0.3em #148;
}
```

Добавляются закругленные края

Создается сплошная тень без размытия под кнопкой

Добавляется легкая тень от текста

Кнопка смещается вниз при нажатии

Уменьшается размер тени для компенсации трансформации

Для этой кнопки я иначе использовал свойство `box-shadow`. Вместо размытия оставил тени четкими. Это и создает эффект толстой нижней границы. Однако отличие все же есть, так как тень в точности повторяет линию закругленных углов элемента.

Я добавил тень и самому тексту. Она ведет себя точно так же, как и тень блока, только ее отбрасывают сами буквы. Очень похож и ее синтаксис: смещение по осям *X* и *Y*, радиус размытия (необязательный параметр) и цвет. Но тень текста не имеет ключевого слова `inset` и радиуса распространения. Я задал для тени темно-синий цвет и установил смещение 1 пиксел для каждого из направлений.

Кое-что новое наблюдается и в активном состоянии. Я применил свойство `transform` с функцией `translateY()`. Оно смещает элемент вниз на 1 em (подробнее расскажу об этом в главе 15, когда будем рассматривать функцию трансформирования). Затем уменьшил вертикальное смещение тени блока на ту же величину (3 em вместо 4 em). Когда вы нажимаете кнопку, она движется, а тень остается на месте. Нажмите кнопку и проверьте сами.

Существует множество способов использовать тени и градиенты. Со временем появляются все новые и новые тренды. Просто иногда, впервые встретив на сайте необычный дизайн, потратьте пару минут, чтобы узнать, как он реализуется. И не бойтесь экспериментировать.

11.3. Режимы смешивания

В большинстве случаев один элемент будет иметь только одно фоновое изображение в виде картинки или градиента. Хотя бывают случаи, когда нужно использовать два изображения или больше. И CSS позволяет это сделать.

Фоновое изображение может принимать сколько угодно аргументов, разделенных запятой:

```
background-image: url(bear.jpg), linear-gradient(to bottom, #57b, #148);
```

Если вы задействуете несколько изображений одновременно, то перечисленные первыми будут находиться поверх указанных после них. В данном примере файл `bear.jpg` перекроет линейный градиент — того не будет видно. Но если вы добавляете два изображения, то, видимо, хотите, чтобы и второе было видно. Это делается с помощью *режима смешивания*.

Если вы работали с программным обеспечением для обработки фотографий, то, возможно, знаете, как действуют режимы смешивания (наложения). Они управляют отображением наложенных изображений относительно друг друга. Эти режимы имеют загадочные имена: «экран», «выжигание» основы», «жесткий свет» и пр. Например, на рис. 11.13 показаны два фона, наложенные друг на друга в режиме умножения (`multiply`). Они используют два одинаковых изображения, смещенных друг относительно друга.

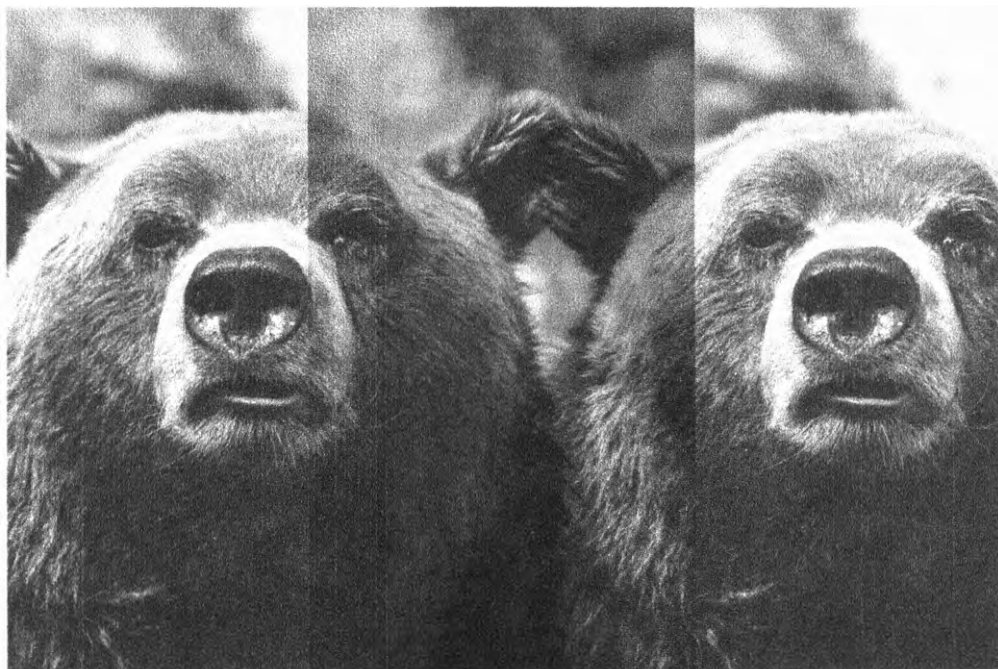


Рис. 11.13. Два фона, наложенные друг на друга в режиме умножения

Это создает интересный эффект: два изображения четко видны, несмотря на то что накладываются друг на друга. И этот режим не вымывает цвет, как свойство прозрачности.

ПРИМЕЧАНИЕ

Если фоновое изображение имеет прозрачные участки, то через них будут видны нижние слои, даже если режим смешивания не был применен. Данный эффект достигается изображениями в формате GIF или PNG с прозрачными областями или градиентом, вместо одного из цветов использовано ключевое слово `transparent`.

Создадим элемент с двумя фоновыми изображениями, такой же как на рис. 11.13. Создайте новую страницу и добавьте на нее элемент, как показано в листинге 11.12. Мы будем задействовать этот макет несколько раз.

Листинг 11.12. Контейнер `div` для смешиваемых изображений

```
<div class="blend"></div>
```

Для начала используем пустой элемент, чтобы продемонстрировать эффект. Добавьте код из листинга 11.13 в таблицу стилей и привяжите ее к своей странице.

Листинг 11.13. Смешение двух фоновых изображений

```
.blend {
  min-height: 400px;
  background-image: url(images/bear.jpg), url(images/bear.jpg);
  background-size: cover;
  background-repeat: no-repeat;
  background-position: -30vw, 30vw;
  background-blend-mode: multiply;
}
```

Два фоновых изображения
через запятую

Определяет одно значение,
которое будет применяться
к обоим изображениям

Определяет разные положения
для двух изображений

Режим
смешивания

Большинство приведенных свойств фона могут принимать несколько аргументов, разделенных запятой. Одно из таких свойств — положение фона. Первое значение будет применено к первому изображению, второе — ко второму. Свойства `background-size` и `background-repeat` также могут принимать несколько аргументов. Но так как мы указали только по одному аргументу, каждое из свойств будет применяться к обоим изображениям. Свойство `min-height` было добавлено для того, чтобы мы смогли увидеть изображение, так как элемент пустой.

Свойство `background-size` в качестве аргумента принимает два специальных ключевых слова — `cover` и `contain`. Первое изменяет размер изображения так, что оно полностью заполняет элемент. Это может привести к тому, что края изображения будут обрезаны. Второе гарантирует, что изображение будет видно полностью, однако при этом некоторые части элемента могут остаться не закрытыми фоновым изображением (эффект «леттербоксинг»). Данное свойство также принимает аргументы длины и ширины, что позволяет явно задать размеры.

Попробуйте изменить режим смешивания, например, на `color-burn` или `difference` и посмотреть, как это повлияет на конечное отображение. Вы можете

долго экспериментировать с этим свойством, но так и не понять, как на самом деле работают режимы смешивания. Поэтому попробуем:

- ☐ вместо второго изображения использовать цвет или градиент;
- ☐ наложить на изображение текстуру, например царапины или зернистость фото-пленки;
- ☐ осветлить, затемнить или уменьшить контрастность изображения, чтобы удобнее было читать текст;
- ☐ наложить текстовый баннер, но так, чтобы изображение виднелось из-под него.

Продедаем все это вместе. А после я кратко опишу работу каждого из режимов смешивания.

11.3.1. Изменение оттенка изображения

Режим смешивания можно использовать, чтобы придать полноцветному изображению определенный оттенок. Для демонстрации я возьму все то же изображение медведя и придам ему голубой оттенок (рис. 11.14).



Рис. 11.14. Изображение с голубым оттенком

Свойство `background-blend-mode` накладывает не только фоновые изображения, но и просто цвета (`background-color`). Затем эти слои смешиваются с помощью режима смешивания, то есть можно в качестве цвета фона выбрать любой понравившийся

оттенков, а затем наложить его на изображение. Измените свой CSS-код, как показано в листинге 11.14.

Листинг 11.14. Наложение цвета на фоновое изображение

```
.blend {
  min-height: 400px;
  background-image: url("images/bear.jpg");
  background-color: #148;
  background-size: cover;
  background-repeat: no-repeat;
  background-position: center;
  background-blend-mode: luminosity;
}
```

Синий цвет фона

Используется режим смешивания luminosity

Режим смешивания `luminosity` принимает яркость переднего слоя (изображение медведя) и смешивает его с оттенком и насыщенностью нижнего слоя (синий цвет фона). Другими словами, в результате получаем цвет фона, но яркость и контрастность — самого изображения.

Важно понимать, что этот и несколько других режимов смешивания могут вести себя по-разному в зависимости от порядка расположения изображений. В данном случае цвет — это нижний слой, а изображение — верхний. Но если вы поместите изображение на задний план, а цвет — на передний и вместо него возьмете градиент, эффект будет совсем другим. В этом случае нужно использовать режим смешивания цветов, обратный режиму «яркость»: тон и насыщенность берутся от переднего слоя, а яркость — от заднего.

11.3.2. Виды режимов смешивания

В CSS поддерживаются 15 режимов смешивания. В каждом из них я применяю собственную математическую формулу, чтобы определить, как именно стоит наложить изображения. Цвет каждого пиксела одного слоя накладывается на каждый пиксел другого слоя, в результате формируется цвет пиксела итогового изображения.

Режимы смешивания представлены в табл. 11.1. Каждый из них относится к одной из пяти категорий: затемнение, осветление, контраст, компонент, сравнение. Одни используются чаще, другие — реже. А выбрать правильный часто можно лишь методом проб и ошибок.

Таблица 11.1. Режимы смешивания

Категория	Режим смешивания	Описание
Затемнение	multiply (умножение)	Чем светлее верхнее изображение, тем лучше будет видно изображение основы
	darken (затемнение)	Выбирает наиболее темный из двух цветов
	color-burn («выжигание» основы)	Затемняет основное изображение, увеличивает контраст

Категория	Режим смешивания	Описание
Осветление	screen (экран)	Чем темнее верхнее изображение, тем лучше будет видно изображение основы
	lighten (осветление)	Выбирает наиболее светлый из двух цветов
	color-dodge (осветление основы)	Осветляет основу изображения, уменьшает контрастность
Контраст	overlay (перекрытие)	Увеличивает контрастность за счет применения значения multiply для темных цветов и screen для светлых
	hard-light (жесткий свет)	Сильно увеличивает контрастность, как и overlay, но с более сильными эффектами multiply и screen
	soft-light (мягкий свет)	Действует так же, как и предыдущий, но вместо multiply/screen использует burn/dodge
Компонент	hue (оттенок)	Применяет оттенок верхнего изображения к нижнему
	saturation (насыщенность)	Применяет насыщенность верхнего изображения к нижнему
	luminosity (яркость)	Применяет яркость верхнего изображения к нижнему
	color (цвет)	Применяет оттенок и насыщенность верхнего изображения к нижнему
Сравнение	difference (вычитание)	Вычитает более темные цвета из светлых
	exclusion (исключение)	Похоже на предыдущий, но с меньшей контрастностью

На момент написания книги большинство режимов поддерживалось большинством браузеров, за исключением Internet Explorer и Edge. Компонентные режимы не поддерживаются также Safari. Актуальную информацию вы найдете на сайте caniuse.com/#feat=css-backgroundblendmode.

11.3.3. Добавление текстуры изображению

Другой способ применения режима смешивания — добавление его изображению текстуры. Изображение может быть четким, современным, но иногда нужно, чтобы оно выглядело совсем иначе, по стилистическим причинам. Можно использовать изображение в градациях серого или искусственно добавить зернистость или любую другую текстуру.

Посмотрите на изображение на рис. 11.15. Это то же самое изображение медведя, с которым мы работали ранее, но на него наложена текстура, создающая эффект грубого полотна. Он получается с помощью одного из режимов смешивания категории «контраст»: **overlay**, **hard-light** или **soft-light**. В данном случае я не хотел менять оттенок снимка, поэтому использовал черно-белое изображение текстуры. Таким образом, оригинальное изображение сохраняет свой цвет.



Рис. 11.15. Изображение с наложенной текстурой

Код для наложения текстуры представлен в листинге 11.15. Изображение текстуры повторяется и наложено поверх изображения медведя. Внесите изменения в свой проект в соответствии с листингом, чтобы увидеть эффект в браузере.

Листинг 11.15. Использование режима мягкого света для придания изображению текстуры

```
.blend {
  min-height: 400px;
  background-image: url("images/scratches.png"), url("images/bear.jpg");
  background-size: 200px, cover;
  background-repeat: repeat, no-repeat;
  background-position: center center;
  background-blend-mode: soft-light;
}
```

Слой текстуры находится
поверх изображения

Изображение текстуры
повторяется каждые 200 пикселей

Режим смешивания —
мягкий свет

Размер изображения текстуры (рис. 11.16) установлен равным 200 пикселям, применено свойство повторения фона. Это заставляет изображение многократно повторяться, чтобы заполнить элемент. Размер же фонового изображения имеет свойство `cover` и не повторяется.

Я выяснил, что режим `soft-light` намного лучше подходит для темных изображений текстуры, тогда как `hard-light` и `overlay` предпочтительно использовать для более светлых изображений (но если поменять изображение и текстуру местами, правило работает наоборот). Вы можете получить другие результаты — все зависит от дизайнерских предпочтений и от того, насколько темное исходное изображение.

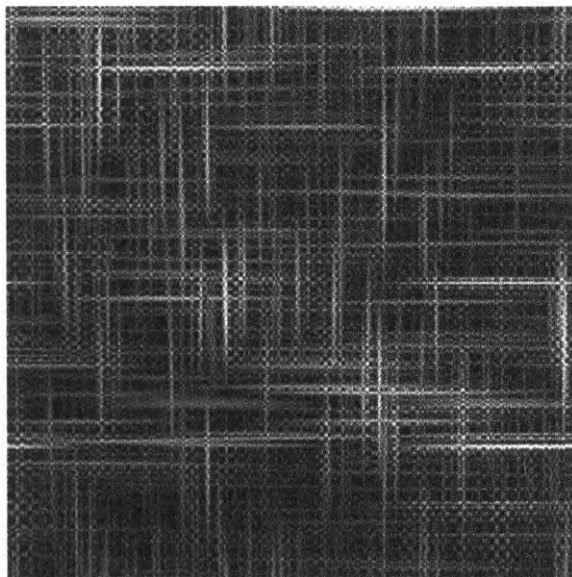


Рис. 11.16. Изображение текстуры в градациях серого

11.3.4. Микширование режимов смешивания

Несмотря на то что свойство `background-blend-mode` позволяет накладывать друг на друга несколько изображений, его использование ограничивается цветами и изображениями лишь одного элемента. Однако существует другое свойство, `mix-blend-mode`, которое обеспечивает наложение нескольких элементов. Это позволяет вам не ограничиваться только наложением изображений. Например, текст и границы одного элемента могут быть наложены на фоновое изображение другого. С помощью данного свойства вы можете наложить заголовок на изображение (рис. 11.17).

Создается интересный эффект: текст выглядит прозрачным, как бы вырезанным на красном полотне. Так происходит потому, что я задействовал режим смешивания `hard-light` и серый цвет шрифта. Контрастные режимы смешивания сильнее влияют как на очень светлые, так и на очень темные тона, однако использование обычного серого (`#808080`) позволяет оттенку фонового изображения в этой области остаться неизменным.

Чтобы добиться такого эффекта в своем проекте, прежде всего необходимо привязать блок заголовка к контейнеру как дочерний. Дополните свой HTML-код следующим образом (листинг 11.16).

Листинг 11.16. Добавление заголовка внутрь блока

```
<div class="blend">
  <h1>Большая медведица</h1>
</div>
```

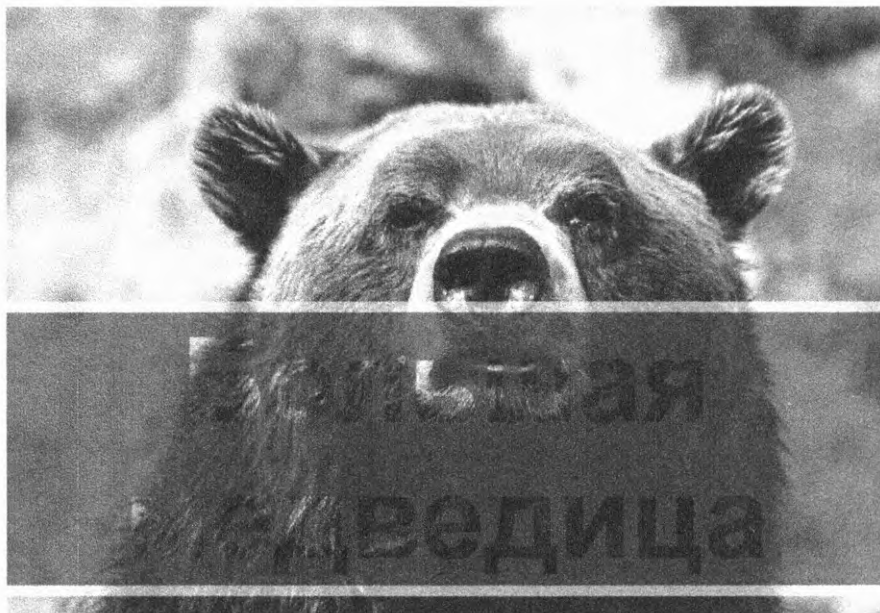



Рис. 11.17. Заголовок, наложенный на изображение

Понадобится отформатировать с помощью стилей элемент `h1` следующим образом: одноцветный красный фон, толстые светло-серые границы сверху и снизу и серый текст. А затем применить режим смешивания. Он рассматривает весь элемент как слой, наложенный на фоновое изображение в блоке, находящемся позади него. Внесите в таблицу стилей следующие изменения (листинг 11.17).

Листинг 11.17. Использование режима смешивания для наложения элементов

```
.blend {
  background-image: url("images/bear.jpg");
  background-size: cover;
  background-position: center;
  padding: 15em 0 1em;
}

.blend > h1 {
  margin: 0;
  font-family: Helvetica, Arial, sans-serif;
  font-size: 6rem;
  text-align: center;
  mix-blend-mode: hard-light;
  background-color: #c33;
  color: #808080;
  border: 0.1em solid #ccc;
  border-width: 0.1em 0;
}
```

Использование режима наложения «жесткий свет»

Устанавливаем цвет текста и фона для верхнего элемента

Цвет заголовка не обладает высокой контрастностью, поэтому будьте осторожны. Мне пришлось сделать текст достаточно большим и жирным, чтобы он оставался читаемым. Либо используйте для фона менее контрастные изображения. В примере я расположил тест в нижней части изображения, так как она менее контрастная.

Режимы смешивания — весьма занятная часть дизайна страницы. Комбинируйте их с градиентами и тенями, это позволит сделать сайты визуально более интересными. Но предупреждаю: не стоит увлекаться, во всем надо знать меру.

Итоги главы

- ❑ Задействуйте градиенты и тени, чтобы добавить странице глубины.
- ❑ Даже плоский дизайн способен выглядеть привлекательно при умелом использовании теней и градиентов.
- ❑ Можно применять градиенты с явными цветовыми узлами, чтобы создать эффект полос.
- ❑ Едва уловимый градиент в качестве фона всегда выигрывает по сравнению с простым цветом.
- ❑ Используйте режимы смешивания, чтобы придать изображениям необходимый оттенок или наложить текстуру.

12

Контраст, цвета и интервалы

В этой главе

- Преобразование дизайнерского макета в HTML- и CSS-разметку.
- Настройка контраста для привлечения внимания к нужным элементам страницы.
- Выбор цветов.
- Управление интервалами.
- Управление высотой строк.

Важная часть работы любого веб-разработчика — это воплощение в жизнь проекта дизайнера с использованием каскадных таблиц стилей. То есть перевод с языка искусства на язык кода. Иногда этот перевод однозначен и прост. А иногда придется долго вести переговоры с дизайнером, чтобы найти компромиссы. Вам надо придумать, как воплотить в CSS каждую деталь, которую дизайнер тщательно продумал, при этом код должен быть логичным и пригодным для внесения поправок. Ваш CSS-код должен быть более практичным, чем шаблон дизайнера.

После того как перевод завершен, именно вам, разработчику придется поддерживать проект, дорабатывая версию, которую предложил дизайнер. Поэтому особенно важно хотя бы в общем понимать идеи, которые дизайнер вложил в цвета, расположение элементов и шрифты. Необходимо знать, как наиболее точно реализовать эти идеи. И только глубокое уважение к работе дизайнера может сделать этот процесс не таким раздражающим.

Я понимаю, что вы не всегда будете пользоваться готовым дизайном. Если вы работаете над проектом небольшого стартапа или вообще над личным проектом, то будете предоставлены сами себе. В данном случае важно разбираться в основах дизайнерского ремесла и применять их на практике.

В этой главе мы рассмотрим шаблон страницы в том виде, который мог бы предложить дизайнер, и превратим его в код. Я сделаю акцент на расположении элементов и цвете. Также остановлюсь на некоторых моментах, относящихся к этим аспектам, которые дизайнер посчитал бы важными. Моя цель при этом, чтобы вы, понимая эти вещи, были способны применить их в своих проектах, даже если работаете без шаблона. Итак, создадим такую же страницу, как представленная на рис. 12.1.

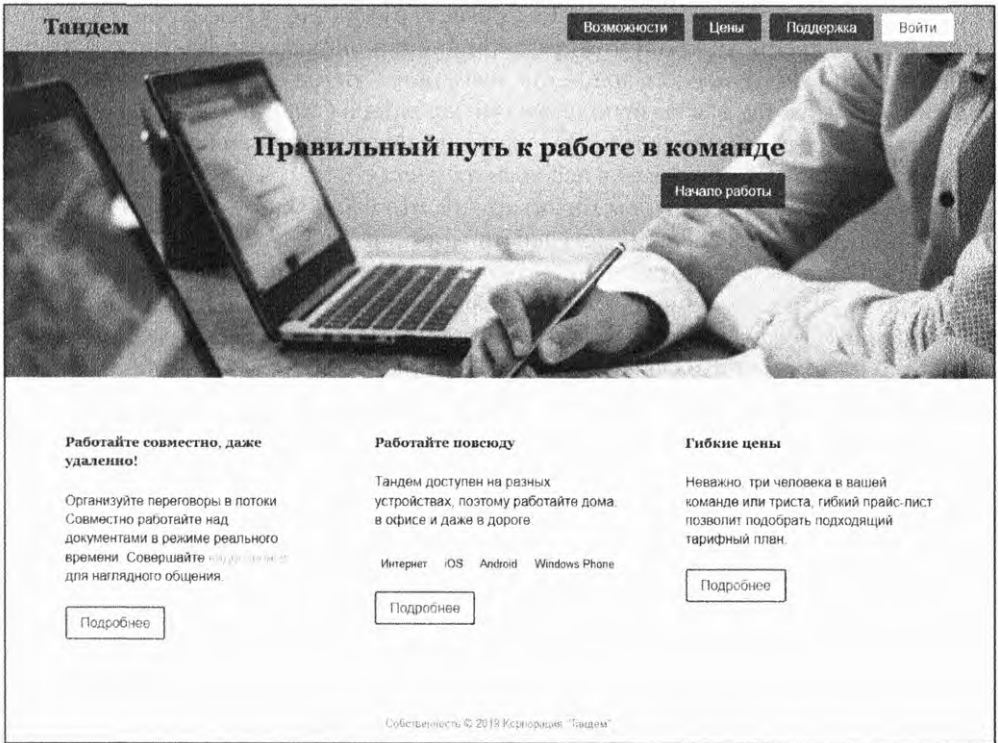


Рис. 12.1. Дизайн страницы, разработанный для компании «Тандем»

Данный снимок экрана демонстрирует итоговый результат. Вероятно, от дизайнера вы получите чуть больше информации о деталях. Но прежде, чем приступить, хочу обратить ваше внимание на несколько моментов по поводу дизайна в целом.

12.1. Царство контраста

Посмотрите еще раз на рис. 12.1. На что вы обращаете внимание в первую очередь? Скорее всего, это слоган компании и кнопка **Начало работы** под ним. На странице вы видите еще несколько элементов: название компании в левом верхнем углу и три текстовых колонки внизу. Но в глаза бросается именно то, что находится в центре. И причина этого — *контраст*.

В дизайне контраст — это способ привлечь внимание к какому-то объекту, заставляя его выделяться. Наши глаза и мозг неосознанно ищут шаблоны. И когда что-то выбивается из шаблона, мы неосознанно обращаем на это внимание.

Чтобы контраст работал так, как мы предполагаем, прежде всего надо создать сам шаблон. Вы не можете применить исключение из правила, если нет правила.

На рис. 12.1 расположение кнопок **Подробнее** гармонично, как и размер и расположение трех колонок. Кроме того, все три кнопки одинаковы. Вы также видите, что на странице всего несколько цветов, и все они — оттенки зеленого. Вот почему в модульном CSS так важно использование шаблонов (см. главы 9 и 10) — вместо того чтобы каждый раз рисовать новую идеально подходящую кнопку, лучше сделать универсальную и установить ее в нескольких местах.

Выбирая второй вариант, вы гарантируете, что сайт будет выдержан в едином стиле. Один из профессиональных секретов дизайнеров — создать общий шаблон, а затем резко менять его, чтобы привлечь внимание к определенным деталям.

Создать контраст можно прежде всего с помощью цвета, расположения или размера. Если несколько предметов светлые, а один темный, вы прежде всего обратите внимание на темный. Когда один из предметов окружен большим количеством неиспользуемого пространства, называемого *воздухом*, он выделяется. А крупные предметы выделяются среди мелких. Для создания еще большего контраста можно комбинировать данные приемы, что я и сделал со слоганом на странице-примере: шрифт крупнее, большое количество воздуха и броская темная кнопка.

Тем не менее слоган не единственный контрастный элемент на этой странице. Вы можете обнаружить иерархию важности содержащейся на ней информации, если проанализируете контраст. Помимо слогана и кнопки **Начало работы**, данный прием очевиден на навигационной панели (рис. 12.2), а также в каждой из колонок внизу страницы (рис. 12.3). Они не так сильно выделяются, как слоган, но привлекают к себе внимание в определенной области. Так как подвал — наименее важная часть страницы с точки зрения информации, он меньше по размеру и менее контрастен.

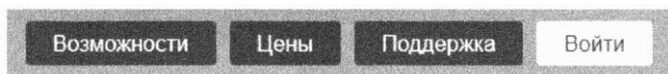


Рис. 12.2. Более светлая кнопка входа привлекает внимание на фоне трех более темных

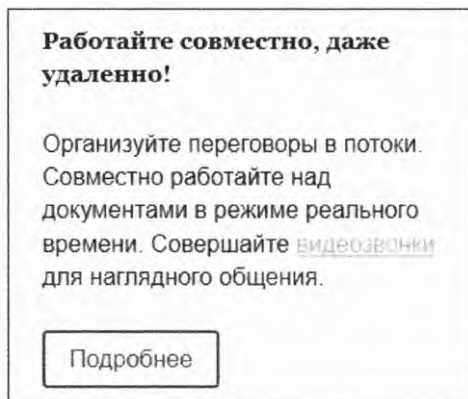


Рис. 12.3. Цветная кнопка выделяется на фоне черно-белого текста

Любая страница создается с определенной целью: иногда просто поведать историю, иногда собрать какую-то информацию или предоставить пользователю некий сервис. В дополнение к основной информации на каждой странице много навигационных элементов, рекламы, абзацев текста, есть также подвал, содержащий сведения об авторских правах и различные ссылки. Работа дизайнера заключается в том, чтобы выделить важную информацию. Ваша работа — не испортить то, что сделал дизайнер.

12.1.1. Создание шаблона

Иногда, разрабатывая макет, дизайнер может проигнорировать некоторые вещи, важные для вас. Он может не учесть, что стоит точно указывать промежутки между элементами, а скругление границ и тени делать одинаковыми для большинства элементов, или вообще не обратить внимания на интервалы между строками и буквами в тексте.

Пример правильного макета показан на рис. 12.4, на нем обозначены точные расстояния в пикселах. Однако иногда необходимость сохранить все эти значения при воплощении проекта в жизнь может показаться утомительной или даже сложной.

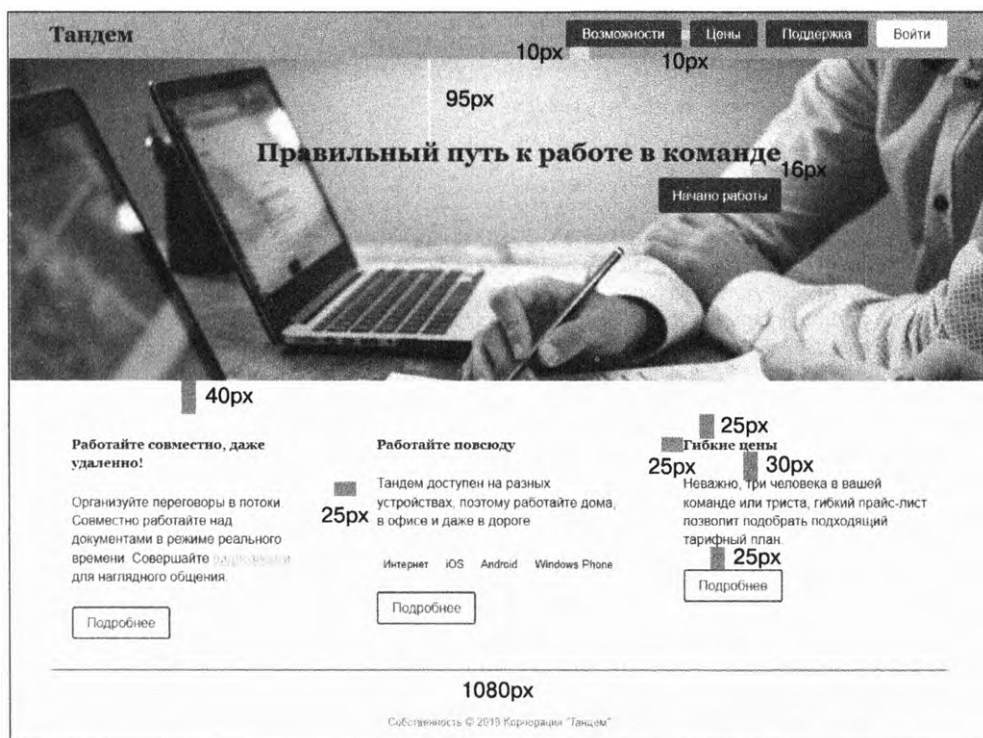


Рис. 12.4. Макет сайта с обозначением точных расстояний

В данном макете с помощью розовых маркеров обозначены интервалы между элементами. Например, 10 пикселей между кнопками на панели навигации, 40 пикселей от нижнего края hero-изображения до края трех белых колонок с текстом, по 30 пикселей между заголовком колонки и основным текстом и т. д. Некоторые размеры тщательно выверены, в дальнейшем это поможет создать тот самый шаблон; 10 и 25 пикселей — наиболее часто применяемые на данной странице размеры.

А теперь внимательно взглянем на общий дизайн — выбор цветов и использование пространства. (Шрифтовое оформление тоже очень важно, но к нему мы вернемся в следующей главе.) Я покажу, как точно воплотить в жизнь макет, представленный на рис. 12.4. При этом необходимо учитывать, что сайт постоянно развивается и меняется. Создание шаблона — это лишь часть работы, надо быть готовыми добавлять новый функционал и содержание, но так, чтобы они вписывались в общий дизайн. Этот аспект мы тоже рассмотрим.

12.1.2. Реализация дизайна

Создайте новую страницу и связанную с ней таблицу стилей. Скопируйте код, представленный в листинге 12.1. Я разбил дизайн на модули, которые мы будем по отдельности рассматривать в течение всей главы.

Листинг 12.1. Разметка страницы

```
<head>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <nav class="nav-container"> ← Контейнер верхней панели навигации
    <ul class="top-nav">
      <li class="top-nav__home"><a href="/">Тандем</a></li>
      <li><a href="/features">Возможности</a></li>
      <li><a href="/pricing">Цены</a></li>
      <li><a href="/support">Поддержка</a></li>
      <li class="top-nav__featured"><a href="/login">Войти</a></li>
    </ul>
  </nav>
  <div class="hero"> ← Крупное hero-изображение
    <div class="hero__inner">
      <h2>Правильный путь к работе в команде</h2>
      <a href="/sign-up" class="button button--cta">Начало работы</a>
    </div>
  </div>
  <div class="container"> ← Строка с трехколоночным блоком
    <div class="tile-row">
      <div class="tile">
        <h4>Работайте совместно, даже удаленно!</h4>
        <p>Организируйте переговоры в потоки. Совместно работайте
          над документами в режиме реального времени. Совершайте
          <a href="/features/video-calling">видеозвонки</a>,
          чтобы видеть собеседника.</p>
      </div>
    </div>
  </div>
```

```
<a href="/collaboration" class="button">Подробнее</a>
</div>

<div class="tile">
  <h4>Работайте повсюду</h4>
  <p>Тандем доступен на разных устройствах, поэтому работайте
    дома, в офисе и даже в дороге:</p>
  <ul class="tag-list">
    <li>Интернет</li>
    <li>iOS</li>
    <li>Android</li>
    <li>Windows Phone</li>
  </ul>
  <a href="/supported-devices" class="button">Подробнее</a>
</div>

<div class="tile">
  <h4>Гибкие цены</h4>
  <p>Неважно, три человека в вашей команде или триста, гибкий
    прайс-лист позволит подобрать подходящий тарифный план.</p>
  <a href="/pricing" class="button">Подробнее</a>
</div>
</div>
</div>

<footer class="page-footer">
  <div class="page-footer__inner">
    Собственность &copy; 2019 Корпорация "Тандем"
  </div>
</footer>
</body>
```

Для именования классов я использовал БЭМ-методологию, в ней сразу понятно, какой элемент принадлежит какому модулю: двойное нижнее подчеркивание обозначает, что элемент является подэлементом модуля (например, `hero__inner`), а двойной дефис — варианты стиля одного элемента в различных модулях (например, `button--cta`) (см. главу 9). Далее мы рассмотрим каждый из модулей. А для начала обратим внимание на цвета, предложенные дизайнером.

12.2. Цвета

Если вы когда-нибудь получали готовый проект от дизайнера, то должны знать, что обычно он представляет собой большой PDF-файл, где выделено несколько разделов. Конечно, большую часть файла будет занимать готовый макет, примерно такой, как на рис. 12.4. Но хороший дизайнер разъяснит и некоторые основополагающие моменты. В файл могут быть включены одна или две страницы с различными вариациями заголовков и основного текста. Также документ, скорее всего, будет содержать детализированную информацию о некоторых элементах пользовательского интерфейса, таких как ссылки и кнопки, включая изменения при наведении и нажатии. А еще там будет представлена цветовая палитра проекта.

Обычно цветовая палитра выглядит примерно так, как на рис. 12.5. Она демонстрирует все цвета, используемые на сайте, и их шестнадцатеричные значения. Также дизайнер может дать названия цветов, которые в дальнейшем будут указываться в спецификации.

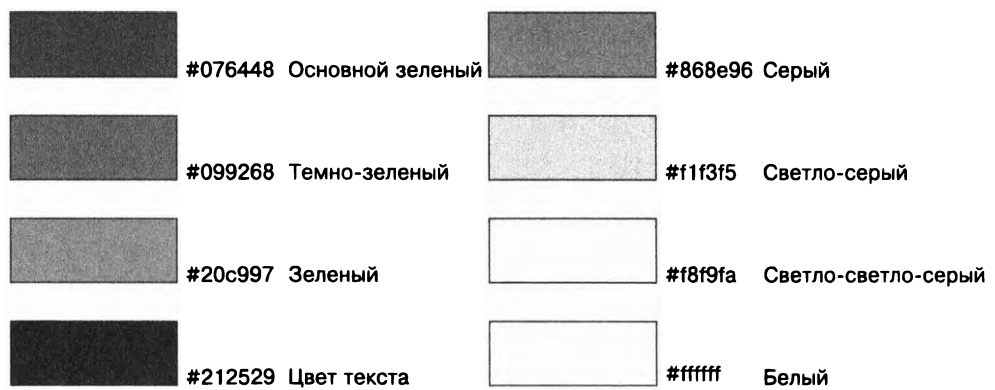


Рис. 12.5. Цветовая палитра сайта

Обычно палитра имеет один основной цвет, вокруг которого строится весь дизайн сайта. Чаще всего в качестве него выбирается основной цвет логотипа компании. На нашей странице базовый цвет — зеленый (верхний левый на рисунке). Остальные цвета — это, как правило, различные оттенки основного или хорошо сочетающиеся с ним цвета. Также обычно присутствуют черный, белый (хотя они могут и отличаться от традиционных #000000 и #ffffff) и несколько оттенков серого.

Поскольку вы многократно используете эти цвета в коде CSS, то сэкономите немало времени, если присвоите нужные значения переменным. Иначе придется множество раз печатать шестнадцатеричные значения, и попробуйте не ошибиться.

Давайте наконец назовем странице базовые стили. Сюда относится создание переменных для цветов, так будет явно удобней. Страница, которую вы видите на рис. 12.6, еще не очень похожа на законченную, но цвета уже соответствуют проекту.

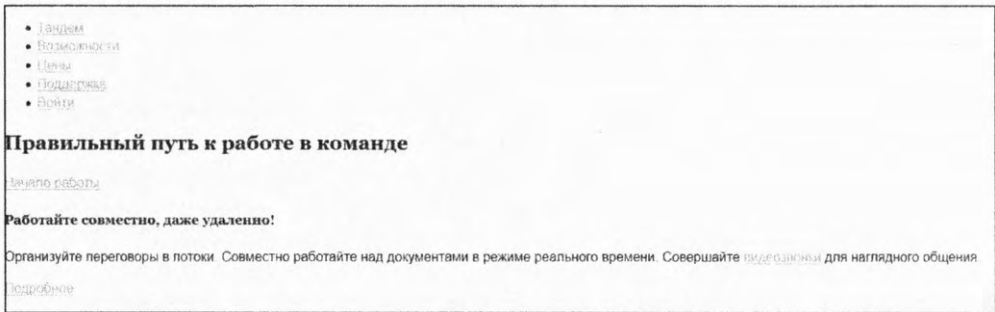


Рис. 12.6. Страница с базовым стилем и цветами

Добавьте приведенные далее стили (листинг 12.2) в свой код.

Я применил пользовательские настройки цветов (если забыли, что это такое, еще раз прочтите раздел 2.6). Использование переменных цвета позволит сэкономить массу времени, если по ходу работы придется изменять цвета. Однажды я работал над проектом, где дизайнер решил поменять основной цвет, когда почти все было готово. К счастью, мне пришлось лишь изменить значение пары переменных, вместо того чтобы искать нужные места в сотнях строк кода и вручную их править.

Листинг 12.2. Базовые стили, включая переменные цвета

```
html {
  --brand-green: #076448;
  --dark-green: #099268;
  --medium-green: #20c997;
  --text-color: #212529;
  --gray: #868e96;
  --light-gray: #f1f3f5;
  --extra-light-gray: #f8f9fa;
  --white: #fff;

  box-sizing: border-box;
  color: var(--text-color);
}
*,
*::before,
*::after {
  box-sizing: inherit;
}

body {
  margin: 0;
  font-family: Helvetica, Arial, sans-serif;
  line-height: 1.4;
  background-color: var(--extra-light-gray);
}

h1, h2, h3, h4 {
  font-family: Georgia, serif;
}

a {
  color: var(--medium-green);
}
a:visited {
  color: var(--brand-green);
}
a:hover {
  color: var(--brand-green);
}
a:active {
  color: red;
}
```

Присвоение
всем переменным
значения цвета

Установка
шрифтов
заголовка

Используются
переменные
для назначения
цвета

Заполнитель для активных ссылок.
Позже нам понадобится красный цвет

ПРИМЕЧАНИЕ

В этом примере я задействовал пользовательские настройки CSS. Чтобы его упростить, не понадобятся дополнительные инструменты. В своих проектах, если нужно будет поддерживать Internet Explorer или другой устаревший браузер, используйте лучше CSS-препроцессоры. В приложении Б приведена более подробная информация о препроцессорах.

Я также добавил заполнитель для активных ссылок. Мы скоро вернемся и ликвидируем этот пробел. А пока придадим нашей странице нормальный внешний вид. Разместим на ней основные модули, зададим цвета и шрифты (рис. 12.7). Пока не обращайте внимания на расстояния между элементами.

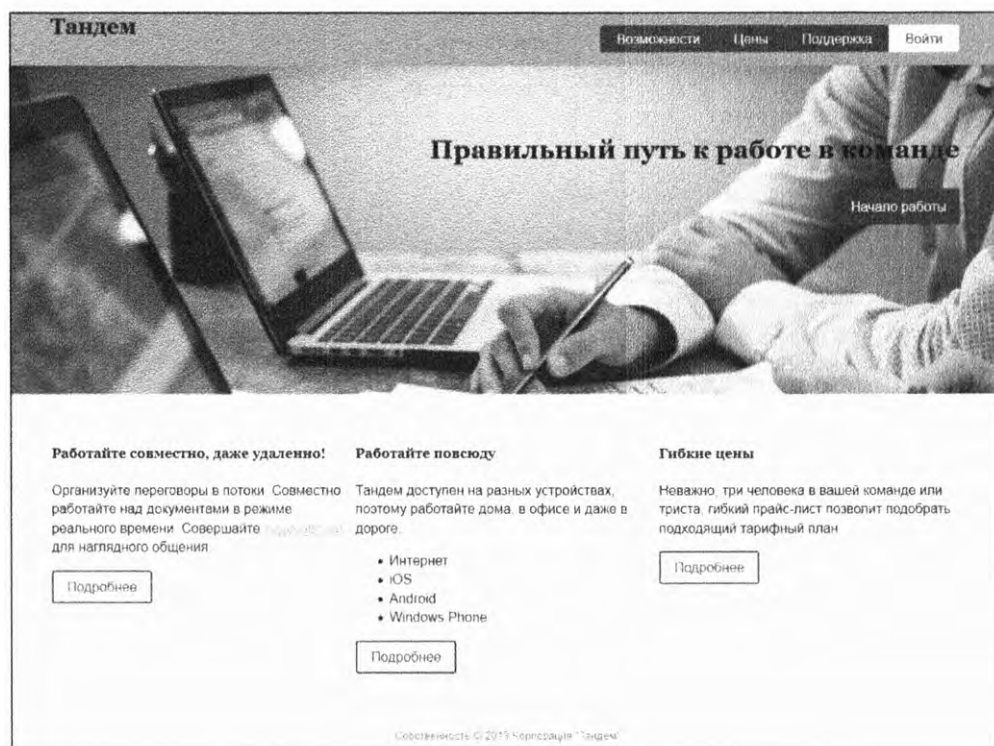


Рис. 12.7. Примерное расположение элементов на странице с применением базовых стилей

Начнем сверху и будем работать над страницей в три этапа: шапка, hero-изображение и основной раздел с тремя колонками текста. В большинстве случаев я использую техники, описанные в предыдущих главах. А затем вернемся чуть назад и придадим странице действительно красивый вид.

Итак, шапка сайта и панель навигации. Эта секция состоит из трех модулей: `nav-container`, `home-link` и `top-nav` (листинг 12.3). Добавьте этот код в свою таблицу стилей.

Листинг 12.3. Стили шапки

```

.nav-container {
  background-color: var(--medium-green);
}
.nav-container__inner {
  display: flex;
  justify-content: space-between;
  max-width: 1080px;
  margin: 0 auto;
}

.home-link {
  color: var(--text-color);
  font-size: 1.6rem;
  font-family: Georgia, serif;
  font-weight: bold;
  text-decoration: none;
}

.top-nav {
  display: flex;
  list-style-type: none;
}

.top-nav a {
  display: block;
  padding: 0.3em 1.25em;
  color: var(--white);
  background: var(--brand-green);
  text-decoration: none;
  border-radius: 3px;
}

.top-nav a:hover {
  background-color: var(--dark-green);
}

.top-nav__featured > a {
  color: var(--brand-green);
  background-color: var(--white);
}

.top-nav__featured > a:hover {
  color: var(--medium-green);
  background-color: var(--white);
}

```

Центрирование контента
и ограничение его по ширине
до 1080 пикселей

Применяем flexbox-верстку,
чтобы разместить элементы навигации в строку

Добавляем цвета и отступы
для каждого элемента навигации

Вся шапка находится внутри контейнера `nav-container`. Здесь я применил шаблон двойного контейнера, чтобы расположить внутренний элемент ровно по центру (эта техника описывается в главе 4). Получается, что цвет распространяется до самых краев страницы, в то время как ширина основного контента ограничена. Этот элемент — flex-контейнер со свойством `justify-content: space-between`, расталкивающий контент к разным краям: `home-link` — влево, а `top-nav` — вправо. Модуль `top-nav` — еще один flex-контейнер, поэтому все ссылки в нем располагаются в строку, а все цвета задаются пользовательскими свойствами.

Теперь зададим стили области hero-изображения. Для этого создадим еще два модуля: для изображения и для кнопки. Добавьте показанный далее код (листинг 12.4) в таблицу стилей.

Листинг 12.4. Стили hero-изображения и кнопки

```
.hero {
  background: url(collaboration.jpg) no-repeat;
  background-size: cover;
  margin-bottom: 2.5rem;
}
.hero_inner {
  max-width: 1080px;
  margin: 0 auto;
  padding: 50px 0 200px;
  text-align: right;
}
.hero h2 {
  font-size: 1.95rem;
}

.button {
  display: inline-block;
  padding: 0.4em 1em;
  color: var(--brand-green);
  border: 2px solid var(--brand-green);
  border-radius: 0.2em;
  text-decoration: none;
  font-size: 1rem;
}
.button:hover {
  background-color: var(--dark-green);
  color: var(--white);
}
.button--cta {
  padding: 0.6em 1em;
  background-color: var(--brand-green);
  color: var(--white);
  border: none;
}
```

Шаблон двойного контейнера

Использование отступов, чтобы примерно позиционировать слоган и кнопку

Стандартные стили кнопок

Вариант стиля для СТА-кнопки

В модуле hero-изображения тоже использован прием двойного контейнера, как и в шапке сайта. У внутреннего элемента также есть поля. Пока их размеры заданы приблизительно. Когда все элементы будут примерно на своих местах, я вернусь немного назад и поясню, почему так важно тщательно прорабатывать дизайнерский макет.

Я также определил модуль Button для кнопки. Стил ь по умолчанию — белая кнопка с зелеными границами и зеленым цветом текста. Это стил ь для кнопок в нижней части страницы, там, где располагаются колонки с текстом. Поэтому я определил для СТА-кнопки стил ь с зеленым однотонным фоном и белым тек-

стом. (СТА означает call to action, или «призыв к действию». Это маркетинговый термин для ключевого элемента, внимание к которому вы хотите привлечь, — в нашем случае для кнопки **Начало работы**.) И наконец, добавим основной блок с тремя колонками (рис. 12.8).

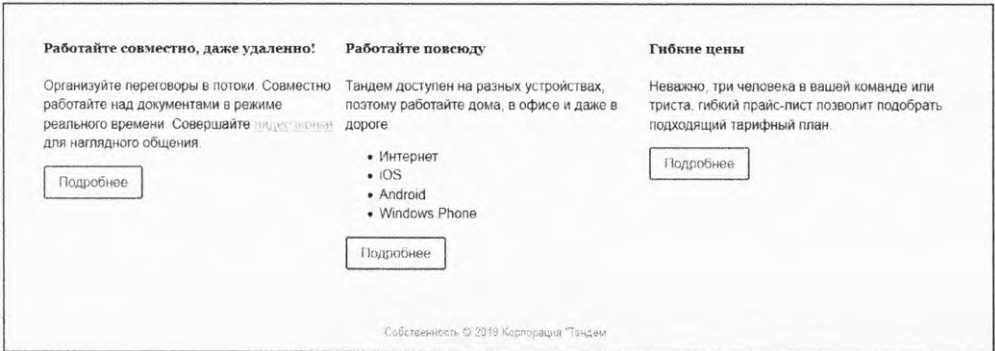


Рис. 12.8. Примерное оформление основного раздела страницы

Основной раздел страницы состоит из контейнера, ограничивающего ширину, свойства `tile-row`, формирующего колонки, и свойства `tile` для создания белых блоков в каждой колонке. Добавьте в таблицу стилей код (листинг 12.5), который также задает стили для подвала. Стиль для кнопок мы уже задали, поэтому нет необходимости делать это второй раз.

Листинг 12.5. Три колонки и блоки с текстом

```
.container {
  margin: 0 auto;
  max-width: 1080px;
}

.tile-row {
  display: flex;
}

.tile-row > * {
  flex: 1;
}

.tile {
  background-color: var(--white);
  border-radius: 0.3em;
}

.page-footer {
  margin-top: 3em;
  padding: 1em 0;
  background-color: var(--light-gray);
}
```

← Контейнер с ограничением ширины 1080 пикселей, как и другие разделы на нашей странице

Все колонки одинаковой ширины

```

    color: var(--gray);
}
.page-footer__inner {
    margin: 0 auto;
    max-width: 1080px;
    text-align: center;
    font-size: 0.8rem;
}

```

← Контейнер с ограничением ширины 1080 пикселей, как и другие разделы на нашей странице

И снова в этом листинге используется вложенный контейнер, ограничивающий ширину до 1080 пикселей. Заданы также белый фон и скругление границ для блоков с текстом.

Подвал страницы — это пример того, как задействовать контраст, чтобы, наоборот, не привлекать внимания. Шрифт здесь не крупный, серые буквы на светло-сером фоне. Это наименее важная составляющая страницы, поэтому она не должна выделяться. Наоборот, она сливается с фоном и как бы намекает пользователю: «Это не то, что ты ищешь».

12.2.1. Разбираемся с нотациями цветов

Цвета в нашей палитре представлены их шестнадцатеричными значениями. Это краткое представление, которое веб-разработчики использовали изначально, но явно не самый простой и удобный способ. Поэтому в CSS есть и другие способы представления, например функции `rgb()` и `hsl()`.

Функция `rgb()` — способ задать цвет как сочетание красного, зеленого и синего, применяя десятичные значения вместо шестнадцатеричных. То есть значения не от 00 до FF, а от 0 до 255: `rgb(0,0,0)` — чисто черный (все равно что `#000`), а `rgb(136,0,0)` — кирпичный цвет (`#800`).

Стоит отметить, что нотация RGB, как и шестнадцатеричные значения, не является интуитивно понятной. Трудно осознать, что за цвет скрывается за `#2097c9` или его RGB-эквивалентом, а тем более угадать, как он будет выглядеть на странице. Можно, конечно, предположить, что если красный принимает значение 20, что довольно мало, зеленый — 97 — среднее значение, а синий — c9, что много, то оттенок будет сине-зеленым. Но насколько темным? Насколько ярким? В общем, RGB интуитивно непонятно. Это представление создавалось для обработки компьютером, а не человеком.

HSL — нотация, которая изначально создавалась для того, чтобы быть понятной человеку. Расшифровывается как *hue* (оттенок), *saturation* (насыщенность), *lightness* (яркость). Синтаксис выглядит следующим образом: `hsl(198, 73%, 46%)` — то же самое, что и `#2097c9`.

Функция `hsl()` принимает три значения. Первое — это оттенок, число между 0 и 359. Оно обозначает 360° окружности, которая плавно меняет цвет: красный (0), желтый (60), зеленый (120), голубой (180), синий (240), фиолетовый (300) и снова красный. Второе значение — насыщенность, процентное представление

интенсивности цвета, где 100 % обозначает яркий, насыщенный цвет, а 0 % — его полное отсутствие, то есть серый. А последнее значение определяет, насколько цвет яркий. Среднее значение 50 %, его увеличение делает цвет светлее (100 % — это белый), а уменьшение — темнее (0 % — это черный). Например, `hsl(198, 73%, 46%)` означает сине-голубой оттенок, довольно насыщенный и средней яркости. То есть в результате получаем богатый голубой, чуть светлее цвета неба.

В табл. 12.1 сравниваются нотации — шестнадцатеричное представление, RGB и HSL, а также даны привычные названия. (Всего в CSS определены около 150 названий цветов, которые также можно использовать.)

Таблица 12.1. Сравнение представлений цветов

Название	Шестнадцатеричное	RGB	HSL
Синий	#0000ff	rgb(0, 0, 255)	hsl(240, 100%, 50%)
Лиловый	#e6e6fa	rgb(230, 230, 250)	hsl(240, 67%, 94%)
Коралловый	#ff7f50	rgb(255, 127, 80)	hsl(16, 100%, 66%)
Золотой	#ffd700	rgb(255, 215, 0)	hsl(51, 100%, 50%)
Зеленый	#008000	rgb(0, 128, 0)	hsl(120, 100%, 25%)
Загар	#d2b48c	rgb(210, 180, 140)	hsl(34, 44%, 69%)

Вообще, наилучший способ начать понимать HSL — немного с ним поэкспериментировать. И вновь я призываю вас посетить сайт hslpicker.com. Он представляет собой интерактивную панель смешения цветов, состоящую из трех ползунков. Есть также четвертый, дополнительный, который отвечает за прозрачность. Смотрите и наслаждайтесь тем, как изменяется цвет, когда вы передвигаете один из ползунков.

ПРИМЕЧАНИЕ

Нотации RGB и HSL имеют аналоги с дополнительным альфа-каналом — `rgba()` и `hsla()`. Они принимают четвертое значение в диапазоне от 0 до 1, отвечающее за прозрачность. Сейчас некоторые браузеры поддерживают восьмизначное hex-значение, где последние два числа также отведены для альфа-канала.

Преобразование цветов в браузере

Преобразуем все шестнадцатеричные значения в HSL. Существует много онлайн-ресурсов, таких как hslpicker.com, которые отображают цвета сразу тремя способами. Однако самый легкий способ — использование инструментов разработчика прямо в браузере, так как они всегда под рукой. Я покажу, как это сделать с помощью Chrome.

Как только страница загрузится, откройте инструменты разработчика (**Cmd+Option+I** в macOS, **Ctrl+Shift+I** в Windows). На панели **Elements** (Элементы) щелкните кнопкой мыши на элементе **html**, чтобы выбрать его. Связанные с ним стили, включая пользовательские, отобразятся на панели **Styles** (Стили) (рис. 12.9).

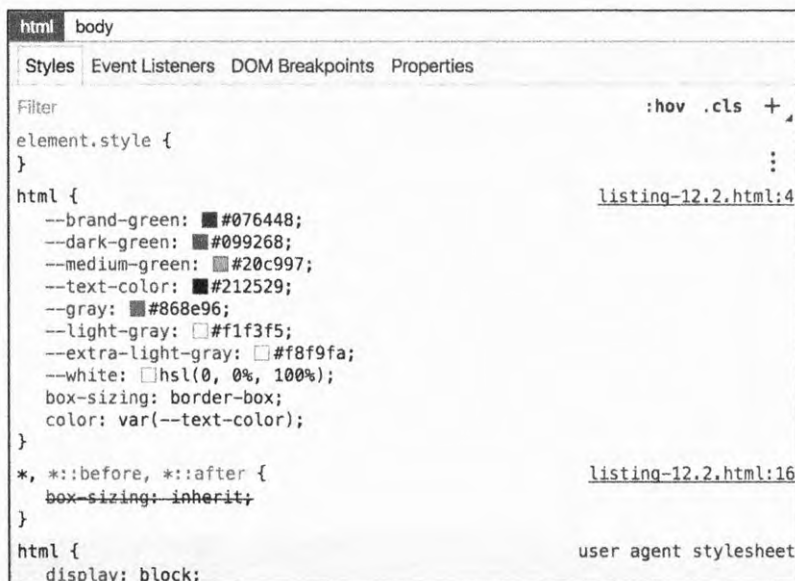


Рис. 12.9. Применяемые стили показаны на панели Styles в инструментах разработчика. Нажав и удерживая клавишу Shift, щелкните на квадратном индикаторе цвета, чтобы выбрать нотацию шестнадцатеричную, RGB или HSL

В строках цветов есть небольшие индикаторы, в которых отображается цвет. Нажав и удерживая клавишу Shift, щелкните кнопкой мыши на индикаторе цвета, чтобы заменить шестнадцатеричное значение на RGB. Если щелкнуть еще раз, то значение RGB сменится на HSL. Третий щелчок вернет шестнадцатеричное значение.

ПРИМЕЧАНИЕ

Данный метод переключения между представлениями работает и в инструментах разработчика Firefox. Но, к сожалению, только для обычных свойств, а не для цветов, заданных с помощью пользовательских настроек, как в нашем проекте.

Если хотите подробнее изучить эту тему, то щелкните кнопкой мыши на квадратном индикаторе цвета. Откроется окно палитры (рис. 12.10). В нем можно изменять цвета, выбирать их из палитры и переключаться между различными нотациями. Также имеется пипетка, которая позволяет захватить цвет прямо со страницы. В браузере Firefox есть похожий функционал, хотя он значительно уже.

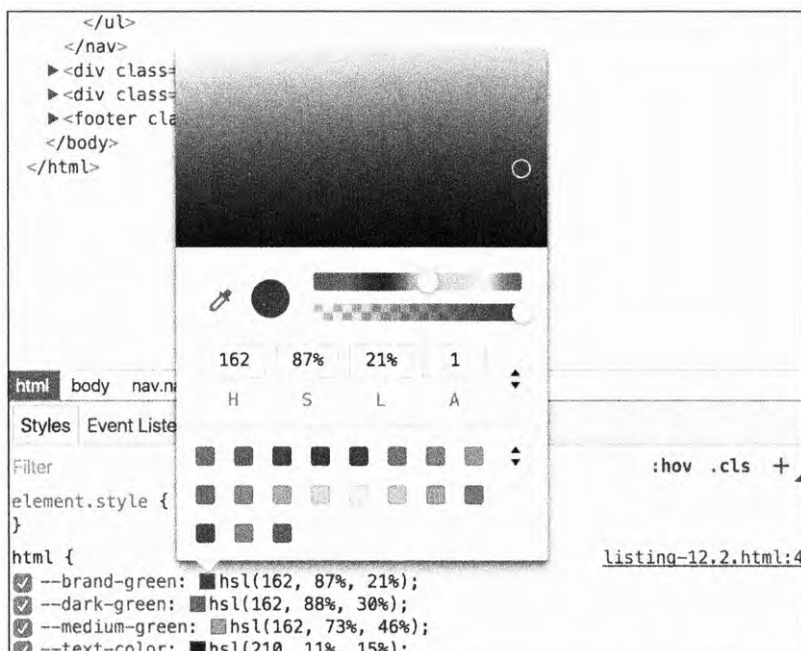


Рис. 12.10. Использование окна палитры для настройки цвета

Применение HSL-значений в таблице стилей

В большинстве случаев шестнадцатеричных значений вполне достаточно. Однако использование HSL дает свободу для экспериментов с цветом, что позволяет найти новые решения для вашего сайта. Представим все цвета на странице в HSL-нотации и немного поэкспериментируем.

Скопируйте HSL-значения из инструментов разработчика в свою таблицу стилей. Ваш CSS-код должен выглядеть следующим образом (листинг 12.6).

Листинг 12.6. Преобразование шестнадцатеричных значений в HSL

```
html {
  --brand-green: hsl(162, 87%, 21%);
  --dark-green: hsl(162, 88%, 30%);
  --medium-green: hsl(162, 73%, 46%);
  --text-color: hsl(210, 11%, 15%);
  --gray: hsl(210, 7%, 56%);
  --light-gray: hsl(210, 17%, 95%);
  --extra-light-gray: hsl(210, 17%, 98%);
  --white: hsl(0, 0%, 100%);

  box-sizing: border-box;
  color: var(--text-color);
}
```

Зеленые цвета имеют
один и тот же оттенок

Цвет текста
и серые цвета
не чисто серые

Как только мы начинаем использовать HSL, сразу становятся очевидными несколько явлений. Первое, чего вы могли не заметить ранее, — все зеленые цвета одного оттенка. Вы, конечно, вряд ли поймете, что это бирюзовый, пока не заглянете в браузер, но определенная закономерность явно просматривается. Вы бы никогда этого не увидели, применяя шестнадцатеричные значения. А зная эту закономерность, смело добавляйте новые цвета в палитру. Например, я попытался добавить чуть более яркий цвет `hsl(162, 50%, 80%)`, а потом немного его подкорректировать с помощью инструментов разработчика.

Еще вы можете увидеть, что серые цвета не совсем серые из-за изменений насыщенности и все опять же одного оттенка. Вряд ли вы могли бы это заметить, даже глядя на сами цвета. Но это небольшой секрет, который позволяет странице выглядеть более гармонично. На самом деле чистый серый цвет редко встречается в природе, поэтому наши глаза лучше воспринимают серые цвета с небольшой примесью, даже если она незаметна.

ПРИМЕЧАНИЕ

Дизайнеры почти всегда включают серый в палитру проекта. Мой опыт свидетельствует, что почти всегда приходится задействовать еще один, иной серый оттенок. И каким бы он ни был, светлым, очень светлым или средним между светло-серым и очень светло-серым, все равно придется это делать. Главная проблема здесь в том, какое имя присвоить переменной. Поэтому я советую использовать в названии числовые значения, например `--gray-50` или `--gray-80`, где число отражает яркость. Таким образом вы всегда без особых проблем сможете добавить еще одно значение.

Итак, в настоящих проектах далеко не всегда стоит использовать HSL для представления абсолютно всех цветов. Но если возникнет такая необходимость, теперь вы знаете, как это сделать. Это сильно упростит работу.

12.2.2. Добавление цветов в палитру

Вы можете столкнуться с ситуацией, когда нужен цвет, не предусмотренный в проекте. Например, требуется красный для сообщения об ошибке или синий для оформления окна информации. Опытные дизайнеры обычно это предусматривают, но все равно у вас, скорее всего, возникнет необходимость добавить новый цвет.

В нашем проекте я оставил место для стиля активной ссылки. Активные ссылки традиционно красные, что определяется браузерными стилями. Но этот красный слишком яркий, почти мультяшный, что совсем не вписывается в дизайн страницы. Давайте найдем другой оттенок, не такой едкий.

Самый простой способ найти цвет, который хорошо сочетается с данным, — взять *комплементарный* к нему. Это цвет, который находится на противоположной стороне цветового круга: комплементарный к синему — желтый, к зеленому — пурпурный, а к красному — бирюзовый.

В нотации HSL очень легко найти комплементарный цвет — нужно прибавить 180 к значению оттенка или столько же вычесть из него. Наш зеленый имеет значение оттенка 162. Прибавив 180, получим 342, что соответствует красному с легким оттенком пурпурного. Можно также вычесть 180, получим -18 , что в целом ровно то же самое, что 342. То есть `hsl(-18, 87%, 21%)` и `hsl(342, 87%, 21%)` отобразятся на странице одинаково. Я все же предпочитаю использовать значения от 0 до 360, это более привычно.

У нас есть оттенок, но по-прежнему надо выбрать насыщенность и яркость. Обычная (неактивная) ссылка на странице выглядит как `hsl(162, 73%, 46%)`, посмотрим, подойдут ли нам эти значения. Но, так как основной цвет — зеленый, мы не хотим, чтобы второстепенные цвета привлекали слишком много внимания к себе. Поэтому насыщенность немного уменьшим, например на 10 %. В результате получим цвет `hsl(342, 63%, 46%)`. На рис. 12.11 показано, как активная ссылка выглядит на странице.

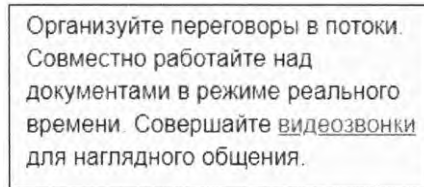


Рис. 12.11. Красная активная ссылка

Добавим этот цвет в таблицу стилей. Включите в проект следующий код (листинг 12.7). Он задает значение переменной `--red`, а затем использует ее для задания цвета активной ссылке.

Листинг 12.7. Добавление красного цвета в палитру

```
html {
  --brand-green: hsl(162, 87%, 21%);
  --dark-green: hsl(162, 88%, 30%);
  --medium-green: hsl(162, 73%, 46%);
  --text-color: hsl(210, 11%, 15%);
  --gray: hsl(210, 7%, 56%);
  --light-gray: hsl(210, 17%, 95%);
  --extra-light-gray: hsl(210, 17%, 98%);
  --white: hsl(0, 0%, 100%);
  --red: hsl(342, 63%, 46%);
  box-sizing: border-box;
  color: var(--text-color);
}
...
a:active {
  color: var(--red);
}
```

← Назначение переменной red

← Использование переменной для активных ссылок

Обновим страницу и посмотрим, как это выглядит. К сожалению, активное состояние ссылки не отображается по умолчанию. Вы можете навести указатель мыши на ссылку, нажать и удерживать кнопку мыши, но, как только ее отпустите, ссылка вернется в обычное состояние. Либо используйте инструменты разработчика, чтобы спровоцировать активное состояние ссылки.

Щелкните правой кнопкой мыши на ссылке и выберите в контекстном меню пункт **Inspect element** (Исследовать элемент). Откроется панель инструментов разработчика. На панели **Elements** (Элементы) щелкните на элементе **a** правой кнопкой мыши и в контекстном меню выберите пункт **Force state ▶ :active** (Принудительное состояние ▶ :active) (рис. 12.12). Браузер отобразит ссылку в активном состоянии.

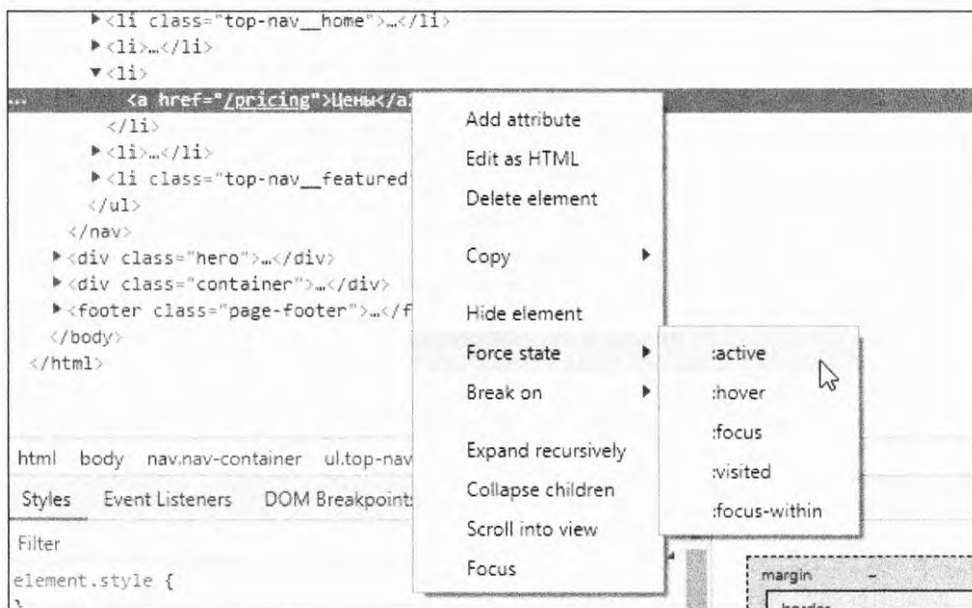


Рис. 12.12. Инструменты разработчика позволяют принудительно переводить элемент в нужное состояние, чтобы вы могли прикинуть, как будут выглядеть стили

Теперь можно оценить, как выглядит красный цвет. И при необходимости немного скорректировать цвет, опять же применяя инструменты разработчика.

Работа с цветом — далеко не точная наука, но HSL ее сильно упрощает. Попробуйте использовать комплементарные цвета к другим оттенкам, задействованным на странице. Поэкспериментируйте с насыщенностью и яркостью, чтобы найти оптимальный вариант.

Если хотите глубже изучить, как работать с цветами, почитайте в Интернете статьи по теории цвета. Очень хорошая статья для новичков написана Натальей Шелберн (Natalya Shelburne): tallys.github.io/color-theory/.

12.2.3. Применение контраста при выборе цвета текста

Вы, вероятно, уже заметили, что для текста используется темно-серый, а не чистый черный (`#000`) цвет. А значение яркости нотации HSL составляет 15%, а не 0. Серый вместо черного — распространенная практика. На мониторах с задней подсветкой чистый черный текст на белом (`#fff`) фоне выглядит чересчур контрастным. От этого глаза быстро устают, особенно когда читаешь большие фрагменты текста. Точно так же и с белым текстом на черном фоне. В этом случае лучше взять темно-серый вместо черного или светло-серый вместо белого. Это незаметно для глаз, но устают они намного меньше.

Слишком резкий контраст для текста использовать не стоит, но и его полное отсутствие — это плохо. Серый текст на светло-сером фоне трудно воспринимать людям, особенно с плохим зрением. Его будет трудно читать на экране смартфона на ярком солнце. Так какой же контраст считать подходящим?

Чтобы понять это, можно обратиться к руководству консорциума W3C по обеспечению доступности веб-контента (Web Content Accessibility Guidelines (WCAG)). Оно определяет как минимальный (AA), так и максимальный (AAA) уровень контрастности. Поскольку более крупный текст читать проще, чем мелкий, оба уровня подразумевают меньший порог контрастности для крупных шрифтов. Рекомендованные уровни контрастности представлены в табл. 12.2.

Таблица 12.2. Рекомендации по уровню контрастности согласно WCAG

Текст	Уровень AA	Уровень AAA
Обычный	4.5:1	7:1
Крупный	3:1	4.5:1

Согласно WCAG крупным считается любой текст с размером обычного шрифта больше 18 пунктов (24 пикселей) или полужирного свыше 14 пунктов (18,667 пиксела). Это означает, что контрастность основного текста на вашей странице должна соответствовать значению для обычного текста, а заголовков — значению для крупного текста или превышать их.

В руководстве WCAG даже имеется математическая формула для расчета нужного уровня контрастности, но я обычно ею не пользуюсь. Существует множество инструментов для проверки контрастности в CSS, доступных онлайн.

У каждого из онлайн-инструментов есть свои плюсы и минусы. Один из моих любимых доступен по адресу leaverou.github.io/contrast-ratio. Он поддерживает любой допустимый формат представления цвета. Просто укажите значения цветов фона и текста, и в автоматическом режиме будет рассчитан уровень контрастности (рис. 12.13). Наведите указатель мыши на это значение, чтобы увидеть, для какого уровня, AA или AAA, и для какого размера шрифта оно допустимо.

В большинстве случаев не стоит пытаться достичь уровня соответствия AAA. И в самом руководстве WCAG это признают. Конечно, неплохо, если основной текст страницы достигает уровня AAA, но не будьте чересчур строгими к себе, когда работаете с цветными метками и декоративным текстом.

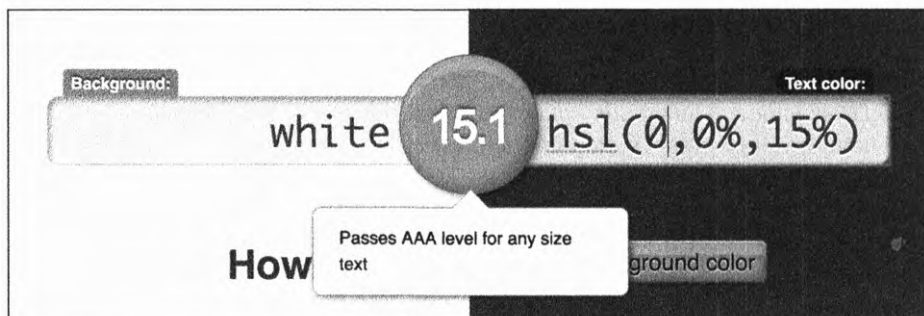


Рис. 12.13. Уровень контрастности фона и шрифта составляет 15.13:1

Помните также, что математика — это, несомненно, хорошо, но далеко не всегда. Некоторые шрифты читать легче других. Особенно важно учитывать это в проектах, где используются тонкие шрифты. На рис. 12.14 представлены два одинаковых абзаца. Читаются они по-разному, хотя цвет один и тот же.

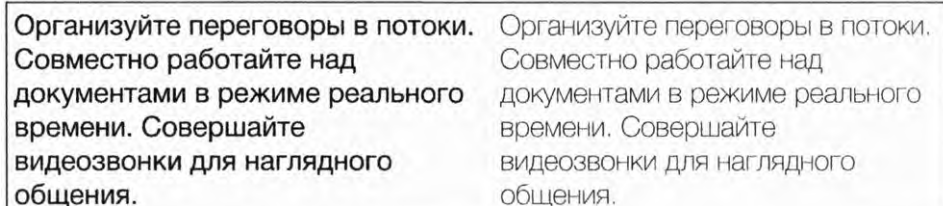


Рис. 12.14. Более тонкий шрифт выглядит менее контрастным, несмотря на то что цвет один и тот же

В обоих случаях используется шрифт Helvetica Neue. Но у текста слева насыщенность шрифта задана 300 (такой обычно называют книжным), а справа — 100 (тонкий). Первый пример с контрастом 7:1 выглядит прекрасно, а вот второй требует усиления.

СОВЕТ

Не все шрифты поддерживают свойство насыщенности. Но уж если вы их применяете, убедитесь, что они хорошо читаются.

12.3. Интервалы

Подробно поговорив о цветах, мы наконец-то можем вернуться к позиционированию объектов на странице с помощью интервалов. Вероятно, это одна из самых нудных частей процесса веб-разработки. Возможно, вам с дизайнером придется несколько раз к ней возвращаться и все менять, пока не устраните все несоответствия.

Большая часть работы сводится к простому заданию полей элементов. Это одна из самых простых задач, с которой и стоит начать, хотя не исключено, что тут придется немного подгонять значения. Перед тем как приступить, обратите внимание на две вещи: использовать или нет относительные единицы измерения и как высота строки влияет на вертикальные отступы.

12.3.1. Единицы em или пикселы?

Вам предстоит принять важное решение — использовать единицы em или пикселы. Обычно дизайнеры работают с пикселями, поэтому преимущества второго варианта очевидны. Хотя, если вы оценили достоинства относительных единиц измерения em или rem, можете воспользоваться ими.

Обратите внимание на значения на панели навигации (рис. 12.15). Дизайн предусматривает интервал 10 пикселей между элементами и 10 пикселей от края кнопки до нижнего края панели.

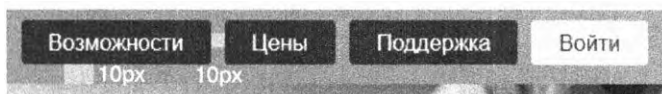


Рис. 12.15. Проект предусматривает интервалы 10 пикселей вокруг каждой кнопки

В главе 2 я уже описывал преимущество использования относительных единиц измерения. В частности, они позволяют задать относительный размер шрифта (`font-size: calc(0.5em + 1vw)`), который изменяется пропорционально другим элементам. При увеличении масштаба страницы будут увеличиваться не только размер шрифта, но и поля элементов, если они заданы в единицах em или rem. А если у пользователя в браузере задан строго определенный размер шрифта, это позволит разметке не разбегаться и сохранить внешний вид страницы.

Но так как эта техника появилась сравнительно недавно, очень немногие дизайнеры применяют ее на практике. Лучше заранее обсудить со специалистом использование относительных единиц измерения. Хотя, скорее всего, вам придется самостоятельно воплощать эту идею в жизнь.

Пикселы, несомненно, облегчат вам работу сегодня, но не в долгосрочной перспективе, так как такой дизайн теряет в гибкости. Возможно, в будущем это доставит куда больше хлопот, чего нельзя знать заранее наверняка. Относительные единицы измерения сложнее в применении, но повышают адаптивность сайта.

Так как задание полей и других размеров в пикселях (например, все поля и отступы по 10 пикселей) — довольно примитивный процесс, я сделаю акцент на использовании em при создании панели навигации.

В макете страницы явно указано, что все расстояния на панели навигации должны составлять 10 пикселей (см. рис. 12.15). Поскольку мы знаем, что размер шрифта равен 16 пикселям, не составит труда посчитать, что нужное расстояние равняется $0,625\text{ em}$, то есть 10, деленное на 16. А теперь добавим следующее объявление (листинг 12.8) в таблицу стилей.

Листинг 12.8. Использование полей и отступов на панели навигации

```
.nav-container {
  background-color: var(--medium-green);
}
.nav-container__inner {
  display: flex;
  justify-content: space-between;
  max-width: 1080px;
  margin: 0 auto;
  padding: 0.625em 0;
}

/* ... */

.top-nav {
  display: flex;
  list-style-type: none;
  margin: 0;
}
.top-nav > li + li {
  margin-left: 0.625em;
}
```

Добавление отступа 10 пикселей
сверху и снизу для всей панели

Удаление стандартного поля,
применяемого браузером по умолчанию

Добавление расстояния 10 пикселей
между всеми элементами навигации

Когда вы работаете с интервалами на странице, важно понимать отличие полей от отступов. В данном случае логичнее применить отступы для выравнивания по вертикали и делать так для всего контейнера `nav-container__inner`, в том числе для названия компании в левом верхнем углу. А поля я использовал, чтобы задать расстояние между кнопками по горизонтали, так как не хотел, чтобы они вплотную примыкали друг к другу.

Задание расстояния от нижнего края hero-изображения до основного блока с текстом тоже очевидно (рис. 12.16). Поскольку оно задается как внешнее, тут понадобятся поля.

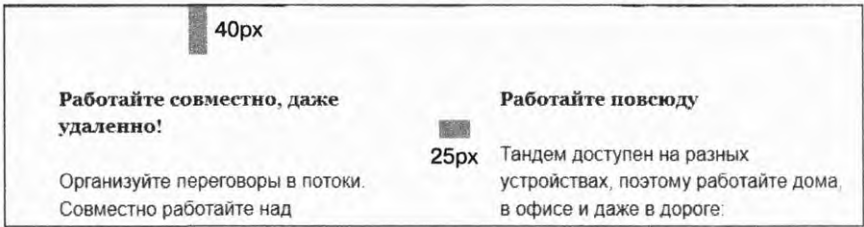


Рис. 12.16. Поля (40 пикселей) ниже hero-изображения и между колонками основного угла текста (25 пикселей)

Точно так же можно перевести это значение в единицы `em`. 40 пикселей под hero-изображением равняются 2,5 `em` ($40 / 16 = 2,5$ — и снова используем поля), а 25 пикселей составляют 1,5625 `em` ($25 / 16$). Добавьте эти поля в свой код (листинг 12.9).

Листинг 12.9. Добавление полей под hero-изображением и между колонками основного текста

```
.hero {
  background: url(collaboration.jpg) no-repeat;
  background-size: cover;
  margin-bottom: 2.5rem;
}

/* ... */

.tile-row {
  display: flex;
}
.tile-row > * {
  flex: 1;
}
.tile-row > * + * {
  margin-left: 1.5625em;
}
```

Интервал 40 пикселей
под hero-изображением

Интервал 25 пикселей
между колонками

Задание интервала между контейнерами, у которых есть фоновое изображение или цвет, довольно очевидно. Немного сложнее дела обстоят с интервалами между строками текста, например абзацем и заголовком.

12.3.2. Вычисление высоты строки

Наш макет предусматривает определенное пространство вокруг текста. На рис. 12.17 представлены конкретные значения. (Возможно, разглядеть трудновато, но на свет-ло-сером фоне есть белый блок для текста. Отступ от краев этого блока до текста составляет 25 пикселей.)

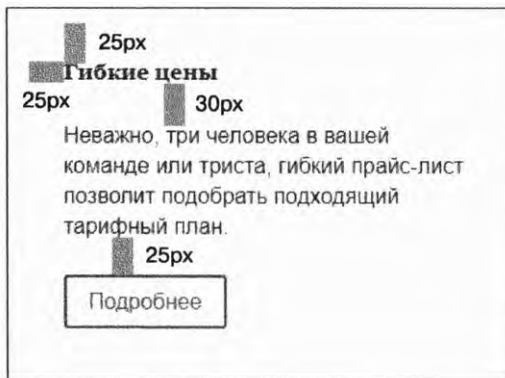


Рис. 12.17. Желаемое расстояние между краями блока и текстом

Создание расстояния 25 пикселей представляет собой не что иное, как задание отступов для блока: $25 / 16 = 1,5625 \text{ em}$. А вот задать 30 пикселей между заголовком

и абзацем — далеко не тривиальная задача. Если бы я просто задал поля 30 пикселей для двух элементов, фактическое расстояние равнялось бы почти 36 пикселям. Чтобы понять, почему так происходит, рассмотрим, как вообще определяется высота элемента.

В блочной модели содержимое блока элемента окружено отступом, затем идет граница элемента, а затем поля. То есть блок для таких элементов, как заголовок и абзац, фактически занимает чуть больше пространства, чем видимый текст: высота строки элемента чуть больше, чем высота букв. Это показано на рис. 12.18. Высота текста составляет 1 em, но высота строки немного превышает это значение.

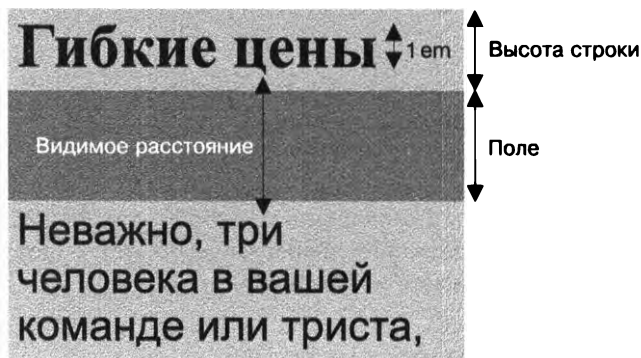


Рис. 12.18. Высота строки определяет высоту блока

На вашей странице высота строки равняется 1,4. Это задано в свойствах элемента `body` и наследуется от него. Собственно, поэтому высота элемента, содержащего одну строку текста, составляет 1,4 em. А сам текст центрирован по вертикали. При размере шрифта 16 пикселей высота блока составляет 22,4 пикселя. Эти дополнительные 6,4 пикселя поровну распределяются над текстом и под ним.

Если вы зададите нижнее поле 30 пикселей, то по факту дополнительно к этому расстоянию получите 3,2 пикселя как отступ под заголовком и еще 3,2 пикселя — как отступ перед абзацем (расстояния одинаковые, так как для абзаца и заголовка заданы одинаковые размер шрифта и высота строки). То есть фактическая высота составляет 36,4 пикселя.

ПРИМЕЧАНИЕ

Дизайнеры привыкли работать с понятием «интерлиньяж», или межстрочное расстояние. В CSS оно определяется высотой строки, что не является прямым аналогом интерлиньяжа. Подробнее рассмотрим это в следующей главе.

Обычно дизайнеры не обращают внимания на такие мелочи, как 1–2 пикселя, но вот 6,5 пикселя могут серьезно испортить внешний вид страницы. А если вы используете еще более крупный размер шрифта хотя бы для одного из элементов, эта погрешность будет еще больше.

Один из способов исправить ситуацию — просто вычесть лишнее из значения, задающего поля. Например, в нашем случае сделать его не 30 пикселей, а 24, то есть отнять 6 пикселей. А разделив на 16, получить значение 1,5 em. Добавьте код из листинга 12.10 в свой проект.

Листинг 12.10. Задание расстояний для блока и текста

```
p {
  margin-top: 1.5em;
  margin-bottom: 1.5em;
}
```

Добавление полей
к абзацам в базовых стилях

```
/* ... */

.tile {
  background-color: var(--white);
  border-radius: 0.3em;
  padding: 1.5625em;
}
```

Добавление отступов
внутри плиток

```
.tile > h4 {
  margin-bottom: 1.5em;
}
```

Добавление поля
под заголовком блока

Мы задали поля 1,5 em для всего документа, что означает: этот стиль будет применяться к любому абзацу на странице. А также повторили данное значение для заголовков в блоках с текстом (`.tile > h4`), для того чтобы расстояние под заголовком всегда было одинаковое, даже если после него нет абзаца с текстом. Поля устроены таким образом, что их значения будут схлопываться, образуя между заголовком и абзацем расстояние ровно 30 пикселей.

Осталось последнее: расположить слоган и кнопку на hero-изображении в соответствии с макетом (рис. 12.19).

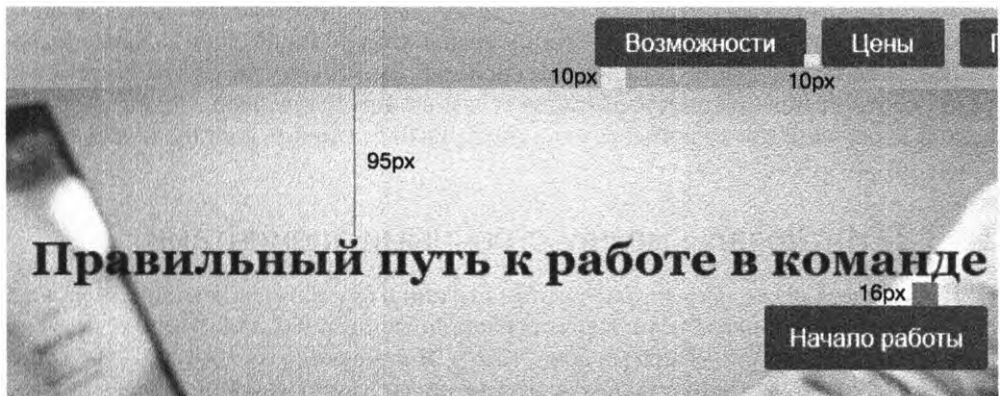


Рис. 12.19. Согласно проекту, расстояние от слогана до верхнего края изображения составляет 95 пикселей, от слогана до кнопки — 16 пикселей

В данном случае тоже придется учесть высоту строки, так как размер шрифта еще крупнее — 1,95 em. Умножим его на 16 и получим 31,2 пиксела. Снова умножим это значение на 1,4, чтобы получить высоту строки, равную 43,68 пиксела, то есть по 6 пикселей над текстом и под ним.

Так как над текстом уже есть 6 пикселей, надо добавить только 89 пикселей, чтобы получить желаемые 95 пикселей. Аналогично достаточно задать нижнее поле равным 10 пикселям, чтобы получить 16 пикселей, как на макете. Это простая арифметика. Хотя иногда лучше подогнать значения интервалов вместе с дизайнером, чтобы все остались довольны.

Сейчас, когда вы знаете, что нужны 89 пикселей над слоганом и 10 пикселей под ним, можете перевести эти значения в относительные единицы измерения и добавить их в свою таблицу стилей: $89 / 16 = 5,5625 \text{ em}$, $10 / 16 = 0,625 \text{ em}$. Обновите код в таблице стилей в соответствии с листингом 12.11, добавив вычисленные значения.

Листинг 12.11. Позиционирование слогана и кнопки на hero-изображении

```
.hero {
  background: url(collaboration.jpg) no-repeat;
  background-size: cover;
  margin-bottom: 2.5rem;
}
.hero__inner {
  max-width: 1080px;
  margin: 0 auto;
  padding: 5.56em 12.5em 12.5em 0;
  text-align: right;
}
.hero h2 {
  font-size: 1.95rem;
  margin-top: 0;
  margin-bottom: 0.625rem;
}
```

Замена примерных расчетов точными

Удаление верхнего поля, так как нужные расстояния заданы отступом в контейнере hero__inner

Настройка интервала между слоганом и кнопкой

Верхний отступ в контейнере hero__inner задает пространство над слоганом. Я добавил отступы к правому и нижнему краям, хотя они и не были заданы в макете. Затем сделал значение верхнего поля слогана равным нулю, чтобы расстояние точно соответствовало внутренним отступам в контейнере hero__inner. А также использовал единицы em вместо em, так как размер шрифта слогана не равняется стандартным 16 пикселям.

12.3.3. Интервалы между строчными элементами

Осталось всего несколько деталей, чтобы страница стала выглядеть как на макете. В центральной колонке перечислены операционные системы, для которых доступно приложение компании «Тандем» (рис. 12.20). Я намеренно отложил это на потом. Давайте наконец настроим стили так, чтобы они соответствовали макету.

Данный вид расположения типичен для списков тегов в блогах или категорий товаров в интернет-магазине. Я намеренно привел этот пример, так как тут есть несколько тонкостей, которые вам стоит знать.



Рис. 12.20. Список элементов, которым нужно задать стиль и расположить в строку

Существует несколько способов расставить элементы таким образом. Первые два, которые приходят в голову, — flexbox-верстка и строчные элементы. Flexbox-верстку мы уже не раз рассматривали в предыдущих главах, поэтому я хотел бы остановиться на втором способе.

Часть стилей тут очевидна. Каждому элементу надо задать свойство `display: inline`, а также отступы, цвет фона, радиус закругления границ. На первый взгляд этого достаточно. Но проблемы возникают, если ширина невелика и контент начинает переползать на новую строку. Такая ситуация представлена на рис. 12.21, это может случиться, например, при изменении размера страницы.

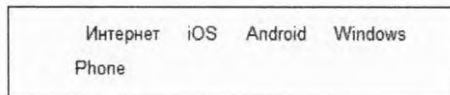


Рис. 12.21. Элементы списка переходят на следующую строку, если им не хватает ширины блока

Серый фон элементов в первой строке сливается с фоном элементов во второй строке. Причина этого — высота строки. Как говорилось ранее, высота строки текста определяется на основе размера шрифта текста и высоты строки. Если вы добавите к данным элементам отступ, это увеличит элемент, но не высоту строки текста.

Исправить эту ситуацию можно, увеличив высоту строки для каждого из элементов. Добавьте в проект следующий код (листинг 12.12), чтобы создать стили для тегов. А теперь измените высоту строки, чтобы понять, как это повлияет на внешний вид и расположение контента на странице.

Это поведение, по сути, характерно только для строчных элементов. Если элемент строчно-блочный (или flex-элемент), высота строки будет увеличиваться автоматически. Правда, вам все равно придется задать поля по вертикали и горизонтали. А строчные элементы позволяют использовать встроенные промежутки между ними, их не требуется задавать вручную.

Листинг 12.12. Форматирование тегов

```
.tag-list {
  list-style: none;
  padding-left: 0;
}
.tag-list > li {
  display: inline;
  padding: 0.3rem 0.5rem;
  font-size: 0.8rem;
  border-radius: 0.2rem;
  background-color: var(--light-gray);
  line-height: 2.6;
}
```

Замещение
браузерных стилей

Увеличение высоты строки,
чтобы добавить вертикальный интервал
и избежать наложения строк

ПРИМЕЧАНИЕ

Текст Windows Phone в строчном элементе может переноситься по словам. В случае с flex-элементами или строчными блоками это невозможно и на следующую строку переносится весь элемент. Поэтому, решая, что использовать, определите, какой вариант наиболее приемлем для вас.

Итак, дизайн страницы реализован. Сейчас она должна полностью соответствовать макету, приведенному на рис. 12.22.

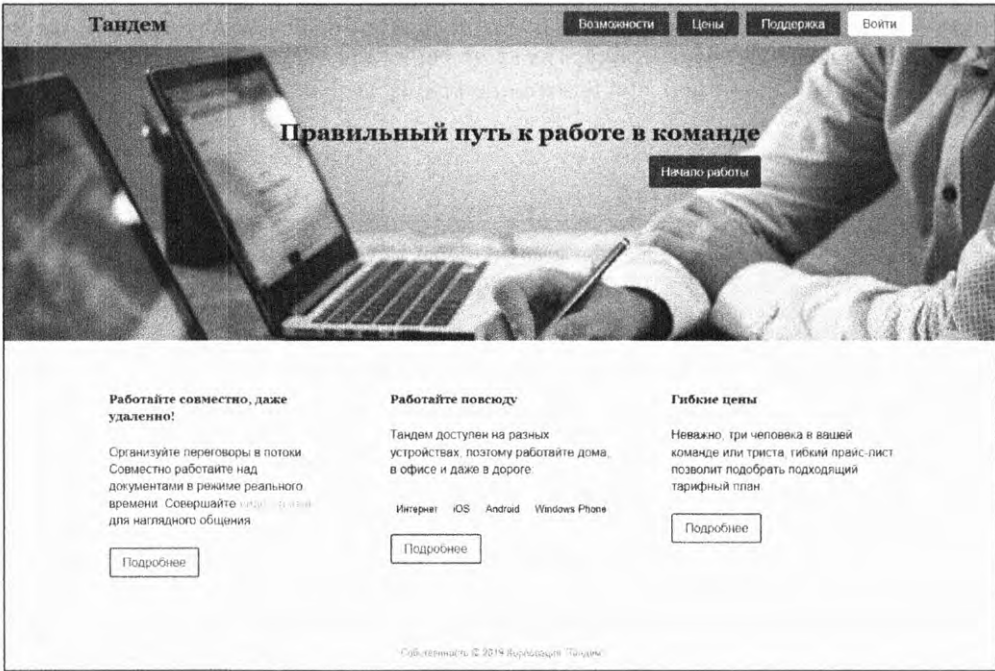


Рис. 12.22. Завершенный дизайн страницы

Мы потратили много времени на то, чтобы подробно рассмотреть каждую деталь. Зачастую веб-разработчики не так внимательны к мелочам, но те, кто не жалеет на это времени, в итоге преуспевают больше. Ведь именно в мелочах кроется разница между хорошим и превосходным сайтами.

Я призываю вас найти время на проработку деталей в CSS. Даже если вы работаете без профессионального дизайнера, доверьтесь своим глазам. Попробуйте чуть увеличить или уменьшить расстояния и посмотрите, как будет лучше. Найдите время поэкспериментировать с различными значениями. И не переборщите с цветами — просто расставьте акценты в тех местах, к которым хотите привлечь внимание. Создайте шаблон, а потом скорректируйте его там, где сосредоточена наиболее важная информация.

Итоги главы

- ❑ Применяйте контраст выборочно, чтобы привлечь внимание к определенным частям страницы.
- ❑ Используйте значения HSL, чтобы упростить работу с цветом и сделать ее более наглядной.
- ❑ Доверяйте дизайнеру в мелочах.
- ❑ Найдите время для создания хорошей разметки.
- ❑ И помните, что высота строки может повлиять на то, как элементы располагаются относительно друг друга по вертикали.

13 Шрифтовое оформление

В этой главе

- Как придать странице уникальный внешний вид, используя веб-шрифты.
- Применение Google Fonts API.
- Тонкая настройка внешнего вида шрифтов (расстояния между буквами и строками).
- Решение проблем с производительностью и оптимизация веб-шрифтов.
- Вспышки нестилизованного и невидимого текста.

С помощью шрифтов можно как придать странице уникальный вид, так и визуально испортить ее. В течение многих лет веб-разработчики были вынуждены ограничиваться узким набором так называемых *безопасных веб-шрифтов*. Это такие шрифты, как Arial, Helvetica и Georgia, установленные на большинстве пользовательских устройств. Браузеры могли отображать только их, поэтому мы вынуждены были ограничивать себя. Конечно, мы могли использовать и что-то более экзотическое, например Helvetia Neue, правда такие шрифты корректно отображались только у тех счастливицков, у которых в системе был установлен данный набор, остальные видели ничем не примечательный стандартный шрифт.

Ситуация полностью изменилась с появлением веб-шрифтов. Они задействуют специальную форму правила `@font-face`, которое сообщает браузеру, где скачать шрифт, использованный на странице. Теперь с помощью шрифтов реально оживить даже самую визуально скучную страницу. Это открывает огромные возможности. Но также предполагает, что надо принимать намного больше решений в области дизайна, чем раньше.

С помощью шрифтов вы можете заставить страницу выглядеть шуточно или серьезно, вызвать у пользователя доверие к вам и вашей информации или создать неформальный дизайн. Посмотрите на рис. 13.1: один и тот же текст представлен тремя разными парами шрифтов. В верхнем примере используются News Cycle для заголовка и EB Garamond для основного текста. Выглядит довольно официально и подойдет, вероятно, для сайта газеты. В верхнем правом применены Forum и Open Sans, что выглядит чуть менее официально. Такой стиль подойдет для личного

блога или сайта небольшой технической компании. В нижнем примере — Anton и Pangolin. Выглядит очень забавно, даже слегка мультяшно, что наиболее уместно для детского сайта. То есть, просто изменяя шрифты, можно придать странице абсолютно разный вид.

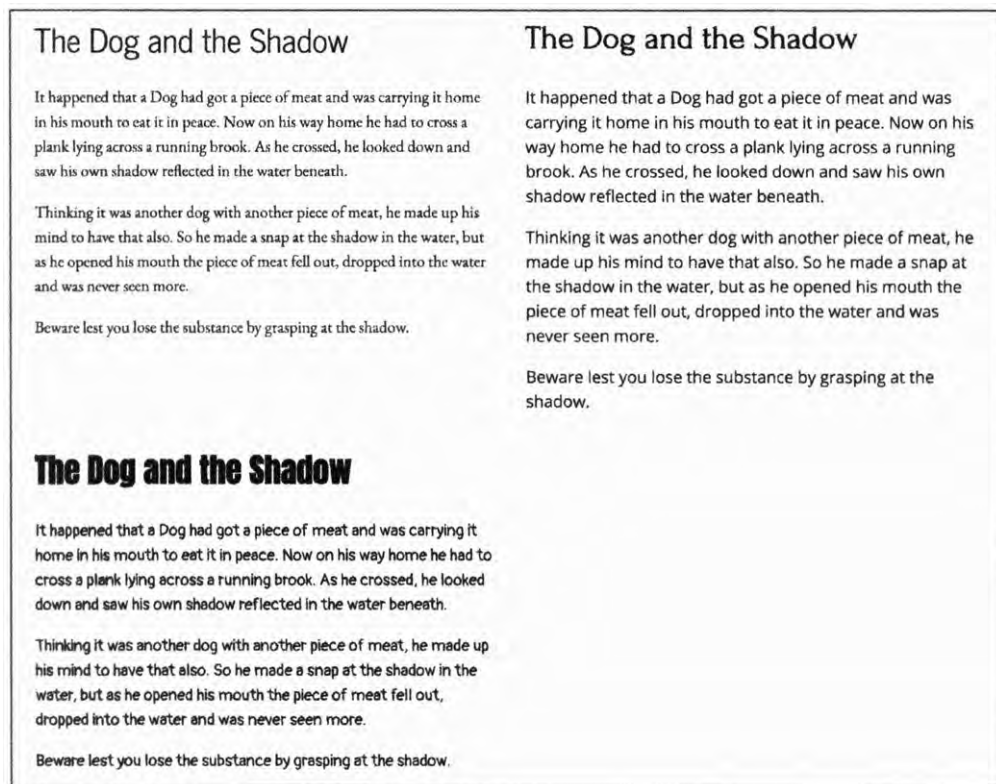


Рис. 13.1. Шрифты оказывают влияние на то, как воспринимается информация

В этой главе рассмотрим шрифты для веб-страниц. Я покажу, как они работают, а также порекомендую пару онлайн-ресурсов, предлагающих широкий выбор шрифтов. Остановимся также на некоторых свойствах CSS, которые позволяют контролировать расположение и размер шрифтов на странице. Понимание этих свойств позволит вам не только улучшить читаемость текста, но и сделать так, чтобы страница точно соответствовала макету, разработанному дизайнером.

Оформление с помощью шрифтов — такое же старое ремесло, как и книгопечатание. Это единственная тема в книге, чья история насчитывает несколько сотен лет. Поэтому я не буду заострять внимание на этом, просто покажу вещи, которые, на мой взгляд, вам необходимо знать, и расскажу, как применить их в современной веб-разработке.

13.1. Веб-шрифты

Самый простой и быстрый способ использовать веб-шрифты — посещение специализированных онлайн-сервисов. Вот самые распространенные:

- ❑ Typekit (www.typekit.com);
- ❑ Webtype (www.webtype.com);
- ❑ Google Fonts (fonts.google.com).

Неважно, платные они или нет, эти сервисы берут на себя решение многих проблем, как технических (хостинг), так и юридических (лицензирование). Каждый из них предлагает большую библиотеку шрифтов. Хотя иногда определенный, подходящий именно вам шрифт можно найти на одном-единственном ресурсе.

Поскольку сервис Google Fonts предлагает высококачественные и бесплатные (да-да, они бесплатные) шрифты, я покажу, как с его использованием добавить новый шрифт на страницу. Google решает за вас множество проблем, так что процесс крайне прост. Но после мы все же рассмотрим, как все это работает.

Возьмем страницу, которую вместе создали в предыдущей главе, и добавим несколько новых шрифтов. После этого она будет выглядеть так, как показано на рис. 13.2. В качестве шрифта для основного текста возьмем Roboto, а для заголовков — Kurale.

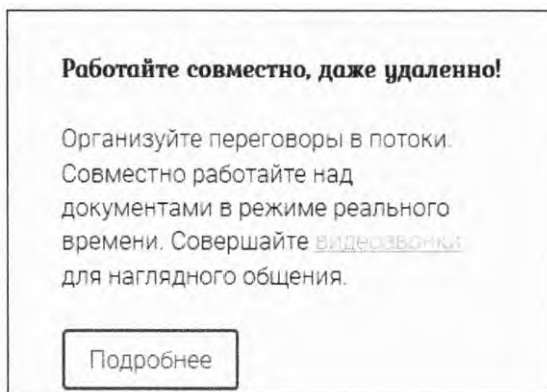


Рис. 13.2. Фрагмент страницы с применением шрифтов Roboto и Kurale

Распространенная практика — использование двух шрифтов: одного для заголовков, другого для основного текста. Часто первый — это *шрифт с засечками (антиква)*, а второй — *без засечек (рубленый, или гротеск)*, как в нашем примере. Также встречаются сайты, где заголовки и основной текст выполнены шрифтами различной насыщенности.



Засечка (англ. serif) в шрифтах — короткий, обычно перпендикулярный штрих на конце буквы, с которого начинается и которым заканчивается основной штрих знака (например, Times New Roman). Шрифты без засечек называют рублеными (например, Arial).

Если вы сделали вместе со мной то, что описано в главе 12, то на вашей странице уже есть все, за исключением веб-шрифтов. (HTML-код представлен в листинге 12.1, а CSS-код можно найти на протяжении всей главы. Скопируйте код из листингов.) Теперь добавим веб-шрифты.

13.2. Сервис Google Fonts

Чтобы увидеть все шрифты, доступные на сайте Google Fonts, перейдите по адресу fonts.google.com. На странице представлены гарнитуры шрифтов в виде блоков (рис. 13.3). Выберите подходящие гарнитуры из предложенных или поищите определенные, используя поиск по сайту — значок лупы в правом верхнем углу.

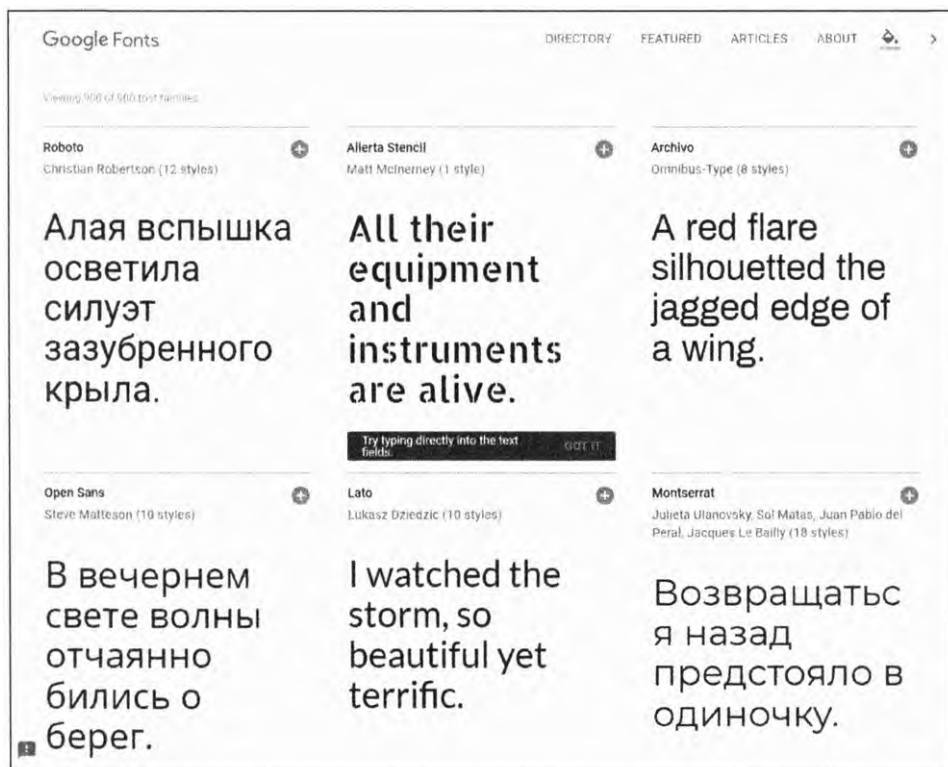


Рис. 13.3. Основной интерфейс выбора шрифтов сервиса Google Fonts

Если вам понравилась какая-то гарнитура, щелкните кнопкой мыши на красном значке +, и Google добавит ее в избранное. В нижней части сайта отобразится панель коллекций, куда попадут все выбранные гарнитуры (рис. 13.4). Красный значок – удаляет гарнитуру из избранного.

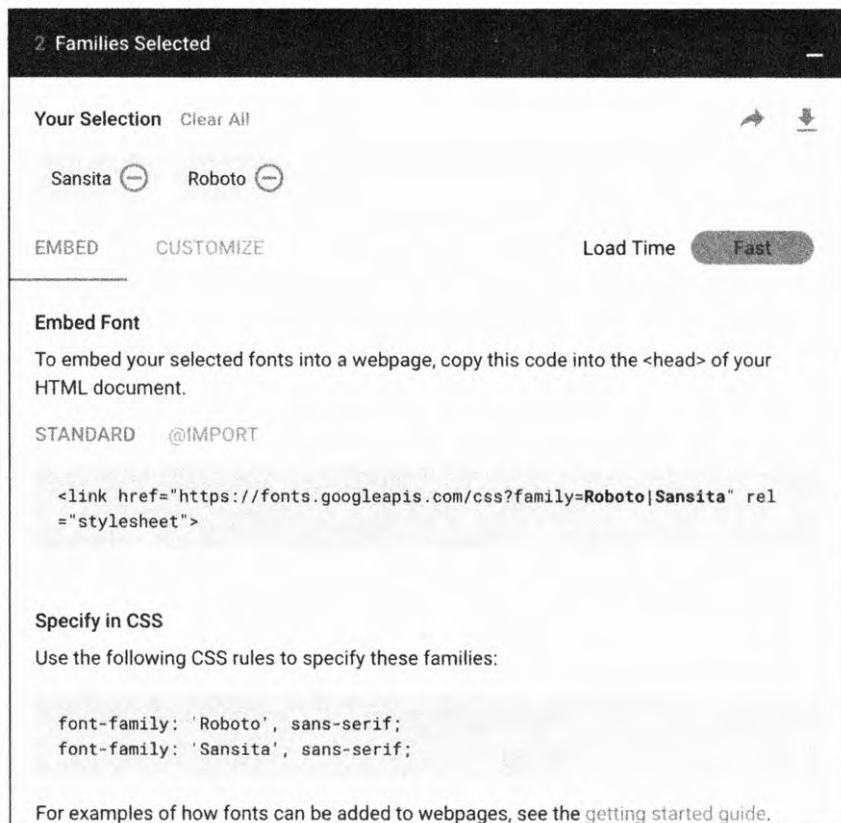


Рис. 13.4. Избранные шрифты, добавленные в коллекции, с примерами кода

Если вы знаете, какие именно шрифты вам нужны, можете искать их по названию. В строке поиска наберите **Kurale**. Так вы отфильтруете все другие гарнитуры на главной странице. Добавьте шрифт в избранное, щелкнув кнопкой мыши на красном значке +. Затем удалите слово **Kurale** из строки поиска и наберите **Roboto**. Google предложит несколько похожих шрифтов, в том числе **Roboto**, **Roboto Condensed** и **Roboto Slab**. Добавьте гарнитуру **Roboto** в избранное.

Если вы откроете панель с избранными гарнитурами, там отобразятся шрифты **Kurale** и **Roboto** с фрагментами кода, который демонстрирует, как их использовать в HTML (в коде самой страницы) и CSS (в таблице стилей). Прежде чем копировать фрагменты кода, необходимо внести некоторые поправки — изменить насыщенность шрифта. Перейдите на вкладку **Customize** (Настроить) (рис. 13.5).

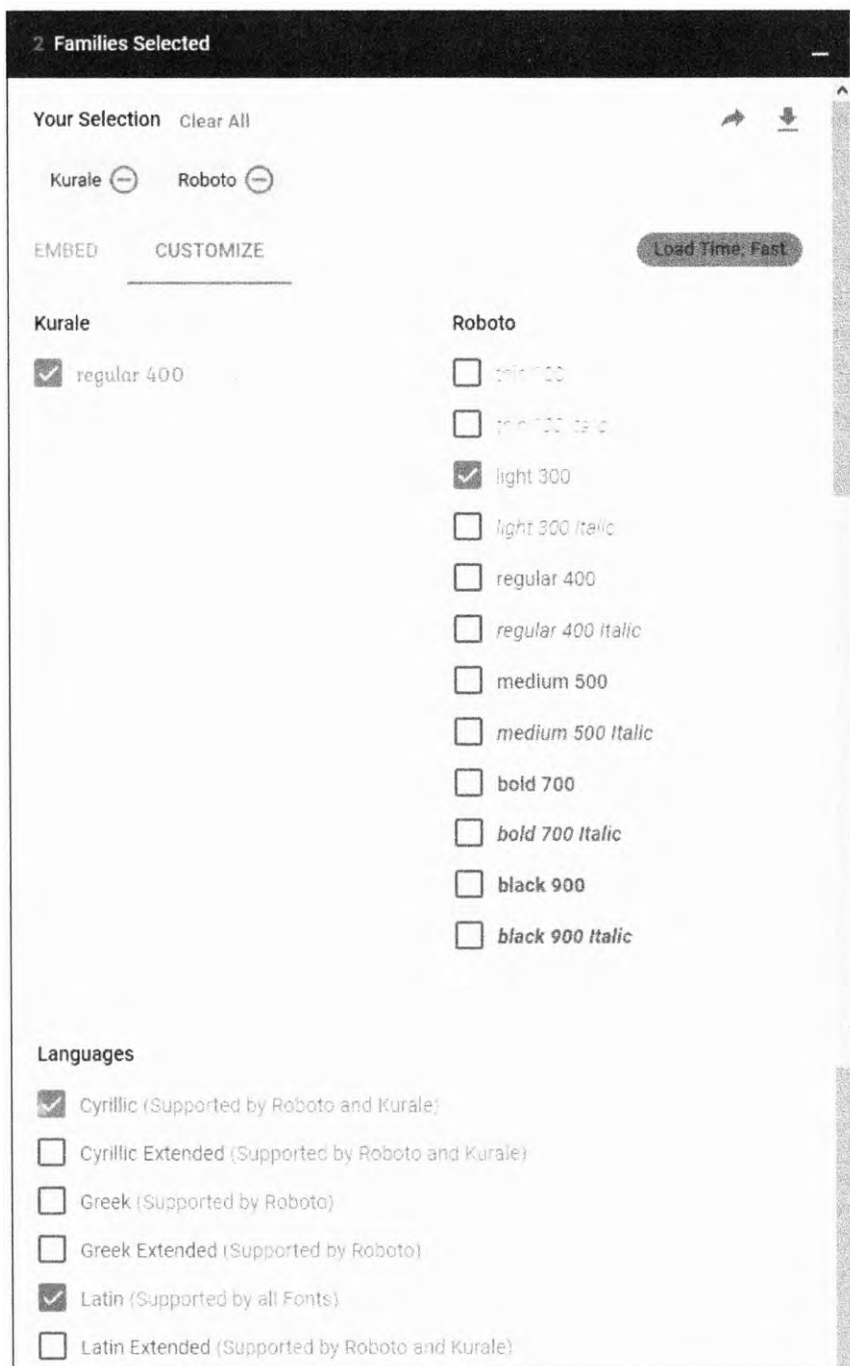


Рис. 13.5. Выберите насыщенность и начертание шрифта

Вы, вероятно, знаете, как работать с обычным и полужирным шрифтами, но для некоторых гарнитур доступно больше вариаций начертания. Например, для Roboto можно выбирать среди шести вариантов насыщенности шрифта, начиная от тонкого и заканчивая черным, существует также курсивный вариант для каждого доступного значения насыщенности. Просто отметьте то, что нужно, установив соответствующие флажки.

ПРИМЕЧАНИЕ

Понятия «гарнитура» и «шрифт» часто путают. Традиционно под гарнитурой понимается целый набор шрифтов (например, Roboto), обычно разработанный одним дизайнером. То есть гарнитура — совокупность нескольких шрифтов различных вариаций (обычный, полужирный, курсив, уплотненный и т. д.).

В идеале можно выбрать все вариации шрифта, что даст вам больше вариантов для использования на странице. Но, сделав так, вы заметите, что индикатор времени загрузки (сверху справа) изменяется с «быстро» до «средне» и «медленно». Чем больше шрифтов выбрано, чем дольше браузер будет их скачивать. А после изображений шрифты — один из наиболее ресурсоемких элементов, которые сильно замедляют загрузку страницы. Стоит подходить к этому процессу разумно и выбирать только действительно нужные начертания шрифта.

Среди начертаний Roboto выберите светлое с насыщенностью 300 (см. рис. 13.5), а среди Kurale — обычное с насыщенностью 400 (единственное доступное). С ними мы будем работать в примере. (Курсив требуется довольно часто, но все же заранее добавлять его не стоит.) Вернитесь на вкладку Embed (Внедрение) и увидите, что фрагменты кода изменились в соответствии с настройками.

Скопируйте элемент link и добавьте его в раздел заголовка страницы (элемент head), как показано в листинге 13.1. Так вы добавите стили, определяющие внешний вид шрифтов, используемых на странице. Сейчас к ней привязаны две таблицы стилей: ваша и таблица стилей шрифтов.

Листинг 13.1. Элемент link, привязывающий таблицу стилей шрифтов с сервиса Google

```
<link href="https://fonts.googleapis.com/css?family=Roboto:300|Kurale&subset=cyrillic" rel="stylesheet">
```

В этой таблице стилей описано все, что нужно для отображения данного веб-шрифта на странице. Теперь можно использовать его в собственных стилях. В результате страница будет выглядеть так, как на рис. 13.6.

Чтобы применить шрифты Roboto и Kurale, необходимо указать их в свойстве font-family. Давайте модифицируем CSS-код. Установим Roboto как основной шрифт элемента body — тела сайта, откуда он будет наследоваться остальными элементами. Затем зададим Kurale в качестве основного шрифта для заголовков и ссылки «Тандем» на главную страницу в левом верхнем углу. Измените соответствующий фрагмент кода, как показано в листинге 13.2.

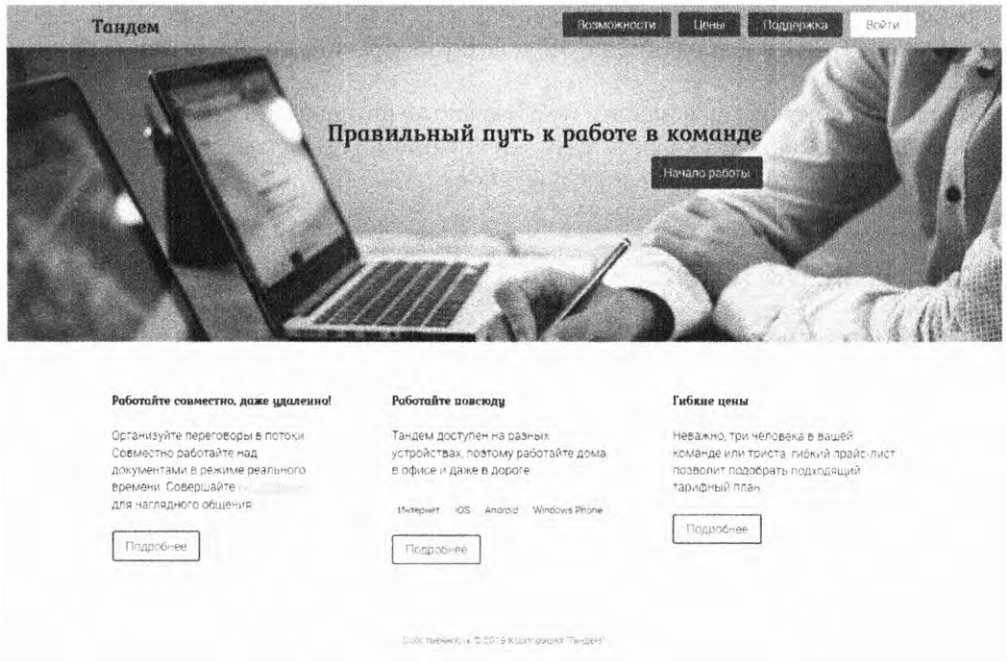


Рис. 13.6. Страница со шрифтами Roboto и Kurale

Листинг 13.2. Использование веб-шрифтов

```
body {
    margin: 0;
    font-family: Roboto, sans-serif;
    line-height: 1.4;
    background-color: var(--extra-light-gray);
}

h1, h2, h3, h4 {
    font-family: Kurale, serif;
}

/* ... */

.home-link {
    color: var(--text-color);
    font-size: 1.6rem;
    font-family: Kurale, serif;
    font-weight: bold;
    text-decoration: none;
}
```

Применение шрифта Roboto
ко всей странице

Установка шрифта Kurale
для заголовков

Установка шрифта Kurale
для ссылки на главную страницу

При использовании сервиса Google Fonts браузер получает информацию, откуда скачивать веб-шрифты, и отображает их на странице. В принципе, любой другой

сервис со шрифтами работает примерно так же. Он предоставляет или URL-адрес, который необходимо включить в CSS, или JavaScript-код, который автоматически сделает все сам.

Дальше я покажу, как изменять межбуквенные интервалы, а также поделюсь соображениями по поводу скорости загрузки страницы. Но сначала посмотрим, что же в действительности делает сервис Google Fonts.

13.3. Как работает свойство @font-face

Сервисы, предоставляющие шрифты, упрощают процесс добавления новых шрифтов на страницу, но все же стоит разобраться, как они работают. Взглянем на CSS-файл, предоставленный Google. Чтобы это сделать, откройте в браузере URL <https://fonts.googleapis.com/css?family=Roboto:300|Kurale&subset=cyrillic>. Фрагмент кода из этого файла я скопировал в листинг 13.3.

Листинг 13.3. Таблица стилей, созданная сервисом Google Fonts

```
/* cyrillic */
@font-face {
  font-family: 'Kurale';
  font-style: normal;
  font-weight: 400;
  src: local('Kurale Regular'), local('Kurale-Regular'),
       url(https://fonts.gstatic.com/s/kurale/v4/4iCs6KV9e9dXjhoKew72j00.woff2)
       format('woff2');
  unicode-range: U+0400-045F, U+0490-0491, U+04B0-04B1, U+2116;
}

/* latin */
@font-face {
  font-family: 'Kurale';
  font-style: normal;
  font-weight: 400;
  src: local('Kurale Regular'), local('Kurale-Regular'),
       url(https://fonts.gstatic.com/s/kurale/v4/4iCs6KV9e9dXjhoKfw72.woff2)
       format('woff2');
  unicode-range: U+0000-00FF, U+0131, U+0152-0153, U+02BB-02BC, U+02C6,
  U+02DA, U+02DC, U+2000-206F, U+2074, U+20AC, U+2122, U+2191, U+2193,
  U+2212, U+2215, U+FEFF, U+FFFD;
}

/* cyrillic */
@font-face {
  font-family: 'Roboto';
  font-style: normal;
  font-weight: 300;
  src: local('Roboto Light'), local('Roboto-Light'),
```

Набор правил @font-face, которые определяют использование шрифта в любом месте на странице

Объявление имени для выбранного шрифта

Определение значений начертания и насыщенности шрифта, которые применяются правилом @font-face

```

url(https://fonts.gstatic.com/s/roboto/v18/KFOlCnqEu92Fr1MmSU5fABc4EsA.woff2) ←
format('woff2');
unicode-range: U+0400-045F, U+0490-0491, U+04B0-04B1, U+2116; ←
}
/* latin */
@font-face {
    font-family: 'Roboto';
    font-style: normal;
    font-weight: 300;
    src: local('Roboto Light'), local('Roboto-Light'),
        url(https://fonts.gstatic.com/s/roboto/v18/KFOlCnqEu92Fr1MmSU5fBBc4.woff2)
        format('woff2');
    unicode-range: U+0000-00FF, U+0131, U+0152-0153, U+02BB-02BC, U+02C6,
        U+02DA, U+02DC, U+2000-206F, U+2074, U+20AC, U+2122, U+2191, U+2193,
        U+2212, U+2215, U+FEFF, U+FFFD;
}

```

Расположение файла шрифта

Набор символов Юникода, к которым применяется @font-face

Набор правил `@font-face` определяет шрифты для браузера, которые необходимо задействовать в CSS. В данном случае первое правило гласит: «Если на странице необходимо отобразить кириллические буквы, используя шрифт Kurale с насыщенностью 400, то возьмите следующий файл шрифта». Второе правило определяет применение такого же шрифта для латинских букв. Третье и четвертое регламентируют использование светлого шрифта Roboto (насыщенность 300).

Свойство `font-family` определяет название шрифта, именно его вы задействуете в таблице стилей. Свойство `src`: предоставляет список адресов файлов, разделенных запятой, среди которых браузер будет искать файл шрифта, начиная со значений `local('Kurale Regular')` и `local('Kurale-Regular')`, что означает использование локальных файлов, если данные шрифты установлены в системе пользователя. Если же нет, то файл в формате WOFF2 будет скачан с указанного URL-адреса (`url()`).

ПРИМЕЧАНИЕ

Файл, расположенный на серверах Google, имеет аналоги для других наборов символов, таких как греческий или вьетнамский алфавиты. Для них созданы отдельные файлы, и они не скачиваются без необходимости. Но принцип точно такой же, поэтому я решил не рассказывать об этом подробнее.

13.3.1. Форматы шрифтов и замена при необходимости

Сервис Google Fonts предполагает, что мой браузер поддерживает формат WOFF2 по умолчанию. Собственно, предполагает он это потому, что проверил версию браузера (Chrome) и определил, что данный формат поддерживается. Но если я открою страницу в браузере, например, Internet Explorer 10, то отобразится немного другой стиль, соответствующий файлу в WOFF-формате.

WOFF расшифровывается как Web Open Font Format (открытый формат веб-шрифтов). Это сжатый формат, разработанный специально для использования в Сети. Все современные браузеры работают с его первой версией, но не все поддерживают формат WOFF2 (усовершенствованная версия с лучшим алгоритмом сжатия и, как следствие, меньшим размером файлов). Вы вряд ли хотите утруждать себя определением версии браузера пользователя, как делает Google. Проще указать ссылки на два файла в форматах WOFF и WOFF2 (листинг 13.4). (Я укоротил ссылки, чтобы пример был нагляднее.)

Листинг 13.4. Объявление файла шрифта WOFF2 с применением альтернативного формата WOFF при необходимости

```
@font-face {
  font-family: "Roboto";
  font-style: normal;
  font-weight: 300;
  src: local("Roboto Light"), local("Roboto-Light"),
       url(https://example.com/roboto.woff2) format('woff2'),
       url(https://example.com/roboto.woff) format('woff');
}
```

Первым будет использоваться этот формат

Если браузер не поддерживает формат WOFF2, то будет задействован этот файл

На заре веб-разработки программисты вынуждены были использовать четыре, а то и пять форматов, так как разные браузеры поддерживали разные форматы. Сейчас WOFF поддерживают практически все браузеры. Но так как WOFF2 значительно быстрее, по возможности указывайте обе ссылки.

13.3.2. Варианты начертания в одной гарнитуре

Если возникает необходимость работать с несколькими шрифтами из одной гарнитуры, то для каждого необходимо отдельное правило `@font-face`. Если выбраны несколько вариантов в интерфейсе Google Fonts, то сервис предложит вам ссылку следующего вида: <https://fonts.googleapis.com/css?family=Roboto:300,700>. Откройте ее в браузере и посмотрите код. Я скопировал часть кода в листинг 13.5.

Листинг 13.5. Определение шрифтов различной насыщенности из одной гарнитуры

```
/* cyrillic */
@font-face {
  font-family: 'Roboto';
  font-style: normal;
  font-weight: 300;
  src: local('Roboto Light'), local('Roboto-Light'),
       url(https://fonts.gstatic.com/s/roboto/v18/KF0lCnqEu92Fr1MmSU5fABc4EsA.woff2)
       format('woff2');
  unicode-range: U+0400-045F, U+0490-0491, U+0480-0481, U+2116;
}

/* latin */
@font-face {
```

Обычный Roboto

```

font-family: 'Roboto';
font-style: normal;
font-weight: 300;
src: local('Roboto Light'), local('Roboto-Light'),
      url(https://fonts.gstatic.com/s/roboto/v18/KFOlCnqEu92Fr1MmSU5fBBc4.woff2)
      format('woff2');
unicode-range: U+0000-00FF, U+0131, U+0152-0153, U+02BB-02BC, U+02C6,
U+02DA, U+02DC, U+2000-206F, U+2074, U+20AC, U+2122, U+2191, U+2193,
U+2212, U+2215, U+FEFF, U+FFFD;
}

/* cyrillic */
@font-face {
  font-family: 'Roboto';
  font-style: normal;
  font-weight: 700;
  src: local('Roboto Bold'), local('Roboto-Bold'),
        url(https://fonts.gstatic.com/s/roboto/v18/KFOlCnqEu92Fr1MmWUlfABc4EsA.woff2)
        format('woff2');
  unicode-range: U+0400-045F, U+0490-0491, U+04B0-04B1, U+2116;
}

/* latin */
@font-face {
  font-family: 'Roboto';
  font-style: normal;
  font-weight: 700;
  src: local('Roboto Bold'), local('Roboto-Bold'),
        url(https://fonts.gstatic.com/s/roboto/v18/KFOlCnqEu92Fr1MmWUlfBBc4.woff2)
        format('woff2');
  unicode-range: U+0000-00FF, U+0131, U+0152-0153, U+02BB-02BC, U+02C6,
U+02DA, U+02DC, U+2000-206F, U+2074, U+20AC, U+2122, U+2191, U+2193,
U+2212, U+2215, U+FEFF, U+FFFD;
}

```

Листинг демонстрирует определение двух различных шрифтов Roboto. Если на странице необходимо отобразить обычный шрифт с насыщенностью 300, будет использовано первое правило, если полужирный — второе.

Если же в стиле страницы указано какое-то другое значение, например `font-weight: 500` или `font-style: italic`, браузер попытается усреднить два данных ему варианта, насколько это возможно. Обычно он просто выбирает правило, чье значение ближе к заданному. Но некоторые браузеры могут имитировать курсив или полужирность шрифта, чтобы достичь желаемого эффекта, приблизительно, конечно. Это происходит за счет геометрического преобразования букв и выглядит хуже, чем создание специального правила с отдельным файлом. Не рекомендую применять данный способ.

При использовании Google Fonts или аналогичного сервиса, как правило, предлагается готовый код. Хотя иногда вам может понадобиться шрифт, которого просто не существует ни на одном из сервисов. В этом случае нужно разместить на сервере собственный файл с шрифтом и определить его в правиле `@font-face`.

13.4. Управление интервалами в целях повышения читаемости

Снова обратимся к нашей странице. Теперь, когда на ней используются новые шрифты, немного доработаем дизайн. Существует два свойства, **line-height** и **letter-spacing**, которые отвечают за интервалы между строками (вертикальные) и между буквами (горизонтальные) соответственно.

Данными свойствами веб-разработчики, как правило, пренебрегают. Но если потратить немного времени и настроить их должным образом, это может сильно улучшить визуальное восприятие страницы. Более того, так можно улучшить читаемость текста на странице, а следовательно, повысить ее посещаемость.

Если интервалы слишком малы, усложняется восприятие информации объемом больше нескольких предложений или даже слов. То же самое верно, если интервалы слишком велики. На рис. 13.7 представлен текст с различными вариациями интервалов.

The Dog and the Shadow

It happened that a Dog had got a piece of meat and was carrying it home in his mouth to eat it in peace. Now on his way home he had to cross a plank lying across a running brook. As he crossed, he looked down and saw his own shadow reflected in the water beneath.

Thinking it was another dog with another piece of meat, he made up his mind to have that also. So he made a snap at the shadow in the water, but as he opened his mouth the piece of meat fell out, dropped into the water and was never seen more.

Beware lest you lose the substance by grasping at the shadow.

The Dog and the Shadow

It happened that a Dog had got a piece of meat and was carrying it home in his mouth to eat it in peace. Now on his way home he had to cross a plank lying across a running brook. As he crossed, he looked down and saw his own shadow reflected in the water beneath.

Thinking it was another dog with another piece of meat, he made up his mind to have that also. So he made a snap at the shadow in the water, but as he opened his mouth the piece of meat fell out, dropped into the water and was never seen more.

Beware lest you lose the substance by grasping at the shadow.

The Dog and the Shadow

It happened that a Dog had got a piece of meat and was carrying it home in his mouth to eat it in peace. Now on his way home he had to cross a plank lying across a running brook. As he crossed, he looked down and saw his own shadow reflected in the water beneath.

Thinking it was another dog with another piece of meat, he made up his mind to have that also. So he made a snap at the shadow in the water, but as he opened his mouth the piece of meat fell out, dropped into the water and was never seen more.

Beware lest you lose the substance by grasping at the shadow.

Рис. 13.7. Интервалы в тексте оказывают сильное влияние на его читаемость

Если вы попытаетесь прочесть сильно сжатый текст в левом верхнем углу, то обнаружите, что надо напрягаться. Из-за этого можно пропустить некоторые строки, а другие прочитать дважды. Вскоре вам захочется бросить такое чтение. К тому же страница выглядит перегруженной. Текст внизу слева чрезмерно растянут. Отдельные буквы привлекают слишком много внимания, и потребуются дополнительные усилия, чтобы сложить их в слова. Текст вверху справа выглядит наиболее комфортным для чтения.

13.4.1. Интервалы основного текста сайта

Найти верные значения свойств `line-height` и `letter-spacing` не всегда просто, так как плотность текста — субъективное ощущение. Лучше всего попробовать несколько вариантов: найти слишком сжатый и разреженный и выбрать промежуточное значение. К счастью, все же есть несколько правил, которые помогут сделать правильный выбор.

Начальное значение для `line-height` задается ключевым словом `normal`, что составляет примерно 1,2 em (оно же используется в файлах шрифтов по умолчанию). В большинстве случаев этого недостаточно. Обычно идеальное значение для основного текста сайта находится в интервале от 1,4 до 1,6 em.

На нашей странице высота строки уже установлена как 1,4 em, это сделано в предыдущей главе. Вся страница наследует это свойство от элемента `body`. Но посмотрите, как бы выглядела страница без этого свойства (рис. 13.8). На рисунке слева используются значения `line-height` и `letter-spacing` по умолчанию, а справа — пользовательские. (Позже вы примените на странице второй вариант.)

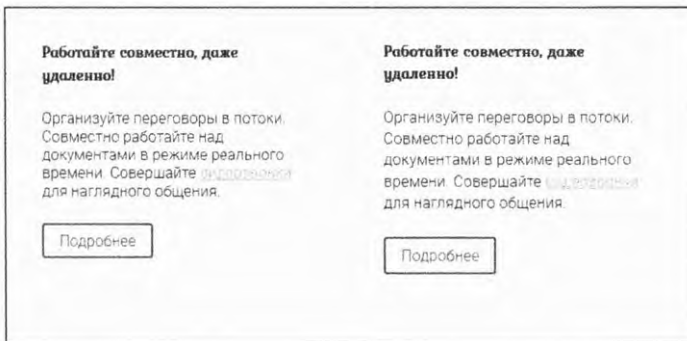


Рис. 13.8. Фрагмент страницы компании «Тандем»: *слева* — с интервалами по умолчанию; *справа* — с заданными вручную

Измените значение `line-height` на 1,3 или 1,5 em. Может, так вам понравится больше, чем предложенный мной вариант 1,4 em.

СОВЕТ

Для длинных строк лучше использовать бóльшую высоту строки. Так глазу проще перемещаться со строки на строку и при этом не теряться в пространстве. В идеале стоит делать строки длиной 45–75 символов, они считаются оптимальными для восприятия.

Теперь рассмотрим свойство `letter-spacing`. Оно принимает в качестве аргумента расстояние, которое необходимо добавить между символами. Здесь даже 1 пиксел существенен, поэтому лучше брать меньшие значения. Подбирая межбуквенный интервал, я обычно применяю сотые доли единицы `em` (например, `letter-spacing: 0.01em`). Добавление этого расстояния с помощью CSS показано в листинге 13.6.

Листинг 13.6. Установка интервала между буквами для основного текста сайта

```
body {
  margin: 0;
  font-family: Roboto, sans-serif;
  line-height: 1.4;
  letter-spacing: 0.01em;
  background-color: var(--extra-light-gray);
}
```

Значения высоты строки и межбуквенного интервала наследуются всеми элементами на странице

Добавление интервала 0,01 em между буквами

Попробуйте увеличить значение до 0,02 или 0,03 `em` и посмотрите, как это будет выглядеть. Конечно, ваш глаз не так наметан, как у дизайнера, но в этом нет ничего страшного. Доверьтесь интуиции. Если сомневаетесь, просто попытайтесь не переусердствовать. Смысл не в том, чтобы привлечь внимание к этим интервалам, а как раз наоборот. Для страницы компании «Тандем» я нахожу наиболее приемлемыми значения 0,01 или 0,02 `em` и предлагаю использовать привычное 0,01 `em`.

Преобразование значений интерлиньяжа и трекинга в CSS-код

В мире дизайна межстрочный интервал называется *интерлиньяжем*, а интервал между буквами — *трекингом*. Если вы работаете с дизайнером, он может обозначить желаемые интервалы, но эти значения могут не соотноситься с используемыми в свойствах `line-height` и `letter-spacing` языка CSS.

Интерлиньяж обычно задается в пунктах. Например, в интерлиньяж 18 пунктов входят высота текста и пространство между ним и следующей строкой. Применяется точно так же, как свойство `line-height` в CSS, только берутся безразмерные единицы. Поэтому сначала нужно перевести это значение в пиксели, а потом в безразмерные единицы.

Чтобы перевести пункты в пиксели, надо умножить значение в пунктах на 1,333 (на 1 дюйм приходится примерно 96 пикселей, или 72 пункта, а $96 / 72 = 1,333$): 18 пунктов $\cdot 1,333 = 24$ пиксела. Это число следует разделить на размер шрифта, чтобы получить значение в безразмерных единицах: 24 пиксела / 16 пикселей = 1,5.

Трекинг задается обычным числом, например 100. Оно представляет собой количество тысячных долей одной единицы `em`, поэтому достаточно просто разделить его на 1000: $100 / 1000 = 0,1 \text{ em}$.

13.4.2. Заголовки, мелкие элементы и интервалы

Интервалы для заголовков не всегда точно такие же, как для основного текста сайта. Задав интервалы для основного текста, проверьте, не нужно ли дополнительно настроить заголовки.

Заголовки обычно короткие, всего несколько слов. Но иногда попадаются и длинные. Распространенная ошибка — тестировать дизайн только для коротких. Сейчас, когда высота строки настроена, временно добавьте текст в некоторые заголовки, чтобы посмотреть, как они будут выглядеть при переносе строк (рис. 13.9).

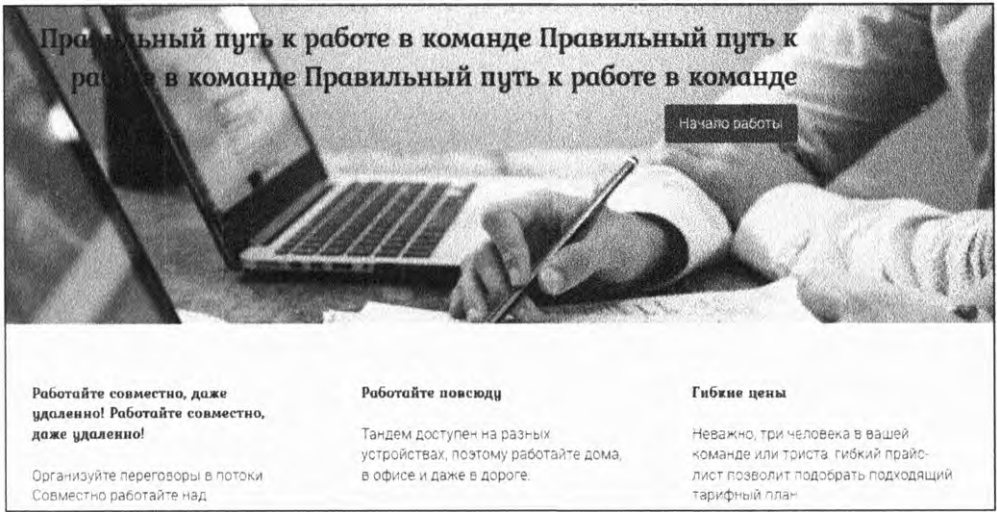


Рис. 13.9. Убедитесь, что высота строки подходит для многострочных заголовков

В данном случае вертикальные интервалы кажутся нормальными, я не буду их трогать. Хотя всегда стоит проверять. Иногда значение 1,4 может казаться слишком большим, особенно для крупных шрифтов. Это зависит от гарнитуры. В некоторых проектах мне приходилось сильно уменьшать интерлиньяж — вплоть до 1,0 em.

А что касается межбуквенного интервала, его можно немного увеличить. Добавьте код из листинга 13.7 в свою таблицу стилей. Изменение будет едва различимым.

Листинг 13.7. Увеличение межбуквенного интервала в заголовках

```
h1, h2, h3, h4 {
  font-family: Kurale, serif;
  letter-spacing: 0.03em;
}
```

```
/* ... */
```

```
.home-link {
  color: var(--text-color);
  font-size: 1.6rem;
  font-family: Kurale, serif;
  font-weight: bold;
  letter-spacing: 0.03em;
  text-decoration: none;
}
```

Увеличение интервала
между буквами
для заголовков

В основном тексте сайта максимальный упор сделан на улучшение читаемости. Но данный аспект чуть менее важен для заголовков и других элементов, содержащих не так много информации, например кнопок. Поэтому тут можно дать волю своей творческой натуре. Допустим, намеренно перестараться с интервалами. Или, наоборот, использовать отрицательное значение, чтобы наложить буквы друг на друга. На рис. 13.10 представлен пример заголовка со свойством `letter-spacing: -0.02em`.

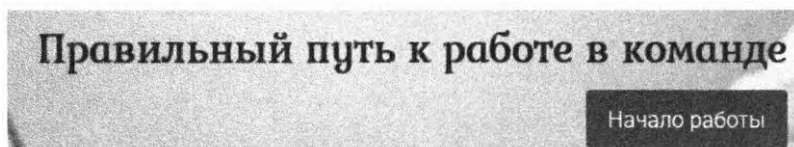


Рис. 13.10. Сжатый интервал между буквами в декоративном элементе

Это значение не подходит. Несколько строк такого текста читать будет невозможно, но небольшие, буквально в несколько слов, фразы читаются легко. Применим тот же стиль и для слогана. Внесите в свою таблицу стилей код из листинга 13.8.

Листинг 13.8. Сжатие интервала между буквами в слогане

```
.hero h2 {
  font-size: 1.95rem;
  letter-spacing: -0.02em;
  margin-top: 0;
  margin-bottom: 0.625rem;
}
```

Отрицательное значение
для уменьшения интервала

То же самое можно сделать и для мелких элементов, таких как кнопки. Лично мне кажется, что они выглядят достаточно крупными, особенно на панели навигации. Попробуем немного поработать с их стилями. На рис. 13.11 показано, как эти кнопки выглядят сейчас (*вверху*) и после того, как я изменил стили (*внизу*).

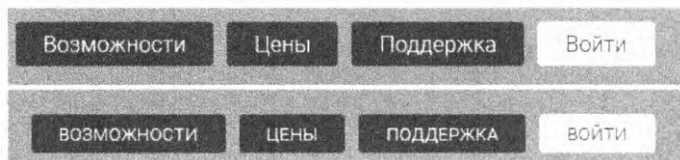


Рис. 13.11. Изменение свойств текста может улучшить внешний вид кнопок на панели навигации

Я слегка уменьшил размер шрифта, изменил регистр текста, применив свойство `text-transform`, и увеличил интервал между буквами.

СОВЕТ

Текст в верхнем регистре лучше смотрится с увеличенным интервалом между буквами.

Добавьте в свою таблицу стилей код из листинга 13.9. Тут определен более мелкий шрифт и для других кнопок на странице, благодаря чему они немного уменьшатся. Хотя в этом листинге представлено только изменение регистра и интервала между буквами.

Листинг 13.9. Настройка размера и интервала для элементов панели навигации

```
.nav-container__inner {
  display: flex;
  justify-content: space-between;
  align-items: flex-end;
  max-width: 1080px;
  margin: 0 auto;
  padding: 0.625em 0;
}

/* ... */

.top-nav a {
  display: block;
  font-size: 0.8rem;
  padding: 0.3rem 1.25rem;
  color: var(--white);
  background: var(--brand-green);
  text-decoration: none;
  border-radius: 3px;
  text-transform: uppercase;
  letter-spacing: 0.03em;
}

...

.button {
  display: inline-block;
  padding: 0.4em 1em;
  color: hsl(162, 87%, 21%);
  border: 2px solid hsl(162, 87%, 21%);
  border-radius: 0.2em;
  text-decoration: none;
  font-size: 0.8rem;
}
```

Выравнивание элементов
контейнера nav по нижнему краю

Уменьшение размера шрифта
ссылок на панели навигации

Изменение единиц измерения
отступов с em на rem

Изменение регистра букв
и увеличение интервала между ними

Уменьшение размера
шрифта на кнопках

Поскольку вы уменьшили размер навигационных ссылок на панели навигации, их высота теперь меньше, чем высота родительского блока `nav-container`. По умолчанию они выровнены по верхнему краю, следовательно, под ними остается внушительное пустое пространство. Выравнивание `flex`-элементов блока `nav-container` по нижнему краю (`flex-end`) решает эту проблему.

А так как изменился размер шрифта элементов панели навигации, то и отступы, ранее определенные в `em`, тоже изменились. Чтобы этого избежать, я заменил

единицы измерения на `rem`. Мы, конечно, могли бы вычислить новое значение для отступов, но в этом нет никакой необходимости.

Со свойством `text-transform` вы прежде не сталкивались. Оно изменяет регистр всего текста на верхний (прописные буквы) независимо от того, как он представлен в HTML-коде. Настоятельно рекомендую пользоваться именно данным способом, а не изменять HTML-код. Это позволит изменить всего одну строку в CSS, а не искать и править их огромное количество в HTML в случае изменения проекта. Хотя если что-то следует написать в верхнем регистре по правилам грамматики (например, аббревиатуру), то лучше сделать это именно в HTML. CSS используется только в целях дизайна.

Еще одно значение, которое принимает `text-transform`, — это `lowercase`, переводящее текст в нижний регистр. Можно также взять значение `capitalize`, оно задает перевод в верхний регистр первой буквы каждого слова, а прочие оставляет соответствующими HTML-коду.

Последнее усилие: вертикальный ритм

В главе 12 я описывал, как важно создавать и использовать согласованные шаблоны, включая интервалы между элементами, при разработке дизайна страницы. Поддержание *вертикального ритма* подразумевает соблюдение этого же правила для строк текста на странице. Он достигается построением *сетки базовых линий*, определяющей расстояния между строками текста. Большая часть текста или он весь должен ложиться на сетку.

Сетка на рисунке далее представляет собой горизонтальные линии, расположенные на одинаковом расстоянии друг от друга. Обратите внимание на то, как на нее ложатся заголовок и метка на кнопке.

Элементы с различным размером шрифта и разными полями создают вертикальную гармонию — накладываются на сетку базовых линий.

Применение данного принципа — весьма трудоемкий процесс, но логичная структура, которую он создает, стоит того. Если вы внимательны к деталям и хотели бы подробнее изучить эту технику, рекомендую вам статью zellwk.com/blog/why-vertical-rhythms/.

Однако предупреждаю: соблюдение вертикального ритма обычно требует использования единиц измерения в `line-height`. Это изменяет принципы наследования (об этом говорилось в главе 2), так что придется явно задать высоту строки везде, где шрифт другого размера.

Работайте совместно, даже удаленно!

Организуйте переговоры в потоки.
Совместно работайте над документами в режиме реального времени. Совершайте для наглядного общения.

Подробнее

13.5. Вспышки нестилизованного и невидимого текста

Прежде чем закончить со шрифтами, необходимо поговорить о скорости загрузки. Файлы шрифтов довольно большие. Я уже говорил, что по возможности надо свести количество используемых шрифтов к минимуму. Но даже тогда могут возникнуть проблемы. У браузера бывает такой момент, когда разметка и контент уже визуализировались, а шрифты — еще нет. Важно понимать, что происходит в этот момент.

Большинство разработчиков браузеров решили, что важнее отобразить страницу так быстро, как только возможно, используя при этом системные шрифты. Затем, когда шрифт наконец подгружается, страница обновляется. Процесс показан на рис. 13.12.



Рис. 13.12. Вспышка нестилизованного текста

Скорее всего, веб-шрифты будут занимать другое количество пространства на экране (больше или меньше), чем системные. Когда происходит второе обновление страницы, ее разметка слегка изменяется и текст смещается. Если задержка между первым и вторым обновлениями мала, то пользователь может ничего и не заметить. Но когда задержка загрузки продолжительная или файлы шрифтов слишком большие, процесс может занять несколько секунд. Некоторых пользователей это раздражает. Он уже мог начать читать, как текст внезапно сместился, из-за чего он потерял то место, где читал. Это явление называют *FOUT* (flash of unstyled text) — *вспышка нестилизованного текста*.

Веб-разработчики не особо восторгались таким поведением браузеров, поэтому производителям пришлось его изменить. Вместо отображения замененного шрифта на странице отображается все, кроме текста. А точнее, текст просто невидим, но все же занимает определенное пространство. Таким образом, блоки располагаются

на своих местах, а пользователь видит страницу в процессе загрузки. Это привело к возникновению эффекта *FOIT* (flash of invisible text) — *вспышки невидимого текста* (рис. 13.13). Видны фоновые цвета и границы, но текст появляется только при втором обновлении, когда веб-шрифты полностью загружены.



Рис. 13.13. Вспышка невидимого текста

Данный подход решает одну проблему, но создает новую: что произойдет, если загрузка шрифтов займет очень много времени? Или они вообще не загрузятся? В этом случае страница останется пустой, с абсолютно бесполезными для пользователя цветными блоками. Так что вы явно предпочли бы вспышку нестилизованного текста.

Веб-разработчики предложили ряд решений данной проблемы. Такое ощущение, что раз в год, а то и чаще появляется новый способ. Суть их всех проста — ни вспышки нестилизованного текста, ни вспышки невидимого текста не удовлетворяют разработчиков. Но в мире веб-шрифтов без них никак не обойтись. Остается лишь минимизировать проблему, насколько это возможно.

К счастью, шум вокруг этой проблемы постепенно утих, что избавило меня от необходимости рассказывать о десятках разных техник. Я просто изложу подход, который считаю наиболее разумным. Он заключается в использовании небольшого JavaScript-скрипта, контролирующего загрузку шрифтов. А также покажу, как это сделать средствами CSS в обход JavaScript. Вы можете применить любую из техник на свое усмотрение или обе сразу.

13.5.1. Библиотека Font Face Observer

Задействуя JavaScript, вы можете контролировать загрузку шрифтов. Это позволит лучше понять разницу между вспышками нестилизованного и невидимого текста. А еще используйте специально разработанную библиотеку, которая сделает всю

работу самостоятельно. Мне очень нравится Font Face Observer (fontfaceobserver.com). Она позволяет дождаться загрузки шрифтов, а затем действует согласно обстановке. Я добавляю класс `fonts-loaded` в элемент `html` с помощью JavaScript, как только загрузятся шрифты. Библиотеку можно применять, чтобы просто изменить стиль страницы, независимо от веб-шрифтов.

Скачайте файл `fontfaceobserver.js` и сохраните его в ту же папку, где находится страница. Добавьте следующий код (листинг 13.10) в нижнюю часть страницы непосредственно перед закрывающим тегом `</body>`.

Листинг 13.10. Использование Font Face Observer для контроля загрузки шрифтов

```
<script type="text/javascript">
  var html = document.documentElement;
  var script = document.createElement("script");
  script.src = "fontfaceobserver.js";
  script.async = true;

  script.onload = function () {
    var roboto = new FontFaceObserver("Roboto");
    var kurale = new FontFaceObserver("Kurale");
    var timeout = 2000;

    Promise.all([
      roboto.load(null, timeout),
      kurale.load(null, timeout)
    ]).then(function () {
      html.classList.add("fonts-loaded");
    }).catch(function (e) {
      html.classList.add("fonts-failed");
    });
  };
  document.head.appendChild(script);
</script>
```

Динамическое создание элемента `script` для добавления библиотеки Font Face Observer на страницу

Создание наблюдателей для шрифтов Roboto и Kurale

Когда шрифты загружены, создается класс `fonts-loaded` внутри элемента `html`

Если шрифты не загрузились, создается класс `fonts-failed`

Приведенный скрипт создает два наблюдателя, по одному на каждый используемый шрифт. Метод `Promise.all()` ожидает загрузки обоих, а затем добавляет класс `fonts-loaded`. Если при загрузке возникает ошибка или время ожидания истекает (через 2 с), вызывается метод `catch`, который создает класс `fonts-failed`. То есть при загрузке страницы скрипт добавляет один из классов, `fonts-loaded` или `fonts-failed`.

ПРИМЕЧАНИЕ

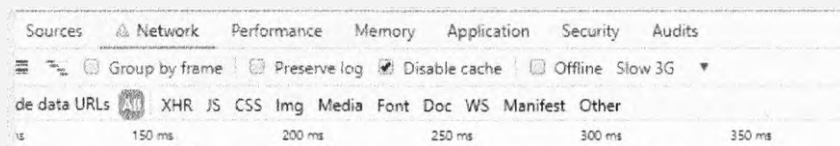
И этот код, и библиотека Font Face Observer используют механизм JavaScript, называемый промисами, который не поддерживается в Internet Explorer. К счастью, библиотека Font Face Observer включает в себя полизаполнение, что позволяет имитировать некоторые функции в устаревших браузерах. Так что если вы готовы написать полизаполнение, то скачайте автономную версию Font Face Observer с сайта.

Далее я покажу, как использовать классы `fonts-loaded` и `fonts-failed` для контроля поведения шрифтов.

Искусственное замедление Сети для тестирования загрузки шрифтов

Если вы используете быстрое сетевое соединение, то сложно протестировать, как страница ведет себя при загрузке. Одно из решений проблемы — намеренно замедлить скорость скачивания файлов с помощью инструментов разработчика.

В Chrome на вкладке **Network** (Сеть) на верхней панели имеется раскрывающийся список, который позволяет задать скорость соединения. Вы можете искусственно замедлить ее, выбрав значение **Slow 3G** (Медленное 3G), как показано на рисунке.



Предлагаю также установить флажок **Disable Cache** (Отключить кэширование). Теперь каждый раз, когда вы обновляете страницу, все файлы скачиваются заново. Это позволит более точно симитировать загрузку страницы пользователем, у которого медленное соединение и который впервые заходит на сайт.

Данные настройки применяются только при открытой панели инструментов разработчика. Но все равно убедитесь, что вернули все настройки в нормальное состояние, чтобы в следующий раз, когда откроете инструменты разработчика, избежать проблем.

13.5.2. Откат к системным шрифтам

Существует два подхода к загрузке шрифтов. Первый заключается в использовании альтернативных шрифтов с помощью CSS: в селекторе `.fonts-loaded` следует определить значение желаемого веб-шрифта. Тем самым вы замените в браузере вспышку невидимого текста вспышкой нестилизованного текста. Второй способ — применение селектора `.fonts-failed`. Это очень похоже на вспышку невидимого текста, с той разницей, что задан период ожидания, по истечении которого браузер загрузит системные шрифты. То есть в случае ошибки при загрузке страница не останется пустой.

Из этих вариантов я чаще предпочитаю второй. Но это субъективное мнение, так как выбор всегда зависит от ситуации и требований конкретного проекта. Даже длительность ожидания — дело вкуса.

Попробуем внедрить второй вариант. Следующий код (листинг 13.11) определяет использование альтернативных шрифтов и класса `.fonts-failed`. Добавьте его в свой CSS-код.

Листинг 13.11. Определение альтернативных шрифтов с нейтрализацией вспышки невидимого текста

```
body {  
  margin: 0;  
  font-family: Roboto, sans-serif;  
  line-height: 1.4;  
  letter-spacing: 0.01em;  
  background-color: var(--extra-light-gray);  
}  
.fonts-failed body {  
  font-family: Helvetica, Arial, sans-serif;  
}  
  
h1, h2, h3, h4 {  
  font-family: Kurale, serif;  
  letter-spacing: 0.03em;  
}  
.fonts-failed h1,  
.fonts-failed h2,  
.fonts-failed h3,  
.fonts-failed h4 {  
  font-family: Georgia, serif;  
}  
...  
.home-link {  
  color: var(--text-color);  
  font-size: 1.6rem;  
  font-family: Kurale, serif;  
  font-weight: bold;  
  letter-spacing: 0.03em;  
  text-decoration: none;  
}  
.fonts-failed .home-link {  
  font-family: Georgia, serif;  
}
```

Если веб-шрифты
не загрузятся,
то будут использованы
системные

Когда происходит ошибка загрузки или истекает период ожидания, на страницу добавляется класс `fonts-failed`, который определяет, что шрифты должны быть заменены альтернативными. То есть при быстром соединении веб-шрифты на страницу загрузятся почти без задержки, при медленном — через пару секунд (после истечения периода ожидания) загрузятся системные.

СОВЕТ

Мы потратили некоторое время, настраивая интервалы между буквами и строками в оформленном с помощью стилей тексте. Вы, возможно, захотите проделать то же самое для системных шрифтов, так как, вероятнее всего, эти значения для них будут иными. Добавьте размеры интервалов в набор правил класса `.fonts-failed`, так они будут применяться, только если не загрузятся веб-шрифты. Идеальный вариант — настроить интервалы для шрифтов замены так, чтобы разница между ними и основными была минимальной. Тогда эффект от вспышек нестилизованного текста будет менее заметным. В этом вам поможет такой инструмент: meowni.ca/font-style-matcher/.

Нет правильного ответа на вопрос, как стоит организовывать загрузку шрифтов. Аналитическая информация о скорости загрузки вашей страницы может помочь с выбором правильного подхода. Вспышки невидимого текста лучше использовать при быстром соединении, а вспышки нестилизованного текста — при медленном. Но только вы решаете, какой из способов выбрать.

13.5.3. И наконец, свойство `font-display`

Новое свойство в CSS `font-display` позволяет контролировать загрузку шрифтов без помощи JavaScript. На момент написания книги оно было доступно только в браузерах Firefox, Chrome, Safari и Opera. Я кратко опишу, как работает данное свойство, возможно, это пригодится вам в будущих проектах.

Свойство описывается в правиле `@font-face`. Оно определяет поведение браузера при загрузке шрифтов. Пример приведен в листинге 13.12.

Листинг 13.12. Пример свойства `font-display`

```
@font-face {  
  font-family: "Roboto";  
  font-style: normal;  
  font-weight: 300;  
  src: local("Roboto Light"), local("Roboto-Light"),  
       url(https://example.com/roboto.woff2) format('woff2'),  
       url(https://example.com/roboto.woff) format('woff');  
  font-display: swap;  
}
```

Использование значения `swap`, соответствующего эффекту вспышки не оформленного с помощью стилей текста

В данном случае браузер должен сразу отобразить системные шрифты, а веб-шрифты — как только они будут загружены. Одним словом, это вспышка нестилизованного текста.

Это свойство поддерживает следующие значения:

- ❑ **auto** — поведение браузера по умолчанию (для большинства браузеров — вспышка невидимого текста);
- ❑ **swap** — отображает системные шрифты, а веб-шрифты — через некоторое время, как только они будут загружены (вспышка нестилизованного текста);
- ❑ **fallback** — компромисс между **auto** и **swap**. Короткое время (примерно 100 мс) текст будет невидим. Если веб-шрифты к этому времени еще не загрузятся, браузер отобразит системные шрифты. Но как только веб-шрифты будут готовы, они появятся на странице;
- ❑ **optional** — очень похоже на **fallback**, но позволяет браузеру самостоятельно определять, какие шрифты использовать, в зависимости от скорости соединения. Обычно это означает, что при медленном соединении даже не будет попытки загрузить веб-шрифты.

Данные опции обеспечивают немного больше возможностей, нежели JavaScript. Для быстрых соединений лучший вариант — `fallback`, который представляет собой быструю вспышку невидимого текста, но способен вызвать вспышку нестилизованного текста, если веб-шрифты не загрузятся в течение 100 мс. Для медленных соединений больше подходит `swap`, так как сразу загружаются системные шрифты. А в проектах, где шрифт не самая важная часть дизайна, используйте `optional`.

Управлять загрузкой веб-шрифтов — непростая задача. Чтобы получить более детальную информацию, советую прочитать книгу Дж. Л. Вагнера (Jeremy L. Wagner) *Web Performance in Action*. Одна из ее глав посвящена скорости загрузки и веб-шрифтам, а несколько важных глав рассказывают о других аспектах производительности в CSS.

Итоги главы

- ❑ Используйте сервисы, такие как Google Fonts, чтобы упростить работу с веб-шрифтами.
- ❑ Строго ограничьте количество применяемых шрифтов и контролируйте размер страницы.
- ❑ Задействуйте `@font-face` для добавления собственных шрифтов.
- ❑ Найдите время на настройку свойств `line-height` и `letter-spacing`.
- ❑ Для управления загрузкой шрифтов используйте библиотеку Font Face Observer или аналогичный JavaScript-сценарий.
- ❑ Следите за внедрением поддержки свойства `font-display`.

14

Переходы

В этой главе

- Создание анимации с помощью переходов.
- Изучение функций времени и выбор подходящих.
- Координация с JavaScript.

В традиционных печатных изданиях страницы статичны. Текст не перемещается по бумаге, цвета не способны меняться. Но Сеть — живая среда, в которой можно сделать гораздо больше. Элементы могут исчезать, меню — сворачиваться. Один цвет — сменяться другим. И самый простой способ добиться этого — использовать *переходы*.

С помощью CSS-переходов вы можете скомандовать браузеру плавно заменить какой-либо параметр другим. Например, если применены синие ссылки, которые становятся красными при наведении указателя мыши, переход заставит их из голубых превращаться в фиолетовые, а затем в красные, когда пользователь наводит на них указатель, и возвращаться к первоначальному цвету при его переводе в другую область. При правильном использовании переходы могут увеличить ощущение интерактивности страницы, а поскольку наши глаза реагируют на движение, переходы способны привлечь внимание пользователя к происходящим изменениям.

Зачастую переходы можно добавить на страницу, выполнив некоторые условия. Из данной главы мы узнаем, как это сделать, а также поговорим о том, какие решения придется принять в процессе. Так как существуют некоторые варианты применения переходов, в которых могут возникнуть сложности, мы также рассмотрим решение этих проблем.

14.1. Отсюда сюда

Переходы задаются с помощью семейства свойств `transition-*`. Если переход применен к элементу, то, когда одно из значений его свойств изменяется, это изменение происходит плавно.

Выполним простой пример с использованием кнопки и затем исследуем, как это работает. При наведении указателя мыши кнопка зеленовато-голубого цвета

с прямыми углами окрашивается в красный цвет, а углы скругляются. Два этих состояния, а также промежуточное состояние во время перехода показаны на рис. 14.1.



Рис. 14.1. Элемент до, в процессе и после перехода

Добавьте кнопку на новую страницу и свяжите ее с таблицей стилей. Разметка приведена далее (листинг 14.1).

Листинг 14.1. Добавление простой кнопки на страницу

```
<button>Наведи на меня указатель мыши</button>
```

Затем включите стили в таблицу стилей. Они определяют как нормальное состояние, так и состояние при наведении указателя мыши. Два свойства перехода инструктируют браузер, что нужно мягко переходить от одного значения к другому (листинг 14.2).

Листинг 14.2. Стили кнопок с переходами

```
button {
  background-color: hsl(180, 50%, 50%); ← Зеленовато-голубая кнопка
  border: 0;
  color: white;
  font-size: 1rem;
  padding: .3em 1em;
  transition-property: all; ← Установка перехода
                             на все изменения свойств
  transition-duration: 0.5s; ← Установка продолжительности
                             перехода 0,5 с
}
button:hover {
  background-color: hsl(0, 50%, 50%); ← Состояние при наведении: кнопка становится
  border-radius: 1em;                 красной со скругленными углами
}
```

Свойство `transition-property` указывает, в каких свойствах должен присутствовать переход. В данном случае ключевое слово `all` означает, что переход задается для всех свойств, которые изменяются. Свойство `transition-duration` указывает продолжительность перехода к конечному значению. В данном случае значение `0.5s` означает 0,5 с.

Загрузите страницу и посмотрите, как протекает переход при наведении указателя мыши на кнопку. Обратите внимание на то, что свойство `border-radius` мягко переходит от значения 0 к 1 em, несмотря на то что вы явно не указывали нулевое значение скругления углов для нормального состояния. Нулевое значение для кнопки устанавливается автоматически, и функция перехода работает с ним. Попробуйте изменить другие свойства состояния при наведении указателя, например `font-size` или `border`.

Переход выполняется каждый раз, когда значение данного элемента меняется. Это может произойти при изменении состояния с помощью псевдокласса `:hover` или если JavaScript-код, например, добавляет или удаляет класс, влияющий на стили элемента.

Заметьте, что вы не применяли свойства перехода в наборе правил `:hover`, а сделали это в селекторе, который связан с элементом всегда, даже когда подразумевается правило `:hover`. Вы хотите, чтобы переход осуществлялся как при наведении указателя мыши на элемент (переход в измененное состояние), так и при его снятии (возврат в нормальное состояние). В то время как другие значения меняются, вы, как правило, не хотели бы, чтобы сами свойства перехода менялись.

Можете также использовать краткую форму записи свойства — `transition`. Ее синтаксис приведен на рис. 14.2. В краткой форме указываются до четырех значений четырех разных свойств: `transition-property`, `transition-duration`, `transition-timing-function` и `transition-delay`.

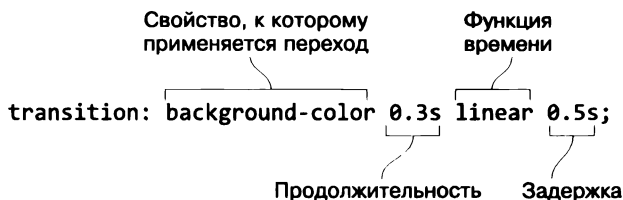


Рис. 14.2. Синтаксис краткой формы свойства `transition`

Первое значение указывает на свойство, к которому должен быть применен переход. Значение по умолчанию — ключевое слово `all`, которое применяет переход ко всем свойствам. Но если нужно, чтобы переход был только у одного свойства, укажите его в этом значении. Например, правило `transition-property: color` применит переход только к свойству `color`, другие же свойства будут меняться мгновенно. Можете также вписать несколько значений, например: `transition-property: color, font-size`.

Второе значение — продолжительность перехода. Оно представляет собой время, выраженное в секундах (например, `0.3s`) или миллисекундах (`300ms`).

ВНИМАНИЕ!

В отличие от длины для времени нельзя указать значение 0. Вы должны задавать единицы измерения для значений времени (`0s` или `0ms`), в противном случае объявление будет считаться недействительным и браузер его проигнорирует.

Третье значение — это функция времени. Она контролирует вычисление промежуточных значений свойства, эффективно управляя ускорением и замедлением изменений в течение перехода. В качестве ее значений используются ключевые

слова, такие как **linear** или **ease-in**, или пользовательская функция. Функция времени — важная часть перехода, и вскоре мы познакомимся с ней ближе.

Последнее значение — задержка перехода — позволяет указать период ожидания от изменения значения свойства до начала перехода. Если вы наведете указатель мыши на кнопку с задержкой перехода 0,5 с, то в течение этого времени не увидите изменений.

Если требуется применить два различных перехода к двум различным свойствам, добавьте несколько правил, разделенных запятыми:

```
transition: border-radius 0.3s linear, background-color 0.6s ease;
```

Или используйте обычную форму записи. Далее приведен аналог:

```
transition-property: border-radius, background-color;  
transition-duration: 0.3s, 0.6s;  
transition-timing-function: linear, ease;
```

В этой главе вы найдете примеры множества переходов.

14.2. Функции времени

Функция времени — это важная часть перехода. Если переход заставляет значение свойства меняться, то функция времени контролирует, *как* это происходит. Будет ли скорость изменения постоянной, или оно начнется медленно и будет ускоряться?

Используйте несколько ключевых слов, таких как **linear**, **ease-in** и **ease-out**, чтобы определить тип движения. С ключевым словом **linear** значение изменяется с постоянной скоростью. С **ease-in** изменение вначале протекает медленно, но ускоряется до самого конца перехода. С ключевым словом **ease-out** изменение начинается быстро, но к концу перехода замедляется. Рисунок 14.3 иллюстрирует, как блок будет двигаться слева направо при использовании различных функций времени.



Рис. 14.3. Переход **linear** задает движение с постоянной скоростью, с **ease-in** движение ускоряется, а с **ease-out** — замедляется

Скорость непросто визуализировать на статичном изображении, поэтому напомним демонстрационный пример, чтобы увидеть ее вживую в браузере. Создайте новую HTML-страницу и добавьте следующую разметку (листинг 14.3).

Листинг 14.3. Пример простой функции времени

```
<div class="container">
  <div class="box"></div> <div class="box"></div> <div class="box"></div>
</div>
```

Блок будет перемещаться по экрану слева направо

Затем отформатируйте блок с помощью стилей, придав ему цвет и размер. После этого задайте ему абсолютное позиционирование и используйте переход, чтобы сменить позицию при наведении указателя мыши. Добавьте на страницу новую таблицу стилей и скопируйте в нее код из листинга 14.4.

Листинг 14.4. Перемещение блока слева направо

```
.container {
  position: relative;
  height: 30px;
}
.box {
  position: absolute;
  left: 0;
  height: 30px;
  width: 30px;
  background-color: hsl(130, 50%, 50%);
  transition: all 1s linear;
}
.container:hover .box {
  left: 400px;
}
```

Установка начальной позиции слева

Применение перехода

Перемещение на 400 пикселей вправо при наведении указателя мыши

В левом верхнем углу страницы вы увидите маленький блок. При наведении на него указателя мыши он переместится вправо. Обратите внимание на то, что он будет двигаться с постоянной скоростью.

ВНИМАНИЕ!

Данный пример иллюстрирует, как элемент с абсолютным позиционированием перемещается по экрану с помощью перехода в свойстве `left`. Тем не менее существуют причины избегать использования переходов в определенных свойствах, включая свойство `left`. Я раскрою эти проблемы в главе 15, а также расскажу о лучшей альтернативе — трансформациях.

Теперь отредактируйте свойство перехода, чтобы увидеть различия в поведении функции времени. Попробуйте задействовать ключевые слова `ease-in` (`transition: all 1s ease-in`) и `ease-out` (`transition: all 1s ease-out`). Они справляются со своей задачей, но иногда вам может понадобиться усилить контроль. Настраивайте изменение скорости перехода, определяя собственные функции времени. Посмотрим, как это делается.

14.2.1. Изучение кривых Безье

Функции времени базируются на математически определенных кривых Безье. Браузер использует их для вычисления значения свойства как функции изменения с течением времени. Кривые Безье для некоторых функций времени и ключевые слова, которые можно задействовать в качестве функций времени, приведены на рис. 14.4.

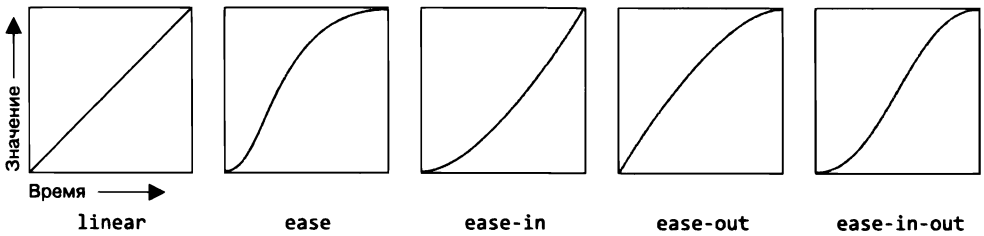


Рис. 14.4. Кривые Безье функций времени иллюстрируют изменение значения с течением времени

Начинаются эти кривые в левом нижнем углу, а заканчиваются — в правом верхнем. Направление хода времени — слева направо. И кривая представляет, как значение будет меняться во времени, прежде чем достигнет конечного значения. Линейная функция времени представляет собой постоянное движение в течение всего перехода, то есть прямую линию. Другие значения имеют искривления, отображая ускорение или замедление.

Однако вы не ограничены приведенными пятью ключевыми словами. Можете определить собственную кривую кубического уравнения Безье для более плавных или резких переходов. Или добавить эффект отскока. Поговорим об этом подробнее.

На созданной странице откройте консоль разработчика и изучите элемент `box`. На панели **Styles** (Стили) (браузер Chrome) или **Rules** (Правила) (браузер Firefox) рядом с функцией времени увидите небольшой значок. Щелкните на нем кнопкой мыши, после чего откроется небольшое всплывающее окно, в котором можно изменить кривую функции времени (рис. 14.5).

В левой части всплывающего окна предлагается на выбор серия готовых кривых Безье. (В браузере Firefox их гораздо больше, чем в Chrome.) Щелкните кнопкой мыши на любой, чтобы выбрать ее. Кривая отобразится в правой части окна.

На обоих концах кривой есть короткие прямые линии с точками на конце — *манипуляторы*. Эти точки называются *контрольными*. Перетащите одну из них, чтобы редактировать форму кривой. Обратите внимание на то, как длина и направление манипулятора влияют на кривую.

Щелкните за пределами всплывающего окна, чтобы закрыть его, и увидите, что функция времени обновилась. Вместо ключевого слова, такого как `ease-out`, появилось значение, выглядящее приблизительно так: `cubic-bezier(0.45, 0.05, 0.55, 0.95)`. Это функция `cubic-bezier()`, а четыре значения в скобках определяют пользовательскую функцию времени.

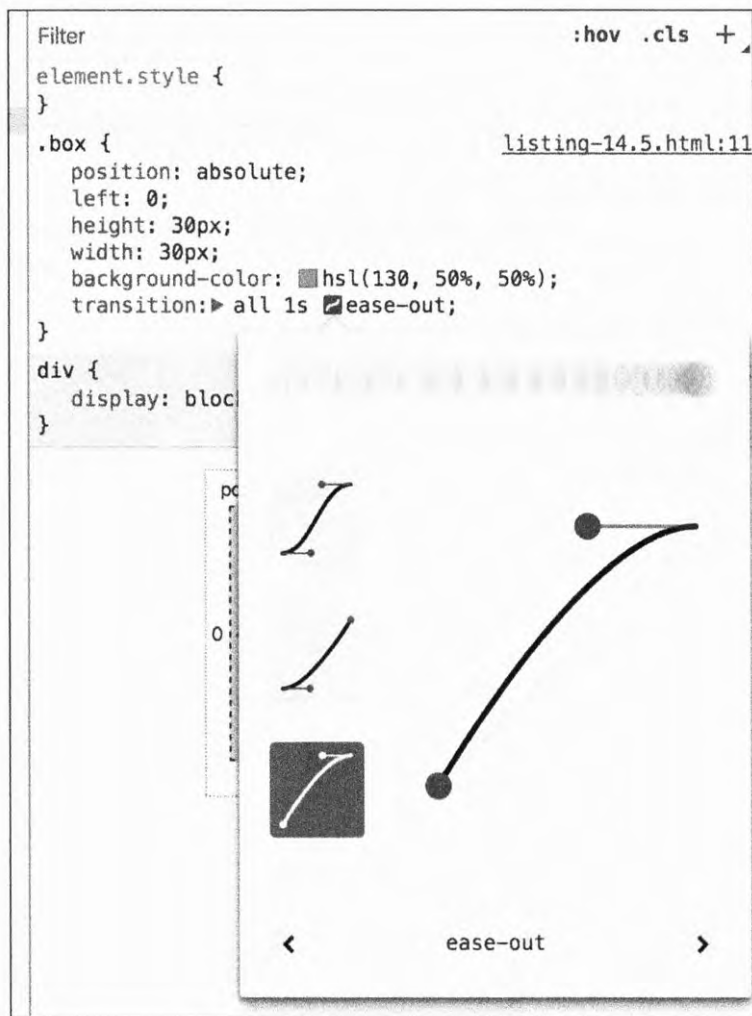


Рис. 14.5. Редактирование кривой Безье в консоли разработчика Chrome

Выбор функции времени

Вне зависимости от того, используете вы ключевые слова в функциях времени или пользовательские кривые Безье, полезно знать, при каких обстоятельствах лучше подходит каждый из вариантов. Любые сайт и приложение могут поддерживать кривые замедления (*ease-out*), кривые ускорения (*ease-in*), а также ключевое слово *linear*. Лучше всего использовать одни и те же кривые для обеспечения последовательного дизайна.

Вы можете применять все три функции в следующих сценариях.

- *Линейное движение* — изменение цвета и эффекты усиления/ослабления.
- *Замедление* — изменения, вызванные действиями пользователя. Когда он нажимает кнопку или наводит указатель мыши на элемент, задействуйте `ease-out` или подобное значение. Так пользователь заметит быструю непосредственную реакцию на свое действие, эффект от которой будет пропадать при остановке элемента.
- *Ускорение* — изменения, вызванные системой. Когда загрузка контента завершается или запускается событие превышения лимита времени, используйте `ease-in` или что-то похожее. Сначала элемент привлечет внимание пользователя анимацией, а потом движение ускорится и завершится.

Правила просты, следование им станет хорошей отправной точкой. Но не бойтесь нарушить их, если сочтете нужным. В некоторых случаях для более продолжительной анимации можно взять и четвертую кривую: используйте ключевое слово `ease-in-out` (ускорение и последующее замедление) или эффект отскока (см. пример в главе 15).

Давайте подробнее изучим, как работает функция `cubic-bezier()`. Другой пример кривой приведен на рис. 14.6.

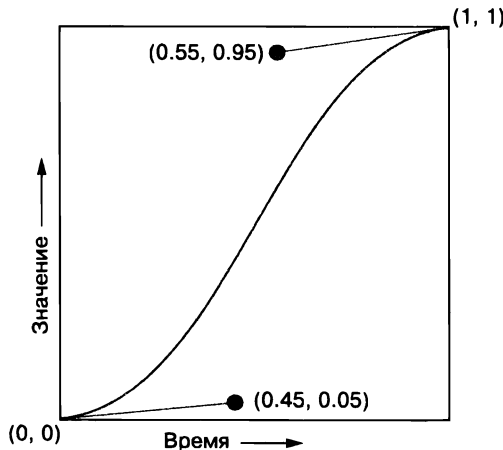


Рис. 14.6. Кривая Безье, представляющая функцию времени

Здесь показана пользовательская кривая Безье. Она вначале ускоряется, сохраняет высокую скорость в середине (самая крутая часть кривой) и замедляется в конце. Кривая существует в декартовой системе координат. Она начинается в точке (0, 0) и заканчивается в точке (1, 1).

При наличии конечных точек положение двух манипуляторов — это все, что нужно для определения кривой. В CSS данная кривая может быть задана как

`cubic-bezier(0.45, 0.05, 0.55, 0.95)`. Четыре значения представляют собой координаты x и y двух контрольных точек манипуляторов.

Зная только эти числа, трудно представить себе форму кривой. Гораздо удобнее работать с ней с помощью графических интерфейсов. По этой причине я люблю редактировать и тестировать переходы в браузере, прежде чем копировать готовое кубическое выражение Безье в таблицу стилей. Для этой задачи предпочитаю пользоваться консолью разработчика. Можно применять также онлайн-ресурсы, например `cubic-bezier.com`.

14.2.2. Шаги

Последний тип функций времени представлен функцией `steps()`. Она задает не плавные переходы от одного значения к другому, основанные на кривых Безье, а переходы посредством отдельных мгновенных шагов.

Функция принимает два параметра: количество шагов и ключевое слово, например `start` или `end`, которое определяет, в начале или в конце шага должно происходить каждое изменение. Некоторые варианты шаговой функции показаны на рис. 14.7.

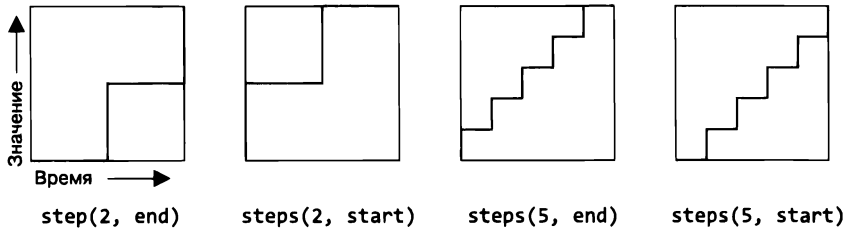


Рис. 14.7. Функция `steps()` изменяет значение пошагово

Обратите внимание на то, что ключевое слово `end` — это значение по умолчанию для второго параметра, поэтому запись `steps(3)` аналогична записи `steps(3, end)`. Чтобы увидеть функцию в действии, внесите изменения в таблицу стилей согласно листингу 14.5.

Листинг 14.5. Использование функции `steps()` для изменения значения

```
.box {
  position: absolute;
  left: 0;
  height: 30px;
  width: 30px;
  background-color: hsl(130, 50%, 50%);
  transition: all 1s steps(3);
}
```

← Переход осуществляется в три шага

Теперь перемещение слева направо в течение одной секунды (продолжительность перехода) не будет плавным — оно разделено по времени на трети, или три шага.

На каждом шаге блок появляется в стартовой позиции — сначала первой трети, затем второй и третьей, пока не переместится в конечную позицию к окончанию секунды.

ПРИМЕЧАНИЕ

По умолчанию значение свойства меняется в конце каждого шага, поэтому переход не начинается немедленно. Вы можете изменить это поведение так, чтобы изменения происходили в начале каждого шага, а не в конце, добавив ключевое слово `start`. Тогда запись будет выглядеть следующим образом: `steps(3, start)`.

На практике функцию `steps()` применяют довольно редко, но существует список умных идей для ее использования по адресу csstricks.com/clever-uses-step-easing/.

14.3. Неанимируемые свойства

Многие переходы отображают движение. Например, можно применить правило `transition: color 200ms linear` к ссылкам, чтобы они меняли цвет при наведении указателя или щелчке кнопкой мыши. Или задать переход для фона блоков, реагирующих на щелчок, или отступов кнопок.

Если JavaScript-код меняет что-либо на странице, подумайте над тем, приемлемо ли здесь добавление переходов. В некоторых случаях это просто добавление свойства перехода к выбранному элементу. В других потребуется более явное координирование. В оставшейся части данной главы вы создадите раскрывающееся меню и примените переходы, чтобы оно разворачивалось беспрепятственно без щелчка на нем.

Сначала вы заставите его появляться, создав переход для значения `opacity`. Затем измените раскрывающийся список, чтобы использовать эффект перехода в свойстве `height`. Оба эффекта вызывают возникновение специфических проблем, требующих некоторых размышлений.

Меню будет выглядеть как на рис. 14.8. Вы начнете с того, что сделаете меню открывающимся и закрывающимся. Затем добавите эффекты переходов. Под меню я добавил ссылку. Обратите внимание на то, как выглядит `drawer`-элемент меню по отношению к этой ссылке, когда меню открыто, это важно.

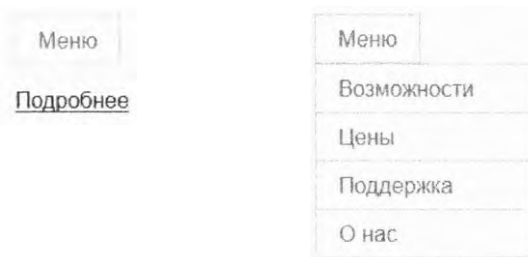


Рис. 14.8. Раскрывающееся меню в закрытом (слева) и открытом (справа) состоянии

Создайте новую страницу для раскрывающегося меню, добавив разметку, приведенную в листинге 14.6. Оно аналогично раскрывающимся меню, созданным в предыдущих главах, и содержит небольшое количество JavaScript-кода для переключения состояний меню с открытого на закрытое и наоборот.

Листинг 14.6. Раскрывающееся меню с переходами

```
<div class="dropdown" aria-haspopup="true">
  <button class="dropdown_toggle">Меню</button>
  <div class="dropdown_drawer">
    <ul class="menu" role="menu">
      <li role="menuitem">
        <a href="/features">Возможности</a>
      </li>
      <li role="menuitem">
        <a href="/pricing">Цены</a>
      </li>
      <li role="menuitem">
        <a href="/support">Поддержка</a>
      </li>
      <li role="menuitem">
        <a href="/about">О нас</a>
      </li>
    </ul>
  </div>
</div>
<p><a href="/read-more">Подробнее</a></p>
```

Появляющийся и исчезающий drawer-элемент меню

Ссылка под раскрывающимся меню

```
<script type="text/javascript">
  (function () {
    var toggle = document.getElementsByClassName('dropdown_toggle')[0];
    var dropdown = toggle.parentElement;
    toggle.addEventListener('click', function (e) {
      e.preventDefault();
      dropdown.classList.toggle('is-open');
    });
  })();
</script>
```

Включает в контейнер или исключает из него класс is-open при нажатии кнопки

Стили до добавления эффекта появления показаны в листинге 14.7. Добавьте их в таблицу стилей и свяжите ее со страницей. Я включил сюда несколько эффектов перехода, поэтому при наведении указателя мыши цвета меняются плавно. Кроме того, в коде есть кое-что новое, но он нужен для установки страницы, чтобы вы могли сосредоточиться на создании эффекта появления.

Откройте страницу в браузере и проверьте ее. Меню должно открываться и закрываться при щелчке на кнопке-переключателе. Обратите внимание на то, как кнопка и ссылки меню плавно меняют цвета при наведении на них указателя мыши и возвращаются в начальное состояние при его переводе в другую область.

Для эффектов наведения указателя мыши я задал продолжительность перехода 0,2 с. Как показывает опыт, большинство переходов должны длиться 200–500 мс. Если переход окажется более продолжительным, пользователям будет казаться, что

страница медленная и слишком долго не откликается. Тем более если такие переходы часто встречаются.

Листинг 14.7. Стили раскрывающегося меню с переходами

```
body {
  font-family: Helvetica, Arial, sans-serif;
}

.dropdown__toggle {
  display: block;
  padding: 0.5em 1em;
  border: 1px solid hsl(280, 10%, 80%);
  color: hsl(280, 30%, 60%);
  background-color: white;
  font: inherit;
  text-decoration: none;
  transition: background-color 0.2s linear;
}

.dropdown__toggle:hover {
  background-color: hsl(280, 15%, 95%);
}

.dropdown__drawer {
  position: absolute;
  display: none;
  background-color: white;
  width: 10em;
}

.dropdown.is-open .dropdown__drawer {
  display: block;
}

.menu {
  padding-left: 0;
  margin: 0;
  list-style: none;
}

.menu > li + li > a {
  border-top: 0;
}

.menu > li > a {
  display: block;
  padding: 0.5em 1em;
  color: hsl(280, 40%, 60%);
  background-color: white;
  text-decoration: none;
  transition: all .2s linear;
  border: 1px solid hsl(280, 10%, 80%);
}

.menu > li > a:hover {
  background-color: hsl(280, 15%, 95%);
  color: hsl(280, 25%, 10%);
}
```

Запуск перехода
при смене цвета фона

Смена цвета фона
при наведении
указателя мыши

Переходы цветов
фона и текста

Смена цвета при наведении
указателя мыши

СОВЕТ

Используйте быстрые переходы для эффектов при наведении указателя мыши, для эффектов появления/исчезновения и уменьшения размеров. Скорость таких переходов не должна превышать 300 мс, иногда можно даже задать 100 мс. Для переходов с продолжительным движением или сложными функциями времени, такими как у эффектов отскока (см. главу 15), задействуйте более длительный промежуток времени — 300–500 мс.

Работая над переходами, я иногда замедляю их до 2–3 с. Это позволяет внимательно рассмотреть, как выполняется переход и соответствует ли ожиданиям его поведение. Если будете применять этот прием, убедитесь, что, закончив, вернули высокую скорость.

14.3.1. Свойства, которые нельзя анимировать

Не все свойства можно анимировать. Одно из них — `display`. Можно лишь переключаться между двумя значениями, `display: none` и `display: block`. Переход между ними выполнить нельзя, поэтому свойства переходов, примененные к `display`, будут проигнорированы.

Если найти свойство в справочнике, допустим, MDN (developer.mozilla.org/ru/), то, как правило, там будет сказано, можно ли его анимировать и какое значение (например, длина, цвет, процент) может быть интерполировано. Описание свойства `background-color` есть по адресу developer.mozilla.org/ru/docs/Web/CSS/background-color, оно показано на рис. 14.9.

Начальное значение	<code>transparent</code>
Применяется к	все элементы. Это также применяется к <code>::first-letter</code> и <code>::first-line</code> .
Наследуется	нет
Отображение	визуальный
Обработка значения	вычисленный цвет
Animation type	цвет
Канонический порядок	уникальный неоднозначный порядок, определённый формальной грамматикой

Рис. 14.9. Документация MDN содержит блоки технического описания для всех свойств

Как показано на рисунке, свойство `background-color` может быть анимировано только в значении цвета, то есть можно выполнить переход от одного цвета к другому, что имеет смысл, поскольку данное свойство устанавливает цвет. Строка **Animation Type** используется как для переходов, так и для анимации, которые будут рассматриваться в главе 16. Документация содержит и другую полезную информацию о свойстве, например: его значение по умолчанию, к каким элементам оно применяется и является ли наследуемым. Если вам нужно хорошее техническое описание того, как работать с этим свойством, найдите его в документации MDN и взгляните на блок описания.

ПРИМЕЧАНИЕ

Большинство свойств, принимающих значения длины, количества, цвета или функцию `calc()`, могут быть анимированы. Большая часть свойств, требующих ключевые слова или другие дискретные значения, такие как `url()`, — нет.

Если бы вам понадобилось найти свойство `display`, вы бы увидели, что его тип анимации обозначен как `discrete`, что означает: свойство способно меняться только между дискретным количеством значений и не может быть интерполировано в анимацию или переход. Если вы хотите, чтобы элемент появлялся или исчезал, то не сможете использовать переход для свойства `display`, но сможете — свойство `opacity`.

14.3.2. Появление и исчезновение

Теперь применим переход прозрачности, чтобы добавить в меню эффект появления и исчезновения при открытии и закрытии. Результат должен выглядеть так, как показано на рис. 14.10.

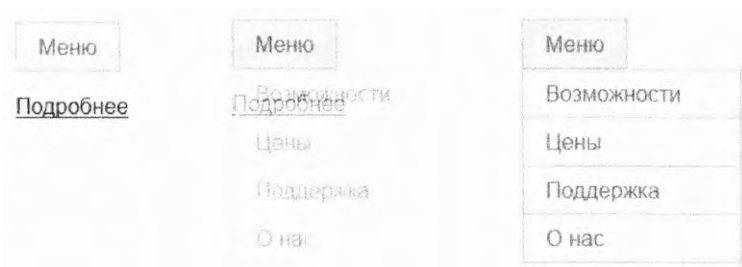


Рис. 14.10. Появление меню

Свойство `opacity` может иметь любое значение от 0 (невидимый) до 1 (непрозрачный). Листинг 14.8 иллюстрирует основную идею. Данный код не будет работать в одиночку по причинам, которые вы скоро поймете. Отредактируйте таблицу стилей, как показано в листинге.

Листинг 14.8. Добавление правил прозрачности и перехода

```

.dropdown__drawer {
  position: absolute;
  background-color: white;
  width: 10em;
  opacity: 0;
  transition: all 0.2s linear;
}
.dropdown.is-open .dropdown__drawer {
  opacity: 1;
}

```

← Замена записи display: none на opacity: 0

← Переход для свойства opacity

← Замена записи display: block на opacity: 1

Теперь меню появляется при открытии и исчезает при закрытии. К сожалению, при этом оно не пропадает, а всего лишь становится невидимым и по-прежнему остается на странице. Так что, попытавшись щелкнуть на ссылке [Подробнее](#), вы по ней не перейдете. У вас получится щелчок на невидимом элементе меню, расположенном перед ссылкой, и вы попадете на страницу [Возможности](#).

Необходимо не только создать переход для свойства `opacity`, но и полностью убрать `drawer`-элемент меню, когда он невидим. Это делается с помощью другого свойства — `visibility`.

Свойство `visibility`, как и свойство `display`, позволяет убрать элемент со страницы. Можно присваивать ему значения `visible` или `hidden`. В отличие от `display` свойство `visibility` анимируется.

ПРИМЕЧАНИЕ

Применение правила `visibility: hidden` к элементу удаляет его с видимой страницы, но он не исчезает из потока, а продолжает занимать пространство. Другие элементы все так же будут обтекать его позицию, оставляя свободное место на странице. В нашем случае это не отразится на меню, поскольку ему уже задано абсолютное позиционирование.

Листинг 14.9. Использование задержки перехода, чтобы установить, когда будет меняться свойство `visibility`

```

.dropdown__drawer {
  position: absolute;
  background-color: white;
  width: 10em;
  opacity: 0;
  visibility: hidden;
  transition: opacity 0.2s linear,
              visibility 0s linear 0.2s;
}
.dropdown.is-open .dropdown__drawer {
  opacity: 1;
  visibility: visible;
  transition-delay: 0s;
}

```

← Панель меню скрыта и невидима, когда меню закрыто

← Задерживает переход visibility на 0,2 с

← Удаляет задержку перехода, когда элементу присвоен класс is-open

Теперь переход разбит на два набора значений. Это определяет проявление эффекта исчезновения. Первый набор значений задает переход прозрачности длительностью 0,2 с. Второй набор использует переход видимости продолжительностью 0 с (мгновенный) после задержки в течение 0,2 с. Это означает, что сначала выполняется переход свойства `opacity`, а по его окончании — переход `visibility`. Это позволяет меню исчезать постепенно, а когда оно станет прозрачным, свойство `visibility` переключается на значение `hidden`. Тогда пользователь сможет щелкнуть на ссылке [Подробнее](#), и меню ему не помешает.

Когда меню появляется, порядок должен быть изменен: видимость должна переключиться сразу после перехода прозрачности. Именно поэтому во втором наборе правил задана задержка перехода 0s. Так свойство `visibility` будет иметь значение `hidden`, пока меню закрыто, но во время переходов появления и исчезновения оно станет меняться на `visible`.

СОВЕТ

Вы можете использовать JavaScript-событие `transitionend`, чтобы выполнить действие после завершения перехода.

Эффект появления/исчезновения реализуется и с помощью JavaScript вместо `transition-delay`, но, по моему мнению, это потребует большего количества кода и может привести к ошибкам. Иногда язык JavaScript все же необходим для достижения желаемого результата (который вы вскоре увидите), но если переход или анимация могут быть получены с помощью одного CSS-кода, это почти всегда предпочтительно.

14.4. Переход к автоматическому выравниванию высоты

Вернемся к раскрывающемуся меню, чтобы реализовать другой часто встречающийся эффект — разворачивание и сворачивание с помощью перехода, примененного к высоте (рис. 14.11).

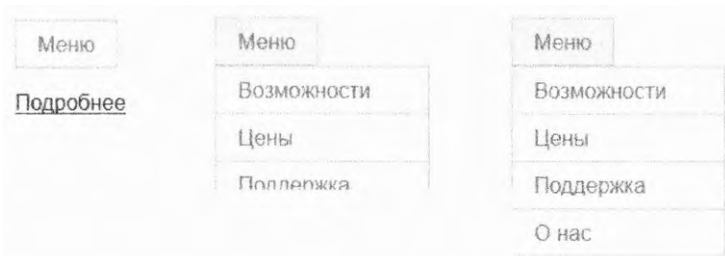


Рис. 14.11. Разворачивание элемента с помощью перехода, примененного к его высоте

Когда меню открывается, происходит переход его высоты от нуля до естественного значения (**auto**). Когда закрывается, она возвращается к нулевому значению. Основная идея показана в листинге 14.10. К сожалению, код не работает. Отредактируйте свой CSS-код, чтобы он соответствовал приведенным правилам, затем мы попытаемся узнать, в чем заключается проблема и как ее решить.

Листинг 14.10. Переход, примененный к высоте

```
.dropdown__drawer {
  position: absolute;
  background-color: white;
  width: 10em;
  height: 0;
  overflow: hidden;
  transition: height 0.3s ease-out;
}
.dropdown.is-open .dropdown__drawer {
  height: auto;
}
```

Закрытый drawer-элемент меню имеет нулевую высоту и скрыт из потока

Переход свойства height

Высота открытого drawer-элемента определяется его содержимым

Свойству **overflow** задано значение **hidden**, чтобы обрезать контент drawer-элемента, когда он закрыт или в нем осуществляется переход. Но код не работает, поскольку значение не может переходить от нулевой длины до автоматической.

Вы можете явно установить высоту, например, 120 пикселей, но проблема в том, что точно неизвестно, какой она должна быть. Это станет понятно, только когда содержимое задано и отображено в браузере, поэтому для определения высоты будет использоваться JavaScript.

После загрузки страницы будет получен доступ к свойству **scrollHeight** элемента DOM. Так вы узнаете его высоту. Затем немного измените код, чтобы установить высоту элемента равной полученному значению. Отредактируйте скрипт на странице в соответствии с листингом 10.11.

Листинг 14.11. Явная установка высоты, чтобы переход работал

```
(function () {
  var toggle = document.getElementsByClassName('dropdown__toggle')[0];
  var dropdown = toggle.parentElement;
  var drawer = document.getElementsByClassName('dropdown__drawer')[0];
  var height = drawer.scrollHeight;

  toggle.addEventListener('click', function (e) {
    e.preventDefault();
    dropdown.classList.toggle('is-open');
    if (dropdown.classList.contains('is-open')) {
      drawer.style.setProperty('height', height + 'px');
    } else {
      drawer.style.setProperty('height', '0');
    }
  });
})();
```

Получение вычисленной автоматической высоты drawer-элемента

Явно устанавливается высота для открытия drawer-элемента

Восстанавливает нулевое значение высоты для закрытия drawer-элемента

Теперь, помимо переключения класса `is-open`, вы явно указали высоту в пикселах, поэтому переход осуществляется к корректной высоте. Затем установили возврат высоты к нулевому значению при закрытии, благодаря чему выполняется обратный переход.

ВНИМАНИЕ!

Свойство элемента `scrollHeight` равно 0, если элемент скрыт с помощью правила `display: none`. Если это произойдет, можете установить для свойства `display` значение `block` (`el.style.display = 'block'`) и получите доступ к свойству `scrollHeight`, затем восстановите значение свойства `display` (`el.style.display = 'none'`).

Иногда переходы требуют координации между CSS и JavaScript. В некоторых случаях оказывается заманчиво полностью переместить логику в JavaScript-код. Например, вы могли бы воспроизвести переход высоты, повторно устанавливая новую высоту только в JavaScript-коде. Но, как правило, необходимо переносить всю возможную нагрузку в CSS-код. Он лучше оптимизирован в браузере, поэтому более производительен и предоставляет некоторые возможности, например плавность, которые могут потребовать большого количества кода.

Мы еще не закончили изучение переходов. Они пригодятся в процессе работы с трансформациями в следующей главе.

Итоги главы

- ❑ Можно использовать переходы для сглаживания резких изменений на странице.
- ❑ Чтобы привлечь внимание пользователя, применяйте ускоряющееся движение.
- ❑ Чтобы показать пользователю, что его действие вступило в силу, используйте замедляющееся движение.
- ❑ Можете задействовать JavaScript, чтобы скоординировать переходы с изменениями имен классов, когда CSS-код не способен обеспечить то, что вам нужно.

15 Трансформации

В этой главе

- Изменение элементов с помощью трансформации для имитации переходов и анимации.
- Создание эффекта «выпрыгивания» при переходе.
- Поток процессов при рендеринге в браузере.
- Рассмотрение 3D-трансформаций и перспективы.

В этой главе рассмотрим свойство `transform`, которое используется для изменения или искажения формы или положения элемента на странице. Сюда относятся вращение, масштабирование, смещение и наклон изображения в двух или трех плоскостях. Трансформация в большинстве случаев применяется вместе с переходами или анимацией, вот почему я раскрываю эту тему между посвященными им главами. В этой и следующей главах вы создадите страницу, которая будет переполнена переходами, трансформациями и анимацией.

Для начала я покажу, как задействовать трансформацию для статичных элементов. Это позволит понять, как она работает сама по себе, прежде чем добавлять ее в переходы. Затем вы создадите небольшое, но довольно сложное меню с множеством эффектов трансформации и переходов. Наконец, мы рассмотрим работу в трехмерном пространстве с использованием перспективы, что даст базовые понятия для более подробного рассмотрения 3D в анимации в следующей главе.

15.1. Вращение, масштабирование, смещение и наклон

Обычное правило трансформации выглядит следующим образом:

```
transform: rotate(90deg);
```

Это правило, применяемое к элементу, поворачивает его на 90° вправо (по часовой стрелке). Функция трансформирования `rotate()` определяет, как именно изменять элемент. Существует еще несколько функций, относящихся к одной из следующих категорий (продемонстрированы на рис. 15.1).

- *Вращение* — поворачивает элемент на определенное количество градусов вокруг оси.

- ❑ *Смещение* — смещает элемент влево, вправо, вверх или вниз (похоже на относительное позиционирование).
- ❑ *Масштабирование* — сжимает или растягивает элемент.
- ❑ *Наклон* — изменяет форму элемента, смещая верхний край элемента в одну сторону, а нижний — в противоположную.

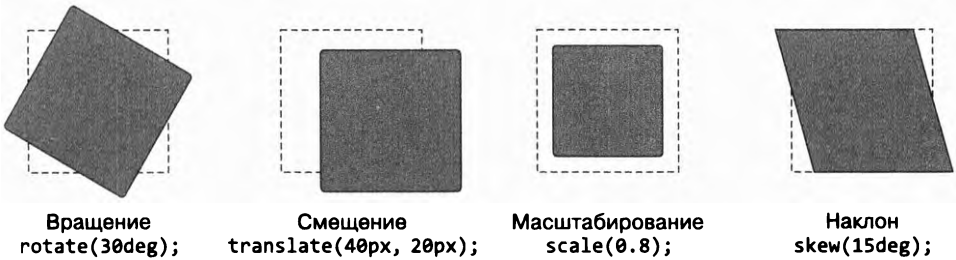


Рис. 15.1. Четыре базовых типа трансформации (пунктирная линия показывает изначальное положение элемента)

Каждая трансформация действует как соответствующая функция в значении свойства `transform`. Создадим простой пример и посмотрим, как все работает в браузере. Это будет простая карточка с изображением и текстом (рис. 15.2), к которой применена трансформация.

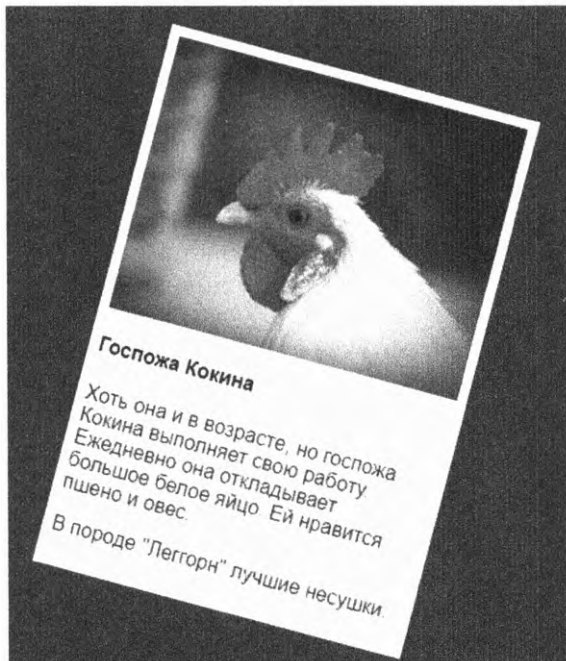


Рис. 15.2. Простая карточка с эффектом трансформации

Создайте новую страницу и связанную с ней таблицу стилей. Вставьте следующий HTML-код (листинг 15.1).

Листинг 15.1. Создание простой карточки

```
<div class="card">
  
  <h4>Госпожа Кокина</h4>
  <p>Хоть она и в возрасте, но госпожа Кокина выполняет свою работу.
    Ежедневно она откладывает большое белое яйцо. Ей нравится пшено и овес.</p>
  <p>В породе "Леггорн" лучшие несушки.</p>
</div>
```

Затем добавьте CSS-код в таблицу стилей. Сюда включены базовые стили, цвета и карточка с эффектами трансформации (листинг 15.2).

Листинг 15.2. Форматирование карточки и применение эффекта трансформации

```
body {
  background-color: hsl(210, 80%, 20%);
  font-family: Helvetica, Arial, sans-serif;
}

img {
  max-width: 100%;
}

.card {
  padding: 0.5em;
  margin: 0 auto;
  background-color: white;
  max-width: 300px;
  transform: rotate(15deg);
}
```

Размещение карточки по центру

Поворот карточки на 15° вправо

Открыв страницу в браузере, вы увидите повернутую карточку. Поэкспериментируйте, чтобы понять, как работает функция `rotate()`. Чтобы повернуть изображение в другую сторону, используйте отрицательное значение в градусах, например `rotate(-30deg)`.

А затем попробуйте заменить функцию вращения какой-либо другой. Применяйте следующие свойства и смотрите, как изменится внешний вид страницы:

- ❑ `skew(20deg)` наклоняет карточку на 20°. Попробуйте задать отрицательное значение, чтобы наклонить в другую сторону;
- ❑ `scale(0.5)` уменьшает карточку в половину. Данная функция принимает в качестве аргумента безразмерные единицы. Значения меньше 1 сжимают элемент, больше — растягивают;
- ❑ `translate(20px, 40px)` смещает изображение на 20 пикселей вправо и на 40 пикселей вниз. Можно использовать отрицательные значения, чтобы сместить изображение в противоположные стороны.

При трансформации стоит учитывать следующее: даже если элемент меняет свое положение на странице, его действительное местоположение не изменяется и это не влияет на другие элементы. Даже если переместить объект за пределы экрана, его изначальное место никто не займет, оно останется пустым. Также при вращении угол элемента может выйти за пределы экрана или перекрыть часть объекта, находящегося за ним (рис. 15.3).

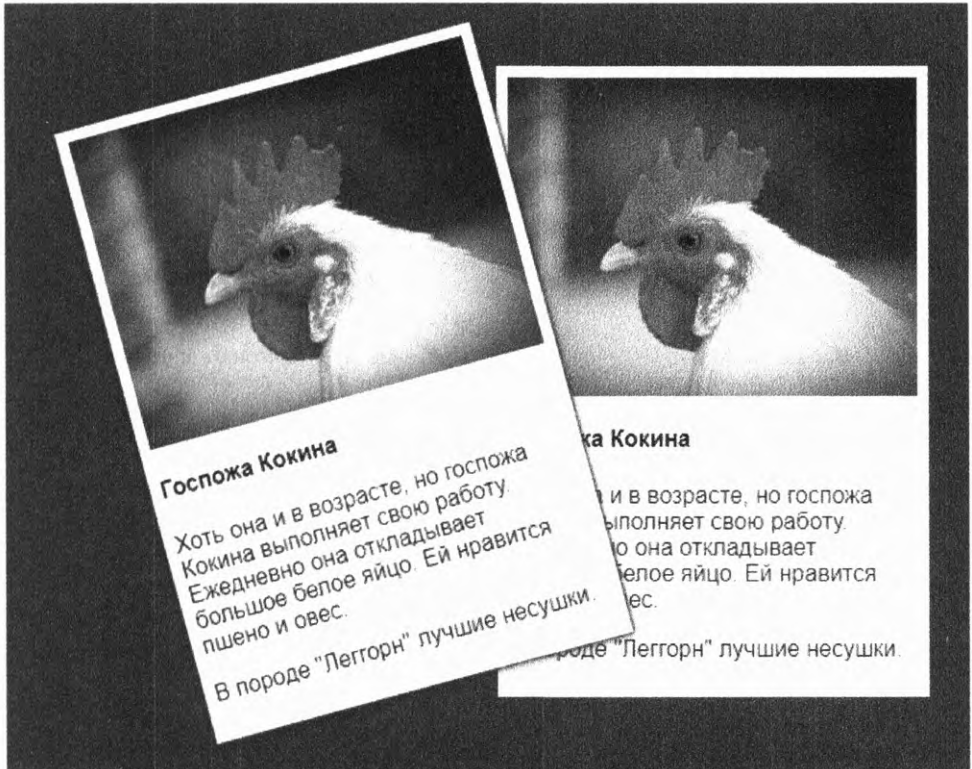


Рис. 15.3. Трансформация одного элемента не влияет на положение других, следовательно, они могут накладываться

В некоторых случаях я задаю большие поля для одного или обоих элементов, чтобы предотвратить нежелательное наложение.

ВНИМАНИЕ!

Трансформацию нельзя применить к некоторым строчным элементам, таким как `span` или `a`. Чтобы их трансформировать, нужно или изменить значение свойства `display` на что-то отличное от `inline` (например, `inline-block`), или заменить элемент на `flex` или сетчатый (применить `display: flex` или `display: grid` для родительского элемента).

15.1.1. Изменение точки трансформации

Любая трансформация выполняется относительно *точки трансформации*. Эта точка служит осью вращения или позицией, откуда начинаются масштабирование и наклон. То есть точка трансформации всегда остается на своем месте, а остальная часть элемента трансформируется относительно нее (не касается `translate()`, так как смещается вся фигура).

По умолчанию точкой трансформации является центр элемента, но ее можно задать самостоятельно с помощью свойства `transform-origin`. На рис. 15.4 представлены несколько элементов, трансформированных вокруг разных точек трансформации.

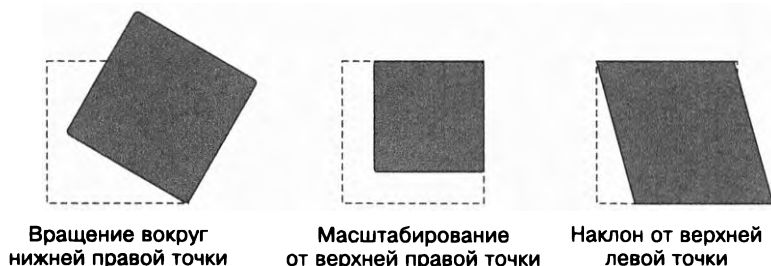


Рис. 15.4. Вращение, масштабирование и наклон с точками трансформации в разных углах элемента

Элемент, расположенный слева, вращается вокруг точки трансформации, заданной свойством `transform-origin: right bottom`. Для элемента по центру — `right top`. А для элемента справа точка трансформации задана свойством `left top` так, что верхний левый угол остается на месте, а остальная часть фигуры смещается.

Точку трансформации можно задать также в процентах, считая от верхнего левого угла элемента. Следующие две строки кода эквивалентны:

```
transform-origin: right center;
transform-origin: 100% 50%;
```

ПРИМЕЧАНИЕ

Можете использовать длину, чтобы задать точку трансформации в пикселах, ем или любых других единицах измерения. Хотя мой опыт показывает, что ключевых слов `top`, `right`, `bottom`, `left` и `center` более чем достаточно в большинстве случаев.

15.1.2. Применение нескольких трансформаций

Свойство трансформации можно задействовать для одного элемента несколько раз, разделяя значения запятой. Трансформации будут применяться по очереди справа налево. Например, если вы зададите `transform: rotate(15deg) translate(15px, 0)`,

элемент сместится на 15 пикселей вправо, а затем будет повернут на 15° по часовой стрелке. Добавьте в свою таблицу стилей следующий код (листинг 15.3), мы с ним немного поработаем.

Листинг 15.3. Применение нескольких трансформаций

```
.card {  
  padding: 0.5em;  
  margin: 0 auto;  
  background-color: white;  
  max-width: 300px;  
  transform: rotate(15deg) translate(20px, 0);  
}
```

Смещает элемент
на 20 пикселей вправо,
а затем поворачивает
на 15° по часовой стрелке

Самый простой способ — открыть инструменты разработчика в браузере и поэкспериментировать со значениями, чтобы увидеть, как изменения отражаются на элементе в реальном времени. Обратите внимание на то, что функция `translate()` смещает элемент чуть по диагонали. Так происходит, потому что вращение применяется после смещения.

Но, например, мне проще, когда смещение происходит последним (первым в исходном порядке свойства `transform`), так что я могу работать с привычными «лево/право», «верх/низ». Чтобы понять, что я имею в виду, поменяйте порядок в своем коде: `transform: translate(20px, 0) rotate(15deg)`.

15.2. Анимированные трансформации

Сами по себе трансформации применяются не так уж часто. Блок с эффектом `skew()` выглядит интересно, но вряд ли текст в нем легко читать. Однако при использовании совместно с анимацией трансформации становятся намного более полезными.

Создадим страницу, в которой реализуем эту идею. Изображение этой страницы вы найдете на рис. 15.5. Но добавьте на нее немного анимации.

Далее сформируем навигационное меню слева. Оно представляет собой четыре обычных значка, расположенных вертикально. Но при наведении на них указателя мыши будет появляться текст. Этот пример предусматривает несколько переходов и пару трансформаций. Для начала создадим страницу, а потом подробнее рассмотрим меню навигации. (В следующей главе вы сделаете карточки, которые находятся по центру, и анимацию для них.)

Создайте новую страницу и таблицу стилей с именем `style.css`. Добавьте в нее разметку (листинг 15.4). Она включает ссылку на веб-шрифты (Russo One и Echo 2) из API Google Fonts. А также внесите разметку шапки страницы и навигационного меню.

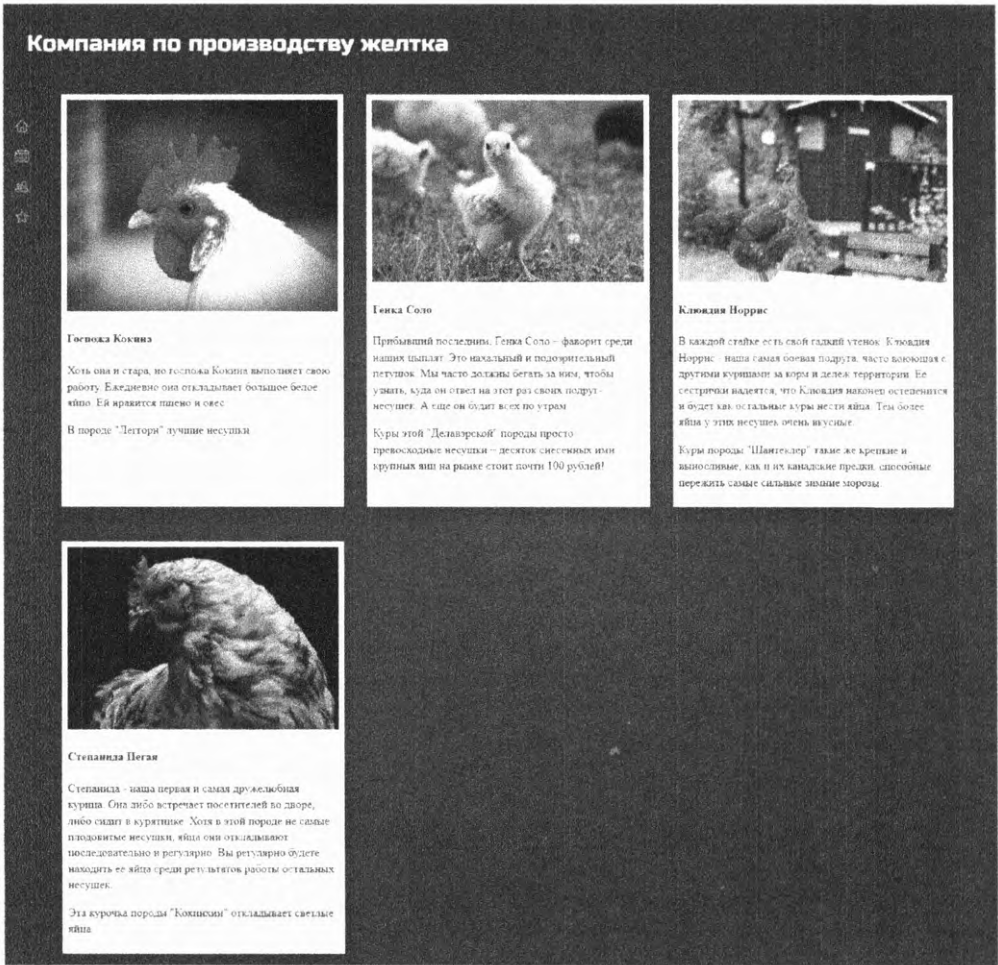


Рис. 15.5. К значкам слева будут применены несколько переходов и трансформаций

Листинг 15.4. Разметка страницы для анимированных трансформаций

```
<!doctype html>
<html lang="en">
  <head>
    <title>Компания по производству желтка</title>
    <link
      href="https://fonts.googleapis.com/css?family=Russo+One|Exo+2"
      rel="stylesheet">
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
```

Добавление на страницу шрифтов Russo One и Exo 2

```

<header>
  <h1 class="page-header">Компания по производству желтка</h1>
</header>
<nav class="main-nav">
  <ul class="nav-links">
    <li>
      <a href="/">
        
        <span class="nav-links__label">Главная</span>
      </a>
    </li>
    <li>
      <a href="/events">
        
        <span class="nav-links__label">События</span>
      </a>
    </li>
    <li>
      <a href="/members">
        
        <span class="nav-links__label">Участники</span>
      </a>
    </li>
    <li>
      <a href="/about">
        
        <span class="nav-links__label">0 нас</span>
      </a>
    </li>
  </ul>
</nav>
</body>
</html>

```

Значки
навигации;
каждый
содержит
изображение
и метку

Элемент `nav` занимает бо́льшую часть разметки. Он состоит из неупорядоченного списка (`ul`) ссылок. Каждая ссылка представляет собой значок и метку. Обратите внимание на то, что значки сохранены в формате *SVG*. Позже это окажется важным. Остальной контент на страницу вы добавите позже, чтобы в следующей главе создать для него стили.



SVG — сокращение от **Scalable Vector Graphics** (масштабируемая векторная графика). Это формат изображения, основанный на XML, который определяет изображение с помощью векторов. А так как изображение определяется математически, его можно сжимать и растягивать до любых размеров. Формат SVG поддерживают все браузеры.

Далее вы добавите несколько базовых стилей, включая фоновый градиент и отступы в заголовке. А также используйте на странице веб-шрифты. Скопируйте или

перепишите в свою таблицу стилей код (листинг 15.5). Здесь только базовые стили и шапка сайта, разметку меню создадите позже.

Листинг 15.5. Базовые стили и заголовок

```
html {
  box-sizing: border-box;
}
*,
*::before,
*::after {
  box-sizing: inherit;
}

body {
  background-color: hsl(200, 80%, 30%);
  background-image: radial-gradient(hsl(200, 80%, 30%),
                                   hsl(210, 80%, 20%));
  color: white;
  font-family: Exo 2, Helvetica, Arial, sans-serif;
  line-height: 1.4;
  margin: 0;
  min-height: 100vh;
}
h1, h2, h3 {
  font-family: Russo One, serif;
  font-weight: 400;
}

main {
  display: block;
}

img {
  max-width: 100%;
}

.page-header {
  margin: 0;
  padding: 1rem;
}

@media (min-width: 30em) {
  .page-header {
    padding: 2rem 2rem 3rem;
  }
}
```

Темно-голубой
фоновый градиент

Убеждаемся, что body охватывает
всю область просмотра,
чтобы градиент заполнил весь экран

Для мобильных устройств
меньшее значение отступов заголовка

Для больших экранов
большее значение
отступов заголовка

В этом примере реализованы несколько идей, описанных в предыдущих главах. Для заливки фона я использовал радиальный градиент. Он добавляет странице некоторую глубину (свойство `background-color` необходимо как альтернативный

вариант для браузера Opera Mini, который не поддерживает радиальные градиенты). Веб-шрифт Russo One применен в заголовках, а Echo 2 — для основного текста сайта. Я также создал адаптивные стили для шапки страницы, которые задают более крупные отступы для больших экранов.

Меню настроим в несколько этапов. Сначала зададим нужное расположение, а затем добавим интерактивное поведение. Для начала выполним все намеченное для мобильных устройств (см. главу 8). Заголовок и меню должны выглядеть как на рис. 15.6.

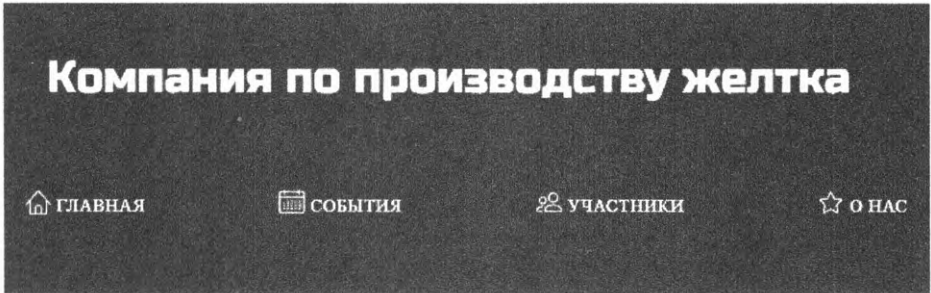


Рис. 15.6. Мобильный дизайн меню навигации

Поскольку для маленьких областей просмотра мы задаем горизонтальное расположение, имеет смысл заняться flexbox-версткой. Вы можете даже задать интервалы между элементами по ширине страницы, используя свойство `align-content: space-between` flex-контейнера. Затем определите цвета шрифта и выравнивание значков. Добавьте в таблицу стилей следующий код (листинг 15.6).

Листинг 15.6. Стили меню навигации для мобильных устройств

```
.nav-links {
  display: flex;
  justify-content: space-between;
  margin-top: 0;
  margin-bottom: 1rem;
  padding: 0 1rem;
  list-style: none;
}
.nav-links > li + li {
  margin-left: 0.8em;
}
.nav-links > li > a {
  display: block;
  padding: 0.8em 0;
  color: white;
  font-size: 0.8rem;
  text-decoration: none;
  text-transform: uppercase;
  letter-spacing: 0.06em;
}
```

Flexbox-верстка для распределения элементов навигации по ширине экрана

Стили текстовых ссылок

```

}
.nav-links__icon {
  height: 1.5em;
  width: 1.5em;
  vertical-align: -0.2em;
}
.nav-links > li > a:hover {
  color: hsl(40, 100%, 70%);
}

```

Небольшое смещение значков и меток вниз к центру

Так меню будет выглядеть в маленьких окнах. Для больших экранов можете применить к нему больше эффектов. Для экранов компьютеров и планшетов мы прикрепим меню навигации к левой стороне, используя фиксированное позиционирование. Это будет выглядеть так, как показано на рис. 15.7.

Меню состоит из двух модулей: внешний элемент я назвал `main-nav`, а внутренний — `nav-links`. Первый служит контейнером, который вы помещаете слева. Он также задает темный фон. Давайте реализуем это.

Добавьте в свою таблицу стилей следующий код (листинг 15.7). Убедитесь, что второе правило `@media` находится *после* созданных стилей `main-nav`, чтобы при необходимости они могли заместить мобильные стили.



Рис. 15.7. Меню навигации для больших областей просмотра прикреплено к левому краю

Листинг 15.7. Позиционирование меню для больших областей просмотра

```

@media (min-width: 30em) {
  .main-nav {
    position: fixed;
    left: 0;
    top: 8.25rem;
    z-index: 10;
    background-color: transparent;
    transition: background-color .5s linear;
    border-top-right-radius: 0.5em;
    border-bottom-right-radius: 0.5em;
  }

  .main-nav:hover {
    background-color: rgba(0, 0, 0, 0.6);
  }
}

/* ... */

@media (min-width: 30em) {
  .nav-links {
    display: block;
  }
}

```

Стили для средних и больших экранов

Меню навигации отображается поверх любого другого контента, добавляемого на страницу позднее

Изначально задается прозрачный цвет фона

Добавляется эффект перехода фону

Добавляется темный полупрозрачный фон при наведении указателя мыши

Замещаются мобильные стили `flexbox`, чтобы расположить значки вертикально

```
padding: 1em;
margin-bottom: 0;
}
.nav-links > li + li {
margin-left: 0;
}
.nav-links__label {
margin-left: 1em;
}
}
```

Свойство `position: fixed` помещает меню в нужную позицию и удерживает его там, даже когда страница прокручивается вниз. Правило `display: block` замещает свойство `display: flex` мобильных стилей, располагая значки меню друг над другом.

А теперь начнем создавать переходы и трансформации.

1. Обеспечим небольшое увеличение значков при наведении на них указателя мыши.
2. Спрячем метки ссылок — они будут плавно появляться, когда пользователь будет наводить на них указатель мыши.
3. Добавим смещение для создания эффекта выплывания совместно с эффектом плавного появления меток.

Итак, начнем.

15.2.1. Масштабирование значков

Посмотрите на структуру ссылок в меню навигации. Все пункты списка содержат ссылку (а), каждая из которых, в свою очередь, содержит значок и метку:

```
<li>
  <a href="/">
    
    <span class="nav-links__label">Главная</span>
  </a>
</li>
```

ПРИМЕЧАНИЕ

Элементы списка вместе с родительским элементом `ul` намного больше и глубже, чем вложенные модули, которые я обычно использую в своих проектах. Я бы попытался найти способ разбить его на более мелкие модули, но для наших целей лучше, чтобы все они были вместе.

Для начала сделаем так, чтобы значок увеличивался при наведении на него указателя мыши. Этого можно добиться масштабированием, добавив переход, чтобы сделать увеличение плавным. На рис. 15.8 представлен значок **События**, на который наведен указатель мыши.



Рис. 15.8. Размер значка по умолчанию (слева). Наведение указателя мыши вызывает увеличение значка (справа)

У значка *События* фиксированные ширина и высота, и вы можете увеличить его, изменяя данные параметры. Но это приведет к тому, что другие элементы сдвинутся, что способно нарушить разметку страницы.

Вместо этого применим трансформацию, которая никак не влияет на окружающие элементы, да и сам значок *События* при этом не смещается вправо. Отредактируйте CSS-код, чтоб добавить эффект увеличения как при наведении указателя мыши, так и при нажатии (листинг 15.8).

Листинг 15.8. Увеличение значка при наведении указателя мыши или нажатии

```
@media (min-width: 30em) {
  .nav-links {
    display: block;
    padding: 1em;
    margin-bottom: 0;
  }
  .nav-links > li + li {
    margin-left: 0;
  }
  .nav-links__label {
    margin-left: 1em;
  }
  .nav-links__icon {
    transition: transform 0.2s ease-out;
  }
  .nav-links a:hover > .nav-links__icon,
  .nav-links a:focus > .nav-links__icon {
    transform: scale(1.3);
  }
}
```

← Переход для свойства трансформации

← Увеличение размера значка

Сейчас, если вы наведете указатель мыши на значок, вы увидите, что он слегка увеличится, показывая, с каким именно элементом вы контактируете. Я намеренно использовал изображения в формате SVG, чтобы избежать пикселизации и других неприятных эффектов. Масштабирование — идеальный вид трансформации в данном случае.

SVG: лучше всего для значков

Значки — важная часть любого дизайна. Приемы, которые использовались для вывода значков, прошли долгий путь развития. Одно время наилучшим считалось поместить все значки в единый файл изображения — так называемый *лист спрайтов*. Затем, задействуя свойства фонового изображения в CSS, размер и позиционирование, можно было вырезать определенный значок (спрайт) и отобразить его в нужном элементе.

Затем стали популярны *текстовые значки*. Вместо того чтобы вставлять значки из листов спрайтов, создавали специальный файл шрифта. В этом пользовательском веб-шрифте каждый символ отображался как значок. Например, сервис Font-Awesome (fontawesome.io) предоставляет сотни таких пользовательских шрифтов со значками, которые очень легко использовать в проектах.

Эти приемы все еще работают, но я призываю задействовать SVG-формат. Такие значки намного более универсальны и производительны, чем все остальные. Вы можете указывать их в качестве источника в элементах `img`, как мы делали в данной главе. Есть и другие способы. Всегда можно создать SVG-лист спрайтов или, так как формат SVG основывается на XML, напрямую внедрять такие значки в HTML-код, например:

```
<li>
  <a href="/">
    <svg class="nav-links__icon" width="20" height="20" viewBox="0 0 20 20">
      <path fill="#ffffff" d="M19.871 12.165l-8.829-9.758c-0.274-0.303-
        0.644-0.47-1.042-0.47 0 0 0 0 0-0.397 0-0.767 0.167-1.042
        0.471-8.829 9.758c-0.185 0.205-0.169 0.521 0.035 0.706 0.096
        0.087 0.216 0.129 0.335 0.129 0.136 0 0.272-0.055 0.371-
        0.165l2.129-2.353v8.018c0 0.827 0.673 1.5 1.5 1.5h11c0.827 0 1.5-
        0.673 1.5-1.5v-8.018l2.129 2.353c0.185 0.205 0.501 0.221 0.706
        0.035s0.221-0.501 0.035-0.706zM12 19h-4v-4.5c0-0.276 0.224-0.5
        0.5-0.5h3c0.276 0 0.5 0.224 0.5 0.5v4.5zM16 18.5c0 0.276 0.224
        0.5-0.5 0.5h-2.5v-4.5c0-0.827-0.673-1.5-1.5-1.5h-3c-0.827 0-1.5
        0.673-1.5 1.5v4.5h-2.5c-0.276 0-0.5-0.224-0.5-0.5v-9.123l5.7-
        6.3c0.082-0.091 0.189-0.141 0.3-0.141s0.218 0.050 0.3 0.141l5.7
        6.3v9.123z"/></path>
    </svg>
    <span class="nav-links__label">Home</span>
  </a>
</li>
```

Этот способ позволит вам обращаться к изображению напрямую из CSS. Можно динамически изменять цвета и даже размер и положение разных частей значка. С помощью

обычного CSS-кода! Кроме того, размер их файлов меньше, и у них нет артефактов масштабирования, как у GIF, PNG и других растровых изображений.

Если вы не знакомы форматом SVG, вот отличный пример того, как использовать его в веб-страницах: css-tricks.com/using-svg/.

Сейчас, когда значки выглядят превосходно, поговорим о метках.

15.2.2. Создание «вылетающих» меток

Метки под значками не обязательно должны быть видны постоянно. По умолчанию их можно спрятать, оставив только значки, показывающие, где находится меню. А когда пользователь наведет указатель мыши на меню или нажмет клавишу **Tab**, метки появятся. Таким образом, когда указатель находится возле меню, оно возникает со всеми эффектами: темный фон и плавно появляющиеся метки, которые смещаются с исходной позиции, расположенной чуть левее (рис. 15.9).

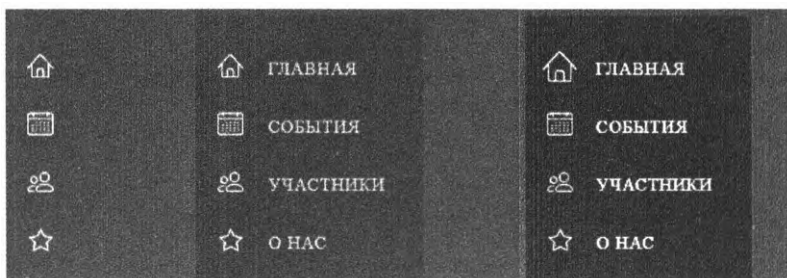


Рис. 15.9. При наведении указателя мыши меню плавно появляется и метки выезжают слева

Данный эффект требует двух переходов одновременно: одного для прозрачности, второго — для трансформации `translate()`. Внесите изменения, указанные в листинге 15.9, в код таблицы стилей.

Листинг 15.9. Переходы для меток `nav-item`

```
@media (min-width: 30em) {
  .nav-links {
    display: block;
    padding: 1em;
    margin-bottom: 0;
  }
  .nav-links > li + li {
    margin-left: 0;
  }
  .nav-links__label {
```

```

display: inline-block;
margin-left: 1em;
padding-right: 1em;
opacity: 0;
transform: translate(-1em);
transition: transform 0.4s cubic-bezier(0.2, 0.9, 0.3, 1.3),
            opacity 0.4s linear;
}
.nav-links:hover .nav-links__label,
.nav-links a:focus > .nav-links__label {
  opacity: 1;
  transform: translate(0);
}

.nav-links__icon {
  transition: transform 0.2s ease-out;
}

.nav-links a:hover > .nav-links__icon,
.nav-links a:focus > .nav-links__icon {
  transform: scale(1.3);
}

```

Скрытие меток

Смещение меток на 1 em влево

Метки становятся строчно-блочными, чтобы к ним можно было применить переход

Добавление переходов к значениям, которые будут изменяться

При наведении указателя мыши метки отображаются и смещаются в нужное положение

Это меню — лишь малая часть работы над обеспечением нужного визуального восприятия страницы, сделать предстоит еще многое. Некоторые из приведенных селекторов действительно длинные и сложные.

Обратите внимание на то, что псевдокласс `:hover` находится на верхнем уровне элемента `nav-links`, в то время как псевдокласс `:focus` — внутри элемента `a` (он обычно применяется только к ссылкам и кнопкам). Таким образом, при наведении указателя мыши все метки появляются одновременно с меню. Но если пользователь перемещается с помощью клавиши `Tab`, метки появляются по одной.

Когда метки скрыты, они смещены влево на 1 em с помощью функции `translate()`. Но когда появляются, они возвращаются на начальную позицию. Я опустил второй параметр для `translate()` и задал только смещение по оси *X*, то есть горизонтальное. Так как нам не надо сдвигать элемент вверх и вниз, используем этот способ.

Стоит рассмотреть и пользовательскую функцию `cubic-bezier()`. Она создает легкий эффект «выпрыгивания»: метки сдвигаются чуть правее, чем должны, а затем занимают нужное положение. Кривая движения представлена на рис. 15.10.

Обратите внимание на то, что кривая выходит за верхний край блока. Это говорит о том, что значение в данный момент выше конечного значения. При переходе от `translate(-1em)` к `translate(0)` метки моментально достигают значения примерно 0,15 em, а затем возвращаются в положение 0. Вы можете задать «выпрыгивание» и в начале любой временной функции, вытягивая кривую за нижний край. Но выполнить данный эффект и в начале, и в конце нельзя. Это даст алогичную кривую.

Откройте браузер и посмотрите, как выглядит данный переход. «Выпрыгивание» едва заметно, возможно, вы захотите замедлить переход, чтобы хорошо рассмотреть эффект. Он добавляет тяжести и инерции надписи, что сделает движение более естественным.

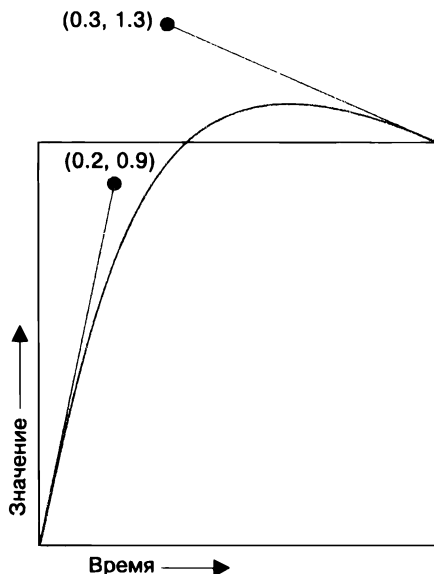


Рис. 15.10. Кривая Безье с «выпрыгиванием» в конце

15.2.3. Поэтапные переходы

Уже сейчас меню выглядит очень здорово. Но, чтобы довести его до совершенства, добавим еще одну деталь. Используем свойство `transition-delay`, чтобы задать разные задержки для каждого элемента. Это заставит их появляться не одновременно, а один за другим, подобно волне (рис. 15.11).

Для воплощения этого приема в жизнь понадобится псевдокласс `:nth-child()`, чтобы обратиться к каждому элементу списка в соответствии с его позицией, а затем задать более длительную задержку для каждого очередного. Добавьте в свою таблицу стилей, сразу после `nav-links`, следующий код (листинг 15.10).

Селектор `:nth-child(2)` обращается ко второму элементу списка, к нему применена небольшая задержка. Для третьего элемента (`:nth-child(3)`) задана чуть более продолжительная задержка. Четвертая и пятая — еще длиннее. К первому элементу это свойство применять не нужно, так как подразумевается, что он появляется сразу.

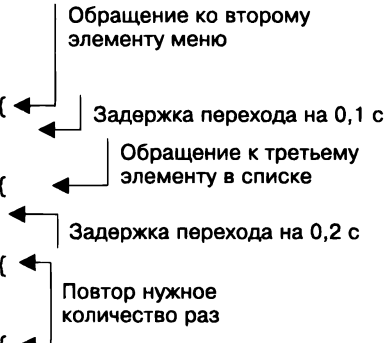


Рис. 15.11. Верхние элементы меню станут возникать раньше нижних

Откройте страницу в браузере и наведите указатель мыши на меню, чтобы увидеть эффект. Создается ощущение, что меню живое. Уберите указатель — оно медленно, элемент за элементом, исчезнет.

Листинг 15.10. Добавление ступенчатой задержки для элементов списка меню

```
.nav-links:hover .nav-links__label,
.nav-links a:focus > .nav-links__label {
  opacity: 1;
  transform: translate(0);
}
.nav-links > li:nth-child(2) .nav-links__label {
  transition-delay: 0.1s;
}
.nav-links > li:nth-child(3) .nav-links__label {
  transition-delay: 0.2s;
}
.nav-links > li:nth-child(4) .nav-links__label {
  transition-delay: 0.3s;
}
.nav-links > li:nth-child(5) .nav-links__label {
  transition-delay: 0.4s;
}
```



Обращение ко второму элементу меню

Задержка перехода на 0,1 с

Обращение к третьему элементу в списке

Задержка перехода на 0,2 с

Повтор нужное количество раз

Однако у данного подхода есть недостаток: количество пунктов меню ограничивается количеством селекторов, которые вы пропишете. Я создал правило и для пятого пункта меню, хотя сейчас их всего четыре. Это просто перестраховка на случай, если в будущем появится еще один элемент. Можно даже добавить шестой, чтобы уж наверняка. Но всегда помните: в какой-то момент число элементов в меню может превзойти количество созданных правил, и тогда придется редактировать CSS-код.

СОВЕТ

Повторяющиеся блоки кода наподобие этого проще создавать с помощью препроцессора. См. приложение Б, чтобы получить более подробную информацию.

Итак, меню готово, но на страницу можно добавить кое-что еще. Мы это сделаем в следующей главе, поэтому не прячьте файлы далеко. Но перед этим рассмотрим еще пару моментов, относящихся к трансформациям, о которых нужно знать.

15.3. Производительность анимации

Существование некоторых форм трансформации способно показаться излишним. Результат смещения можно задать с помощью относительного позиционирования, а в случае изображений или SVG результат масштабирования совпадает с явным заданием высоты и ширины.

Трансформации намного менее требовательны к ресурсам. Если вы анимируете позиционирование элемента, например смещение влево, то, вероятно, заметите

значительное снижение производительности. Особенно если работаете с большим сложным элементом или множеством элементов одновременно. Это характерно как для переходов (описаны в главе 14), так и для анимации (рассмотрим в главе 16).

При использовании любого вида перехода или анимации стоит отдать предпочтение трансформации перед явным заданием позиции и размера. Чтобы понять, почему так, подробнее разберем, как этот процесс происходит в браузере.

15.3.1. Рендеринг страницы

После того как браузер понял, какие стили нужно применять к каким элементам, ему предстоит преобразовать эти стили в пиксели на экране. Этот процесс называется *рендерингом (визуализацией)*. Его можно разбить на три стадии: разметка, окрашивание и компоновка (рис. 15.12).

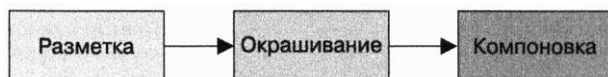


Рис. 15.12. Три стадии рендеринга

Разметка

На первой стадии, в ходе *разметки*, браузер вычисляет, сколько места будет занимать каждый элемент. Из-за того, как работает позиционирование элементов, размер и положение одного могут повлиять на множество других элементов на странице. За это отвечает стадия разметки.

Каждый раз, когда вы изменяете ширину или высоту элемента или его расположение (выравнивание, скажем, по верхнему или левому краю), приходится заново пересчитывать всю разметку. То же самое происходит, если элемент создается или удаляется с помощью DOM JavaScript. Когда разметка изменяется, браузер вынужден перекраивать страницу, заново вычисляя положение всех элементов, которые в результате сдвигаются или размер которых меняется.

Окрашивание

После разметки наступает стадия *окрашивания*. Это процесс заполнения пикселей: отрисовываются текст и изображения, приобретают цвет границы и тени. Но все это не физически отображается на странице, а скорее сохраняется в памяти. Части изображения раскрашиваются *по слоям*.

Если вы измените, например, цвет фона, элемент должен быть перерисован. Но так как цвет не влияет ни на его размер, ни на положение на странице, это не влечет за собой пересчет разметки. Изменение цвета требует меньших вычислительных мощностей, чем изменение размера.

В идеале каждый элемент на странице помещается на собственный слой. При этом он отрисовывается независимо от других слоев на странице. Браузер управляет эти слои в GPU (графический процессор) на рендеринг, а не обрабатывает их в центральном процессоре (CPU), как это делается для главного слоя. Это дает преимущество, так как GPU лучше оптимизирован для операций такого рода.

Данный процесс часто называют *аппаратным ускорением*, поскольку он основан на использовании аппаратного обеспечения и значительно ускоряет процесс рендеринга. Большее количество слоев означает задействование большего количества памяти, в то же время это значительно сокращает время, затраченное на рендеринг.

Компоновка

На стадии *компоновки* браузер собирает все отрисованные слои и создает единое изображение, которое и отобразится на экране. Они собираются в определенном порядке, чтобы одни слои отображались перед другими, как нужно там, где они накладываются.

Использование измененных свойств `opacity` и `transform` значительно сокращает время рендеринга. Когда вы применяете одно из них к элементу, браузер помещает его на отдельный цветовой слой и может задействовать аппаратное ускорение. А так как элемент находится на отдельном слое, при анимации основной слой остается неизменным, значит, не требует повторных вычислений.

Если вы вносите на страницу единственное изменение, разница в оптимизации, вероятно, будет почти незаметной. Но если используете анимацию, экран обновляется десятки раз в секунду, и в этом случае скорость имеет значение. Большинство экранов обновляются с частотой 60 раз в секунду. В идеале изменения при анимации должны вычисляться с такой же или большей частотой, чтобы сделать движение плавным. И чем больше вычислений приходится делать браузеру, тем сложнее достичь такой скорости.

Контроль слоев с помощью свойства `will-change`

Браузеры прошли долгий путь оптимизации процесса рендеринга и максимального разделения элементов по слоям. Если вы применяете свойство `opacity` или `transform` для создания анимации, современные браузеры способны выбрать нужные алгоритмы, основываясь на ряде факторов, таких как системные ресурсы, чтобы сделать анимацию плавной. Хотя иногда все же можно увидеть мерцающую или дергающуюся анимацию.

Если вы столкнулись с подобным, то используйте свойство `will-change`, которое позволяет контролировать слои. Оно заранее сообщает браузеру, каких изменений элемента стоит ожидать. Обычно это говорит о том, что элемент будет помещен на отдельный слой. Например, применение `will-change: transform` означает, что для него ожидается изменение свойства `transform`.

Однако не стоит задействовать это свойство бездумно по всей странице до тех пор, пока вы реально не столкнетесь с проблемами с производительностью, так как оно требует дополнительных системных ресурсов. Протестируйте производительность до и после использования `will-change` и оставьте его, только если ощущаете реальный прирост скорости. Чтобы подробнее рассмотреть, как работает это свойство и стоит ли с ним работать, прочитайте потрясающую статью Сары Сюдан (Sara Soueidan): dev.opera.com/articles/css-will-change-property/.

Но считаю нужным отметить, что с момента выхода статьи один момент все же изменился: там говорится, что только 3D-трансформации создают отдельный слой для элемента. Это давно не так. Современные браузеры задействуют аппаратное ускорение и для 2D-трансформаций.

При создании переходов или анимации, которую мы рассмотрим в следующей главе, старайтесь работать только со свойствами `opacity` или `transform`. Тогда изменения коснутся только формирования слоев, а не разметки в целом. Изменяйте свойства, которые затрагивают разметку, только если нет другого выхода, и всегда думайте о возможных проблемах с производительностью. Чтобы лучше понять, какие свойства относятся к какой стадии — разметке, цвету или компоновке, — прочитайте материал на сайте csstriggers.com.

15.4. Трехмерные (3D) трансформации

Все трансформации, которые мы рассматривали прежде, были двумерными. Они наиболее широко распространены, так как сама страница двумерная и работать с ними наиболее просто. Но вы вовсе не ограничены этим. Вращения и перемещения могут производиться во всех трех измерениях — X , Y и Z .

Можно использовать функцию `translate()`, как вы уже видели, для горизонтального и вертикального (измерения X и Y) смещения. То же самое выполняется с помощью отдельных функций `translateX()` и `translateY()`. Следующие две строки кода дадут одинаковый результат:

```
transform: translate(15px, 50px);
transform: translateX(15px) translateY(50px);
```

Вы также можете выполнить смещение по оси Z с помощью функции `translateZ()`, которая перемещает элемент ближе к пользователю или дальше от него. Таким же образом вращайте объект вокруг трех осей. Функция `rotateZ()` эквивалентна просто `rotate()`, так как выполняет вращение вокруг оси Z . Функции `rotateX()` и `rotateY()` вращают элемент вокруг горизонтальной оси X (наклоняя элемент вперед или назад) и вертикальной оси Y (поворачивая — или *вращая* — элемент вправо и влево) соответственно. Рисунок 15.13 иллюстрирует эти примеры.

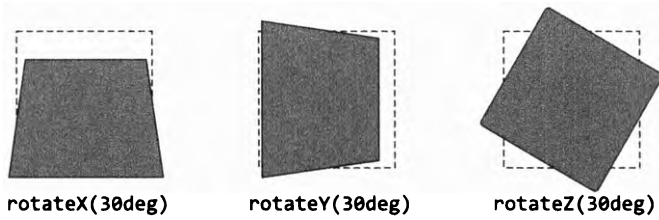


Рис. 15.13. Вращение по каждой из трех осей на 300 пикселей (пунктирной линией обозначено начальное положение элемента)

15.4.1. Контроль перспективы

Прежде чем добавлять 3D-эффект на страницу, рассмотрим важную вещь — *перспективу*. Трансформированные элементы вместе образуют 3D-сцену. Браузер вычисляет двумерную картинку для этой 3D-сцены и отображает ее на экране. Вы можете рассматривать перспективу как расстояние между камерой и сценой. Перемещение камеры влияет на то, как выглядит итоговое изображение.

Если камера ближе, то есть перспектива небольшая, то 3D-эффекты намного сильнее. Если камера далеко (перспектива больше), этот эффект различим хуже. Несколько вариантов перспективы представлены на рис. 15.14.

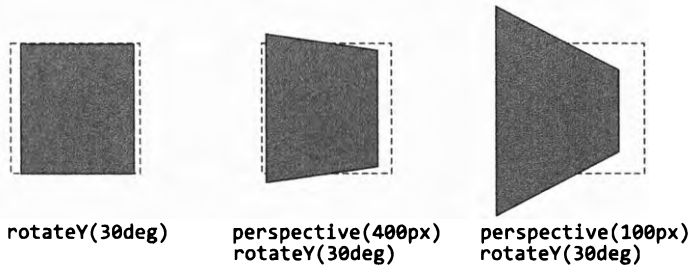


Рис. 15.14. Один и тот же угол поворота при разных значениях перспективы

Развернутый элемент слева, без перспективы, вовсе не выглядит как 3D. Кажется, что его просто сжали по вертикали, нет ощущения глубины. 3D-трансформации без перспективы выглядят плоскими, такими же, как эта: части элемента, которые находятся дальше, не уменьшаются. Значение перспективы на центральном изображении — 400 пикселей. Правый край — тот, который дальше от пользователя, — выглядит меньше, а ближний — больше. Перспектива справа намного короче — 100 пикселей. Это увеличивает эффект, так что отдаляющийся край значительно меньше ближнего.

Можете задать значение перспективы двумя способами: используя трансформацию `perspective()` или же свойство `perspective`. Их поведение немного различается. Рассмотрим пример, чтобы понять, в чем заключается разница. Он будет примитивным, просто чтобы показать эффект перспективы.

Для начала создайте четыре элемента и поверните их, отклонив назад с помощью `rotateX()` (рис. 15.15). Все элементы повернуты одинаково и имеют одно и то же значение функции трансформации `perspective()`. То есть на странице все они выглядят одинаково.

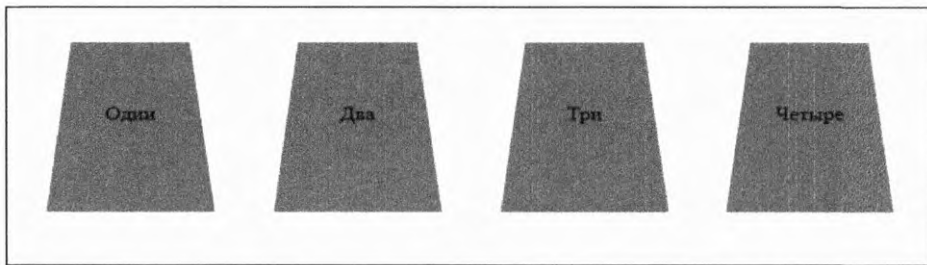


Рис. 15.15. Четыре элемента, повернутые по оси X, для каждого задана функция трансформации perspective(200px)

Создайте новую страницу для этого примера, скопируйте HTML-код (листинг 15.11).

Листинг 15.11. Четыре блока для иллюстрации 3D-трансформаций и перспективы

```
<div class="row">
  <div class="box">Один</div>
  <div class="box">Два</div>
  <div class="box">Три</div>
  <div class="box">Четыре</div>
</div>
```

А теперь примените ко всем блокам 3D-трансформацию и перспективу. Также добавьте цвет и отступы, чтобы сделать эффект более наглядным. Скопируйте код из листинга 15.12 в свою таблицу стилей.

Листинг 15.12. Применение 3D-трансформации к блокам

```
.row {
  display: flex;
  justify-content: center;
}

.box {
  box-sizing: border-box;
  width: 150px;
  margin: 0 2em;
  padding: 60px 0;
  text-align: center;
  background-color: hsl(150, 50%, 40%);
  transform: perspective(200px) rotateX(30deg);
}
```

Поворот блока на 30°
и добавление перспективы

В этом примере все блоки выглядят одинаково. Каждый имеет собственную перспективу, заданную функцией трансформации `perspective()`. Данный метод применяет перспективу к одному элементу. В этом примере мы ее задействовали напрямую для каждого блока. Это все равно, что сделать для каждого блока четыре разные фотографии с одного ракурса.

Иногда вам может понадобиться создать общую перспективу для нескольких объектов, будто они находятся в едином 3D-пространстве (рис. 15.16). Это те же самые четыре блока, но все они стремятся в единую точку. Это все равно что сделать общую фотографию сразу четырех объектов. Чтобы добиться такого эффекта, необходимо задействовать свойство `perspective` для родительского элемента.

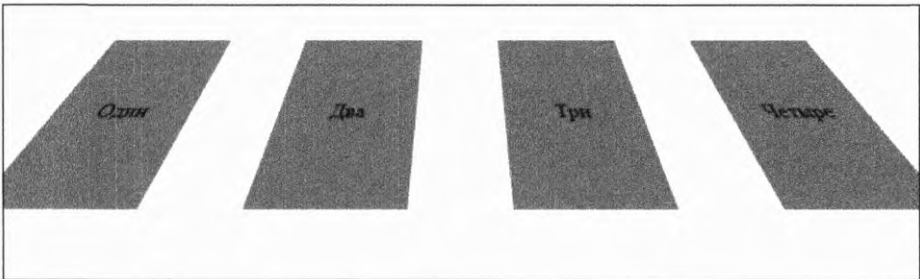


Рис. 15.16. Использование общей перспективы для нескольких объектов с помощью свойства `perspective` родительского элемента

Необходимо убрать функцию `perspective()` для каждого блока и задать свойство `perspective` для их контейнера. Изменения показаны в следующем коде (листинг 15.13).

Листинг 15.13. Создание общей перспективы

```
.row {  
  display: flex;  
  justify-content: center;  
  perspective: 200px;  
}  
  
.box {  
  box-sizing: border-box;  
  width: 150px;  
  margin: 0 2em;  
  padding: 60px 0;  
  text-align: center;  
  background-color: hsl(150, 50%, 40%);  
  transform: rotateX(30deg);  
}
```

Добавление перспективы
к контейнеру

Не применяйте
функцию
трансформирования
для блоков

Когда будет задана общая перспектива для родительского или другого общего элемента, все элементы, которые он содержит, приобретут этот же эффект 3D-трансформации.

Создание перспективы — важная часть 3D-трансформации. Без нее отдаленные от наблюдателя элементы не будут казаться меньше, а ближние — больше. Этот пример довольно прост. В главе 16 рассмотрим те же техники для создания анимации вылета элементов на страницу из глубины.

15.4.2. Профессиональные приемы 3D-трансформации

Есть еще несколько свойств, которые могут оказаться полезными при создании трехмерности. Я не стану уделять им слишком много внимания, приведу лишь несколько примеров из практики. Вам будет полезно знать об их существовании, даже если раньше они не были нужны. А еще дам ссылки на несколько статей, которые стоит прочесть, если вы захотите изучить эту тему глубже.

Точка перспективы

По умолчанию перспектива отображается так, как если бы наблюдатель (или камера) находился прямо по центру элемента. Свойство `perspective-origin` смещает камеру вправо или влево, вверх или вниз. На рис. 15.17 представлен предыдущий пример, но камера смещена вниз и влево.

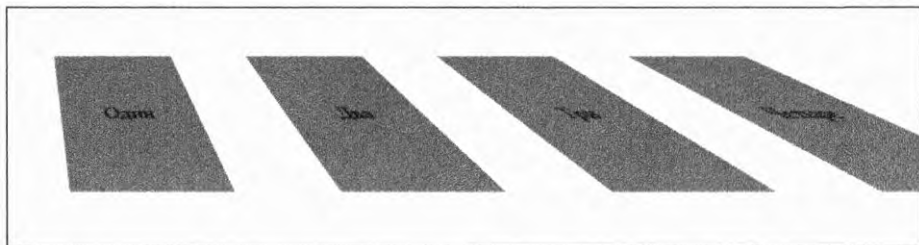


Рис. 15.17. Смещение точки перспективы увеличивает искажение элементов ближе к краям

Добавьте код из листинга 15.14, чтобы увидеть этот эффект у себя на странице.

Листинг 15.14. Использование `perspective-origin` для изменения положения камеры

```
.row {
  display: flex;
  justify-content: center;
  perspective: 200px;
  perspective-origin: left bottom;
}
```

← Смещение камеры в нижний левый угол элемента

Здесь точно такое же расстояние перспективы, как и в предыдущем примере, но точка перспективы смещена так, будто все блоки находятся справа от наблюдателя.

Вы можете задать положение с помощью ключевых слов `top`, `left`, `bottom`, `right` или `center`, а также использовать процентные значения или любые единицы измерения. Отсчет начинается с верхнего левого угла (например, `perspective-origin: 25% 25%`).

Свойство перевернутого изображения

Если задать для функций `rotateX()` или `rotateY()` значение свыше 90° , произойдет нечто интересное: изображение больше не обращено к вам лицевой стороной. Оно будет развернуто, и вы увидите его отраженным. Элемент на рис. 15.18 трансформирован с помощью функции `rotateY(180deg)`. Он выглядит словно зеркальное отражение исходного.

Это изнанка изображения. По умолчанию она видна, но это можно изменить, применив к элементу свойство `backface-visibility: hidden`. Таким образом, он будет виден, только если повернут к вам лицом.

Один из способов применения этого эффекта — сложить два изображения «спиной» друг к другу, как две стороны карточки. Лицевая сторона будет видна, а оборотная — нет. Затем вы можете заставить их контейнер повернуться так, что будет видна оборотная, а не лицевая сторона. Пример найдете здесь: desandro.github.io/3dtransforms/docs/card-flip.html.



Рис. 15.18. Разворот элемента, видна его оборотная сторона

Свойство transform-style (preserve-3D)

Свойство `transform-style` будет полезно, когда вы захотите создать сложную сцену с вложенными 3D-элементами. Представим, что вы применили перспективу к блоку, а затем задали 3D-трансформацию для содержащихся в нем элементов. Этот контейнер будет отображаться как 2D-представление сцены, подобно фотографии 3D-объекта. Выглядит неплохо, так как все равно экран плоский.

Но если после этого вы примените 3D-вращение к контейнеру, оно не будет отображаться так, как должно. Вместо этого оно станет воздействовать на сцену целиком. Это все равно что вращать 2D-фотографию 3D-объекта. Перспектива окажется неправильной, а со страницы пропадет эффект глубины (рис. 15.19).

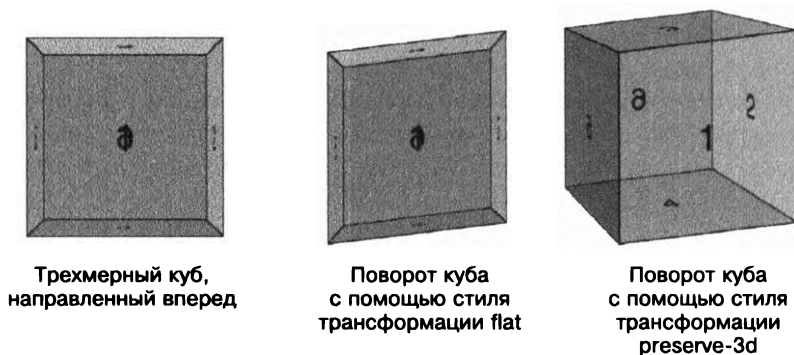


Рис. 15.19. Если применить 3D-трансформацию к родительскому элементу того элемента, который уже подвергнут 3D-трансформации, то лучше использовать значение `preserve-3d` (справа)

Сцена, отображенная слева, представляет 3D-куб, созданный трансформированием шести граней. Среднее изображение демонстрирует, что произойдет, если вы попытаетесь трансформировать весь куб (то есть родительский элемент). Чтобы это исправить, необходимо применить свойство `transform-style: preserve-3d` к родительскому элементу (справа).

ВНИМАНИЕ!

Значение `preserve-3d` поддерживается не всеми версиями Internet Explorer.

Более полная информация по этой теме, а также конкретные примеры имеются в уроках Анны Тюдор (Ana Tudor): davidwalsh.name/3d-transforms. С этими примерами весело экспериментировать, но мне не доводилось использовать `preserve-3d` в настоящих проектах. Если вы все же решите поэкспериментировать с 3D, чтобы просто посмотреть, что с его помощью можно создать, эти уроки могут оказаться полезными.

Итоги главы

- ❑ Используйте трансформацию, чтобы масштабировать, вращать или наклонять элементы в двух или трех плоскостях.
- ❑ Трансформации очень важны в переходах и анимации с точки зрения производительности.
- ❑ Поймите, как работает поток процессов рендеринга, и всегда учитывайте его при создании анимации.
- ❑ Задействуйте пользовательские функции, чтобы добавить эффект покачивания при переходе.

16

Анимация

В этой главе

- Добавление на страницу сложного движения с помощью ключевых кадров.
- Воспроизведение анимации при загрузке страницы.
- Использование вращающегося значка для обеспечения обратной связи.
- Привлечение внимания пользователя к кнопке сохранения.

В предыдущих двух главах вы создали несколько переходов, которые переводили элементы из одного состояния в другое. Это позволило добавить анимацию на страницу и сделать пользовательский опыт визуально более интересным. Но иногда простого перехода недостаточно.

Вместо простого перемещения из одного места в другое вам может понадобиться обеспечить движение элемента по определенной траектории или его возвращение в исходное положение. Это нельзя сделать с помощью перехода. Для реализации более точного контроля над изменениями на странице CSS предусматривает возможность создания анимации с помощью ключевых кадров.

Ключевой кадр (keyframe) представляет собой определенную точку анимации. Вы задаете некоторое количество ключевых кадров, а браузер выполняет интерполяцию, вставляя между ними дополнительные кадры (рис. 16.1).



Рис. 16.1. Вы задаете ключевые кадры, а браузер заполняет промежутки между ними дополнительными кадрами

Концептуально переход напоминает анимацию с помощью ключевых кадров: вы задаете первый (начальная точка) и последний (конечная точка) кадры, а браузер вычисляет все промежуточные значения, между которыми должны плавно переключаться элементы. Но при создании анимации с помощью ключевых кадров вы не ограничены определением лишь двух точек. Можете задать любое их количество.

Браузер будет заполнять промежутки между ними дополнительными значениями, создав ряд плавных переходов, пока не достигнет последнего ключевого кадра.

В этой главе я покажу, как создавать анимацию с помощью ключевых кадров. Вы добавите несколько кадров на страницу, которую начали создавать в главе 15, а затем изучите несколько других способов их использования. Анимация добавляется на страницу не только для того, чтобы сделать последнюю более привлекательной, она, что важно, может предоставлять пользователю обратную связь.

16.1. Ключевые кадры

В CSS анимация состоит из двух частей: правила `@keyframes`, которое определяет ее, и свойства `animation`, применяющего ее к элементу.

Создадим простую анимацию, чтобы познакомиться с синтаксисом. Она будет состоять из трех ключевых кадров (рис. 16.2). В первом кадре элемент красный. Во втором кадре он светло-голубой и сдвинут вправо на 100 пикселей. В последнем кадре он светло-фиолетовый и стоит в исходном положении.

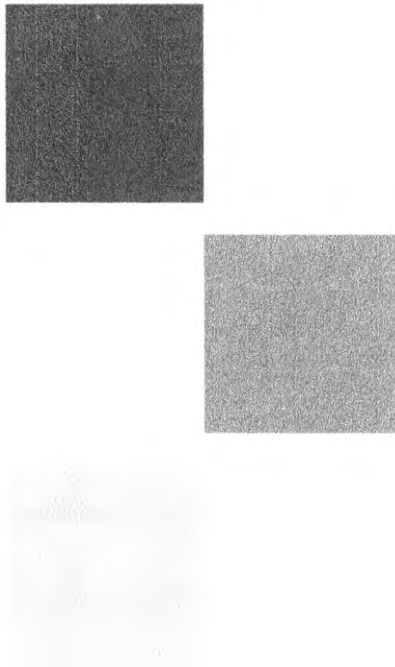


Рис. 16.2. Три ключевых кадра, обеспечивающих анимацию цвета и положения элемента

Эта анимация предполагает изменение двух свойств: `background-color` и `transform`. Правило `@keyframes` для этого приведено в листинге 16.1. Создайте таблицу стилей `styles.css` и добавьте в нее следующий код.

Листинг 16.1. Определение правила @keyframes

```

@keyframes over-and-back {
  0% {
    background-color: hsl(0, 50%, 50%);
    transform: translate(0);
  }

  50% {
    transform: translate(50px);
  }

  100% {
    background-color: hsl(270, 50%, 90%);
    transform: translate(0);
  }
}

```

Установка имени анимации

Установка первого ключевого кадра

Второй ключевой кадр находится в середине анимационной последовательности

Последний ключевой кадр

Анимация, основанная на ключевых кадрах, нуждается в имени. В этом примере создается анимация, названная *over-and-back*. Далее задаются три ключевых кадра с использованием процентных значений, которые указывают место каждого кадра в анимационной последовательности: один находится в начале (0 %), один в середине (50 %) и один в конце (100 %). Объявления внутри каждого из этих блоков определяют вид ключевого кадра.

В примере одновременно анимируются два свойства, однако обратите внимание на то, что они не указываются в каждом ключевом кадре. Изменение свойства *transform* сдвигает элемент с исходного положения вправо, а затем возвращает обратно. Однако свойство *background-color* не указано для ключевого кадра в положении 50 %. Это означает, что цвет элемента будет плавно меняться от красного (в точке 0 %) до светло-фиолетового (в точке 100 %). В точке 50 % элемент будет окрашен в цвет, находящийся точно посередине между этими двумя цветами.

Добавим на страницу код, чтобы посмотреть, как он работает. Создайте HTML-документ и добавьте следующий код (листинг 16.2).

Листинг 16.2. Страница с одним блочным элементом для анимации

```

<!doctype html>
<html lang="en">
<head>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="box"></div>
</body>
</html>

```

Элемент, который предстоит анимировать

Теперь добавьте стили в свою таблицу стилей, чтобы оформить с их помощью блок и применить анимацию. Скопируйте код из листинга 16.3.

Листинг 16.3. Применение анимации к блоку

```
.box {  
  width: 100px;           | Установка высоты и ширины  
  height: 100px;          | элемента в целях демонстрации  
  background-color: green;  
  animation: over-and-back 1.5s linear 3; ← Применение анимации к элементу  
}
```

Откройте страницу в своем браузере. Анимация должна повториться три раза, а затем остановиться. Свойство `animation` — сокращение для нескольких свойств. В этой демонстрации вы указали четыре из них:

- ☐ `animation-name (over-and-back)` — задает имя анимации, определяемое правилом `@keyframes`;
- ☐ `animation-duration (1.5s)` — задает длительность анимации, в данном случае 1,5 с;
- ☐ `animation-timing-function (linear)` — задает временную функцию, описывающую ускорение и/или замедление воспроизведения анимации. Ее значение может быть задано кривой Безье или ключевым словом (`ease-in`, `ease-out` и т. д.);
- ☐ `animation-iteration-count (3)` — задает количество повторений анимации. Если эта функция опущена, используется значение 1.

Перезагрузите страницу, чтобы снова просмотреть анимацию. Проанализируем пару нюансов ее поведения.

Во-первых, цвет изменяется плавно от красного в точке 0 % до светло-фиолетового в точке 100 %, но когда анимация начинает повторяться, цвет мгновенно становится красным. Если планируете повторить анимацию, убедитесь, что конечное значение совпадает с начальным, чтобы изменение было плавным.

Во-вторых, после последней итерации цвет фона меняется на зеленый — значение, указанное в основном наборе правил. Однако обратите внимание на то, что во время воспроизведения анимации это объявление переопределяется значениями, указанными в правиле `@keyframes`. С точки зрения принципа каскадности правила анимации имеют приоритет над остальными.

Как вы помните из главы 1 (см. подраздел 1.1.1), самым высоким приоритетом обладает таблица стилей. Авторские стили имеют более высокий приоритет по сравнению с браузерными. Однако приоритет определений, применяемых анимацией, еще более высок. Процесс анимации свойства сопровождается переопределением стилей, указанных в любом другом месте таблицы стилей, вне зависимости от специфичности селекторов. Это гарантирует согласованность анимации всех ключевых кадров независимо от других правил, примененных к элементу за пределами анимации.

ВНИМАНИЕ!

Анимация хорошо поддерживается браузерами, однако некоторые мобильные браузеры требуют использования префикса `-webkit-` как для свойства `animation` (`-webkit-animation`), так и для правила `@keyframes` (`@-webkit-keyframes`). Это вызывает необходимость дублирования всего кода как с префиксом, так и без него. Решить эту задачу можно с помощью инструмента Autoprefixer (см. врезку «Вендорные префиксы» в главе 5).

16.2. Анимация 3D-трансформаций

Далее вы добавите анимацию на страницу, которую начали создавать в предыдущей главе. После добавления кода из листинга 15.10 у вас должна получиться страница с синим фоном и навигационным меню слева. Остальная ее часть будет заполнена несколькими карточками, содержащими некий контент. Сначала вы создадите общую структуру макета, а затем добавите анимацию.

16.2.1. Создание макета без анимации

В этом примере вы добавите в основную область страницы несколько карточек (рис. 16.3). Затем создадите анимацию, чтобы заставить их влетать в область просмотра с помощью 3D-трансформаций.



Рис. 16.3. Дополнительные карточки для основной области страницы

Разметка для этого контента показана далее. Добавьте код из листинга 16.4 на страницу после элемента `nav`. (Я сократил текстовый контент карточек, чтобы укоротить листинг. Можете увеличить текст, если хотите сделать свою страницу более похожей на рис. 16.3.)

Листинг 16.4. Создание сетки flyin-grid и нескольких карточек

```

<main class="flyin-grid">
  <div class="flyin-grid_item card">
    
    <h4>Госпожа Кокина</h4>
    <p>
      Хотя госпожа Кокина и стара, но она выполняет
      свою работу. Ежедневно она откладывает большое
      белое яйцо. Ей нравятся пшено и овес.
    </p>
    <p>В породе «леггорн» лучшие несушки.</p>
  </div>
  <div class="grid_item card">
    
    <h4>Генка Соло</h4>
    <p>
      Прибывший последним, Генка Соло – фаворит среди
      наших цыплят. Это нахальный и подозрительный
      петушок. Мы часто должны бегать за ним, чтобы
      узнать, куда он отвел на этот раз своих подруг-несушек.
      А еще он будит всех по утрам.
    </p>
    <p>Куры этой делавэрской породы просто превосходные
      несушки – десяток снесенных ими крупных яиц на рынке
      стоит почти 100 рублей!</p>
  </div>
  <div class="grid_item card">
    
    <h4>Клювдия Норрис</h4>
    <p>
      В каждой стайке есть свой гадкий утенок. Клювдия Норрис –
      наша самая боевая подруга, часто воюющая с другими
      курицами за корм и территорию. Ее сестрички
      надеются, что Клювдия наконец остепенится и будет, как
      остальные куры, нести яйца. Тем более яйца у этих
      несушек очень вкусные.
    </p>
    <p>Куры породы «шантеклер» такие же крепкие и выносливые,
      как и их канадские предки, способные пережить самые
      сильные зимние морозы.</p>
  </div>
  <div class="grid_item card">
    
    <h4>Степанида Пегая</h4>
    <p>
      Степанида – наша первая и самая дружелюбная курица.
      Она либо встречает посетителей во дворе, либо сидит в курятнике.
      Хотя в этой породе не самые плодовые несушки, яйца они
      откладывают последовательно и регулярно. Вы постоянно будете
      находить ее яйца среди результатов работы остальных несушек.
    </p>
    <p>Эта курочка породы «кохинхин» откладывает светлые яйца.</p>
  </div>
</main>

```

← Контейнер сетки

← Карточки тоже являются элементами сетки

Эта часть страницы состоит из двух модулей. Внешний модуль, Flyin-Grid, обеспечивает компоновку элементов в сетке, включая 3D-эффект вылета, который я опишу далее. Каждый элемент сетки является также экземпляром внутреннего модуля Card. Данный модуль отвечает за оформление внешнего вида с помощью стилей: задание белого фона, отступов и цвета шрифта.

Этот макет — превосходный пример сетки, поэтому именно его вы и будете использовать. Следует подумать также о мобильном макете и резервном flexbox-макете для старых браузеров, которые не поддерживают сетки. Сначала вы разработаете макет для мобильного устройства, затем добавите стили для flex-контейнеров, а затем — стили для сетки.

Мобильный макет показан на рис. 16.4. На маленьких экранах карточки будут заполнять всю ширину экрана, оставляя небольшие поля слева и справа.



Рис. 16.4. В мобильном макете карточки будут заполнять всю ширину экрана, выстраиваясь друг за другом под меню

Добавьте следующие мобильные стили (листинг 16.5) в свою таблицу стилей.

При таком размере экрана о сетке можно не беспокоиться, поскольку ее элементы будут корректно организованы как обычные блочные элементы. Стили карточек задают их общий вид и белый фон. Затем макет будет усложнен с помощью медиа-запроса.

Листинг 16.5. Мобильные стили для карточек

```
.flyin-grid {
  margin: 0 1rem;
}

.card {
  margin-bottom: 1em;
  padding: 0.5em;
  background-color: white;
  color: hsl(210, 15%, 20%);
  box-shadow: 0.2em 0.5em 1em rgba(0, 0, 0, 0.3);
}

.card > img {
  width: 100%;
}
```

← Добавление небольших полей
слева и справа от контейнера

Установка цвета
и других свойств
карточки

← Заставляет изображение
заполнять всю ширину карточки

Создайте с помощью флекс-контейнеров резервный макет, применяемый только для больших контрольных точек. Это приблизит вас к окончательной версии макета (см. рис. 16.3). Добавьте следующий код CSS (листинг 16.6) в свою таблицу стилей.

Листинг 16.6. Создание резервного flexbox-макета

```
.flyin-grid {
  margin: 0 1rem;
}

@media (min-width: 30em) {
  .flyin-grid {
    display: flex;
    flex-wrap: wrap;
    margin: 0 5rem;
  }

  .flyin-grid_item {
    flex: 1 1 300px;
    margin-left: 0.5em;
    margin-right: 0.5em;
    max-width: 600px;
  }
}
```

← Адаптивная
контрольная точка

Создание флекс-контейнера
с обертыванием

← Увеличение
отступа по бокам

← Активизация свойства flex-grow
и установка значения 300 пикселей
для свойства flex-basis

Код в этом листинге создает адаптивный flexbox-макет. Свойство `flex-wrap: wrap` обеспечивает перенос флекс-элементов на следующую строку, когда они перестают уместиться на одной. Значение 300 пикселей для свойства `flex-basis` определяет минимальную ширину, а свойство `max-width` — максимальную. В соответствии

с этими ограничениями элементы будут переноситься на следующую строку по мере необходимости. Значение 1 свойства **flex-grow** позволяет карточкам растягиваться, заполняя все пространство.

Модуль Card не требует каких-либо изменений сверх уже добавленных мобильных стилей, все цвета и стилистические элементы сохраняются.

На экранах определенных размеров отображение карточек точно соответствует готовому макету. Но когда в последней строке их меньше по сравнению с предыдущими, карточки могут быть разной ширины. Эта проблема проиллюстрирована на рис. 16.5.

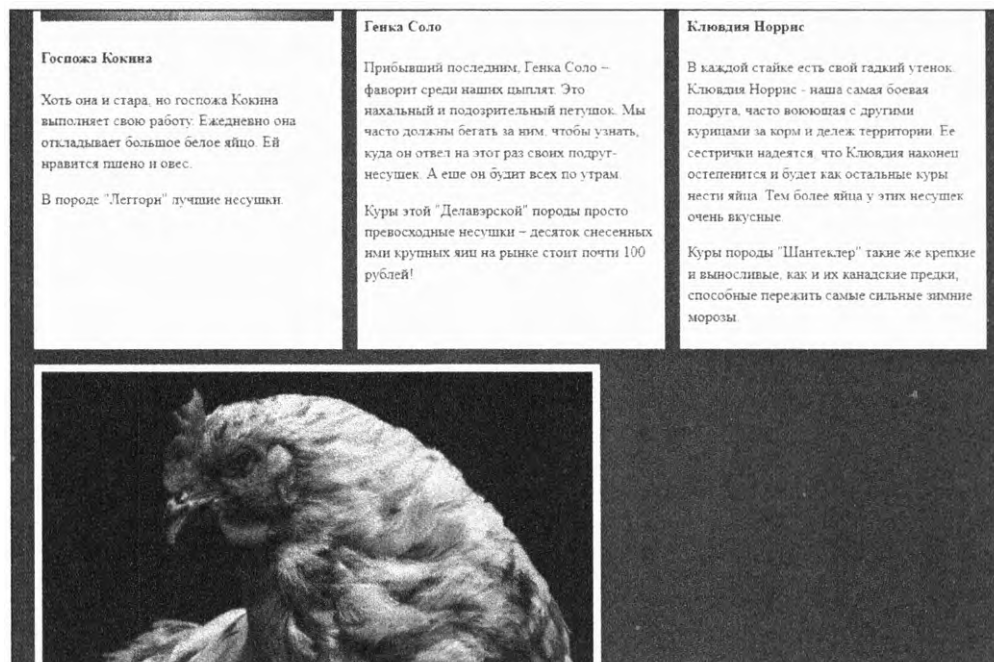


Рис. 16.5. При использовании flexbox-верстки карточки на последней строке не всегда той же ширины, что на предыдущих строках

При такой ширине области просмотра (около 1000 пикселей) на первой строке умещаются три карточки, а одна переходит на вторую. Ширина последнего элемента увеличивается до максимального значения 600 пикселей, в результате чего его размер превышает размер остальных. Когда экран позволяет уместить по две карточки на двух строках, их размеры окажутся одинаковыми. Однако на экранах других размеров может возникнуть проблема. В зависимости от количества карточек вероятны вариации: например, шесть карточек удачно впишутся в две строки по три карточки на строку, но на более крупных экранах на первой строке могут разместиться четыре карточки, а на второй — две.

Этот flexbox-макет все еще работает, поскольку все в нем понятно и доступно для использования, однако он неидеален. Есть два варианта дальнейших действий: определить несколько контрольных точек для более точного управления шириной

flex-элементов или решить, что существующий вариант вполне подходит в качестве резервного макета, и переопределить его, создав сетку для поддерживающих его браузеров.

Применим второй вариант. После создания flexbox-макета вы используете запрос для проверки поддержки сетки и добавления переопределяющих стилей. Обновите таблицу стилей, дополнив ее следующим CSS-кодом (листинг 16.7).

Листинг 16.7. Добавление сетки для поддерживающих ее браузеров

```
@media (min-width: 30em) {  
  .flyin-grid {  
    display: flex;  
    flex-wrap: wrap;  
    margin: 0 5rem;  
  }  
  
  .flyin-grid__item {  
    flex: 1 1 300px;  
    margin-left: 0.5em;  
    margin-right: 0.5em;  
    max-width: 600px;  
  }  
  
  @supports (display: grid) {  
    .flyin-grid {  
      display: grid;  
      grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));  
      grid-gap: 2em;  
    }  
  
    .flyin-grid__item {  
      max-width: initial;  
      margin: 0;  
    }  
  }  
}
```

Резервные стили
остаются прежними

Запросы для проверки поддержки
сетки в блоке медиазапроса

Установка
ширины колонок

Удаление полей,
примененных
резервным макетом

Теперь последние версии браузеров будут использовать идеальный макет. Колонки сетки гарантируют одинаковую ширину всех ее элементов. Функции `repeat()` и `auto-fit` позволяют сетке определить количество колонок, уместящихся в области просмотра. При таком подходе старые браузеры будут применять flexbox-макет, а на маленьких экранах отобразится еще более простой мобильный макет.

16.2.2. Добавление анимации в макет

Теперь, когда макет страницы готов, добавим анимацию. При загрузке страницы в область просмотра карточки должны вылетать, как показано на рис. 16.6. Сначала они покажутся вдалеке повернутыми на 90° вокруг вертикальной оси. Затем полетят к зрителю и ближе к концу анимации повернутся к нему лицевой стороной. На рис. 16.6 показаны три ключевых кадра анимации.

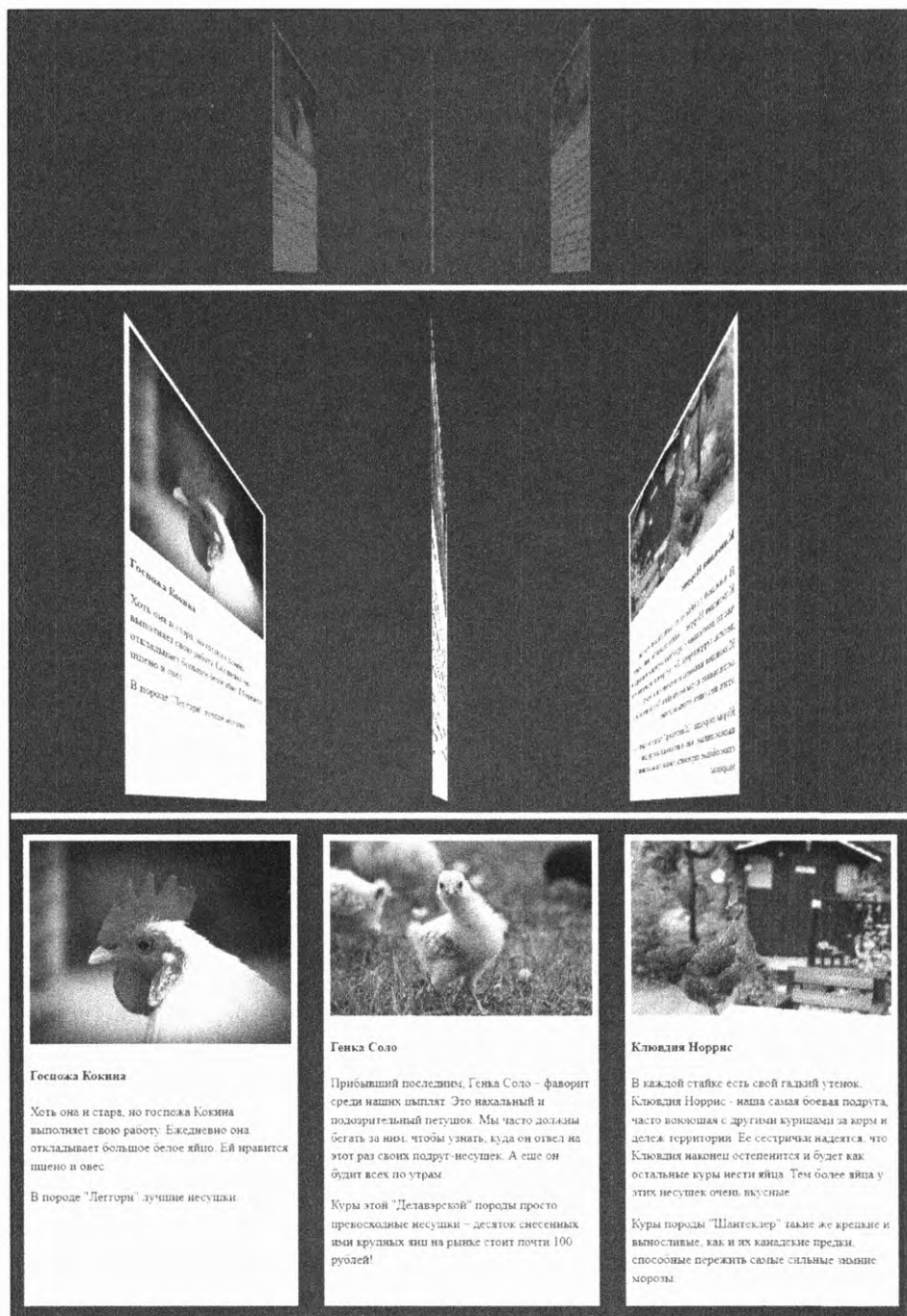


Рис. 16.6. 3D-трансформации для создания эффекта приближения карточек

Эта анимация включает в себя две трансформации: `translateZ()` запускает движение карточек издалека, а `rotateY()` вращает их. В листинге 16.8 приведен код, который применяет к контейнеру эффект перспективы, определяет ключевые кадры и обеспечивает для каждого элемента сетки анимационный эффект. Я также добавил непрозрачность, чтобы в процессе анимации элементы проявлялись.

Листинг 16.8. Добавление анимации

```
.flyin-grid {
  margin: 0 1rem;
  perspective: 500px;
}

.flyin-grid__item {
  animation: fly-in 600ms ease-in;
}

@keyframes fly-in {
  0% {
    transform: translateZ(-800px) rotateY(90deg);
    opacity: 0;
  }
  56% {
    transform: translateZ(-160px) rotateY(87deg);
    opacity: 1;
  }
  100% {
    transform: translateZ(0) rotateY(0);
  }
}
```

Применение эффекта перспективы к контейнеру

Применение анимационного эффекта к каждому элементу

Начальное удаленное положение элемента, он повернут

Приближенное положение элемента, он по-прежнему повернут

Конечное положение элемента

Этот CSS-код применяет эффект перспективы к контейнеру, поэтому все его элементы имеют одинаковую перспективу. Затем он задает для каждого элемента анимационный эффект. Загрузите страницу, чтобы воспроизвести анимацию.

В начале анимации элемент находится вдалеке в повернутом состоянии. В промежутке между начальным и средним ключевыми кадрами происходит большая часть его масштабирования — с 800 до 160 пикселей — и из прозрачного он становится непрозрачным. В промежутке между средним и последним ключевыми кадрами выполняются окончательное масштабирование и основное вращение элемента.

16.3. Задержка запуска анимации и режим заполнения

Запуск анимации можно задержать с помощью свойства `animation-delay`, которое ведет себя подобно `transition-delay`. Используйте его, чтобы задержать воспроизведение анимации подобно тому, как в предыдущей главе притормаживали переходы навигационного меню. Задерживая анимацию каждого из элементов на

разные промежутки времени, вы можете заставить их по очереди вылетать в область просмотра (рис. 16.7).

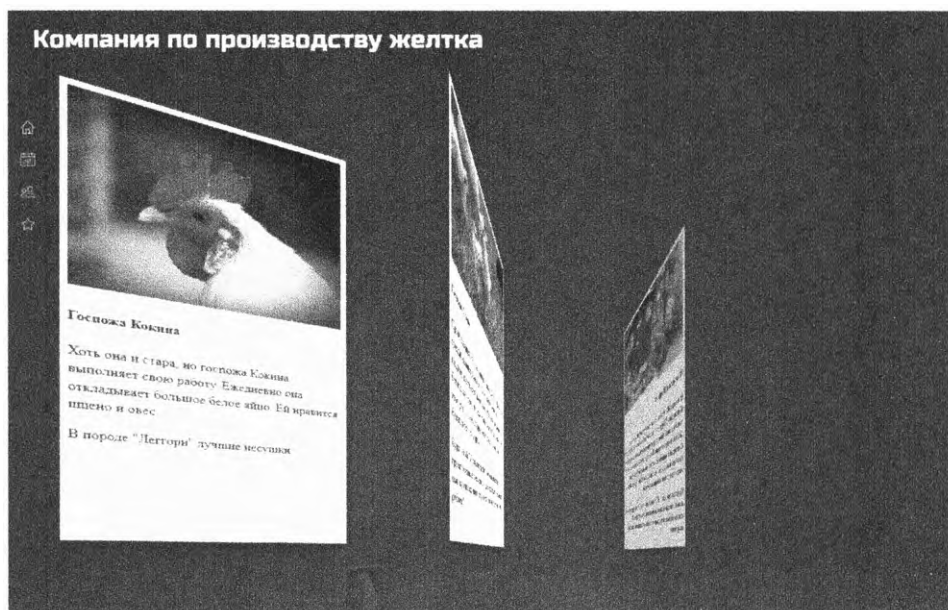


Рис. 16.7. Элементы по очереди вылетают в область просмотра

Следующий код (листинг 16.9) обеспечивает задержку анимации четырех элементов сетки. Однако он не будет работать так, как вы хотите. Добавьте этот код в таблицу стилей, а затем мы рассмотрим проблему и способ ее решения.

Листинг 16.9. Задержка запуска анимации

```
.flyin-grid_item {
  animation: fly-in 600ms ease-in;
}
```

```
.flyin-grid_item:nth-child(2) {
  animation-delay: 0.15s;
}
```

```
.flyin-grid_item:nth-child(3) {
  animation-delay: 0.3s;
}
```

```
.flyin-grid_item:nth-child(4) {
  animation-delay: 0.45s;
}
```

Обеспечивает последовательно возрастающую задержку запуска анимации каждого из элементов

Если вы откроете эту страницу в своем браузере, то заметите проблему. Анимация воспроизводится в ожидаемое время, однако некоторые элементы появляются на странице заранее. Через мгновение они исчезают и запускается их анимация

(рис. 16.8). Это немного раздражает и не обеспечивает тот эффект, на который мы рассчитываем. Ведь нам требуется, чтобы все элементы изначально были невидимыми и отображались только во время анимации.

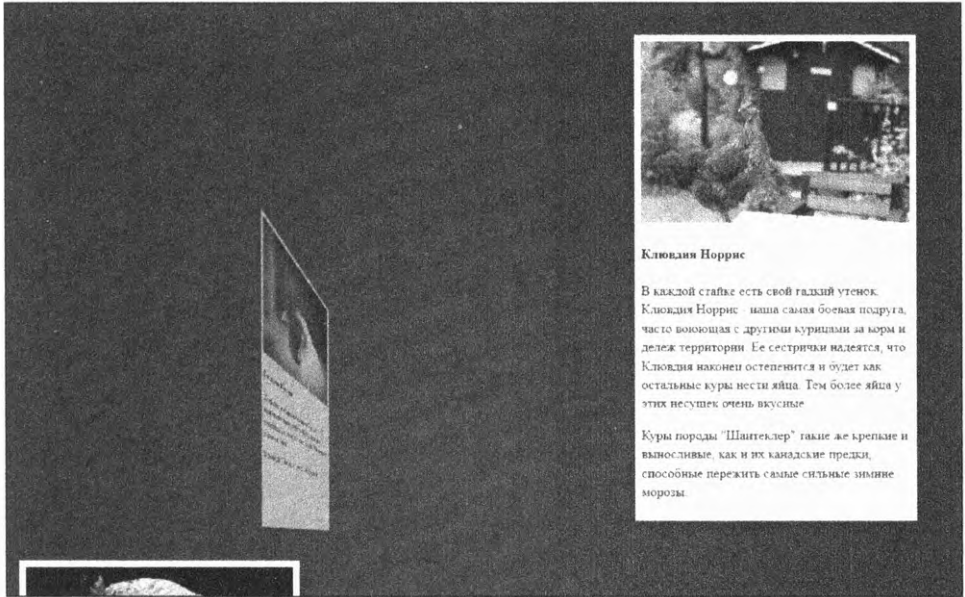


Рис. 16.8. Следующие элементы отображаются в своем окончательном положении до воспроизведения их анимации

Эта проблема связана с тем, что свойства `transform` и `opacity` работают только во время анимации. До ее запуска элементы сетки отображаются на странице в своем конечном положении. Когда начинается анимация, они перемещаются в положение, соответствующее ключевому кадру в точке 0 %. Нужно применить стили так, чтобы до запуска анимация была как бы приостановлена на первом кадре. Это можно сделать с помощью свойства `animation-fill-mode` (рис. 16.9).

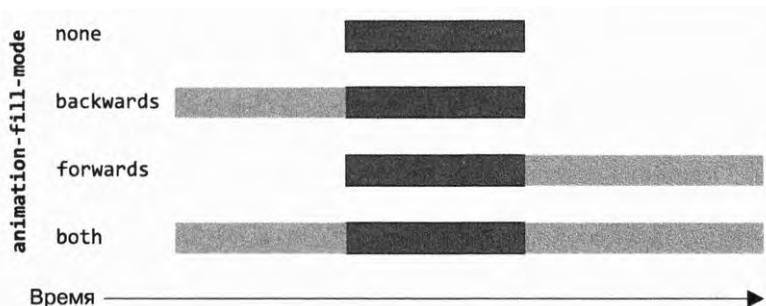


Рис. 16.9. Используйте свойство `animation-fill-mode` для применения стилей анимации до или после ее воспроизведения

Темные прямоугольники в данном случае обозначают длительность анимации. Исходным значением свойства `animation-fill-mode` является `none`. Это означает, что стили анимации не применяются к элементу до и после ее воспроизведения. При использовании режима `animation-fill-mode: backwards` браузер берет значения из первого ключевого кадра и применяет их к элементу перед воспроизведением анимации. Когда задействован режим `forwards`, значения последнего кадра продолжают применяться после завершения анимации. Режим `both` обеспечивает заполнение в обе стороны.

Используйте на своей странице режим обратного заполнения, чтобы решить проблему, возникающую в начале воспроизведения анимации. Обновите таблицу стилей, добавив в нее код из листинга 16.10.

Листинг 16.10. Применение режима обратного заполнения

```
.flyin-grid_item {
  animation: fly-in 600ms ease-in;
  animation-fill-mode: backwards;
}
```

← Применяет стили первого кадра перед запуском анимации

Фактически этот код приостанавливает воспроизведение анимации на первом кадре до момента ее запуска. Теперь до начала действия анимации элемент сетки остается прозрачным, передвинутым назад на 800 пикселей и повернутым на 90°.

Поскольку по окончании анимации элемент оказывается в своем нормальном положении, не нужно использовать режим заполнения `forwards`: карточка плавно переходит из состояния, соответствующего конечному кадру анимации, в статическое положение.

16.4. Передача смысла с помощью анимации

Бытует заблуждение, что анимация добавляется на страницу просто для развлечения и не имеет практической цели. Иногда это так (как в предыдущем примере), но не всегда. Лучшая анимация не добавляется в качестве последнего штриха — она интегрируется в опыт. Она передает пользователю конкретную информацию о каком-то элементе страницы.

16.4.1. Реакция на действие пользователя

С помощью анимации пользователю можно сообщить о нажатии кнопки или о получении сообщения. Если когда-либо, отправив форму, вы сомневались в том, был ли зарегистрирован щелчок кнопкой мыши, то знаете, насколько это важно.

На новой странице создадим небольшую форму с кнопкой отправки. Затем вы добавите вращающийся индикатор, сообщающий пользователю о том, что форма отправляется, а браузер ожидает ответа от сервера. Форма (рис. 16.10) состоит из метки, текстовой области и кнопки.

Рис. 16.10. Простая форма с кнопкой Сохранить

Создайте страницу и пустую таблицу стилей для этой формы. Добавьте приведенный далее HTML-код (листинг 16.11).

Листинг 16.11. Форма с кнопкой Сохранить

```
<!doctype html>
<html lang="en">
  <head>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <form>
      <label for="trip">Расскажите нам о впечатлениях
        от посещения нашего зоопарка:</label>
      <textarea id="trip" name="about-my-trip" rows="5"></textarea>
      <button type="submit" id="submit-button">Сохранить</button>
    </form>
  </body>
</html>
```

Текстовая область

Кнопка отправки данных

Сначала вы добавите CSS-код для создания макета и оформления. После этого разработаете несколько вариантов анимации, которые помогут пользователю понять, что происходит в результате его действий. Добавьте в таблицу стилей следующий код (листинг 16.12).

Листинг 16.12. Создание макета и форматирование формы с помощью стилей

```
body {
  font-family: Helvetica, Arial, sans-serif;
}

form {
  max-width: 500px;
}

label,
textarea {
  display: block;
  margin-bottom: 1em;
}
```

Ограничение ширины формы


```

textarea {
  width: 100%;
  font-size: inherit;
}

button {
  padding: 0.6em 1em;
  border: 0;
  background-color: hsl(220, 50%, 50%);
  color: white;
  font: inherit;
  transition: background-color 0.3s linear;
}
button:hover {
  background-color: hsl(220, 45%, 40%);
}

```

Синяя кнопка
с белым текстом

Затемнение кнопки
при наведении на нее
указателя мыши

Предположим, что эта форма — часть более крупного веб-приложения. Когда пользователь нажимает кнопку **Сохранить**, оно отправляет данные на сервер и, вероятно, получает ответ, а затем добавляет на страницу новый контент. Однако на все это уходит некоторое время. Пока пользователь ожидает ответа, ему будет спокойнее, если на экране появится какой-нибудь индикатор, сообщающий о том, что форма отправлена и вскоре появится ответ. Для этого часто используется анимация.

Вы можете изменить кнопку **Сохранить**, добавив ей состояние «загружается». При этом надпись «Сохранить» исчезнет, а на ее месте появится вращающийся значок (рис. 16.11). Когда пользователь отправит форму, вы примените код JavaScript, чтобы добавить кнопке класс **is-loading**, запустив тем самым анимацию.

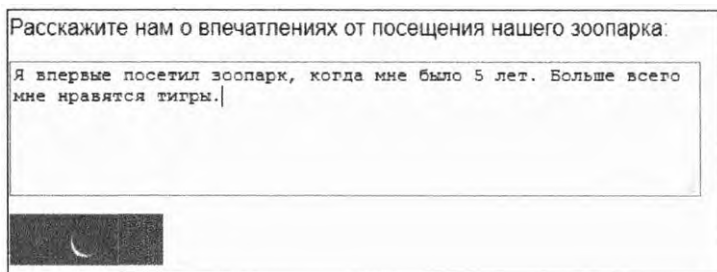


Рис. 16.11. Когда пользователь нажимает кнопку **Сохранить**, на ней появляется вращающийся значок загрузки

Существует множество вариантов вращающегося значка. Я предпочитаю использовать лаконичный, но хорошо передающий смысл вращающийся полумесяц. Чтобы добавить его, нужно внести в CSS-код два изменения: создать форму полумесяца с помощью свойств **border** и **border radius**, а затем реализовать эффект анимации для обеспечения ее вращения. Также нужно с помощью кода JavaScript добавить класс **is-loading**, позволяющий применить стили в момент нажатия кнопки.

Соответствующий CSS-код приведен в листинге 16.13. Эта разметка применяет эффект анимации к абсолютно позиционированному псевдоэлементу на кнопке. Добавьте код в таблицу стилей.

Листинг 16.13. Добавление эффекта вращения и состояния `is-loading`

```
button.is-loading {
  position: relative;
  color: transparent;
}
button.is-loading::after {
  position: absolute;
  content: "";
  display: block;
  width: 1.4em;
  height: 1.4em;
  top: 50%;
  left: 50%;
  margin-left: -0.7em;
  margin-top: -0.7em;
  border-top: 2px solid white;
  border-radius: 50%;
  animation: spin 0.5s linear infinite;
}
@keyframes spin {
  0% {
    transform: rotate(0deg);
  }
  100% {
    transform: rotate(360deg);
  }
}
```

Скрытие текста на кнопке

Позиционирование псевдоэлемента в центре кнопки

Обеспечение циклического воспроизведения анимации вращения

Установка одного полного оборота за итерацию

Этот код определяет для кнопки состояние `is-loading`. Его применение делает текст на ней невидимым благодаря свойству `color: transparent`, а псевдоэлемент помещается в центр кнопки с помощью абсолютного позиционирования.

В данном случае позиционирование может показаться довольно сложным: свойства `top` и `left` смещают псевдоэлемент вниз на половину высоты кнопки и вправо на половину ее ширины. При этом в центре кнопки располагается *верхний левый угол* псевдоэлемента. Затем отрицательные значения полей передвигают псевдоэлемент вверх и влево на 0,7 em, что соответствует половине его высоты и ширины. Вместе эти четыре свойства центрируют псевдоэлемент внутри кнопки. На время добавьте класс `is-loading` и поэкспериментируйте с этими значениями с помощью инструментов разработчика в своем браузере, чтобы понять, как они обеспечивают центрирование псевдоэлемента.

После позиционирования псевдоэлемента примените эффект анимации. При этом для подсчета итераций будет использоваться новое ключевое слово `infinite`. Это означает, что анимация станет повторяться до тех пор, пока к кнопке применяется класс `is-loading`. Анимация предусматривает трансформацию вращения от 0 до 360°. Это обеспечивает полный оборот псевдоэлемента. В конце анимации

элемент занимает первоначальное положение, поэтому переход между итерациями оказывается незаметным.

Вставьте на свою страницу элемент `script` из листинга 16.14, чтобы сценарий JavaScript добавил класс `is-loading` при нажатии кнопки. Поместите этот код перед закрывающим тегом `</body>`.

Листинг 16.14. Добавление класса `is-loading` при нажатии кнопки

```
<script type="text/javascript">
var input = document.getElementById('trip');
var button = document.getElementById('submit-button');

button.addEventListener('click', function(event) {
  event.preventDefault();
  button.classList.add('is-loading');
  button.disabled = true;
  input.disabled = true;
});
</script>
```

Предотвращение отправки
данных формы

Отображение
вращающегося
значка загрузки

Код, находящийся здесь, обеспечивает отправку
данных формы с помощью JavaScript

Нажатие кнопки **Сохранить** останавливает обычный процесс отправки формы с помощью функции `preventDefault()`. Это позволяет пользователю оставаться на одной и той же странице, пока приложение отправляет данные формы с помощью JavaScript. В то же время отключается возможность ввода данных, а к кнопке добавляется класс `is-loading`, отображающий вращающийся значок. Загрузите страницу и нажмите кнопку, чтобы появился этот значок.

В данном случае вы не отправляете данные формы, поскольку в демонстрационном примере нет сервера, на который их отправляют. Однако в настоящем приложении можете после ответа сервера включить вероятность ввода данных и удалить класс `is-loading`. В примере же обновите страницу, чтобы очистить форму и удалить класс `is-loading`.

16.4.2. Привлечение внимания пользователя

Анимация применяется также для того, чтобы привлечь внимание пользователя к чему-либо. Если вы ожидаете, что пользователь напишет в текстовой области больше нескольких предложений, напомните ему о необходимости часто сохранять написанное. Для этого можно реализовать анимационный эффект тряски кнопки (рис. 16.12).

Этот эффект обеспечивается многократным быстрым перемещением элемента влево и вправо. Для этого нужно создать анимацию на основе ключевых кадров и применить ее к кнопке, используя класс `shake`. Добавьте следующий код (листинг 16.15) в свою таблицу стилей.



Рис. 16.12. Обеспечьте быстрое перемещение кнопки влево и вправо, чтобы создать эффект тряски

Листинг 16.15. Создание анимационного эффекта тряски

```

.shake {
  animation: shake 0.7s linear;
}

@keyframes spin {
  0% {
    transform: rotate(0deg);
  }
  100% {
    transform: rotate(360deg);
  }
}

@keyframes shake {
  0%,
  100% {
    transform: translateX(0);
  }
  10%,
  30%,
  50%,
  70% {
    transform: translateX(-0.4em);
  }
  20%,
  40%,
  60% {
    transform: translateX(0.4em);
  }
  80% {
    transform: translateX(0.3em);
  }
  90% {
    transform: translateX(-0.3em);
  }
}

```

Используются значения одного и того же ключевого кадра в нескольких точках анимации

Смещение элемента влево

Смещение элемента вправо

Уменьшение амплитуды последнего движения

В этой анимации я сделал кое-что новое — несколько раз применил значение одного и того же ключевого кадра.

В начальном (0 %) и конечном (100 %) ключевых кадрах элемент находится в положении по умолчанию. Поскольку в этих ключевых кадрах используются одинаковые значения, можете разделить их запятой и задать значения их свойств один раз. Я сделал то же самое с ключевыми кадрами в точках 10, 30, 50 и 70 %, которые сдвигают элемент влево. Кадры в точках 20, 40 и 60 % сдвигают элемент вправо. Ключевые кадры в точках 80 и 90 % сдвигают элемент вправо и влево соответственно, но с меньшей амплитудой.

Эта анимация сдвигает элемент из стороны в сторону четыре раза, причем последнее движение выражено чуть слабее прочих, что создает впечатление замедления. На время добавьте кнопке класс **shake**, чтобы воспроизвести анимацию при загрузке страницы.

ПРИМЕЧАНИЕ

Анимацию можно применять в таблице стилей несколько раз, поэтому ее код не обязательно располагать вместе с кодом модуля, который ее использует. Я предпочитаю сохранять все свои определения @keyframe в одном месте, ближе к концу таблицы стилей.

Наконец, задействуйте JavaScript для воспроизведения анимации, когда захотите напомнить пользователю о необходимости сохранения своей работы. Для этого примените обработчик событий `keyup` и функцию `timeout`: когда пользователь введет символ в текстовую область, вы зададите односекундный тайм-аут, который добавит кнопке класс `shake`. Если до истечения секунды пользователь введет другой символ, вы удалите тайм-аут и установите новый. Обновите элемент `script` на своей странице в соответствии с кодом из листинга 16.16.

Листинг 16.16. Добавление класса `shake` после односекундной задержки

```
<script type="text/javascript">
var input = document.getElementById('trip');
var button = document.getElementById('submit-button');

var timeout = null;

button.addEventListener('click', function(event) {
  event.preventDefault();
  clearTimeout(timeout);
  button.classList.add('is-loading');
  button.disabled = true;
  input.disabled = true;
});

input.addEventListener('keyup', function() {
  clearTimeout(timeout);
  timeout = setTimeout(function() {
    button.classList.add('shake');
  }, 1000);
});

button.addEventListener('animationend', function() {
  button.classList.remove('shake');
});
</script>
```

← Определение переменной,
на которую будет ссылаться тайм-аут

← Отмена текущего
тайм-аута (если есть)

← Добавление класса `shake`
после односекундной задержки

← Удаление класса `shake`
по окончании анимации

Теперь загрузите страницу и введите что-нибудь в текстовую область. Через 1 с кнопка **Сохранить** начнет дрожать. Пока вы продолжаете вводить текст, таймер постоянно сбрасывается, а анимация не будет воспроизводиться до следующей паузы, продолжительность которой превысит 1 с. В этом случае тряска кнопки не будет постоянно отвлекать пользователя — анимация начинается только тогда, когда он прекращает ввод текста.

В этом коде мы также использовали событие JavaScript `animationend`. Оно запускается после окончания воспроизведения анимации. В этот момент класс

`shake` удаляется с кнопки, поэтому его можно снова добавить для повторного воспроизведения анимации в следующий раз, когда пользователь прекратит вводить текст.

Добавление и удаление классов, вероятно, самый простой способ применения анимации с помощью JavaScript. Но если вы хорошо знакомы с этим языком, воспользуйтесь специальным API-для взаимодействия с CSS-анимацией, предусматривающим возможность ее приостановки, отмены и запуска в обратном направлении. Для получения дополнительной информации об этом API-интерфейсе обратитесь к документации MDN по адресу developer.mozilla.org/ru/docs/Web/API/Animation.

Рассмотренная анимация — индикатор загрузки и дрожащая кнопка **Сохранить** — сообщает пользователю очень многое. При этом ему не нужно читать какие-либо пояснения. Анимация немедленно передает смысл, делая интерфейс менее навязчивым.

В процессе создания веб-приложения всегда старайтесь оценить, способна ли анимация предоставить пользователям полезную обратную связь. Например, можно сделать так, что при отправке электронного сообщения текстовая область будет вылетать за край экрана, а удаляемый черновик — сжиматься и исчезать. Анимация не обязательно должна быть впечатляющей, чтобы сообщить пользователю: его действия привели к нужному ему результату.

Для получения доступа к фантастическому набору готовых для использования ключевых кадров посетите сайт animista.net. Там вы найдете большую библиотеку анимационных эффектов, включая подпрыгивание, выкатывание и дрожание, напоминающее желе.

16.5. Совет напоследок

Многим веб-разработчикам CSS кажется слишком сложным языком. Одной ногой он стоит в мире дизайна, а другой — в мире кода. Некоторые аспекты языка не являются интуитивно понятными, особенно тем, кто изучает его самостоятельно. Я надеюсь, что эта книга помогла вам найти свой путь.

Мы подробно рассмотрели фундаментальные части языка, а также некоторые из наиболее сложных фрагментов макета страницы. Обсудили множество тем, начиная с организации CSS-кода, упрощающей его поддержание, и заканчивая новейшими подходами к созданию макетов. Мы вступили в мир дизайна и создали интерфейс не только функциональный, но и интуитивно понятный и визуально привлекательный.

Напоследок хочу посоветовать вам оставаться любознательными. Я продемонстрировал широкий набор инструментов CSS. Однако способы их использования неисчислимы. Столкнувшись с поразившей вас веб-страницей, откройте инструменты разработчика своего браузера и постарайтесь понять, как она работает. Подпишитесь на ресурсы разработчиков и дизайнеров, которые демонстрируют творческие решения или публикуют интересные уроки. Пробуйте новые подходы. И продолжайте учиться.

Итоги главы

- ❑ Применяйте ключевые кадры для определения основных точек анимации.
- ❑ Используйте режимы заполнения для обеспечения корректного начала и окончания анимации.
- ❑ JavaScript позволяет запускать анимацию в нужное время.
- ❑ Анимация обогащает пользовательское взаимодействие с веб-страницей не только визуальными эффектами, но и смыслом.

Приложения



Селекторы

Селекторы позволяют нацелиться на определенные элементы страницы для их оформления с помощью стилей. Язык CSS предусматривает широкий набор разных типов селекторов.

А.1. Базовые селекторы

- ❑ **tagname** — селектор типа или селектор тега. Он соответствует имени тега элементов, на которые происходит нацеливание. Имеет специфичность (0,0,1). Примеры: `p`, `h1`, `strong`.
- ❑ **.class** — селектор класса. Нацеливается на элементы, частью атрибута класса которых является указанное имя класса. Имеет специфичность (0,1,0). Примеры: `.media`, `.nav-menu`.
- ❑ **#id** — селектор идентификатора. Нацеливается на элемент с указанным атрибутом ID. Имеет специфичность (1,0,0). Пример: `#sidebar`.
- ❑ ***** — универсальный селектор. Нацеливается на все элементы. Имеет специфичность (0,0,0).

А.2. Комбинаторы

Комбинаторы объединяют несколько простых селекторов в один сложный. Например, в селекторе `.nav-menu li` пробел между двумя простыми селекторами называется *комбинатором потомков*. Он указывает на то, что целевой элемент `li` — это потомок элемента с классом `nav-menu`. Это самый распространенный комбинатор, однако есть и другие, описывающие конкретную взаимосвязь между указанными элементами.

- ❑ *Комбинатор прямых потомков (>)* — нацеливается на элементы, которые являются прямыми потомками другого элемента. Пример: `.parent > .child`.
- ❑ *Комбинатор смежных элементов (+)* — нацеливается на элементы, которые непосредственно следуют за другим. Пример: `p + h2`.
- ❑ *Общий комбинатор смежных элементов (~)* — нацеливается на все смежные элементы, следующие за указанным элементом. Обратите внимание на то, что он не нацеливается на смежные элементы, предшествующие указанному элементу. Пример: `li.active ~ li`.

Составные селекторы. Несколько простых селекторов можно объединить (без пробелов и других комбинаторов) в *составной* селектор, например: `h1.page-header`. Составной селектор нацеливается на элементы, соответствующие *всем* входящим в него простым селекторам. Например, селектор `.dropdown.is-active` нацеливается на `<div class = "dropdown is-active">`, но не на `<div class = "dropdown">`.

А.3. Селекторы псевдоклассов

Селекторы псевдоклассов используются для нацеливания на элементы, находящиеся в определенном состоянии. Это состояние может обуславливаться действием пользователя или положением элемента в документе относительно его родительских или родственных элементов. Селекторы псевдоклассов всегда начинаются с двоеточия (:). Они имеют ту же специфичность, что и селектор классов (0,1,0).

- ❑ `:first-child` — нацеливается на элемент, который является первым дочерним элементом его родительского элемента.
- ❑ `:last-child` — нацеливается на элемент, являющийся последним дочерним элементом его родительского элемента.
- ❑ `:only-child` — нацеливается на элемент, который является единственным дочерним элементом его родительского элемента (родственные элементы отсутствуют).
- ❑ `:nth-child($an+b$)` — нацеливается на элементы, исходя из их положения в ряду родственных элементов. Формула $an+b$, где a и b — целые числа, указывает, на какие элементы необходимо произвести нацеливание. Чтобы понять принцип работы этой формулы, представьте себе ее решение для всех целых значений n , начиная с нуля. Результаты решения этого уравнения указывают на дочерние элементы, на которые произведено нацеливание. Некоторые примеры показаны в следующей таблице.

Селектор	Целевые элементы	Результат	Описание
<code>:nth-child(n)</code>	0, 1, 2, 3, 4	○○○○○○○○○○	Каждый элемент
<code>:nth-child(2n)</code>	0, 2, 4, 6, 8	●○○○○○○○○●	Четные элементы
<code>:nth-child(3n)</code>	0, 3, 6, 9, 12	●○○○○○○○○●	Каждый третий элемент
<code>:nth-child(3n+2)</code>	2, 5, 8, 11, 14	●○○○○○○○○●	Каждый третий элемент, начиная со второго
<code>:nth-child(n+4)</code>	4, 5, 6, 7, 8	●○○○○○○○○●	Все элементы, начиная с четвертого
<code>:nth-child(-n+4)</code>	4, 3, 2, 1, 0	●○○○○○○○○●	Первые четыре элемента

- ❑ `:nth-last-child($an+b$)` — напоминает селектор `:nth-child()`, только отсчет ведется не от первого элемента вперед, а от последнего элемента назад. Формула в круглых скобках работает по той же схеме, что и в случае селектора `:nth-child()`.

- ❑ **:first-of-type** — напоминает селектор **:first-child**, за исключением того, что вместо учета позиции в ряду всех дочерних элементов он учитывает номер позиции элемента только в ряду других дочерних элементов с тем же именем тега.
- ❑ **:last-of-type** — нацеливается на последний дочерний элемент каждого типа.
- ❑ **:only-of-type** — нацеливается на элемент, который является единственным дочерним элементом данного типа.
- ❑ **:nth-of-type($an+b$)** — нацеливается на элементы соответствующего типа, исходя из их порядкового номера и указанной формулы. Подобен селектору **:nth-child**.
- ❑ **:nth-last-of-type($an+b$)** — нацеливается на элементы соответствующего типа, исходя из определенной формулы, при этом отсчет ведется от последнего элемента в обратном направлении. Подобен селектору **:nth-last-child**.
- ❑ **:not(<селектор>)** — нацеливается на элементы, не соответствующие селектору в круглых скобках. Селектор в круглых скобках должен быть простым: он может относиться только к самому элементу, вы не должны использовать этот селектор для исключения предков. Он также не должен содержать другого отрицающего селектора.
- ❑ **:empty** — нацеливается на элементы, не имеющие дочерних элементов. Помните о том, что он не нацеливается на элементы, содержащие пробелы, поскольку пробел представлен в DOM в качестве дочернего элемента текстового узла. На момент написания этой книги консорциум W3C рассматривает псевдокласс **:blank**, который ведет себя подобным образом, но выбирает также элементы, содержащие только пробел. Псевдокласс **:blank** пока не поддерживается ни в одном браузере.
- ❑ **:focus** — нацеливается на элементы, которые получили фокус с помощью щелчка кнопкой мыши, касания экрана или нажатия клавиши **Tab**.
- ❑ **:hover** — нацеливается на элементы, на которые наведен указатель мыши.
- ❑ **:root** — нацеливается на корневой элемент документа. В HTML это элемент **html**. Однако код CSS может применяться и к другим XML и XML-подобным документам наподобие SVG. В этом случае данный селектор работает как более общий.

Есть несколько псевдоклассов, имеющих отношение к полям формы. Некоторые из них были введены или усовершенствованы в спецификации селекторов 4-го уровня, поэтому не работают в IE10 и некоторых других браузерах. Информацию о поддержке найдете на сайте caniuse.com.

- ❑ **:disabled** — нацеливается на отключенные элементы, в том числе поля ввода, выделенные элементы и кнопки.
- ❑ **:enabled** — нацеливается на включенные элементы, то есть элементы, которые могут быть активизированы или получить фокус.
- ❑ **:checked** — нацеливается на выбранные флажки, переключатели или пункты раскрывающегося списка.

- ❑ `:invalid` — нацеливается на элементы, входные значения которых недопустимы с точки зрения заданного типа входных данных, например `<input type="email">`, значением которого не является действительный адрес электронной почты (уровень 4).
- ❑ `:valid` — нацеливается на элементы с допустимыми значениями (уровень 4).
- ❑ `:required` — нацеливается на элементы с установленным атрибутом `required` (уровень 4).
- ❑ `:optional` — нацеливается на элементы, для которых не установлен атрибут `required` (уровень 4).

Это далеко не все псевдоклассы. Их полный список можно найти в документации MDN по адресу developer.mozilla.org/ru/docs/Web/CSS/Псевдо-классы.

А.4. Селекторы псевдоэлементов

Псевдоэлементы напоминают псевдоклассы, однако вместо того, чтобы выбирать элементы в конкретном состоянии, они нацеливаются на определенную часть документа, которая непосредственно не соответствует какому-либо элементу HTML. Они могут нацеливаться на часть элемента и даже внедрять контент в то место страницы, где разметка его не предусматривает.

Эти селекторы начинаются с двойного двоеточия (`::`), хотя большинство браузеров поддерживают синтаксис и с одним двоеточием для обеспечения обратной совместимости. Псевдоэлементы имеют такую же специфичность, как и селектор типа `(0,0,1)`.

- ❑ `::before` — создает псевдоэлемент, который становится первым дочерним элементом сопутствующего. По умолчанию это встроенный элемент. Он может использоваться для вставки текста, изображений или других фигур. Чтобы этот элемент появился, необходимо указать свойство `content`. Пример: `.menu::before`.
- ❑ `::after` — создает псевдоэлемент, который становится последним дочерним элементом сопутствующего. По умолчанию это встроенный элемент. С его помощью вставляются текст, изображения или другие фигуры. Чтобы этот элемент появился, необходимо указать свойство `content`. Пример: `.menu::after`.
- ❑ `::first-letter` — позволяет указать стили только для первого текстового символа в сопутствующем элементе. Пример: `h2::first-letter`.
- ❑ `::first-line` — позволяет указать стили для первой строки текста в сопутствующем элементе.
- ❑ `::selection` — позволяет задать стили для любого текста, который пользователь выделил с помощью указателя мыши. Часто применяется для изменения свойства `background-color` выделенного текста. Использовать можно лишь несколько свойств, включая `color`, `background-color`, `cursor` и `text-decoration`.

A.5. Селекторы атрибутов

Селекторы атрибутов используются для нацеливания на элементы исходя из их HTML-атрибутов. Они имеют ту же специфичность, что и селектор класса (0,1,0).

- ❑ `[attr]` — нацеливается на элементы, для которых указан атрибут `attr` вне зависимости от его значения. Пример: `input[disabled]`.
- ❑ `[attr="значение"]` — нацеливается на элементы с указанным атрибутом `attr`, значение которого соответствует заданному строковому значению. Пример: `input[type="radio"]`.
- ❑ `[attr^="значение"]` — селектор атрибутов «начиная с». Нацеливается по атрибуту и значению, которое начинается с указанной строки. Пример: `a[href^="https"]`.
- ❑ `[attr$="значение"]` — селектор атрибутов «заканчивая». Нацеливается по атрибуту и значению, которое заканчивается указанной строкой. Пример: `a[href$=".pdf"]`.
- ❑ `[attr*="значение"]` — селектор атрибутов «содержит». Нацеливается по атрибуту и значению, которое содержит указанную строку. Пример: `[class*="sprite-"]`.
- ❑ `[attr~="значение"]` — селектор атрибутов «список значений, разделенных пробелами». Нацеливается по атрибуту и значению, представляющему собой список разделенных пробелами значений, одно из которых соответствует указанной строке. Пример: `a[rel="author"]`.
- ❑ `[attr|="значение"]` — нацеливается по атрибуту и значению, которое либо соответствует указанной строке, либо начинается с нее и сопровождается дефисом (-). Полезен для установки атрибута языка, указывающего или не указывающего его субкод (например, мексиканский испанский, `es-MX`, или испанский вообще, `es`). Пример: `[lang|"es"]`.

Селекторы атрибутов, нечувствительные к регистру. Все предыдущие селекторы атрибутов чувствительны к регистру. В спецификации селекторов 4-го уровня появился нечувствительный к регистру модификатор, который можно добавить к любому селектору атрибутов. Для этого поставьте `i` перед закрывающей квадратной скобкой. Пример: `input[value="search" i]`.

Многие браузеры пока не поддерживают и просто игнорируют его. Поэтому при использовании нечувствительных к регистру модификаторов обязательно добавьте в качестве резервной обычную чувствительную к регистру версию.

Б

Препроцессоры

Использование препроцессоров — важная часть работы над CSS. Препроцессор обеспечивает ряд приспособлений, позволяющих оптимизировать написание и поддержание кодовой базы. Например, вы можете написать фрагмент кода один раз, а затем многократно применять его в своей таблице стилей.

Препроцессор берет написанный вами исходный файл и преобразует его в итоговый файл, который представляет собой обычную таблицу стилей CSS. В большинстве случаев исходный файл напоминает обычный CSS-код, но с дополнительными функциями. Простой пример работы с переменной препроцессора выглядит так:

```
$brand-blue: #0086b3;
```

```
a:link {  
  color: $brand-blue;  
}
```

```
.page-heading {  
  font-size: 1.6rem;  
  color: $brand-blue;  
}
```

Этот фрагмент кода определяет переменную с именем **\$brand-blue**, которая используется далее в таблице стилей в двух разных местах. При прогоне через препроцессор Sass переменная заменяется во всей таблице стилей, в результате чего получается следующий CSS-код:

```
a:link {  
  color: #0086b3;  
}
```

```
.page-heading {  
  font-size: 1.6rem;  
  color: #0086b3;  
}
```

Важно отметить, что на выходе получается обычный CSS-код, так что препроцессор не добавляет в язык новых функций, когда речь идет о браузере. Тем не менее он обеспечивает удобство вам как разработчику.

В приведенном примере использование переменной для представления цвета позволяет многократно задействовать этот цвет, не требуя копирования и вставки конкретного шестнадцатеричного кода. Препроцессор выполняет копирование за вас, когда генерирует выходной файл. Это также означает, что вы можете скорректировать значение в одном месте и это изменение распространится на всю таблицу стилей.

Два наиболее популярных препроцессора — *Sass* (sass-lang.com) и *Less* (lesscss.org), хотя есть и другие. Препроцессор *Sass* — самый популярный, поэтому в данном приложении я сосредоточусь главным образом на нем. Однако препроцессоры *Sass* и *Less* похожи друг на друга, имеют лишь минимальные синтаксические различия. Например, для обозначения переменных *Sass* использует символ `$` (`$brand-blue`), а *Less* — символ `@` (`@brand-blue`). Все функции *Sass*, описанные далее, поддерживаются и в *Less*. Чтобы узнать о различиях синтаксиса, обратитесь к документации препроцессора *Less*.

Б.1. Препроцессор Sass

Перед началом работы с *Sass* нужно принять несколько решений. Вначале предстоит выбрать реализацию. Препроцессор *Sass* написан на языке *Ruby*, однако эта реализация работает довольно медленно при компиляции больших таблиц стилей, поэтому я рекомендую использовать *LibSass* — порт C/C++ компилятора *Sass*.

Если вы хорошо знакомы с *JavaScript* и средой *Node*, то можете получить *LibSass*, установив пакет `node-sass` через менеджер пакетов `npm`. Если вы еще не установили *Node.js*, сделайте это бесплатно по адресу nodejs.org. Скачайте и установите его в соответствии с приведенной там инструкцией. Я покажу необходимые для этого команды, но если вы хотите больше узнать об `npm` или вам нужна помощь по устранению неполадок, посетите страницу docs.npmjs.com/getting-started.

Б.1.1. Установка препроцессора Sass

Чтобы установить препроцессор *Sass*, создайте каталог проекта и перейдите к нему в своем терминале. Затем выполните следующие команды:

- ❑ `npm init -y` — инициализирует новый проект `npm`, создавая файл `package.json`. Для получения дополнительной информации об этом файле обратитесь к главе 10 (подраздел 10.1.1);
- ❑ `npm install --save-dev node-sass` — устанавливает пакет `node-sass` и добавляет его в `package.json` в качестве зависимости для разработки.

ПРИМЕЧАНИЕ

В ОС Windows вам нужно также установить пакет `node-gyp`. Для получения дополнительной информации посетите страницу github.com/sass/node-sass#install.

Второе решение, которое следует принять, касается синтаксиса. Препроцессор Sass поддерживает два: Sass и SCSS. Они предусматривают одни и те же функции, однако в синтаксисе Sass опущены все фигурные скобки и точки с запятой, а структура кода задается с помощью отступов, например:

```
body
  font-family: Helvetica, sans-serif
  color: black
```

Это напоминает такие языки программирования, как Ruby и Python. Синтаксис SCSS использует фигурные скобки и точки с запятой, поэтому он больше похож на обычный код CSS, например:

```
body {
  font-family: Helvetica, sans-serif;
  color: black;
}
```

Синтаксис SCSS используется чаще. Если вы испытываете какие-либо сомнения, рекомендую выбрать SCSS, который я применяю в данном приложении.

ПРИМЕЧАНИЕ

Файлы SCSS имеют расширение `.scss`, а файлы Sass — расширение `.sass`.

Б.1.2. Запуск препроцессора Sass

Теперь, когда препроцессор Sass установлен, используем его для создания таблицы стилей. В каталоге проекта создайте два подкаталога с именами `sass` и `build`. Вы поместите исходные файлы в каталог `sass`, а препроцессор Sass задействует их для создания файла CSS в каталоге `build`. Затем отредактируйте файл `package.json`. Измените запись `script` в соответствии с кодом из листинга Б.1.

Листинг Б.1. Добавление команды `sass` в файл `package.json`

```
"scripts": {
  "sass": "sass sass/index.scss build/styles.css"
},
```

Этот код определяет команду `sass`, которая после запуска скомпилирует файл по адресу `sass/index.scss` в новый файл по адресу `build/styles.css`. Файла `sass/index.scss` в проекте еще не существует. Создайте его. В этот файл будет добавлен

ваш Sass-код. Запуск `npm run sass` обеспечивает выполнение этой команды, создавая (или переписывая) таблицу стилей по адресу `build/styles.css`.

СОВЕТ

Для таких систем организации задач, как Grunt, Gulp и Webpack, существуют плагины вроде `gulp-sass`. Если хотите использовать плагин, найдите тот, который интегрирует Sass или Less в наиболее комфортный для вас рабочий процесс.

Б.1.3. Важные функции препроцессора Sass

Я привел один пример переменной Sass `$brand-blue`. Добавьте код из листинга Б.2 в файл `index.scss`, чтобы увидеть, как Sass скомпилирует его.

Листинг Б.2. Переменная Sass

```
$brand-blue: #0086b3;
a:link {
  color: $brand-blue;
}
.page-heading {
  font-size: 1.6rem;
  color: $brand-blue;
}
```

Определение переменной

Использование переменной

Запустите `npm run sass`, чтобы скомпилировать этот код в CSS. Выходной файл `build/styles.css` будет выглядеть следующим образом:

```
a:link {
  color: #0086b3; }

.page-heading {
  font-size: 1.6rem;
  color: #0086b3; }

/*# sourceMappingURL=styles.css.map */
```

Переменные были заменены шестнадцатеричным значением, поэтому теперь браузер может их понять. Кроме того, препроцессор Sass создал файл карты кода и добавил комментарий в конец таблицы стилей, указав путь к *карте кода*.



Карта кода — файл, который компьютер использует для отслеживания источника (Sass) каждой сгенерированной строки кода (в нашем случае CSS). С этим файлом могут работать некоторые отладчики, в том числе инструменты разработчика в браузере.

Обратите внимание на то, что скомпилированный код отформатирован не слишком аккуратно: закрывающие фигурные скобки перенесены на предыдущую строку, а некоторые пустые строки удалены. Это нормально, поскольку пробелы не имеют значения для браузера.

Однако я скорректирую форматирование остальных примеров в этом приложении, чтобы прояснить их смысл.

Строчные вычисления

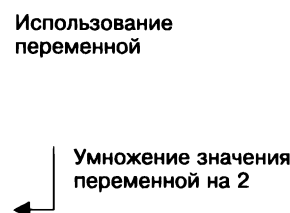
Препроцессор Sass поддерживает также функцию встроенных арифметических вычислений с использованием операторов `+`, `-`, `*`, `/` и `%` (для деления по модулю). Это позволяет получать несколько значений из одного исходного, как показано далее (листинг Б.3).

Листинг Б.3. Применение встроенных вычислений

```
$padding-left: 3em;

.note-author {
  left-padding: $padding-left;
  font-weight: bold;
}

.note-body {
  left-padding: $padding-left * 2;
}
```



Используйте команду `npm run sass` для компиляции этого кода, чтобы получить следующий результат:

```
.note-author {
  left-padding: 3em;
  font-weight: bold;
}

.note-body {
  left-padding: 6em;
}
```

Эта функция полезна, когда два значения связаны друг с другом, но не совпадают. В данном случае `note-body` всегда будет иметь в два раза больший левый отступ, чем `note-author`, вне зависимости от значения `$padding-left`.

Вложенные селекторы

Препроцессор Sass позволяет вкладывать селекторы в другие блоки объявлений. Эта возможность используется для группировки кода в одном и том же блоке, как показано в листинге Б.4.

Листинг Б.4. Вложенные селекторы

```
.site-nav {
  display: flex;
  > li {
    margin-top: 0;
    &.is-active {
      display: block;
    }
  }
}
```

Вложенный селектор

Амперсанд указывает место, в которое будет добавлен внешний селектор

Препроцессор Sass объединяет вложенные селекторы с селекторами внешнего блока (-ов) объявлений. При компилировании этого кода получается следующий результат:

```
.site-nav {
  display: flex;
}

.site-nav > li {
  margin-top: 0;
}

.site-nav > li.is-active {
  font-weight: bold;
}
```

По умолчанию внешний селектор `.site-nav` предваряет каждый селектор в скомпилированном коде, а в место соединения селекторов добавляется пробел. Чтобы это изменить, используйте амперсанд (&) для указания места вставки внешнего селектора.

ВНИМАНИЕ!

Вложение увеличивает специфичность полученных в результате селекторов. Будьте осторожны и избегайте слишком большой глубины вложенности.

Вы также можете вкладывать в блок объявления медиазапросы. Благодаря этому не обязательно повторять один и тот же селектор (листинг Б.5).

Листинг Б.5. Вложение медиазапроса

```
html {
  font-size: 1rem;
  @media (min-width: 45em) {
    font-size: 1.25rem;
  }
}
```

Медиазапрос внутри блока объявления

В результате компилирования этого кода получается:

```
html {  
  font-size: 1rem;  
}  
  
@media (min-width: 45em) {  
  html {  
    font-size: 1.25rem;  
  }  
}
```

Таким образом, в случае изменения селектора не придется изменять соответствующий селектор в медиазапросе.

Фрагментирование (@import)

Фрагментирование позволяет разделить стили на несколько отдельных файлов, а Sass объединит их в один файл. Используя эту возможность, упорядочите свои файлы любым способом и при этом передайте браузеру только один файл, тем самым уменьшив количество сетевых запросов.

Создайте новый файл проекта `sass/button.scss`. Добавьте в него стили, показанные в листинге Б.6.

Листинг Б.6. Частичная таблица стилей button

```
.button {  
  padding: 1em 1.25em;  
  background-color: #265559;  
  color: #333;  
}
```

Затем в `index.scss` импортируйте частичную таблицу стилей, используя правило `@import`, как показано в листинге Б.7.

Листинг Б.7. Импорт частичного файла

```
@import "button";
```



Путь к частичному файлу

После запуска Sass частичный файл будет скомпилирован и вставлен в то место, которое вы указали с помощью правила `@import`.


На мой взгляд, это самая важная функция препроцессора. По мере увеличения таблицы становится все сложнее прокручивать тысячи строк кода для нахождения нужной ее части. Эта функция позволяет разбить таблицу стилей на небольшие логические модули, не теряя производительности. Для получения более подробной информации обратитесь к врезке «Препроцессоры и модульный CSS-код» в главе 9.

Миксины

Миксин — это небольшой фрагмент CSS-кода, который можно многократно использовать в таблице стилей. Это бывает полезно при наличии определенного стиля шрифта, согласованность которого следует обеспечить в разных местах, или задействовании часто повторяющихся правил вроде `clearfix` (см. раздел 4.2).

Миксин определяется с помощью правила `@mixin` и применяется с правилом `@include`. Далее приведен пример миксина `clearfix` (листинг Б.8).

Листинг Б.8. Миксин `clearfix`



```

@mixin clearfix {
  &::before {
    display: table;
    content: " ";
  }

  &::after {
    clear: both;
  }
}

.media {
  @include clearfix;
  background-color: #eee;
}

```

Определение миксина `clearfix`

Вложенные селекторы

Применение миксина

Препроцессор берет код из миксина и вставляет его вместо правила `@include`. Полученный в результате код выглядит следующим образом:

```

.media {
  background-color: #eee;
}
.media::before {
  display: table;
  content: " ";
}
.media::after {
  clear: both;
}

```

Обратите внимание на то, что в полученном коде `clearfix` не упоминается. Содержимое миксина добавляется только в те места таблицы стилей, где используется.

Вы также можете определить миксины, которые принимают параметры подобно функции в обычном программировании. В листинге Б.9 показан миксин, определяющий окно предупреждения. Он принимает параметры `$color` и `$bg-color`, являющиеся переменными, определенными в пределах области видимости миксина.

Листинг Б.9. Миксин с параметрами

```

@mixин alert-variant($color, $bg-color) {
  padding: 0.3em 0.5em;
  border: 1px solid $color;
  color: $color;
  background-color: $bg-color;
}
.alert-info {
  @include alert-variant(blue, lightblue)
}
.alert-danger {
  @include alert-variant(red, pink)
}

```

Определяет миксин с двумя параметрами

Параметры-переменные могут использоваться в пределах миксина

Передает значения в миксин

При каждом использовании миксину передаются разные значения. Они присваиваются этим двум переменным. Приведенный фрагмент кода выдает следующий CSS:

```

.alert-info {
  padding: 0.3em 0.5em;
  border: 1px solid blue;
  color: blue;
  background-color: lightblue;
}
.alert-danger {
  padding: 0.3em 0.5em;
  border: 1px solid red;
  color: red;
  background-color: pink;
}

```

Миксин позволяет многократно задействовать один и тот же фрагмент кода, однако в данном случае он создает две версии одного и того же кода. Эти различия обусловлены переданными значениями.

ВНИМАНИЕ!

Ранее миксины использовались в основном для добавления версий свойств с префиксом поставщика. Например, миксин `border-radius` может применяться для задания свойств `-webkit-border-radius`, `-moz-border-radius` и `border-radius`. Я не рекомендую задействовать миксины с этой целью, лучше воспользуйтесь инструментом Autoprefixer (подробнее о нем узнаете далее из раздела Б.2 «PostCSS»).

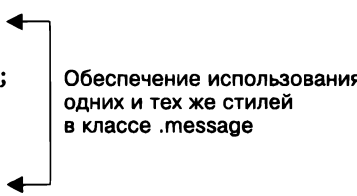
Расширения

Препроцессор Sass предусматривает также правило `@extend`. Оно похоже на миксин, но использует другой способ компиляции. Вместо многократного копирования одних и тех же объявлений, Sass группирует селекторы в один набор правил. Это проще

всего объяснить на примере. В листинге Б.10 набор правил `.message` расширен двумя другими наборами правил.

Листинг Б.10. Расширение базового класса

```
.message {  
  padding: 0.3em 0.5em;  
  border-radius: 0.5em;  
}  
  
.message-info {  
  @extend .message;  
  color: blue;  
  background-color: lightblue;  
}  
  
.message-danger {  
  @extend .message;  
  color: red;  
  background-color: pink;  
}
```



Обеспечение использования
одних и тех же стилей
в классе `.message`

Это дает следующий результат:

```
.message,  
.message-info,  
.message-danger {  
  padding: 0.3em 0.5em;  
  border-radius: 0.5em;  
}  
  
.message-info {  
  color: blue;  
  background-color: lightblue;  
}  
  
.message-danger {  
  color: red;  
  background-color: pink;  
}
```

Обратите внимание на то, что Sass скопировал селекторы `.message-info` и `.message-danger` в первый набор правил. Преимущество этого приема заключается в том, что в разметке нужно сослаться на один класс вместо двух: `<div class="message message-info">` превращается в `<div class="message-info">`, поскольку класс `message-info` теперь включает и все стили для класса `message`, что делает использование класса `message` избыточным.

ВНИМАНИЕ!

В отличие от миксина правило `@extend` перемещает селектор в более раннюю позицию в таблице стилей. Это означает, что исходный порядок ваших объявлений не обязательно совпадет с ожидаемым, что повлияет на каскадность.

При использовании правила `@extend` обычно получается более короткий код, чем с применением миксина. Можно посчитать это преимуществом, поскольку полученная в результате таблица стилей более компактна и, следовательно, быстрее загружается. Однако важно отметить, что миксины создают много повторяющегося кода, который очень хорошо сжимается с помощью архиватора. Пока ваш сервер сжимает весь сетевой трафик (что он и должен делать), это преимущество оказывается не слишком существенным.

Не пренебрегайте миксинами, а правило `@extend` задействуйте исключительно для оптимизации производительности. Учитывайте организацию кода при выборе миксинов или расширений для каждого конкретного случая. В общем, вероятно, следует делать выбор в пользу миксинов. Используйте правило `@extend` только тогда, когда хотите сократить количество имен классов в своем HTML-коде, как сделано в листинге B.10.

Управление цветом

Еще одно удобное приспособление Sass — это серия функций, позволяющих манипулировать цветом. Если вам нужны два связанных друг с другом цвета, например светло-зеленый и темно-зеленый, используйте функции, приведенные в листинге B.11, чтобы получить их.

Листинг B.11. Функции Sass для манипулирования цветом

<code>\$green: #63a35c;</code>		Цвет становится на 10 % темнее
<code>\$green-dark: darken(\$green, 10%);</code>	←	
<code>\$green-light: lighten(\$green, 10%);</code>	←	Цвет становится на 10 % светлее
 <code>\$green-vivid: saturate(\$green, 20%);</code>		
<code>\$green-dull: desaturate(\$green, 20%);</code>		Настройка степени насыщенности цвета
 <code>\$purple: adjust-hue(\$green, 180deg);</code>		
<code>\$yellow: adjust-hue(\$green, -70deg);</code>		Выбор значения оттенка на цветовом круге
 <code>\$green-transparent: rgba(\$green, 0.5);</code>	←	
		Настройка степени прозрачности

Эти функции позволяют, отредактировав значение одной переменной, повлиять на связанные цвета. Сохранять значение в переменной не нужно. Можете скорректировать его непосредственно в свойстве, в котором оно понадобилось:

```
.page-header {
  color: $green;
  background-color: lighten($green, 50%);
}
```

Для более сложных манипуляций существует еще несколько цветовых функций, подробнее о которых вы узнаете по адресу jackiebalzer.com/color.

Циклы

Используйте циклы для создания небольших вариаций значения. В главе 15 я применил несколько селекторов `:nth-child()` для последовательного нацеливания на пункты меню, задав во всех случаях разные значения свойства `transition-delay` (см. листинг 15.10). Этот фрагмент кода можно сократить с помощью цикла Sass, который задействует правило `@for`, как показано в листинге Б.12.

Листинг Б.12. Итеративная обработка серии значений

```
@for $index from 2 to 5 {
  .nav-links > li:nth-child(#{ $index }) {
    transition-delay: (0.1s * $index) - 0.1s;
  }
}
```

Итеративное изменение значения \$index с 2 до 4

Использование переменной в селекторе

Умножение значения переменной на значение задержки

Это создает несколько копий одного и того же блока кода, в каждом из которых значение переменной `$index` увеличено на 1. Обратите внимание на то, что я использовал переменную в селекторе, экранировав ее с помощью нотации `#{ }`. Полученный в результате код выглядит следующим образом:

```
.nav-links > li:nth-child(2) {
  transition-delay: 0.1s;
}

.nav-links > li:nth-child(3) {
  transition-delay: 0.2s;
}

.nav-links > li:nth-child(4) {
  transition-delay: 0.3s;
}
```

Изменение этого шаблона с помощью простого CSS оказалось бы слишком утомительным. Если бы нужно было увеличивать задержку на 0,15 с, то пришлось бы последовательно вручную заменить значения на 0,15, 0,3 и 0,45 с. А если бы я решил добавить еще одну итерацию, пришлось бы вручную скопировать блок и изменить все значения. Однако при использовании цикла Sass для внесения этих изменений достаточно отредактировать математическую формулу или изменить значение счетчика итераций.

Все это каскадные таблицы стилей

Препроцессоры не меняют основ CSS. Все рассмотренное в книге по-прежнему применимо. Я не задействовал Sass ранее, поскольку хотел ясно показать, что обсуждаемые темы представляют собой основы самого языка, а не конкретного препроцессора. Вам нужно разбираться в CSS, чтобы использовать Sass. Однако Sass (или Less) значительно облегчает работу с CSS. Sass — чрезвычайно полезный инструмент. Я рекомендую познакомиться с ним.

Б.2. PostCSS

PostCSS (postcss.org) — еще один вид препроцессора. Подобно Sass и Less, он обрабатывает исходный файл и выдает обработанный файл CSS. Однако PostCSS полностью основан на плагинах. Без каких-либо плагинов итоговый файл представлял бы собой точную копию исходного файла.

Функциональность, обеспечиваемая PostCSS, полностью определяется плагинами, которые вы используете. Можете применить несколько плагинов, которые обеспечат те же функции, что и Sass, или 1–2 плагина и пропустить код через Sass и PostCSS. А при желании — даже написать собственные плагины на языке JavaScript.

Важно отметить, что PostCSS запускает плагины последовательно. При использовании нескольких плагинов порядок их запуска может повлиять на результат, поэтому, чтобы заставить PostCSS работать так, как нужно, иногда требуется пойти путем проб и ошибок. Дополнительную информацию о конфигурировании плагинов вы найдете в документации по PostCSS.

ПРИМЕЧАНИЕ

Вначале PostCSS назывался постпроцессором, поскольку обычно запускался после препроцессора. Однако со временем этот термин перестали использовать, так как он не позволял получить представление обо всех возможностях инструмента.

Б.2.1. Использование инструмента Autoprefixer

Вероятно, самый важный плагин для PostCSS — это Autoprefixer, добавляющий в CSS-код нужные префиксы поставщиков. (Подробнее о префиксах поставщиков говорится во врезке «Вендорные префиксы» в главе 5.)

Допустим, исходный код выглядит так:

```
.example {  
  display: flex;  
  transition: all 0.5s;  
  background: linear-gradient(to bottom, white, black);  
}
```

Инструмент Autoprefixer добавит дополнительные объявления, предусмотрев резервные версии для старых браузеров, в результате чего получится:

```
.example {  
  display: -webkit-box;  
  display: -ms-flexbox;  
  display: flex;  
  -webkit-transition: all .5s;  
  transition: all .5s;  
  background: -webkit-gradient(linear, left top, left bottom,  
    from(white), to(black));  
  background: linear-gradient(to bottom, white, black);  
}
```

Писать все эти префиксы поставщиков вручную было бы утомительно и чревато ошибками. Кроме того, это внесло бы много хаоса в исходный код, о котором вам, вероятно, не хотелось бы думать в процессе работы.

Вы можете сконфигурировать инструмент Autoprefixer, задав список нужных браузеров, и он добавит префиксы поставщиков там, где необходимо, для обеспечения поддержки этих браузеров. Например, конфигурирование с помощью массива `["ie >= 10", "last 2"]` гарантирует, что код будет совместим (когда это возможно) с IE10 и выше, а также двумя последними версиями прочих браузеров. Для определения того, когда эти префиксы необходимы, Autoprefixer использует последние данные из базы данных caniuse.com.

Я настоятельно рекомендую работать с Autoprefixer, даже если вы не задействуете другие плагины PostCSS. Я не включил префиксы поставщиков в приведенные в книге примеры кода, исходя из предположения, что вы позволите инструменту Autoprefixer сделать это самостоятельно.

Б.2.2. Применение cssnext

Другой популярный плагин PostCSS, вернее, набор плагинов — `cssnext` (cssnext.io). Эти плагины пытаются эмулировать будущие синтаксисы CSS, которые пока не поддерживаются всеми браузерами (а некоторые из них еще даже не доработаны в спецификации CSS). Во многом это напоминает полизаполнение для будущих функций CSS.

Многие из функций этого плагина аналогичны функциям Sass: вложенные селекторы, миксин-подобное поведение с использованием правила `@apply` и функции манипуляции цветом. Инструмент Autoprefixer также включен в этот комплект. Полный список функций плагина найдете по адресу cssnext.io/features.

Имейте в виду: некоторые из этих функций все еще находятся на ранних этапах разработки в W3C и их окончательная версия почти наверняка будет отличаться от текущей. Примените `cssnext`, если хотите получить представление о некоторых современных функциях CSS, однако не превращайте его в единственный набор правил препроцессорной обработки. По мере добавления в браузеры встроенной поддержки некоторых функций `cssnext` может оказаться сложно перейти от их обработки с помощью PostCSS к использованию непосредственно в браузере. Имеет смысл хранить правила препроцессорной обработки отдельно от правил полизаполнения.

Б.2.3. Использование cssnano

`cssnano` (cssnano.co) представляет собой минификатор на основе PostCSS. *Минификатор* удаляет из кода лишние пробелы и делает его максимально компактным, сохраняя неизменным синтаксический смысл.

ПРИМЕЧАНИЕ

Минификация не альтернатива сжатию `gzip`, которое должен применять ваш сервер. Как правило, рекомендуется минифицировать и сжимать CSS-код для ускорения его загрузки.

Существует несколько минификаторов CSS, однако целесообразнее, вероятно, было бы провести данную процедуру в ходе сборки PostCSS, а не на отдельном этапе. cssnano позволяет вам сделать именно это.

Б.2.4. Использование PreCSS

PreCSS (github.com/jonathantneal/precss) представляет собой пакет плагинов PostCSS, который обеспечивает несколько Sass-подобных функций. К ним относятся переменные \$, встроенные вычисления, циклы и миксины.

Если обработка кода с помощью препроцессора Sass и PostCSS кажется вам неэффективной, подумайте о том, чтобы заменить Sass пакетом плагинов PreCSS. Тем не менее он не идентичен Sass, поэтому, если собираетесь пойти по этому пути, обратитесь к документации PreCSS. Кроме того, инструмент относительно новый, поэтому может оказаться не столь стабильным, как Sass.

Кит Грант

CSS для профи

Перевел с английского С. Черников

Заведующая редакцией	<i>Ю. Сергиенко</i>
Руководитель проекта	<i>О. Сивченко</i>
Ведущий редактор	<i>Н. Гринчик</i>
Научный редактор	<i>С. Черников</i>
Литературный редактор	<i>Н. Рощина</i>
Художественный редактор	<i>С. Заматевская</i>
Корректоры	<i>Е. Павлович, Е. Рафалюк-Бузовская</i>
Верстка	<i>Г. Блинов</i>

Изготовлено в России Изготовитель ООО «Прогресс книга»

Место нахождения и фактический адрес: 194044, Россия, г. Санкт-Петербург,

Б. Сапсониевский пр., д. 29А, пом. 52. Тел. +78127037373

Дата изготовления 02.2019 Наименование: книжная продукция Срок годности не ограничен

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58 11 12 —

Книги печатные профессиональные, технические и научные

Импортер в Беларусь ООО «ПИТЕР М», 220020, РБ, г. Минск, ул. Тимирязева, д. 121/3, к. 214, тел./факс 208 80 01

Подписано в печать 25.01.19 Формат 70×100/16 Бумага офсетная Усл. п. л. 39,990 Тираж 1200 Заказ 896

Отпечатано в АО «Первая Образцовая типография» Филиал «Чеховский Печатный Двор»

142300, Московская область, г. Чехов, ул. Полиграфистов, 1

Сайт www.chpd.ru, E-mail sales@chpd.ru

тел. 8(499) 270-73-59

ВАША УНИКАЛЬНАЯ КНИГА

Хотите издать свою книгу? Она станет идеальным подарком для партнеров и друзей, отличным инструментом для продвижения вашего бренда, презентом для памятных событий! Мы сможем осуществить ваши любые, даже самые смелые и сложные, идеи и проекты.

МЫ ПРЕДЛАГАЕМ:

- издать вашу книгу
- издание книги для использования в маркетинговых активностях
- книги как корпоративные подарки
- рекламу в книгах
- издание корпоративной библиотеки

Почему надо выбрать именно нас:

Издательству «Питер» более 20 лет. Наш опыт — гарантия высокого качества.

Мы предлагаем:

- услуги по обработке и доработке вашего текста
- современный дизайн от профессионалов
- высокий уровень полиграфического исполнения
- продажу вашей книги во всех книжных магазинах страны

Обеспечим продвижение вашей книги:

- рекламой в профильных СМИ и местах продаж
- рецензиями в ведущих книжных изданиях
- интернет-поддержкой рекламной кампании

Мы имеем собственную сеть дистрибуции по всей России, а также на Украине и в Беларуси. Сотрудничает с крупнейшими книжными магазинами.

Издательство «Питер» является постоянным участником многих конференций и семинаров, которые предоставляют широкую возможность реализации книг.

Мы обязательно проследим, чтобы ваша книга постоянно имелась в наличии в магазинах и была выложена на самых видных местах.

Обеспечим индивидуальный подход к каждому клиенту, эксклюзивный дизайн, любой тираж.

Кроме того, предлагаем вам выпустить электронную книгу. Мы разместим ее в крупнейших интернет-магазинах. Книга будет сверстана в формате ePub или PDF — самых популярных и надежных форматах на сегодняшний день.

Свяжитесь с нами прямо сейчас:

Санкт-Петербург – Анна Титова, (812) 703-73-73, titova@piter.com
Москва – Сергей Клебанов, (495) 234-38-15, klebanov@piter.com





КНИГА-ПОЧТОЙ



ЗАКАЗАТЬ КНИГИ ИЗДАТЕЛЬСКОГО ДОМА «ПИТЕР» МОЖНО ЛЮБЫМ УДОБНЫМ ДЛЯ ВАС СПОСОБОМ:

- на нашем сайте: **www.piter.com**
- по электронной почте: **books@piter.com**
- по телефону: **(812) 703-73-74**

ВЫ МОЖЕТЕ ВЫБРАТЬ ЛЮБОЙ УДОБНЫЙ ДЛЯ ВАС СПОСОБ ОПЛАТЫ:

-  Наложным платежом с оплатой при получении в ближайшем почтовом отделении.
-  С помощью банковской карты. Во время заказа вы будете перенаправлены на защищенный сервер нашего оператора, где сможете ввести свои данные для оплаты.
-  Электронными деньгами. Мы принимаем к оплате Яндекс.Деньги, Webmoney и Qiwi-кошелек.
-  В любом банке, распечатав квитанцию, которая формируется автоматически после совершения вами заказа.

ВЫ МОЖЕТЕ ВЫБРАТЬ ЛЮБОЙ УДОБНЫЙ ДЛЯ ВАС СПОСОБ ДОСТАВКИ:

- Псылки отправляются через «Почту России». Отработанная система позволяет нам организовывать доставку ваших покупок максимально быстро. Дату отправления вашей покупки и дату доставки вам сообщат по e-mail.
- Вы можете оформить курьерскую доставку своего заказа (более подробную информацию можно получить на нашем сайте www.piter.com).
- Можно оформить доставку заказа через почтоматы, (адреса почтоматов можно узнать на нашем сайте www.piter.com).

ПРИ ОФОРМЛЕНИИ ЗАКАЗА УКАЖИТЕ:

- фамилию, имя, отчество, телефон, e-mail;
- почтовый индекс, регион, район, населенный пункт, улицу, дом, корпус, квартиру;
- название книги, автора, количество заказываемых экземпляров.

- БЕСПЛАТНАЯ ДОСТАВКА:**
- курьером по Москве и Санкт-Петербургу при заказе на сумму **от 2000 руб.**
 - почтой России при предварительной оплате заказа на сумму **от 2000 руб.**

CSS

ДЛЯ ПРОФИ

Кит Грант

Как вы понимаете, что зашли на хороший сайт?

Это происходит практически мгновенно, с первого взгляда.

Такие сайты привлекают внимание картинкой — отлично выглядят, а кроме этого они интерактивны и отзывчивы. Сразу видно, что такую страничку создавал CSS-профи, ведь именно каскадные таблицы стилей (CSS) отвечают за все наполнение и оформление сайта — от расположения элементов до неувимых штрихов. Дело за малым — стать CSS-профи, а для этого придется разобраться в принципах CSS, научиться воплощать в жизнь идеи дизайнеров, не забывать о таких важных «мелочах», как красиво подобранный шрифт, плавные переходы и сбалансированная графика.

Перед вами прямой путь в высшую лигу веб-разработки. Книга «CSS для профи» подарит вам не только свежие идеи, но и вдохновит на подвиги, а облегчить этот тернистый путь помогут новейшие технические достижения — адаптивный дизайн, библиотеки шаблонов и многое другое.

Кит Грант — опытный веб-разработчик, создающий и поддерживающий сайты премиум-класса. Его шедевр — сайт Нью-Йоркской фондовой биржи.



Заказ книг:

тел.: (812) 703-73-74
books@piter.com

WWW.PITER.COM

каталог книг и интернет-магазин



[instagram.com/piterbooks](https://www.instagram.com/piterbooks)



[youtube.com/ThePiterBooks](https://www.youtube.com/ThePiterBooks)



vk.com/piterbooks



facebook.com/piterbooks

ВЫ НАУЧИТЕСЬ:

- Избегать распространенных ловушек, свойственных CSS
- Правильно использовать нестандартные возможности
- Пользоваться сетчатой компоновкой и flex-элементами
- Создавать адаптивные макеты для любых устройств
- Обеспечивать долговечность кода

ISBN 978-5-4461-0909-8



9 785446 110909 8