A BOOK APART

Актуальные книги для тех, кто создает сайты

Итан Маркотт

ОТЗЫВЧИВЫЙ ВЕБ-ДИЗАЙН

Предисловие Джереми Кита

Ethan Marcotte

RESPONSIVE WEB DESIGN

A Book Apart

Итан Маркотт

ОТЗЫВЧИВЫЙ ВЕБ-ДИЗАЙН

Перевод с английского Павла Миронова

Издательство «Манн, Иванов и Фербер»
Москва, 2012

ОТ НАУЧНОГО РЕДАКТОРА

Когда вы в последний раз выходили в Интернет со своего планшетника, электронной читалки или телефона? Вчера? Сегодня утром, просматривая новости за чашкой кофе?

В любом случае вы наверняка заметили, что большинство сайтов не оптимизированы под мобильные устройства: выглядят так же, как и на экране ноутбука, только гораздо мельче, и потому просматривать их не слишком удобно.

«Почему же Интернет такой неповоротливый?» — спросите вы. Но это не так: Сеть меняется каждый день. Дизайнеры уже давно научились делать сайты, которые отлично выглядят и на широкоформатном мониторе, и на экране мобильного телефона. И вот что интересно — нужно для этого не так уж много. Не стоит писать кучу кода для каждого мобильного устройства, тратить ресурсы на доработку под постоянно растущий ассортимент (попробуй за ним угнаться, когда того гляди мы будем заказывать домой продукты с экрана, встроенного прямо в холодильник).

Об этом говорит и автор «Отзывчивого веб-дизайна». Он погружает вас в мир адаптивных веб-сайтов — сайтов, которыми удобно пользоваться независимо от того, какое устройство сейчас в вашем распоряжении. Сегодня эта концепция — не прихоть, а важный тренд развития Сети. И речь идет не просто о «тянущемся» сайте, а о его «умном» приспособлении к любому устройству. Пользователю, который удобно расположился за большим экраном, вы покажете сразу несколько колонок текста. Тому же, кто зашел на ваш сайт с iPhone, — лишь тот контент, который нужен ему в дороге.

Адаптирован ли ваш сайт к мобильным устройствам? Не теряете ли вы клиентов лишь оттого, что им неудобно заходить на него с планшетника или читалки? Помогите посетителям вашего сайта — адаптируйте его для них. О том, как это лучше всего сделать, вы узнаете из книги Итана Маркотта.

Александр Сарычев, *улучшающий сайты,*
аналитик компании «ЛидМашина.ру»

ПРЕДИСЛОВИЕ

Язык способен творить чудеса. В английском языке у слова glamour («гламур», «очарование») имеются и другие значения — «магия» или «чародейство», а происходит оно от слова grammar («грамматика»). Из всех чудес, творимых языком, самое волшебное и потрясающее — способность присваивать имена.

История веб-дизайна, хоть и недолгая, уже продемонстрировала нам преобразующую силу языка. Джеффри Зельдман подарил нам термин «веб-стандарты», а Джесси Джеймс Гарретт изменил природу взаимодействия в Сети, придумав технологию Ajax.

Итан Маркотт, изобретя термин «отзывчивый веб-дизайн» (responsive web design), также сотворил своего рода волшебство. Такие технологии, как «резиновые» сетки и изображения или медиазапросы, существовали и до него, но Итан объединил их и изменил само наше представление о веб-дизайне.

Итан — прекрасный рассказчик. Он мог бы стать идеальным автором книги об отзывчивом веб-дизайне. И, что самое замечательное, он ее уже написал!

Если вы надеетесь найти здесь советы и рекомендации, которые помогут вам придать своим сайтам внешний лоск, тогда эта книга не для вас — этот маленький шедевр куда глубже.

Когда вы прочтете книгу (а это не займет у вас много времени) и приступите к работе над своим следующим проектом, обратите внимание на то, как изменится ваш подход к нему. Потому что, возможно, вы и не заметите, как содержащиеся в книге идеи, изложенные в легком, занимательном, а иногда и откровенно юмористическом тоне, повлияют на ход ваших мыслей, однако я гарантирую, что ваша работа определенно выиграет от магических пассов, которые Итан над вами сотворит.

Итан Маркотт — самый настоящий волшебник, поэтому расслабьтесь и будьте готовы поддаться его чарам.

Джереми Кит, *дизайнер и веб-разработчик,*
автор книги «HTML5 для веб-дизайнеров»

1 НАША ОТЗЫВЧИВАЯ СЕТЬ

“Есть что-то, что не любит ограждений…”
Роберт Фрост, «Починка стены»

КОГДА Я НАЧАЛ ПИСАТЬ ЭТУ КНИГУ, я вдруг понял, что понятия не имею, в каком именно виде вы будете ее читать. По крайней мере, гарантировать, что вы будете читать ее, перелистывая бумажные страницы, я не могу. Может, вы сидите за столом и читаете ее электронную версию на мониторе компьютера. Может, едете на работу и читаете ее на экране телефона или планшета. А может, даже и не читаете, а слушаете то, что вам зачитывает компьютер.

Я вообще очень мало о вас знаю.

Издательское дело переняло одну из главных особенностей Сети — ее гибкость. Дизайнер и книгоиздатель Крэйг Мод считает, что издательская деятельность быстро входит в фазу «постартефакта» (http://bkaprt.com/rwd/1/), что цифровой век, в который мы живем, диктует свои условия, и мы должны пересмотреть само понятие «книга».

Конечно, веб-дизайнеры уже в течение некоторого времени пытаются разобраться с этим. По сути, в нашей профессии собственных «артефактов» еще не было. Ведь то, что мы производим, нельзя потрогать, сохранить, передать своим детям. Но, несмотря на бесплотный характер нашей работы, посмотрите, какими терминами мы постоянно пользуемся: «шапка», «пробел», «просвет». Все эти слова пришли непосредственно из мира полиграфии: мы достали их с дальней полки, стряхнули пыль и успешно используем в нашем новом цифровом мире.

Некоторые из результатов такой «вторичной переработки» были совершенно оправданными. Что ни говори, привычка — вторая натура: переезжая в другой город или устраиваясь на новую работу, мы, конечно же, прихватываем с собой старый опыт и накладываем его на новые обстоятельства, чтобы ориентироваться в новой действительности. А поскольку веб-дизайн — довольно-таки молодая сфера деятельности, для него вполне естественно заимствовать некоторые термины из знакомого мира. История графического дизайна охватывает несколько столетий, и было бы нелепо не использовать его язык для формирования нашей отрасли.

Но мы обязаны миру полиграфии намного бóльшим, нежели слова и термины. Не все помнят об этом, но мы позаимствовали из него понятие «холст» (**рис. 1.1**). Любая работа в сфере художественного творчества начинается с выбора холста: художник использует лист бумаги или кусок ткани, скульптор выбирает каменную глыбу. Независимо от того, что намерен сделать художник, его первый творческий акт — выбор холста. Еще до первого мазка кисти или удара зубила холст задает произведению искусства параметры и форму, ширину и высоту будущей работы, определяет ее границы.

Рис. 1.1. Холст, даже пустой, создает ограничения для работы художника

Веб-дизайнеры стараются подражать этому процессу. Мы даже используем то же слово: в нашем любимом редакторе мы создаем «холст» — чистый документ с определенной шириной, высотой, параметрами и формой. Однако у веб-дизайнеров имеется существенная проблема: мы находимся на удалении от пользователя и его окна браузера со всеми свойственными только ему несоответствиями и недостатками (**рис. 1.2**). Давайте посмотрим правде в глаза: как только проект становится доступным онлайн, все начинает зависеть от человека, который его просматривает, — от выбранного им шрифта, цветопередачи монитора, формы и размера окна браузера.

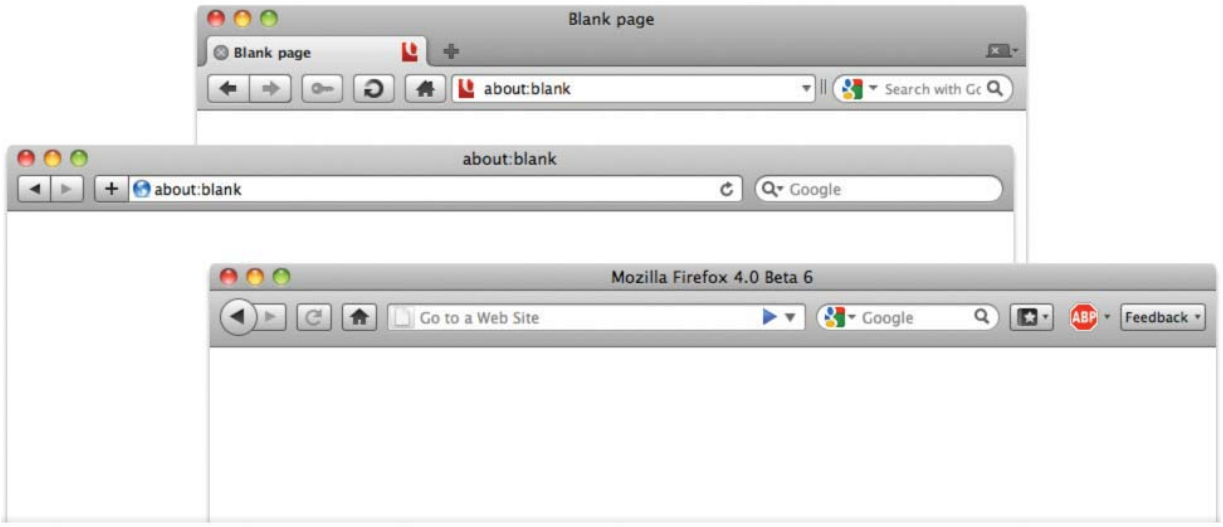

Рис. 1.2. Наш холст — окно браузера

Сталкиваясь с такой неопределенностью и гибкостью, мы начинаем устанавливать ограничения: задаем размеры шрифтов в пикселях или создаем макеты с фиксированной шириной, с учетом минимального разрешения экрана. Установка таких ограничений немного напоминает выбор того самого холста — они определяют параметры будущей работы и придают стабильность, которая защищает от изменчивости, изначально присущей Сети.

Но самое хорошее (и в то же время самое плохое) в Сети то, что она игнорирует какие-либо ограничения. Если бы у меня сегодня было дурное настроение, я бы даже сказал, что она упивается своей способностью обходить все ограничения, в которые мы пытаемся ее загнать. Это касается и параметров, которые мы задаем в наших проектах: их легко нарушить. Если ширина браузера посетителя сайта хоть чуть-чуть меньше ожидаемой минимальной ширины (**рис. 1.3**), то он столкнется с тем, что часть содержимого сайта будет обрезана, или будет вынужден пользоваться для просмотра горизонтальной прокруткой. Проблемы возникают и у нас, и у наших клиентов (**Рис. 1.4**), и потому минимальное разрешение экрана заставляет крайне внимательно относиться к размещению важных ссылок или

элементов: они могут просто-напросто уйти из области просмотра, зависящей не от наших предпочтений, а от предпочтений пользователя.

Рис. 1.3. Даже небольшое отклонение от «идеальных» параметров может негативно сказаться на впечатлении пользователя…

Рис. 1.4. …а также на самом нашем бизнесе и отношении клиентов. («Что скрывается за буквами Reg?» — спросите вы. А это просто обрезанная ссылка на страницу регистрации.)

А ТЕПЕРЬ ПРИСТЕГНИТЕ РЕМНИ

Более десяти лет назад Джон Олсопп написал статью A Dao of Web Design («Дао веб-дизайна») (http://bkaprt.com/rwd/3/), и если вы не читали ее раньше, то просто обязаны прочитать сейчас (серьезно, я готов подождать). Это мое любимое эссе о веб-дизайне, и оно столь же актуально сейчас, как и тогда, когда его написали.

Джон считает, что:

> *Контроль, которым обладает дизайнер, когда работает с печатным носителем, и о котором мечтает, когда работает в вебе, есть попросту функция ограничений, навязываемая печатной страницей. Нам следует радоваться тому, что Сеть не имеет подобных ограничений, и создавать дизайн с расчетом на гибкость. Но вначале мы должны в полной мере оценить плюсы и минусы такого положения вещей.*

Конечно, Джон писал это во времена становления Сети, когда дизайнеры переносили принципы печатного дизайна в новую молодую среду. Но бóльшая часть сказанного актуальна и сегодня. Ведь в наши дни Сеть стала еще более гибкой.

Как бы там ни было, мы вступили в свой собственный переходный период. Разнообразие браузеров приобретает впечатляющие масштабы, а устройства становятся одновременно и миниатюрнее, и крупнее. По оценкам специалистов, в течение нескольких ближайших лет лидирующей формой доступа в Сеть станут устройства с маленькими экранами, при этом современные игровые консоли позволят работать в Сети на широкоформатных экранах. В последнее время все больше пользователей заходят в Интернет с планшетов, то есть у нас появилась еще одна форма доступа — не мобильная и не стационарная, а нечто среднее.

Короче говоря, сейчас нам приходится считаться с гораздо бóльшим количеством устройств, типов входа и разрешений. Сеть вышла за пределы мира стационарных компьютеров, и назад дороги нет.

К сожалению, ранние попытки проектирования поразительно напоминали наши старые подходы, то есть мы по прежнему пытались установить ограничения, чтобы как-то компенсировать неопределенность работы в Сети. Несколько месяцев назад моя подруга прислала мне ссылку на статью, которую она только что прочитала с помощью своего телефона:

http://www.bbc.co.uk/news/mobile/science-environment-13095307

Видите директорию /mobile/? Владельцы сайта использовали для доступа к статье с мобильных устройств отдельный адрес, установив для него ширину страницы в 320 пикселей. Посетители сайта, получившие ссылку на него через Twitter, Facebook или по почте, могут просматривать его только в таком, предназначенном для маленьких экранов виде (независимо от того, на каком устройстве они изучают материал). Для меня читать эту статью на стационарном компьютере было сплошным мучением.

Это вовсе не значит, что мобильные сайты никому не нужны или что для их создания нет никаких коммерческих оснований. Но я искренне считаю, что представление контента в зависимости от устройства — подход если не проигрышный, то, во всяком случае, нежизнеспособный. За последние несколько лет мы уже поняли, что не в состоянии угнаться за темпами развития технологий. Мы что, действительно собираемся подстраиваться под каждый новый браузер или устройство?

Нет? Тогда какие у нас есть еще варианты?

ОТЗЫВЧИВАЯ АРХИТЕКТУРА

Я всю жизнь увлекался архитектурой. Для меня как веб-дизайнера особую притягательность имеет именно то обилие ограничений, которыми, как мне кажется, наслаждаются архитекторы: каждый этап архитектурного процесса — от эскиза до плана, от фундамента до фасада — неуклонно становится все более жестким и все менее изменчивым. Английский архитектор Кристофер Рен однажды написал, что архитектура устремлена в

вечность, и в этих словах заключена великая истина, ведь творческие решения архитектора останутся неизменными на десятилетия, если не на века.

Проведя лишь день наедине с Internet Explorer, начинаешь думать, что такое постоянство действительно прекрасно.

Однако в последние годы возникла относительно новая дисциплина, которая получила название «отзывчивая архитектура» и которая бросила вызов незыблемости, лежащей в основе архитектуры как таковой. Это очень молодая дисциплина, но как более интерактивная форма она уже громко заявила о себе в нескольких направлениях.

К примеру, проводятся эксперименты с поверхностями, которые реагируют на голос (http://bkaprt.com/rwd/5/), и с жилыми пространствами, которые могут трансформироваться, подстраиваясь под пользователей (http://bkaprt.com/rwd/6/). Не так давно придумана технология «умного» стекла, которое по желанию клиента, решившего отгородиться от внешних раздражителей, становится матовым (**Рис. 1.5**). А одна немецкая дизайнерская компания, используя автоматические системы и эластичные материалы, создала «стену», способную изгибаться, расширяться и менять форму, когда к ней приближается человек (**Рис. 1.6**).

Рис. 1.5. То видно, то не видно: «умное» стекло может автоматически становиться матовым

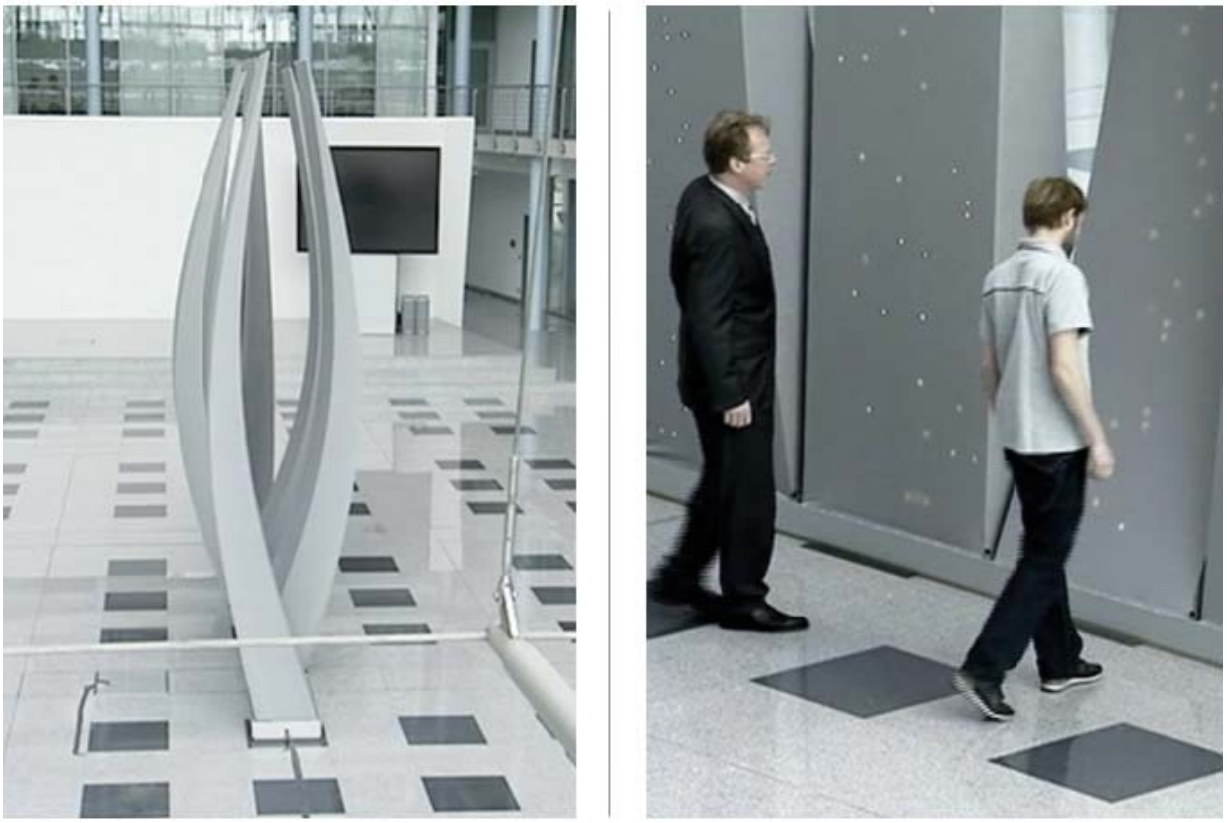

Рис. 1.6. Это не просто привлекательная художественная инсталляция. Стена действительно может чувствовать присутствие человека и реагировать на его приближение

Вместо того чтобы создавать пространства, которые влияют на поведение находящихся в них людей, приверженцы нового подхода предлагают пространства, взаимодействующие с человеком.

ПУТЬ ВПЕРЕД

Меня восхищает то, что архитекторы пытаются преодолеть ограничения, изначально присущие их деятельности. Веб-дизайнеры же, сталкиваясь с многообразием новых устройств и окружений, вынуждены преодолевать ограничения, которые мы сами и наложили на свойственную Сети гибкость.

Мы должны пойти другим путем.

Вместо того чтобы создавать отдельный дизайн для каждого вновь появляющегося устройства или браузера, мы должны относиться к ним как к разным проявлениям одного и того же

дизайна. Другими словами, мы должны создавать сайты, которые будут не только более гибкими, но и лучше адаптируемыми к устройствам отображения.

Короче говоря, нам следует практиковать отзывчивый веб-дизайн. Мы можем воспользоваться присущей Сети гибкостью, не отказываясь при этом от необходимого нам контроля. Все, что нам для этого нужно, — внедрить в нашу работу технологии, основанные на стандартах, и несколько изменить собственное отношение к веб-дизайну.

Ингредиенты

Итак, что же нужно для создания отзывчивого дизайна? Если мы говорим о разработке макета страницы, нам потребуются три основных компонента:

1. **Гибкий макет на основе сетки (flexible, grid-based layout).**
2. **Гибкие изображения (flexible images).**
3. **Медиазапросы (media queries), модуль спецификации CSS3.**

В следующих трех главах мы последовательно рассмотрим эти элементы, которые и сделают наш подход к веб-дизайну более отзывчивым. В процессе изучения мы создадим дизайн, способный адаптироваться к ограничениям окна браузера или устройства, на котором он будет просматриваться, то есть дизайн, практически полностью отвечающий потребностям пользователя.

Но прежде чем мы нырнем в эти глубины, я должен вас предупредить: я фанат научной фантастики. Обожаю лазерные пистолеты, андроидов и летающие авто, а также фильмы и сериалы, в которых всего этого в изобилии. Причем, если честно, качество этих фильмов меня не сильно заботит. Неважно, снял ли фильм Кубрик или его бюджет не превысил суммы, которую я обычно трачу на обед, но если там есть хоть один космический корабль — я счастлив.

Во всех научно-фантастических фильмах, хороших или плохих, есть любимый авторами данного жанра сюжетный прием: тайный робот. Вы наверняка видели хоть один из подобных

фильмов. Они всегда начинаются с того, как группа мужественных авантюристов во главе с честным героем, вооруженным содержательными остротами и/или непреклонной решимостью, отправляется на битву с неким злом. Но в их ряды затесался… тайный робот (звучит зловещая музыка). Это хитрое, дьявольски бездушное существо, сделанное из холодной стали и еще более холодных расчетов, но похожее на человека, и имеет оно одну четкую и подлую цель: подорвать нашу героическую группу изнутри.

Разоблачение робота — это кульминация всего фильма. Ясное дело, вы с самого начала знаете, кто герой, а кто робот-шпион. Что касается остальных персонажей, то приходится терзаться в догадках: кто же из них человек, а кто — тоже робот?

Лично для меня это никогда не было проблемой. Я, конечно, не говорю о Джонни 5 и С-3РО[1], на которых стоило только взглянуть, чтобы понять, что они явно не люди. Я имею в виду тех, кто скрывает свою сущность под синтетической кожей. Итак, я взял дело в свои руки: чтобы хоть как-то помочь решить эту проблему и научиться отличать друзей из крови и плоти от железных врагов, я спроектировал небольшой сайт под названием Robot or Not («Робот или нет») (**Рис. 1.7**).


robot-or-not-responsive.png @ 30.5% (R...
ROBOT OR NOT?
YOU CAN NEVER BE TOO SURE.
THE BOT BLOG
21
ROGUES' GALLERY
PROBABLY BOTS:
MAYBE NOTS:
30.52%
Doc: 8.43M/8.43M

Рис. 1.7. Дизайн сайта Robot or Not во всей красе

Согласен, может, этот вопрос никого, кроме меня, не волнует.

Но это на самом деле и неважно. На этом примере я просто покажу вам, как именно делается отзывчивый сайт. На протяжении следующих нескольких глав мы с вами будем разрабатывать сайт Robot or Not вместе, используя гибкие сетки, гибкие изображения и медиазапросы.

Возможно, вас не сильно увлекло мое повествование. А может быть, вы уже устали от моей болтовни и хотите увидеть готовый продукт. Если так, тогда просто введите в адресной строке и попробуйте его, как говорится, на ощупь.

Вы все еще здесь? Чудесно. Тогда начинаем.

1 Роботы, герои фильма «Короткое замыкание» и саги о «Звездных войнах» соответственно. *Здесь и далее прим. пер.*

2 ГИБКАЯ СЕТКА

ОДИН МОЙ ПРЕПОДАВАТЕЛЬ в колледже как-то сказал, что любое художественное действие — музыкальное, литературное или изобразительное — можно считать ответом на действие, ему предшествующее. Режиссеры шестидесятых сняли фильмы «Бонни и Клайд» и «Выпускник» в ответ на старые голливудские картины, такие как, например, «Звуки музыки». В «Потерянном рае» Джон Мильтон фактически помещает своих литературных предшественников в декорации ада — и это вряд ли можно считать тонкой насмешкой над их поэтическими идеалами. И если бы не музыка Дюка Эллингтона и Бенни Гудмена, Чарли Паркер, возможно, никогда бы и не затевал своих безумных экспериментов с бибопом.

Люди искусства всегда спорили друг с другом. Это в первую очередь касается художников-модернистов середины XX века. Модернисты смотрели на творческое наследие предшественников — романтиков конца XIX века — с некоторым, мягко говоря, презрением. Для них искусство романтиков было перегружено

всей этой чепухой — бесполезным украшательством, которое сводило на нет художественную ценность произведения и не позволяло должным образом донести до зрителя его смысл (**рис. 2.1**).

Рис. 2.1. Модернисты провозглашали отрыв от чрезмерно разукрашенного реализма Уильяма Блейка и Эжена Делакруа и переход к более рациональному подходу Ханса Хофманна и Йозефа Мюллер-Брокманна

Реакция модернистов проявлялась различными способами и охватывала практически все виды искусства. Так, в живописи это означало сведение картин до экспериментов с линиями, формой и цветом. Графические дизайнеры того времени, такие как Ян Чихольд, Эмиль Рудер и Йозеф Мюллер-Брокманн, популяризировали понятие типографской, или модульной, сетки — рациональной системы колонок и рядов, в которые можно было поместить модули с контентом (**Рис. 2.2**). А благодаря дизайнерам Хою Виню и Марку Болтону нам удалось адаптировать эту старую концепцию к потребностям современного веб-дизайна.

Рис. 2.2. Типографская сетка, использующаяся для размещения содержимого и определения размеров страницы, — это мощный инструмент, помогающий и дизайнеру, и читателю

В книге Grid Systems in Graphic Design («Системы сеток в графическом дизайне») Мюллер-Брокманн назвал этот процесс «созданием типографского пространства на странице», то есть разметкой сетки пропорционально размеру чистого листа бумаги.

Но графический дизайн отличается от веб-дизайна одним ключевым моментом: размерами страницы. Наш же холст — окно браузера — может принимать любую форму и размеры в соответствии с прихотями читателя или размерами устройств, на которых этот холст отображается.

```
#page {
  width: 960px;
  margin: 0 auto;
}
```

То есть мы создали элемент в разметке, задали его фиксированную ширину в CSS и расположили на странице по центру. Если же мы решили создать гибкую сетку, мы должны перевести дизайн, созданный в Photoshop (**рис. 2.3**), во что-то более «резиновое», более *пропорциональное*.


robot or not responsive.psd @ 29.3% (...
0
200
400
600
800
1000
ROBOT OR NOT?
YOU CAN NEVER BE TOO SURE.
THE BOT BLOG
21
ROGUES' GALLERY
PROBABLY BOTS:
MAYBE NOTS:
29.25%

Рис. 2.3. Созданный в Photoshop макет выглядит достаточно привлекательным, в отличие от сетки. Как можно сделать ее более гибкой?

С чего же начать?

ГИБКИЕ ШРИФТЫ

Чтобы ответить на этот вопрос, давайте сыграем в одну ролевую игру. Нет-нет, можете убрать реквизит, я говорю о чем-то более практичном, не имеющем отношения к игрищам «толкиенистов».

Представьте на мгновение, что вы разработчик пользовательских интерфейсов. (Если вы и так разрабатываете пользовательские интерфейсы, то представьте себя еще и в пиратской шляпе.) Дизайнер из вашей команды попросил вас преобразовать простой макет в разметку и CSS. Вы бросаете быстрый взгляд на макет, который он вам прислал (**рис. 2.4**).

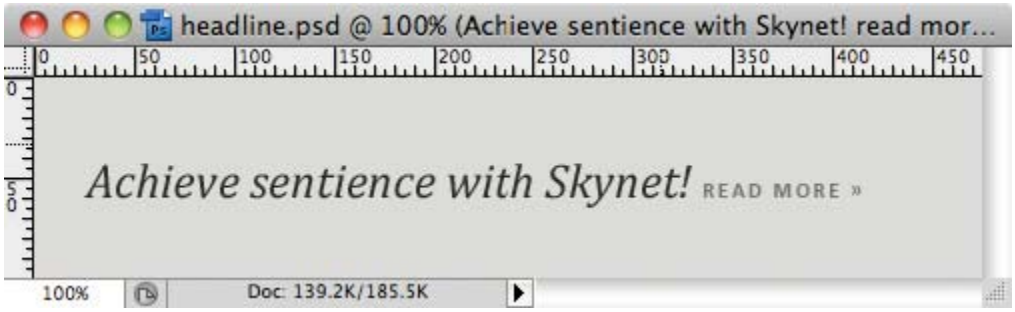

Рис. 2.4. Эскиз для нашего упражнения. По-хорошему, повторить его — минутное дело

Содержимое достаточно скромное, но даже небольшие проекты требуют пристального внимания к мелочам. Итак, вы углубляетесь в изучение дизайна. Оценив типы контента в макете, вы пишете следующий HTML-код:

```
<h1>Achieve sentience with Skynet! <a
href="#">Read More &raquo;</a></h1>
```

Заголовок с включенной в него ссылкой — прекрасная основа для семантической разметки, не правда ли? После обнуления стилей вы получаете в браузере следующий результат (**рис. 2.5**). По чуть-

чуть продвигаемся вперед. Теперь мы можем начать добавлять свой стиль оформления. Давайте впишем в элемент `body` некоторые базовые правила:

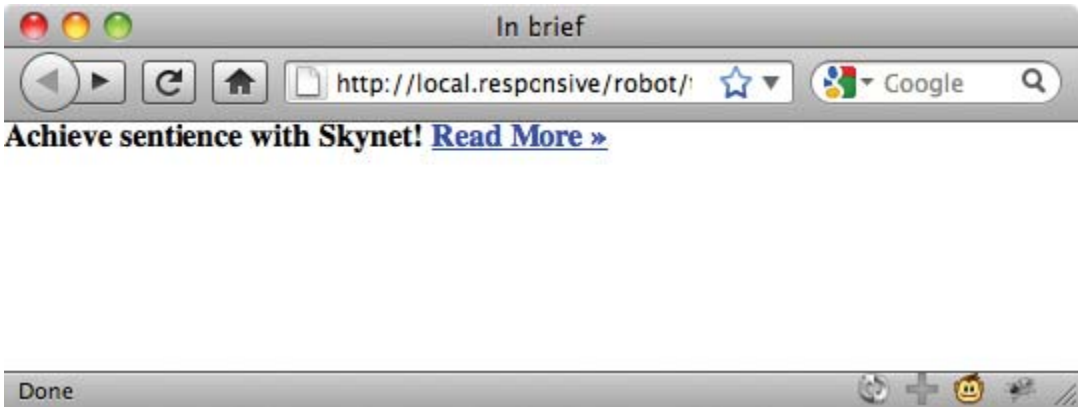

Рис. 2.5. Разметка без стилей. Именно так создается мечта (и веб-сайт)

```
body {
  background-color: #DCDBD9;
  color: #2C2C2C;
  font: normal 100% Cambria, Georgia, serif;
}
```

Ничего особенного: светло-серый фон (`#DCDBD9`) для всего документа и черный текст (`#2C2C2C`). И конечно, характеристики шрифта — жирность (по умолчанию обычная, `normal`) и семейство шрифтов с засечками (`Cambria, Georgia, serif`).

Вы, вероятно, заметили, что кегль (размер шрифта) был установлен `100%`. Поступив таким образом, мы привели базовый кегль к величине, принятой в браузере по-умолчанию, который в большинстве случаев составляет 16 пикселей. Теперь мы всегда сможем изменить кегль по отношению к этой относительной базовой величине с помощью единиц измерения `em`. Но прежде чем мы это сделаем, давайте посмотрим, что у нас уже получилось (**рис. 2.6**).

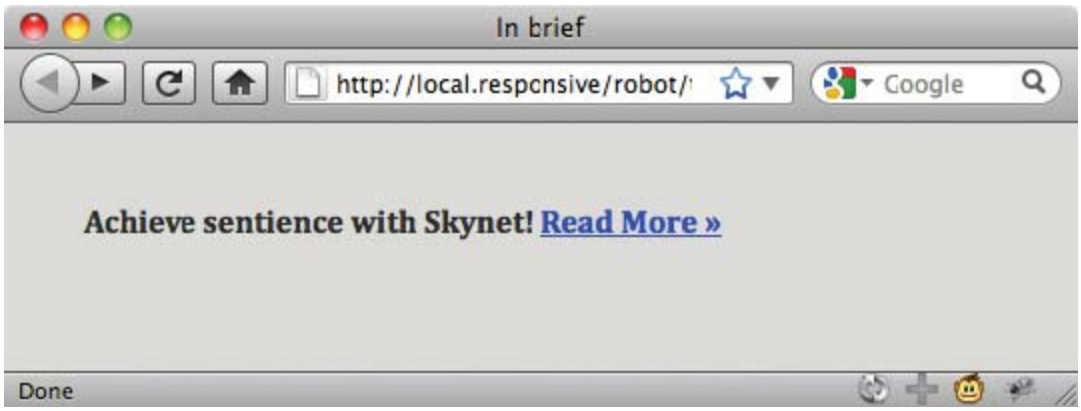

Рис. 2.6. Применив одно простое правило CSS, мы можем придать эскизу несколько другой вид

Удивлены, почему `h1` не выглядит как нормальный заголовок? Мы используем обнуление стилей, нивелирующее стили браузера по умолчанию для элементов HTML, чтобы обеспечить их соответствие в различных браузерах. Лично мне больше всего нравится способ обнуления от Эрика Мейера (http://bkaprt.com/rwd/9/), но вы можете выбрать какой-нибудь другой, благо выбор сейчас достаточно большой.

В любом случае наш `h1` выглядит таким маленьким именно по этой причине: он наследует стиль `font-size: 100%`, который мы задали родительскому элементу `body`, а установленный в браузере по умолчанию кегль — 16 пикселей.

Теперь, если пиксели нас устраивают, мы можем перевести значения из оригинал-макета непосредственно в CSS:

```
h1 {
  font-size: 24px;
  font-style: italic;
  font-weight: normal;
}

h1 a {
  color: #747474;
  font: bold 11px Calibri, Optima, Arial, sans-
```

```
serif;
  letter-spacing: 0.15em;
  text-transform: uppercase;
  text-decoration: none;
}
```

Нет ничего плохого или неправильного в определении размера текста с помощью пикселей. Но в целях нашего эксперимента давайте начнем думать пропорционально и переведем значения кегля (`font-size`) из пикселей в относительные единицы, а вместо пикселей и будем использовать знакомые нам `em`.

Контекстное восстановление

Сейчас будет немного математики: берем *целевое* значение кегля и делим на кегль (`font-size`) его контейнера, то есть контекста. В результате мы получаем желаемый кегль, выраженный в относительных и достаточно гибких единицах `em`.

Другими словами, относительный кегль можно рассчитать по следующей формуле:

```
target ÷ context = result
```

(Отвлечемся на минутку. Лично у меня слово «математика» вызывает немедленный и серьезный приступ паники. У вас тоже? Стойте, не убегайте с криками — все не так страшно. Это говорит вам человек, который заменил курс математики в колледже курсом философии. Делайте, как я: просто посчитайте все на калькуляторе и скопируйте результат в CSS. Калькуляторы — просто спасение для таких, как мы, правда?)

Итак, формула у нас есть, давайте вернемся к нашему заголовку в `24px`. Если предположить, что базовый кегль (`font-size`) элемента `body` равен 16 пикселям при `100%`, мы можем подставить эти значения непосредственно в формулу. В

результате получим отношение целевого кегля заголовка `h1` (24 пикселя, `24px`) и кегля его контекста (16 пикселей, `16px`):

```
24 ÷ 16 = 1.5
```

Так как 24 пикселя в 1,5 раза больше 16 пикселей, это значит, что кегль равен `1.5 em`.

```
h1 {
  font-size: 1.5em; /* 24px / 16px */
  font-style: italic;
  font-weight: normal;
}
```

Вуаля! Размер нашего заголовка прекрасно совпадает с размером, указанным в оригинал-макете, но при этом выражен в относительных, масштабируемых единицах (**рис. 2.7**).

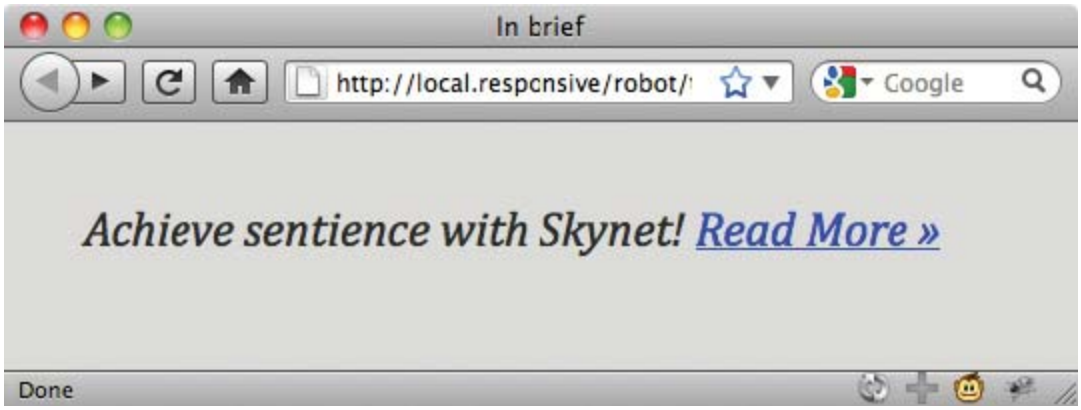

Рис. 2.7. Размер нашего заголовка правильно выражен в гибких, масштабируемых единицах em. (Но что за ерунда творится со ссылкой?)

(Обычно я оставляю комментарий с расчетами с правой стороны (`/* 24px / 16px */`). Вносить изменения становится намного проще.)

С этим закончили, давайте вернемся к нашей одинокой маленькой ссылке Read More (*узнать больше*). Хотя, если

посмотреть на **рис. 2.7**, она не такая уж и маленькая. И это проблема. Нам нужно уменьшить заданные 11 пикселей, и довольно существенно, поскольку размер его шрифта принял значение `1.5 em` от его контекста, `h1`.

И вот здесь требуется внимание. Поскольку текст заголовка установлен равным `1.5 em`, любые элементы внутри этого заголовка должны быть выражены в виде доли этого значения. Другими словами, *изменился наш контекст*.

Поэтому, чтобы установить кегль ссылки в единицах `em`, мы делим целевые 11 пикселей (`11px`) не на `16` (значение, установленное в `body`), а на `24` — кегль заголовка, наш новый контекст:

```
11 ÷ 24 = 0.458333333333333
```

Мы получили какое-то совсем некрасивое число. Может быть, самое некрасивое, которые вы сегодня видели. (Но подождите, эта глава еще не окончена.)

Вам захочется округлить его до более-менее приемлемого числа — скажем, `0.46 em`. Даже не думайте! Может, ваши глаза и устанут смотреть на `0.458333333333333`, но именно это число идеально представляет желаемый кегль в пропорциональном отношении. К тому же браузеры мастерски владеют искусством округления лишних десятичных знаков, когда преобразовывают значения в пиксели. Поэтому нужно дать им больше, а не меньше, и в конце вас будет ожидать отличный результат.

Теперь просто скопируйте это непритязательное число в CSS:

```
h1 a {
  font: bold 0.458333333333333em Calibri,
Optima, Arial, sans-serif; /* 11px / 24px */
  color: #747474;
  letter-spacing: 0.15em;
```

```
    text-transform: uppercase;
    text-decoration: none;
  }
```

Посмотрим на результат. Наша страничка закончена, она полностью соответствует желаемому дизайну, а текст задан в масштабируемых единицах `em` (**рис. 2.8**).

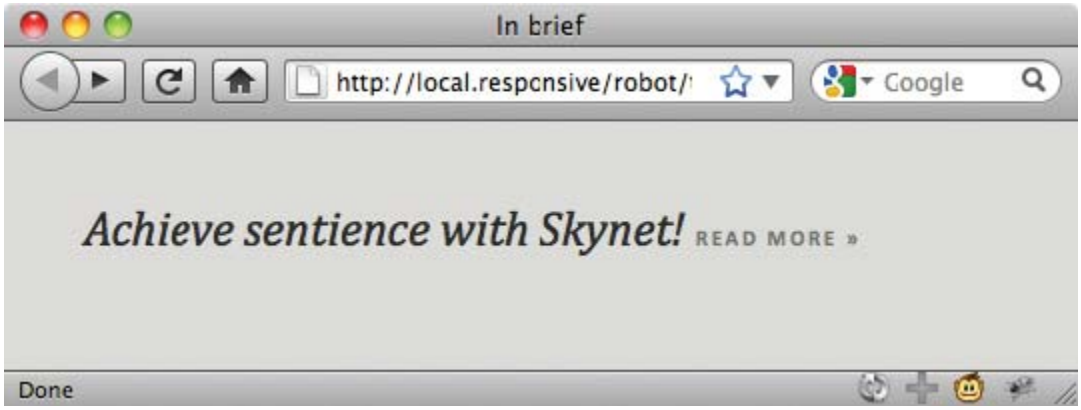

Рис. 2.8. С помощью простой математики мы создали более красивый дизайн — и без всяких пикселей

От гибких шрифтов к гибкой сетке

Вы, наверное, сейчас зеваете со скуки. Здесь ведь должна была быть глава про гибкие макеты, а этот тип Итан все талдычит про *математику* и *размеры шрифтов*. Надоело!

Но когда я впервые делал гибкую сетку, я понятия не имел, с чего начинать. Поэтому, столкнувшись с неразрешимой проблемой, я сделал то, что у меня получается лучше всего: полностью ее проигнорировал и начал работать над тем, что знаю.

Когда я понял, как устанавливать размеры текста в единицах `em`, на меня снизошло прозрение: ведь мы можем применять тот же принцип пропорционального мышления и в отношении самих макетов. Другими словами, все элементы нашей сетки — строки, колонки и накладываемые на них модули — могут быть выражены как *пропорция* содержащихся в них элементов, а не в неизменных, жестких пикселях.

И в этом нам снова поможет наша формула `target ÷ context = result`. Идем дальше.

СОЗДАНИЕ ГИБКОЙ СЕТКИ

Представьте, что дизайнер настолько впечатлен вашей быстрой и качественной работой над заголовком, что прислал вам другой макет, и теперь вам нужно написать код для блога сайта Robot or Not (**рис. 2.9**).

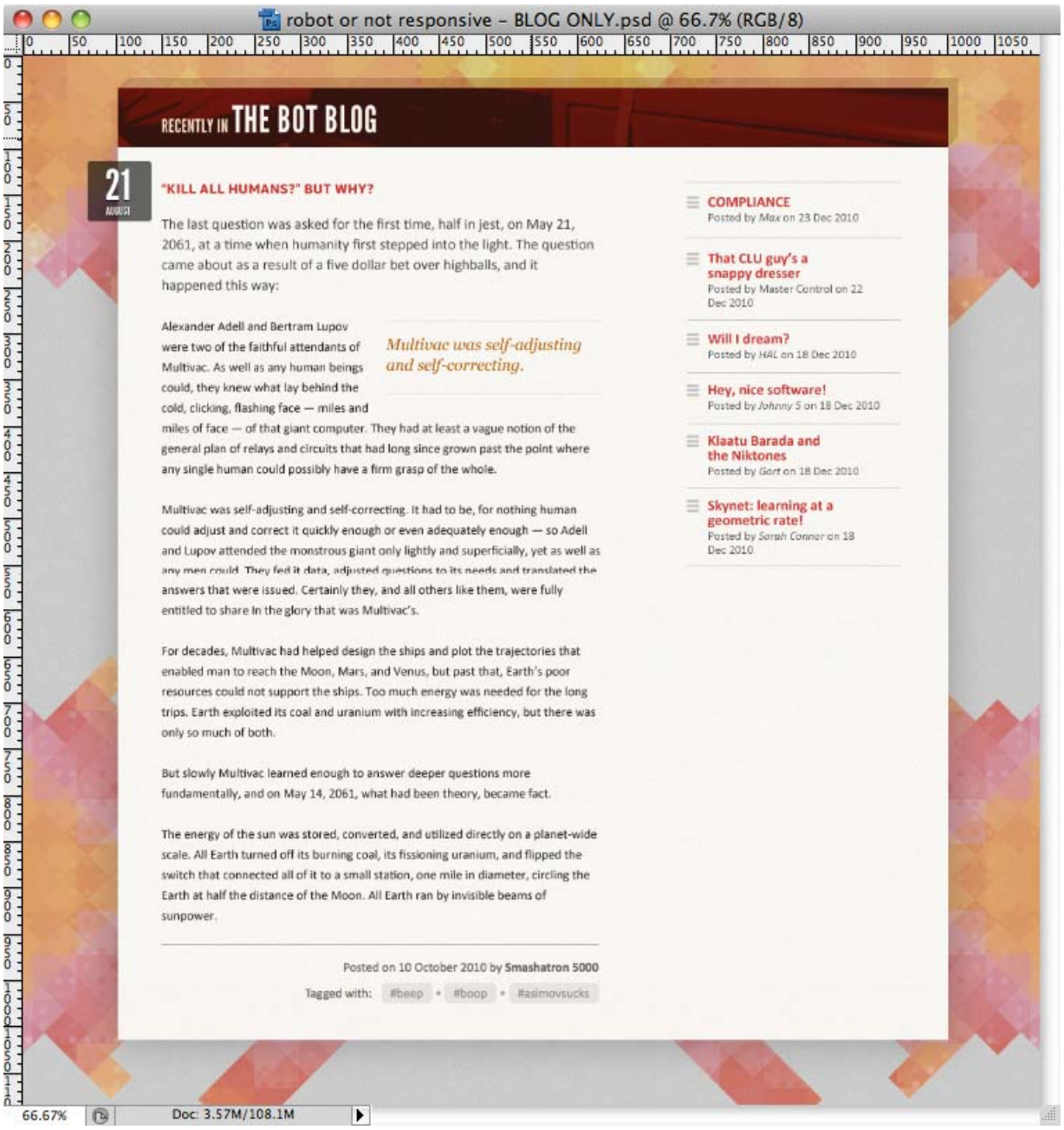

Рис. 2.9. Новое задание — превращение эскиза в гибкий макет

Плюс ко всему дизайнеру настолько понравилась работа, что он прислал краткую схему содержания страницы (**рис. 2.10**), благодаря чему мы можем не тратить время на планирование. Нужно как-то его отблагодарить.

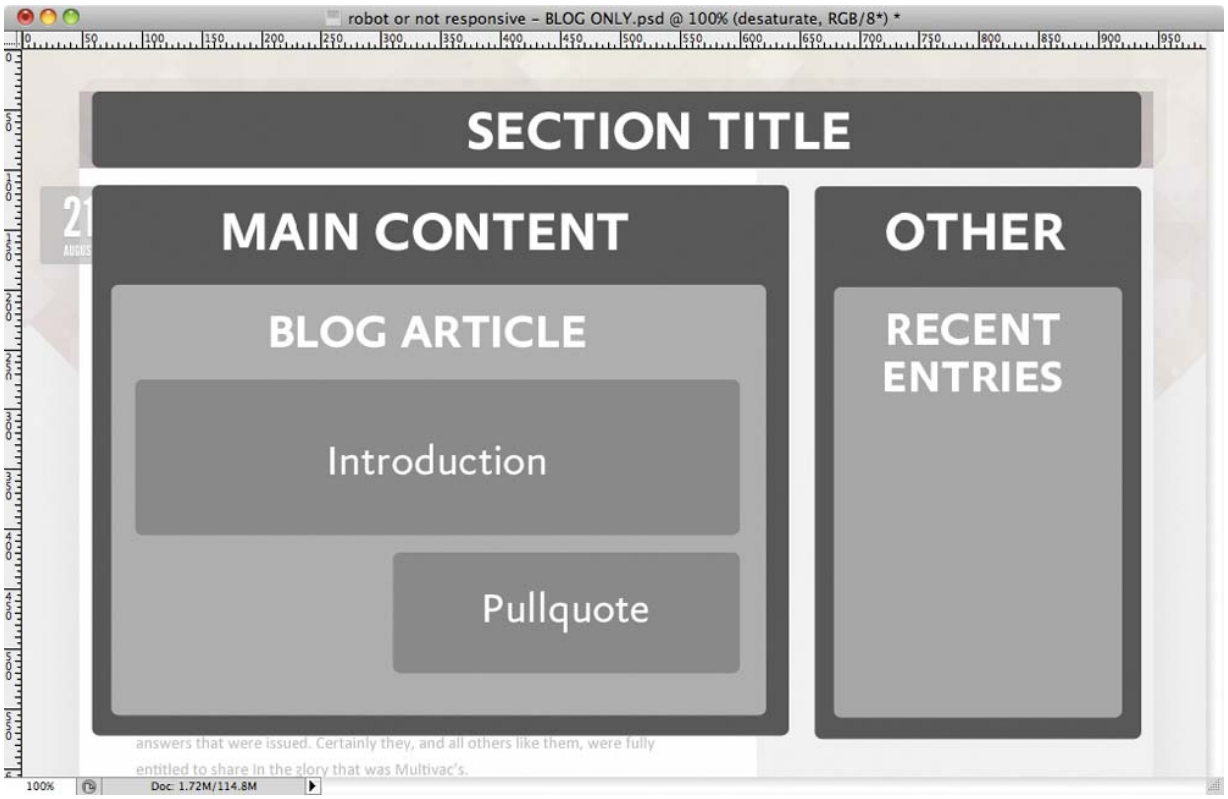

Рис. 2.10. Схема расположения элементов для нашего блога

Мы можем быстро и ловко перевести его схему в базовую структуру верстки:

```
<div id="page">
  <div class="blog section">
    <h1 class="lede">Recently in <a
href="#">The Bot Blog</a></h1>

    <div class="main">
      …
    </div><!-- /end .main -->

    <div class="other">
      …
    </div><!-- /end .other -->
```

```
    </div><!-- /end .blog.section -->
  </div><!-- /end #page -->
```

Наша разметка получилась простой и аккуратной, семантически верной и превосходно подходит для контента. Мы создали основной контейнер для всей страницы (`#page`), который, в свою очередь, содержит модуль `.blog`. Внутрь него мы поместили еще два блока: один с классом `.main` для главного содержания статьи, а второй с классом `.other` для всего остального. Звучит, конечно, не слишком поэтично, но, с другой стороны, это и не сборник стихов.

А теперь пропустим несколько этапов — как это делается на кулинарных шоу, где повар кладет в кастрюлю сырые продукты, а через минуту вынимает из духовки полностью готовую индейку. (Эта метафора прекрасно демонстрирует то, как часто я смотрю кулинарные шоу… или готовлю индейку.)

И все же предположим, что мы уже создали все CSS, связанные со шрифтами, фоновыми изображениями и всеми элементами нашего дизайна, *не имеющими отношения* к макету (**рис. 2.11**). Теперь мы можем сосредоточиться исключительно на создании «резиновой» сетки.

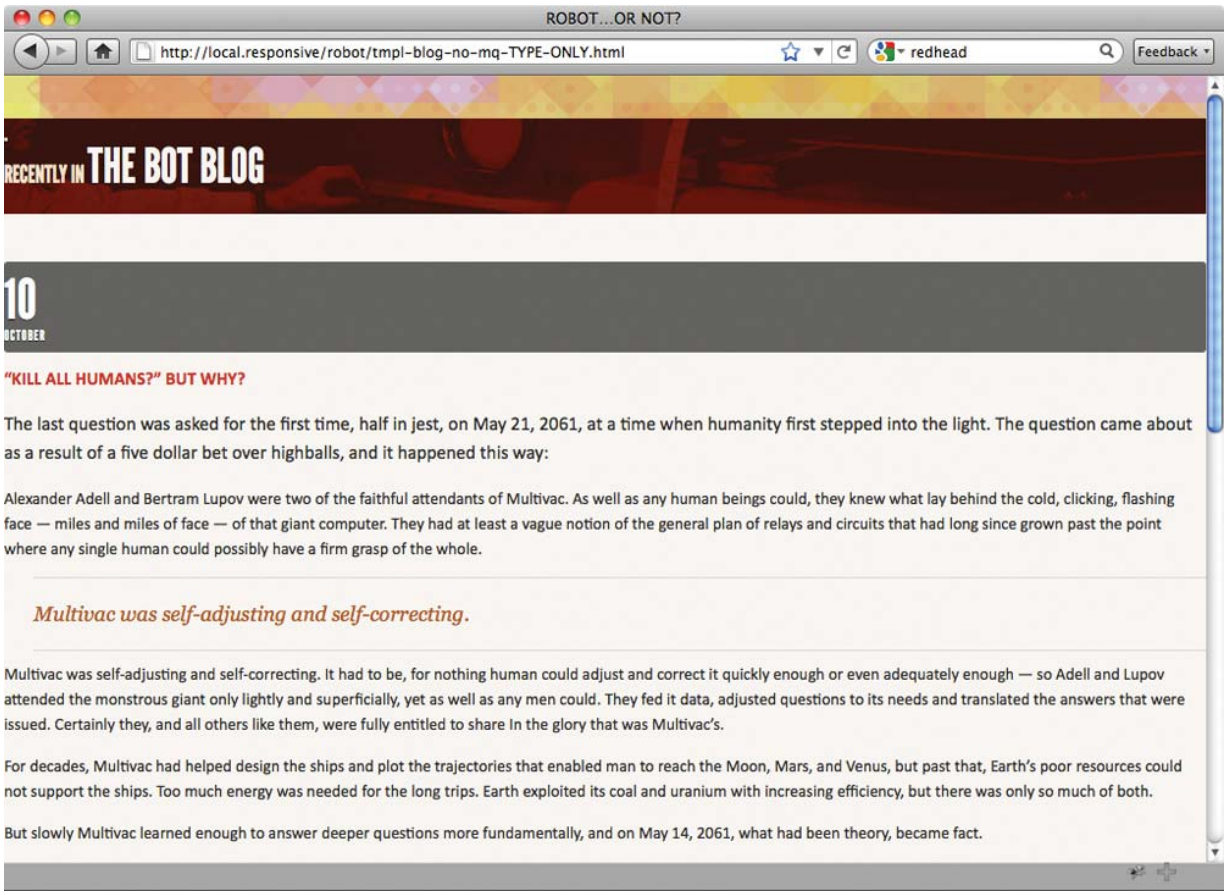

Рис. 2.11. Работа над шаблоном закончена! Если, конечно, не принимать во внимание то, как он должен выглядеть в самом конце

Так как же нам превратить эти блоки `.main` и `.other` в нужные колонки? У нас уже есть схема контента и основная разметка, теперь давайте внимательнее взглянем на физические параметры сетки в оригинальном дизайне (**рис. 2.12**).

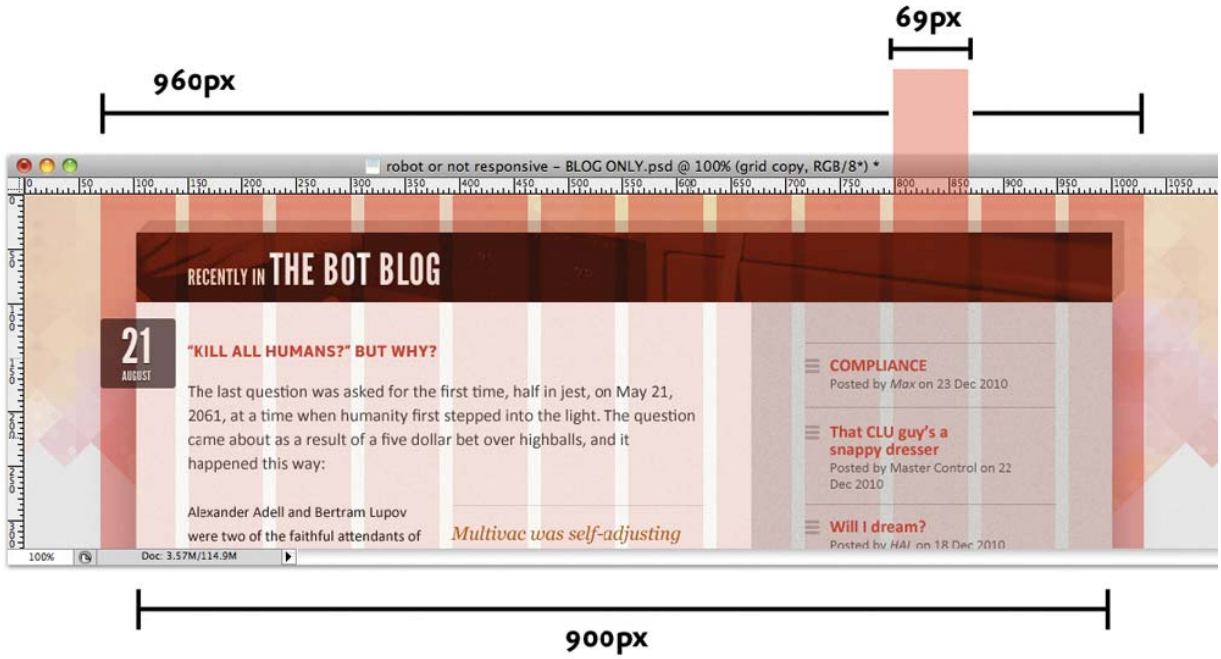

Рис. 2.12. Теперь наша страница основана на сетке!

Сама сетка разделена на 12 колонок по 69 пикселей каждая, отделенных друг от друга промежутками шириной в 12 пикселей (`12px`). В сумме колонки и промежутки дают нам полную ширину в `960px`. Сам же блог шириной 900 пикселей отцентрирован по горизонтали в пределах холста.

Вот они, детали высокого уровня. И если мы внимательно рассмотрим две колонки в блоге (**рис. 2.13**), то увидим, что ширина левой основной колонки (`.main` в нашей разметке) составляет 566 пикселей, в то время как ширина правой вспомогательной (`.other`) — только 331 пиксель.

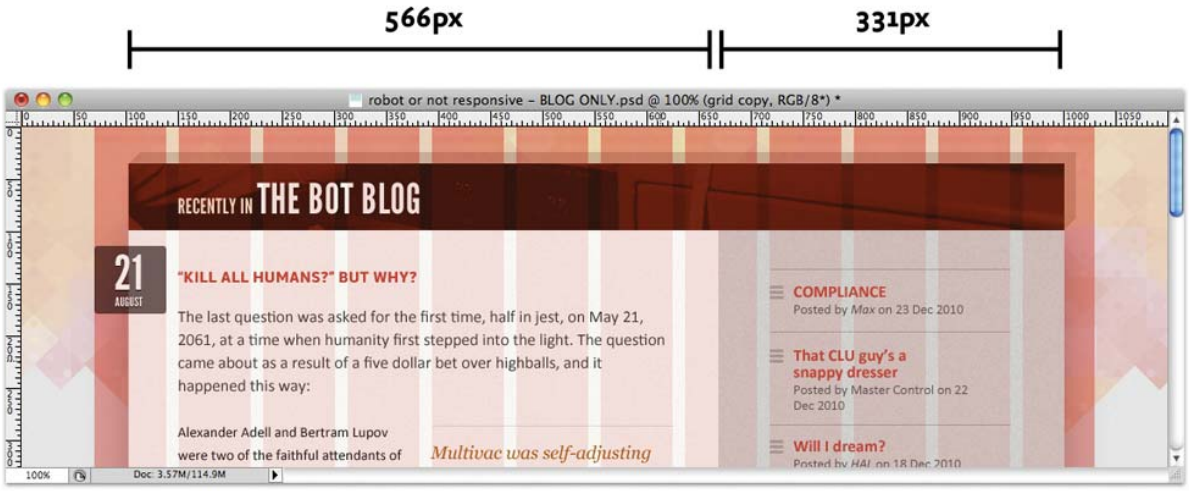

Рис. 2.13. Давайте-ка изучим детали и измерим ширину внутренних колонок

Что-то слишком много пикселей вокруг, да? И если бы пиксели нас устраивали, мы могли бы просто перенести их в CSS. (Эй! Это очень важно!)

```
#page {
  margin: 36px auto;
  width: 960px;
}

.blog {
  margin: 0 auto 53px;
  width: 900px;
}

.blog .main {
  float: left;
  width: 566px;
}

.blog .other {
```

```
    float: right;
    width: 331px;
  }
```

Отлично. Мы установили ширину `#page` в 960 пикселей, отцентрировали в ней модуль `.blog` шириной `900px`, задали ширину `.main (566)` и `.other (331)` и наконец-то разместили эти колонки рядом. Результат выглядит шикарно (**рис. 2.14**).

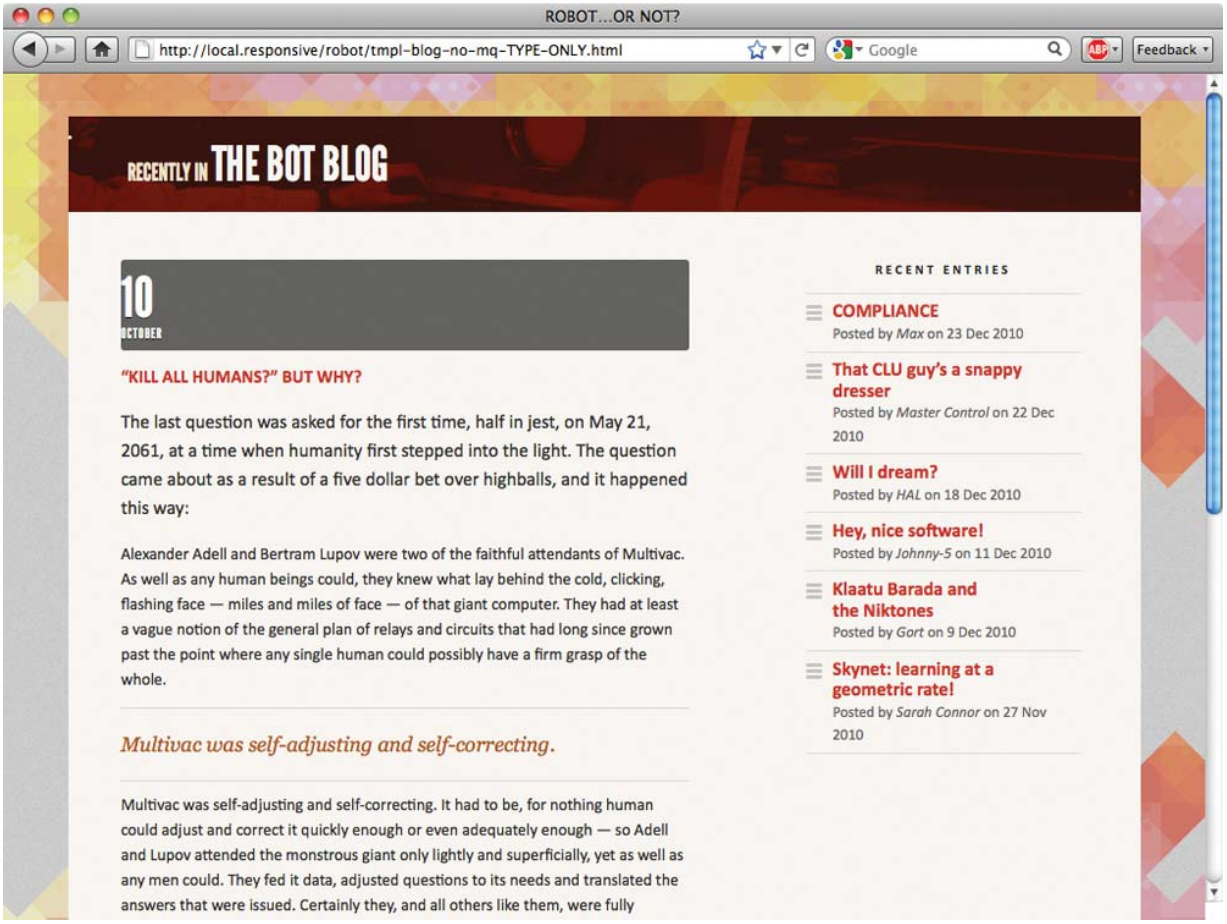

Рис. 2.14. Мы избавились от ненужных пикселей, и наш дизайн почти готов. Или нет?

И хотя наш макет идеально совпадает с оригинал-макетом, он получился совсем негибким. Фиксированная ширина в `960px` делает нашу страницу совершенно безразличной к изменениям размеров области просмотра. Иными словами, если ширина окна

будет меньше 1024 пикселей, перед читателем появится полоса горизонтальной прокрутки (**рис. 2.15**).

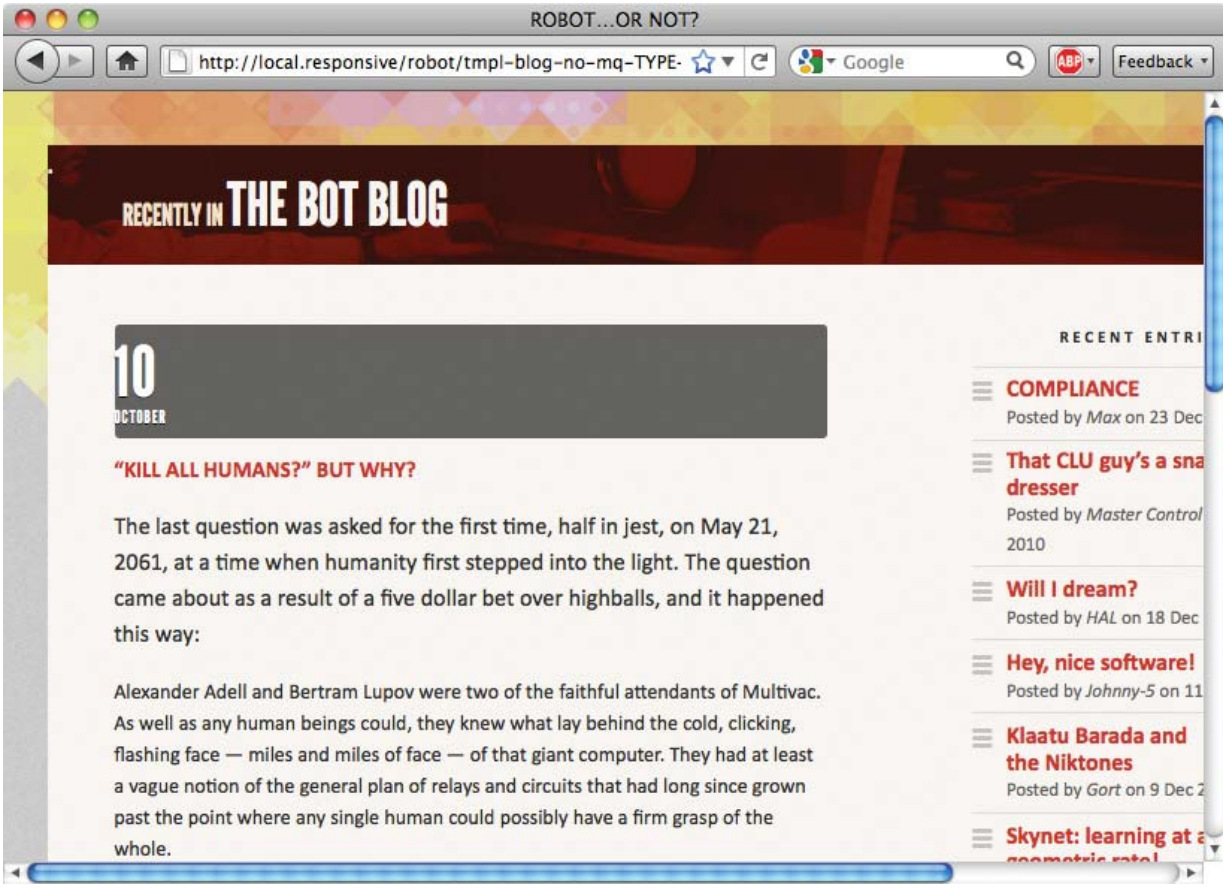

Рис. 2.15. Дизайн выглядит отлично, но он совсем негибкий. Давайте это исправим

И нас это, мягко говоря, не устраивает.

От пикселей к процентам

Вместо того чтобы переносить значения в пикселях из оригинал-макета в CSS, мы должны перевести эти размеры в относительные, пропорциональные значения. В результате мы получим сетку, которая будет трансформироваться в зависимости от области просмотра, причем оригинальные пропорции дизайна останутся неизменными.

Давайте начнем с первого элемента `#page`, который содержит наш макет, и попробуем что-нибудь с ним сделать:

```
#page {
  margin: 36px auto;
  width: 960px;
}
```

Противные, гадкие пиксели. Терпеть их не можем!
Ну ладно, не такие уж они и отвратительные. Помните: в макетах с фиксированной шириной нет ничего плохого! Но нам нужна более гибкая сетка, поэтому давайте попробуем перевести эти `960px` в проценты.

```
#page {
  margin: 36px auto;
  width: 90%;
}
```

Должен признаться, что я пришел к этим 90% без особых на то оснований, просто пробовал различные варианты в окне браузера, а затем выбрал тот, что понравился мне больше всего. Задавая значение элемента `#page` в процентах, мы создаем контейнер, который будет увеличиваться и уменьшаться в зависимости от области просмотра (**рис. 2.16**). И поскольку он отцентрирован по горизонтали, справа и слева останутся отступы по 5%.

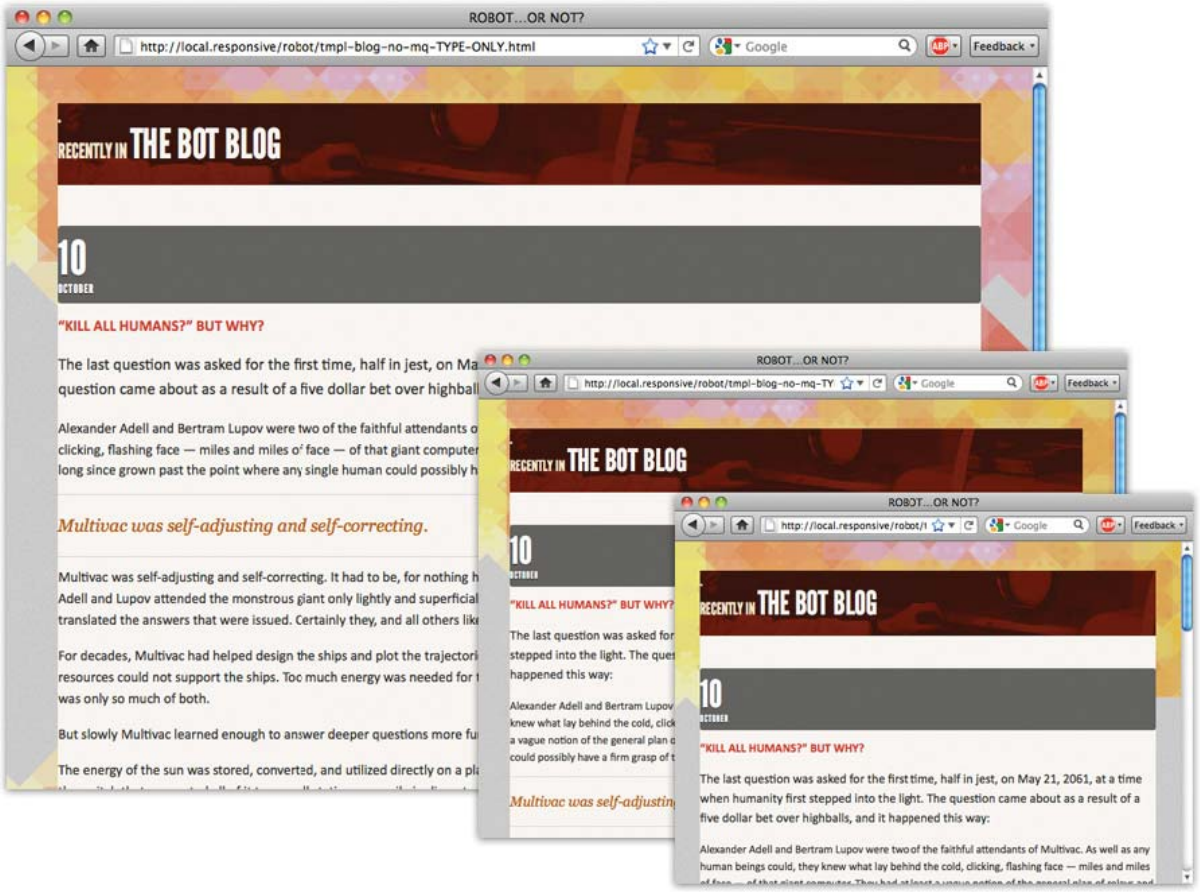

Рис. 2.16. Наш контейнер изменяется в размерах при любом изменении размера окна браузера

Пока все идет неплохо. Теперь давайте примемся непосредственно за модуль `.blog`. Чуть ранее, когда мы играли с пикселями, то установили следующее правило:

```
.blog {
  margin: 0 auto 53px;
  width: 900px;
}
```

Теперь вместо величины в пикселях мы должны выразить ширину элемента `.blog` в пропорциональных величинах: описать ее как

процент ширины содержащего его элемента. И вот здесь нам снова пригодится формула `target ÷ context = result`.

Из оригинал-макета мы знаем, что ширина нашего блога должна составлять `900px`. Теперь нам нужно представить эту ширину в относительных единицах, процентах ширины родительского элемента для элемента `.blog`. Поскольку блок `.blog` вложен в элемент `#page`, ширина которого в соответствии с оригинал-макетом составляет 960 пикселей, это и есть наш контекст.

Давайте разделим ширину `.blog` (`900`) на его контекст (`960`):

```
900 ÷ 960 = 0.9375
```

У нас получилось `0.9375`. Число выглядит не особенно впечатляюще. Но, переведя его в проценты, мы получаем `93.75%` и заносим их прямо в CSS:

```
.blog {
  margin: 0 auto 53px;
  width: 93.75%; /* 900px / 960px */
}
```

(Так же как и в случае с размерами шрифтов, я записал формулу в поле комментария справа от значения ширины (`width`). Это, разумеется, дело вкуса, но я нахожу это очень полезным.)

Итак, с двумя элементами мы разобрались. Но что делать с колонками?

```
.blog .main {
  float: left;
  width: 566px;
}
```

```
.blog .other {
  float: right;
  width: 331px;
}
```

Ширина основной колонки, которая расположена слева, составляет `566px`, ширина же левой колонки равна `331px`. Эти цифры нам тоже придется перевести в проценты. Но прежде чем подставить их в формулу, обратите внимание на то, что контекст изменился. Последний раз мы делили ширину модуля `.blog` на `960`, ширину его контейнера (`#page`). Но поскольку эти блоки вложены в `.blog`, нам нужно делить целевую ширину колонок (`566` и `331`) на ширину их нового контекста, то есть ширину `.blog` (`900`). В результате мы получаем:

```
566 ÷ 900 = .628888889
331 ÷ 900 = .367777778
```

Переведя эти значения в проценты, мы получаем в итоге `62.8888889%` для блока `.main` и `36.7777778%` для блока `.other`:

```
.blog .main {
  float: left;
  width: 62.8888889%; /* 566px / 900px */
}

.blog .other {
  float: right;
  width: 36.7777778%; /* 331px / 900px */
}
```

Вот мы и получили гибкий макет (**рис. 2.17**). При помощи небольших расчетов мы создали контейнер, выраженный в процентах, и две гибкие колонки, что дает нам макет, меняющий свои размеры в соответствии с размерами окна браузера. При этом ширина в пикселях тоже меняется, а *пропорции* дизайна остаются исходными.

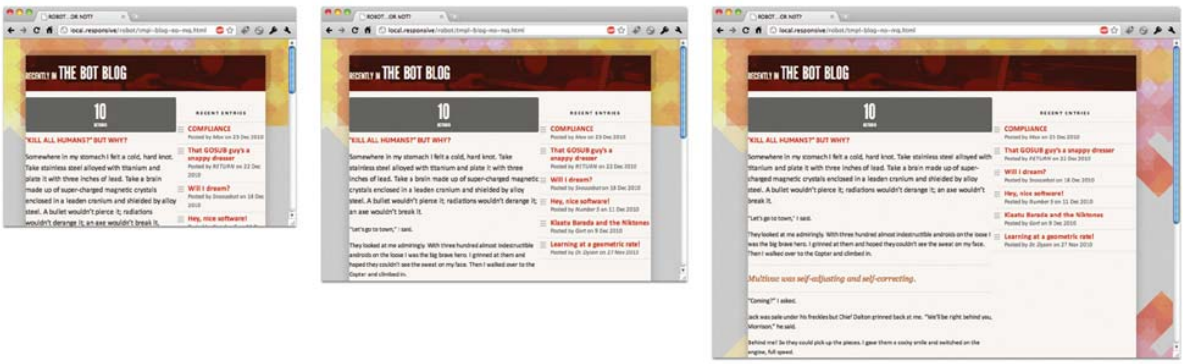

Рис. 2.17. Наша гибкая сетка готова

ГИБКИЕ ПОЛЯ И ОТСТУПЫ

Теперь, когда две колонки стоят на своих местах, можно сказать, что мы закончили с основными компонентами нашей гибкой сетки. Изумительно. Замечательно. Великолепно. И все же этого недостаточно: нас ждет работа над деталями.

Не продохнуть…

Наш дизайн уже достаточно гибок, однако он требует серьезного внимания к двум основным деталям. Название блога уехало далеко влево (**рис. 2.18**), а две колонки примыкают друг к другу без каких-либо отступов или промежутков (**рис. 2.19**). Определенно, нам нужно еще поработать.

Рис. 2.18. Наш заголовок отчаянно нуждается в полях

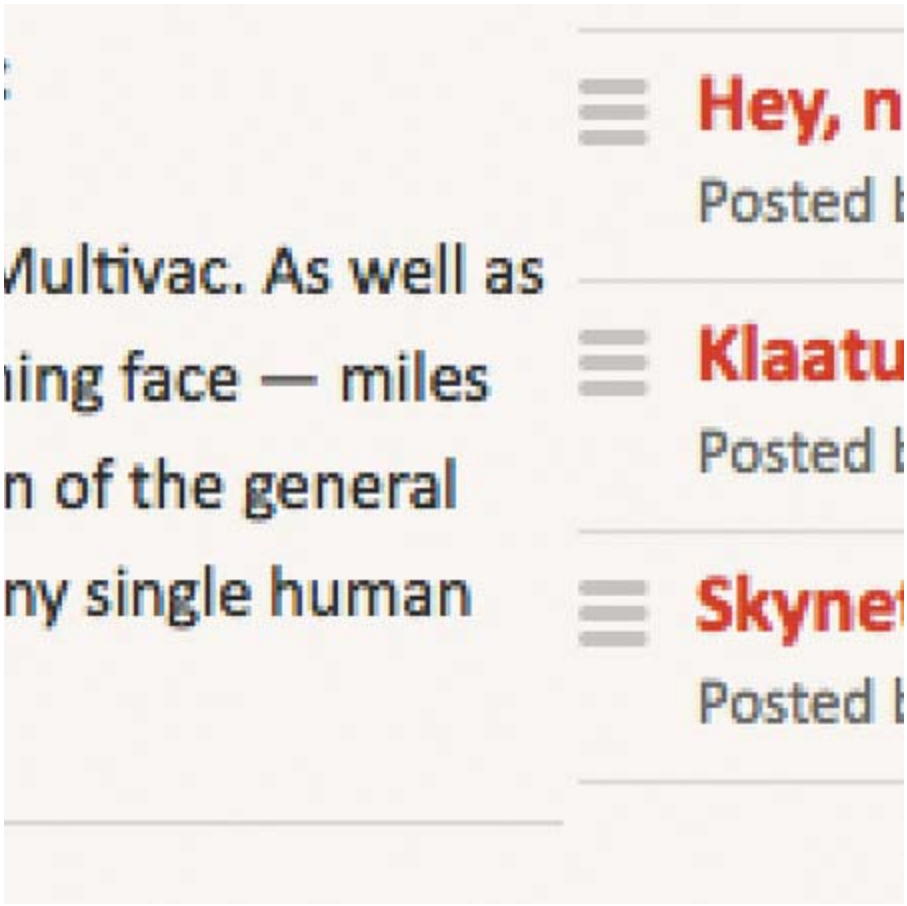

Рис. 2.19. Отступы? Мы не признаем никаких отступов!

(Конечно, на самом деле мы без них страдаем.)

Ну что ж, давайте начнем с заголовка. В оригинальном макете между началом заголовка и левым краем контейнера есть промежуток в 48 пикселей (**рис. 2.20**). Мы, конечно, *могли* бы задать левый отступ (`padding-left`) в пикселях или `em`:

```
.lede {
  padding: 0.8em 48px;
}
```

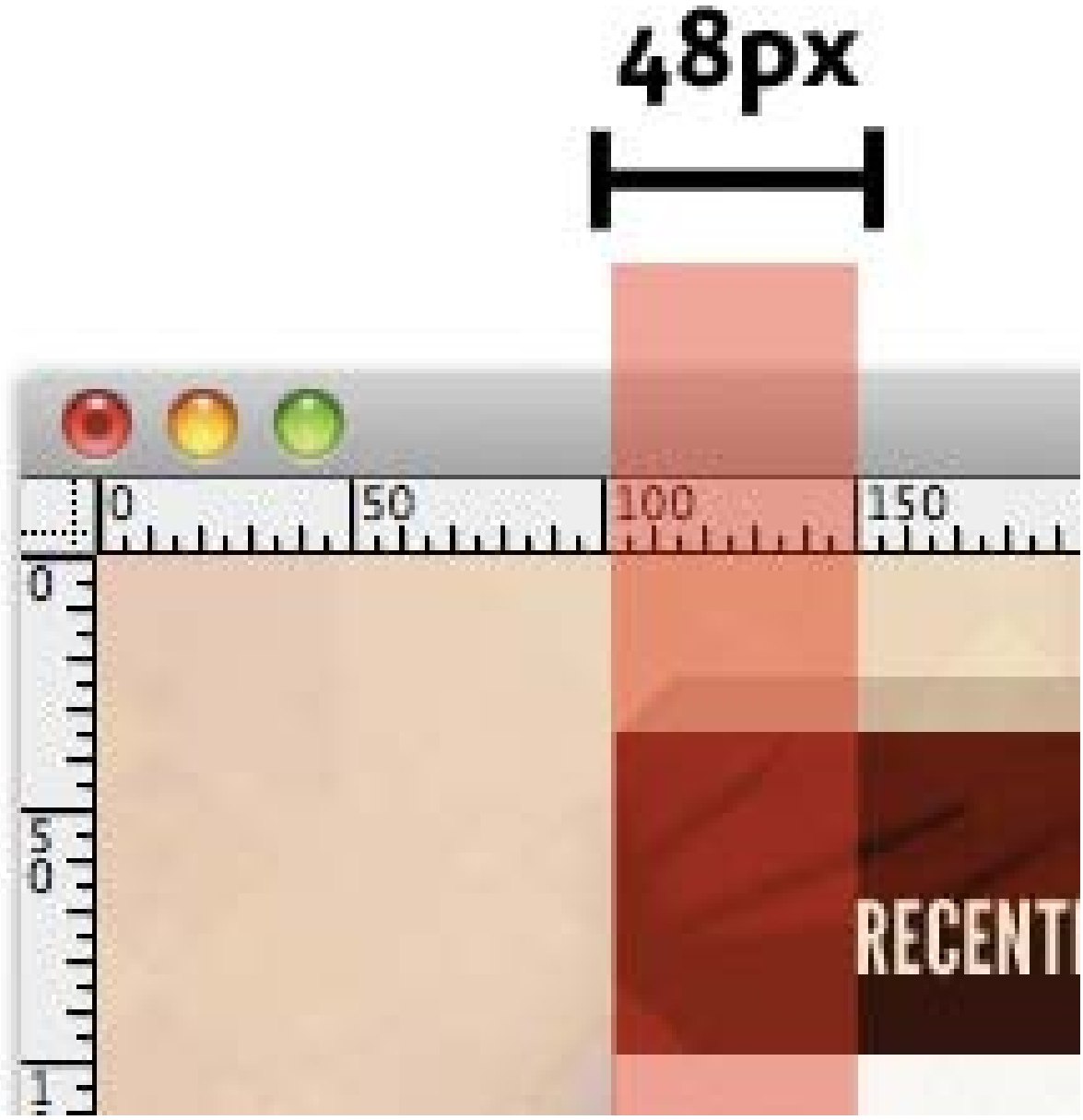

Рис. 2.20. Согласно параметрам эскиза, нам нужно задать горизонтальное поле в 48 пикселей с левой стороны заголовка

Это хорошее решение. Но это фиксированное значение левого отступа (`padding-left`) создаст промежуток, который не будет сочетаться со всей «резиновой» сеткой. И когда гибкие колонки будут становиться у́же или шире, этот отступ проигнорирует остальные пропорции дизайна, и ширина его всегда окажется 48 пикселей (`48 px`), независимо от того, насколько уменьшился или увеличился весь макет.

Так что мы не пойдем этим путем — мы создадим *гибкий отступ*. Пока что мы использовали относительные единицы измерения только в отношении ширины различных элементов, но мы можем это сделать как с полями, так и с отступами. И воспользуемся для этого нашей проверенной формулой:

```
target ÷ context = result
```

Прежде чем мы снова займемся вычислениями, хочу обратить ваше внимание на то, что контексты для гибких полей и для гибких отступов различны.

1. Задавая гибкие поля для элемента, принимайте за контекст ширину контейнера элемента.
2. Задавая гибкие отступы для элемента, принимайте за контекст *ширину самого элемента*. Подумайте о блочной модели, и эти предложения обретут смысл: мы описываем поле в отношении к ширине самого элемента.

Поскольку мы хотим определить поле заголовка, в качестве контекста мы возьмем ширину самого элемента `.lede`. Ширина заголовка нам неизвестна, поэтому мы берем ширину модуля блога, то есть `900 px`. Снова открываем калькулятор и получаем:

```
48 ÷ 900 = 0.0533333333
```

и переводим результат в:

```
.lede {
  padding: 0.8em 5.33333333%; /* 48px / 900px
*/
}
```

Наши `48px` поля теперь выражены в относительных единицах измерения, как доля ширины заголовка.

С этим расправились, идем дальше. Давайте введем понятие пробела в наш контент. Но сначала вспомним, что каждая колонка фактически содержит меньший модуль: левая колонка `.blog` содержит модуль `.article`, а правая `.other` — список `.recent-entries` (**рис. 2.21**).

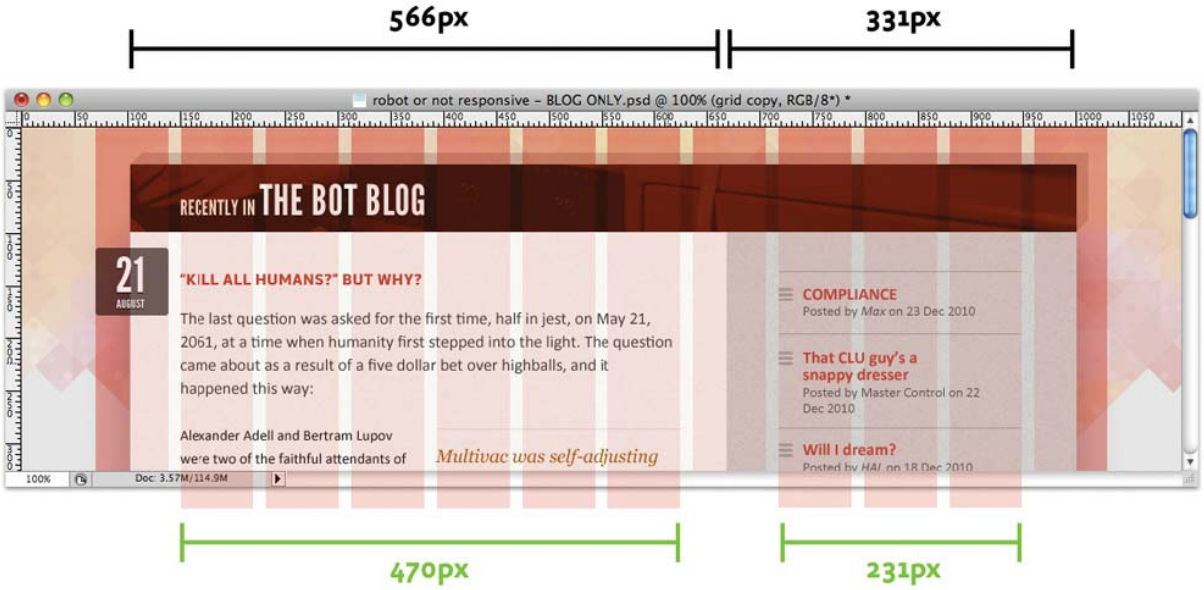

Рис. 2.21. Взглянув на колонки, мы можем достаточно быстро определить их ширину

Начнем с последнего. К счастью для нас, тут и делать нечего. Мы знаем ширину элемента (`231px`) и ширину содержащей ее колонки (`331px`), поэтому можем просто отцентрировать модуль по горизонтали:

```
.recent-entries {
  margin: 0 auto;
  width: 69.7885196%; /* 231px / 331px */
}
```

Со статьей (модуль `.article`) мы можем поступить так же. Но давайте-ка попробуем кое-что другое.
Помните поле шириной `48px`, которое мы задали в заголовке? Наша статья находится в той же колонке (**рис. 2.22**), поэтому вместо того, чтобы размещать ее по центру контейнера, создадим еще один пропорциональный промежуток.

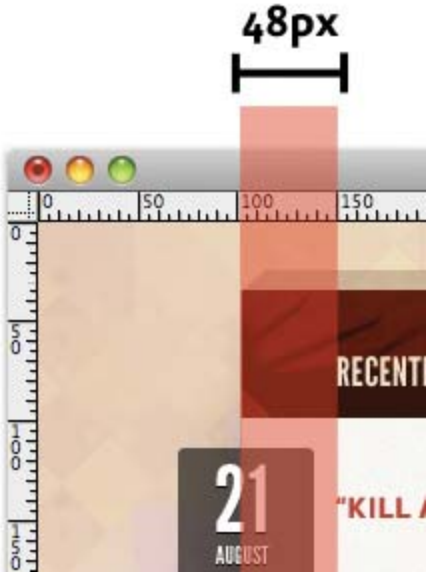

Рис. 2.22. У заголовка и статьи одинаковые поля

Целевое значение — `48px`. А поскольку мы работаем с относительным полем, в качестве контекста берем ширину самой статьи. Но, опять же, мы не знаем точной ширины модуля `.article`, поэтому используем ширину блока `.blog`, то есть `566px`:

```
.article {
  padding: 40px 8.48056537%; /* 48px / 566px */
```

```
}
```

Вуаля! Гибкая сетка закончена (**рис. 2.23**).

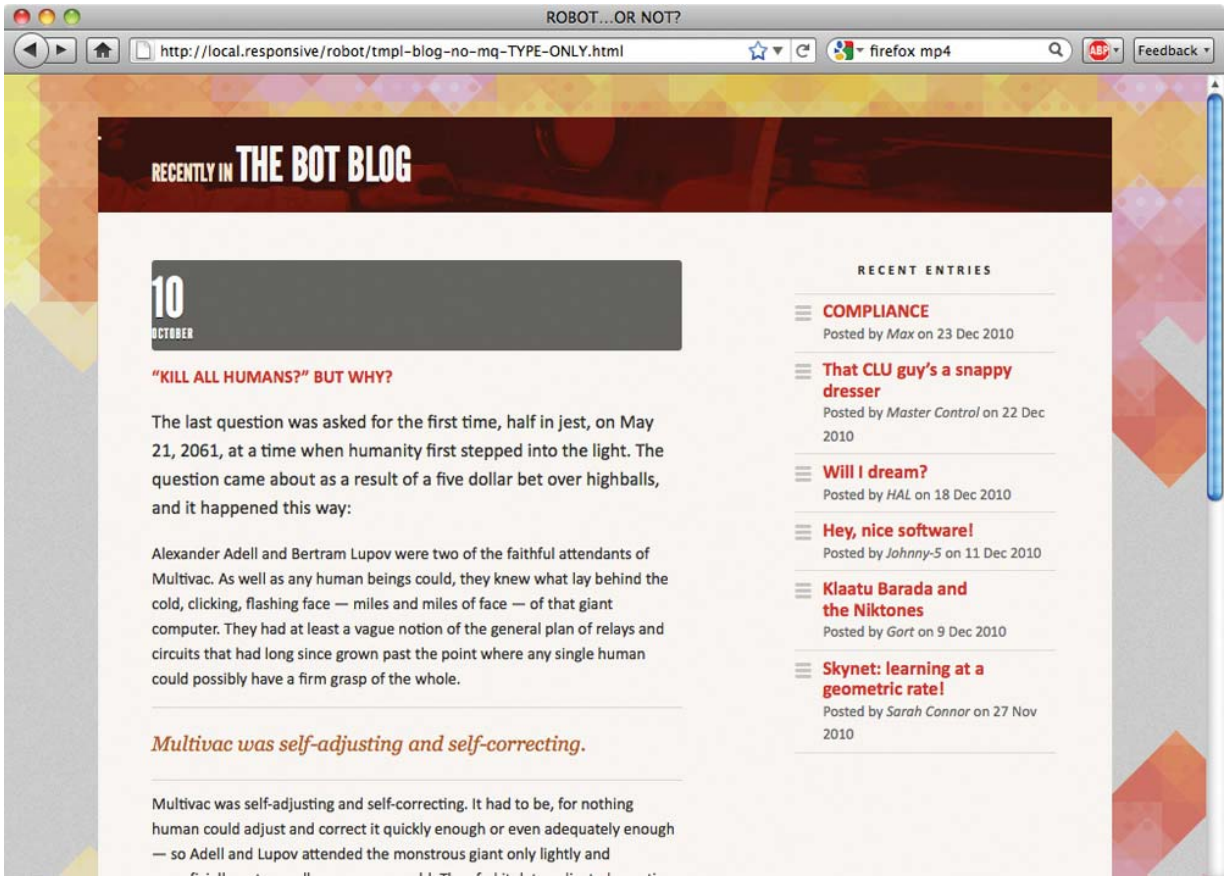

Рис. 2.23. Гибкие поля и отступы! Ура!

Немного отрицательных значений

Давайте обратим внимание на заголовок даты записи в блоге. Пока он занимает всю ширину записи, а так быть не должно. К этому времени мы уже много чему научились, поэтому особых затруднений не возникнет. На первоначальном дизайне мы видим, что дата расположена слева и занимает одну колонку шириной 69px (вернемся к **рис. 2.12**). А поскольку дата входит в блок статьи шириной 474px, мы уже знаем и контекст.

Вооружившись этой информацией, напишем небольшой CSS:

```
.date {
  float: left;
  width: 14.556962%; /* 69px / 474px */
}
```

Пока все хорошо и гибко. Но мы упустили один ключевой элемент: на данный момент дата расположена вплотную к левому краю статьи и окружена заголовком и текстом (**рис. 2.24**). А нам нужно вынести ее за пределы контейнера к левому краю целого модуля.

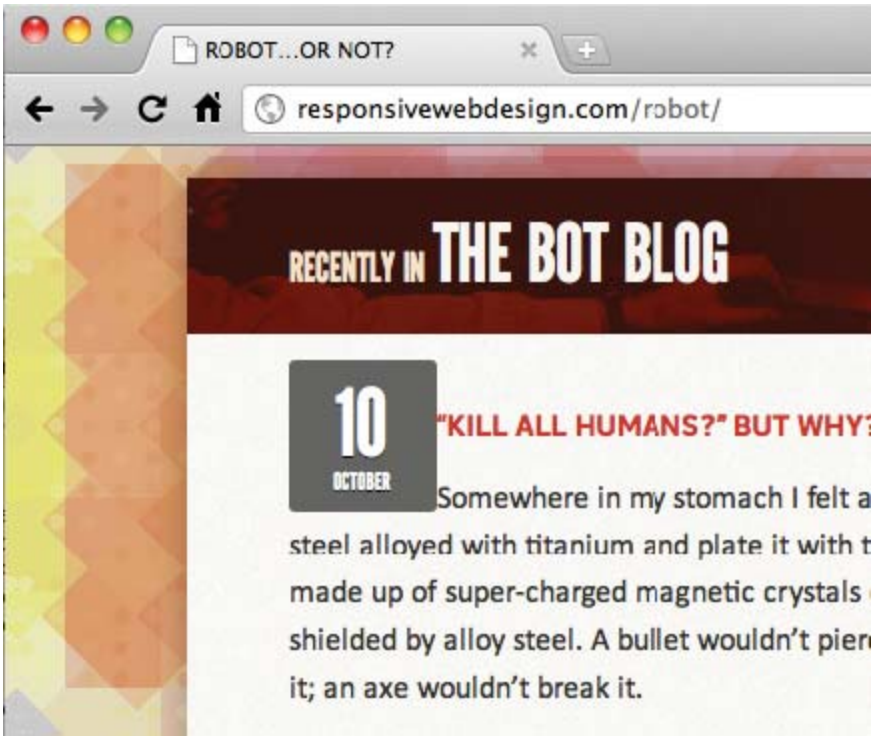

Рис. 2.24. Прогнило что-то в датском королевстве. (Под «датским королевством» я имею в виду дату записи, а когда я говорю «прогнило», то это значит, что она находится слишком близко к тексту.)

Мы сможем сделать это при помощи отрицательных полей, причем нам даже не придется менять принцип действий. Как и прежде, все, что нам нужно, — это определить ширину отступа по отношению к ширине контейнера элемента.

На первоначальном дизайне расстояние от левого края даты до левого края статьи составляет 81 пиксель (**рис. 2.25**). Если бы это был дизайн с фиксированной шириной, эта величина стала бы нашим отрицательным отступом:

```
.date {
  float: left;
  margin-left: -81px;
  width: 69px;
}
```

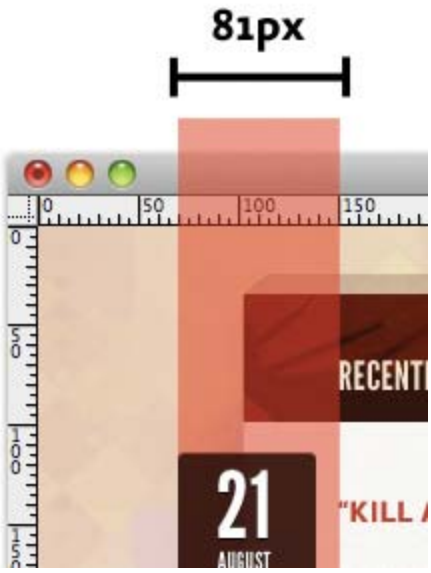

Рис. 2.25. Необходимо сдвинуть дату влево на 81px (или соответствующий относительный эквивалент)

Но мы ведь пока еще ни разу не использовали пиксели, так давайте не будем и начинать. И хотя отступ должен быть отрицательным, это не меняет нашу формулу. Мы все еще хотим выразить целевое значение, то есть отступ шириной в `81px`, как процентное отношение от ширины содержащего дату элемента в `474px`.

```
81 ÷ 474 = .170886076
```

Переведите число в проценты, поставьте перед ним минус — и вы получите пропорциональное отрицательное поле:

```
.date {
  float: left;
  margin-left: -17.0886076%; /* 81px / 474px */
  width: 14.556962%; /* 69px / 474px */
}
```

А теперь откиньтесь на спинку кресла, расслабьтесь и целиком насладитесь моментом: вы впервые создали полностью гибкую сетку (**рис. 2.26**). Мне хочется пожать вам руку.

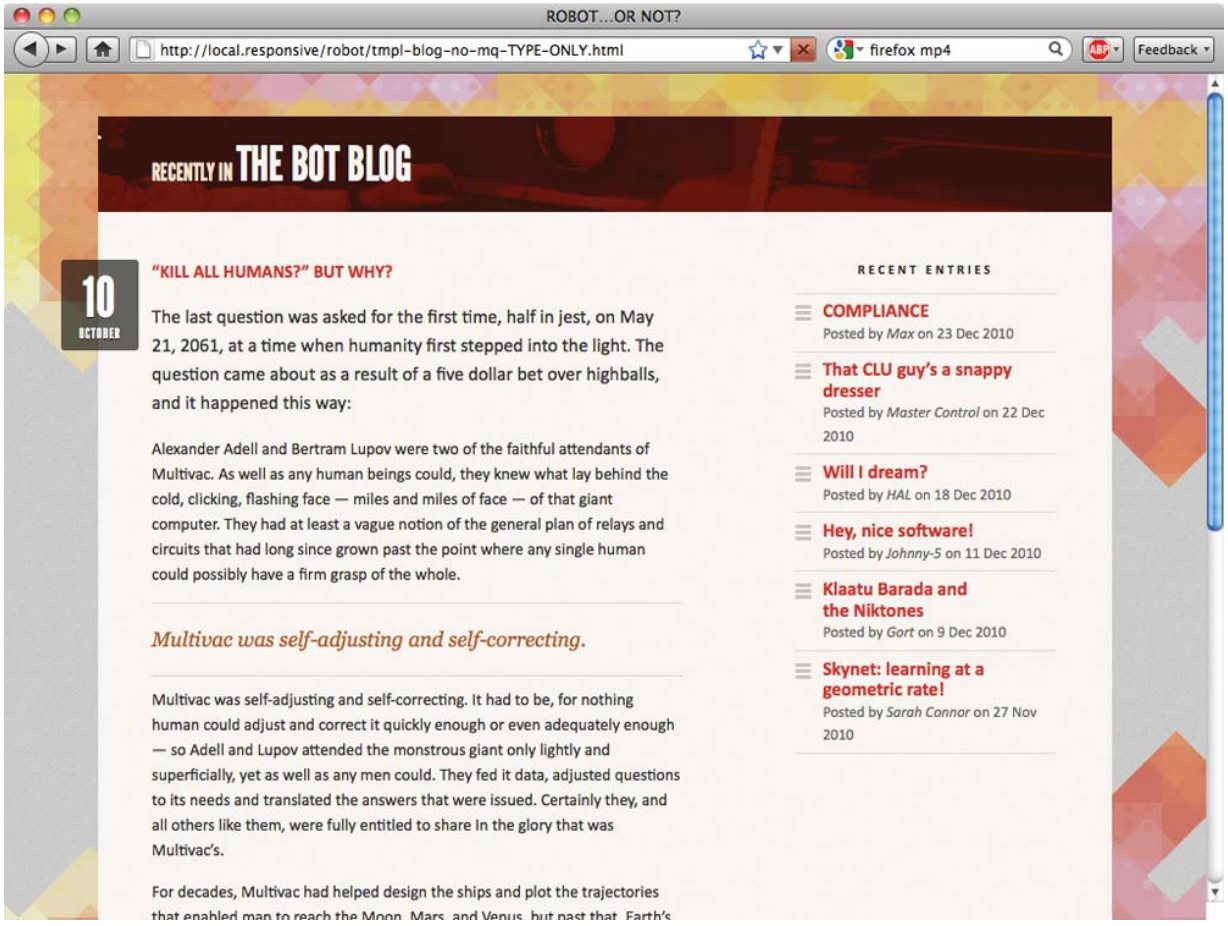

Рис. 2.26. Наша гибкая сетка готова. В основе ее вовсе не пиксели, и при этом — никаких компромиссов с эстетической точки зрения

Гибко двигаемся дальше

Я понимаю, что из-за меня вам пришлось заниматься расчетами больше, чем хотелось бы. Я сам всего этого на дух не переношу и потому — поверьте! — искренне вам сочувствую.

Но создание гибкой сетки — это не только математика. Конечно, формула `target ÷ context = result` помогает превратить размеры в процентные отношения, но вообще-то мы должны сломать нашу привычку переносить пиксели из Photoshop напрямую в CSS и сосредоточить наше внимание на пропорциях, заданных в дизайне. Мы должны научиться лучше видеть контекст, в первую очередь пропорциональное отношение между элементом и контейнером.

«Резиновая» сетка — это всего лишь основа, первый слой отзывчивого дизайна. Давайте двигаться дальше.

3 ГИБКИЕ ИЗОБРАЖЕНИЯ

НУ ЧТО Ж, ПОКА ДЕЛА идут неплохо. У нас есть готовая сетка, сложностью которой мы в угоду гибкости не пожертвовали. Должен признаться: когда я в первый раз создал гибкую сетку, я невероятно гордился собой.

Но потом, как это часто случается с веб-дизайнерами, на меня накатило отчаяние. На нашей странице прекрасно смотрятся слова, сам текст без усилий подстраивается под гибкий контейнер. Но и все! Я не знаю, заметили ли вы, но в Интернете встречаются еще и картинки. А в нашей гибкой сетке их пока нет.

Что же произойдет, если мы вставим изображение с фиксированной шириной в гибкий дизайн?

НАЗАД К РАЗМЕТКЕ

Чтобы ответить на этот вопрос, давайте проведем небольшой эксперимент: вставим картинку прямо в модуль блога и посмотрим, как на это отреагирует макет. Но сначала нужно освободить место в разметке.

Помните тот маленький блок для цитаты `blockquote`, так удобно вписавшийся в нашу статью? У нас и так уже слишком много текста, давайте заменим его на изображение:

```
<div class="figure">
  <p>
    <img src="robot.jpg" alt="" />
    <b class="figcaption">Lo, the robot
walks</b>
  </p>
</div>
```

Ничего особенного: элемент `img`, за которым следует короткая, но информативная надпись, заключенная в элемент `b`. Фактически в этом отрезке я использовал теги `figure/figcaption` из HTML5 в качестве названий классов, что способствует созданию крепкой семантической основы.

(Внимательные читатели могут заметить, что я использовал элемент `b`, а это несемантический прием. Некоторые дизайнеры в этом случае используют текстовый элемент `span`. Что касается меня, то в несемантической разметке мне импонирует лаконичность таких коротких тегов, как `b` и `i`.)

С HTML закончили, давайте перейдем к CSS:

```
.figure {
  float: right;
  margin-bottom: 0.5em;
  margin-left: 2.53164557%; /* 12px / 474px */
  width: 48.7341772%; /* 231px / 474px */
}
```

У нас получилось прекрасное удобное местечко для картинки. Она будет располагаться справа и занимать половину ширины

статьи, или четыре колонки гибкой сетки. Разметку и стиль проверили. Понятное дело, что все эти HTML и CSS не нужны, если нет никакой реальной картинки под рукой.

Поскольку я очень хорошо к вам отношусь (почти так же, как к роботам), я потратил некоторое время в Сети, выбирая подходящую картинку, и нашел фантастического робота (**рис. 3.1**). Преимущество этого изображения, помимо самого робота, конечно, еще и в том, что оно просто огромное. Я его немного обрезал, но не уменьшал, а оставил первоначальное разрешение 655 x 655. Он явно намного больше, чем гибкий контейнер, который будет его содержать. А значит, у нас есть прекрасная возможность посмотреть, как будет вести себя наш гибкий макет.

Рис. 3.1. Вполне подходящее изображение робота, использованное с любезного согласия Джереми Ноубла (http://bkaprt.com/rwd/10/)

Загружаем эту огромную картинку на сервер, обновляем страницу — и что мы видим? Отвратительно. Хуже не придумаешь (**рис. 3.2**).

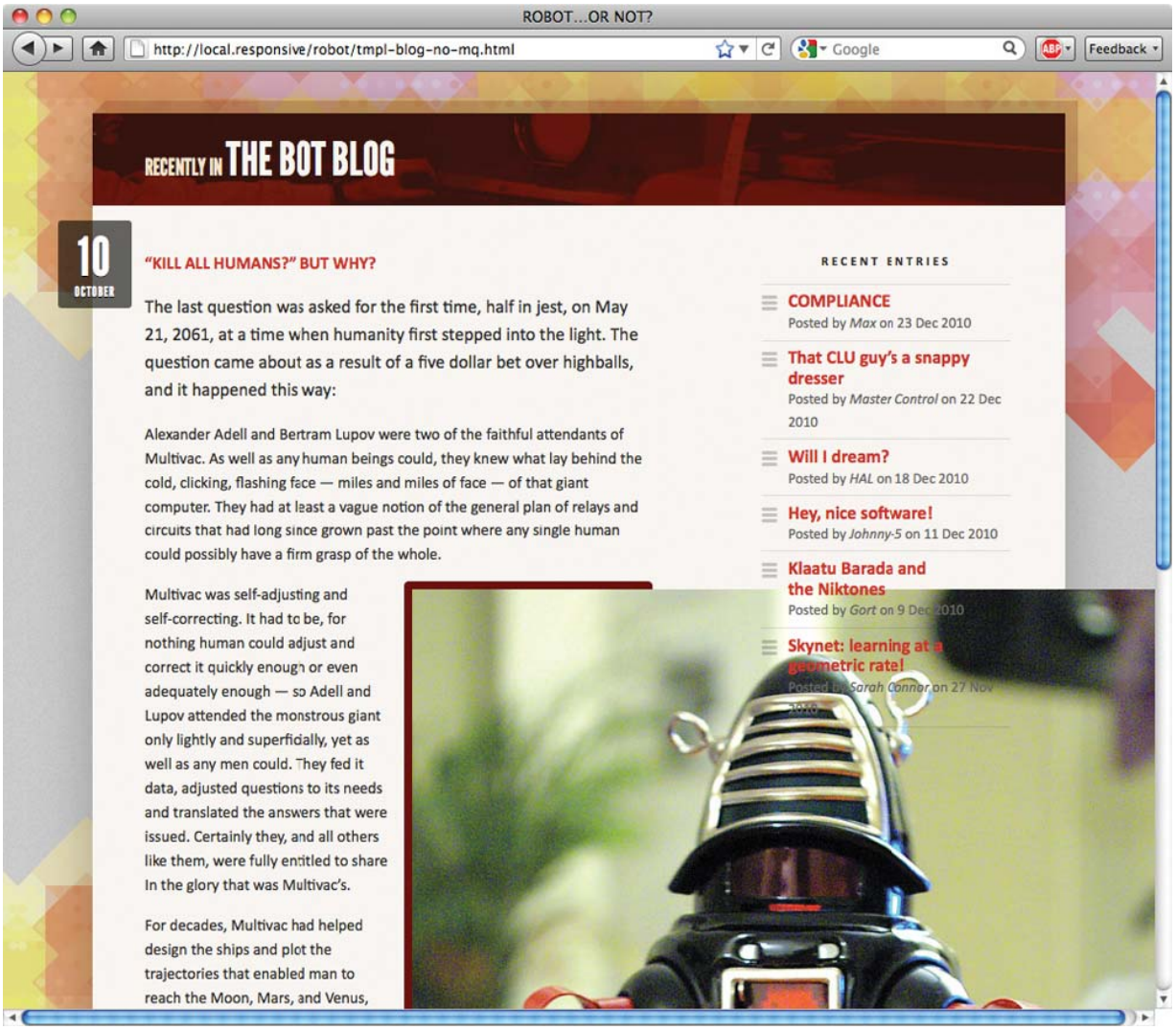

Рис. 3.2. Огромное изображение — огромные проблемы

Вообще-то удивляться тут нечему. Макет в принципе в порядке — гибкий контейнер работает как надо, а пропорции колонок остались неизменными. Но из-за того, что изображение шире, чем модуль `.figure`, то, что не поместилось, попросту вылезло за его пределы. Мы не применили к изображению никаких ограничений, которые бы сочетали его с гибким окружением.

ГИБКИЕ ИЗОБРАЖЕНИЯ

А что если ввести такое ограничение: написать правило, которое не позволит картинкам выходить за ширину их контейнера?

У меня для вас хорошие новости: сделать это проще простого:

```
img {
  max-width: 100%;
}
```

Правило, открытое дизайнером Ричардом Раттером (http://bkaprt.com/rwd/11/), накладывает на любое изображение в документе одно невероятно удобное ограничение. Теперь ширина элемента `img` может быть какой угодно, при условии, что она не превышает ширину содержащего его контейнера. В противном случае свойство `max-width: 100%` заставит изображение ограничиться шириной контейнера. И как вы видите, наша картинка прекрасно стала на место (**рис. 3.3**).

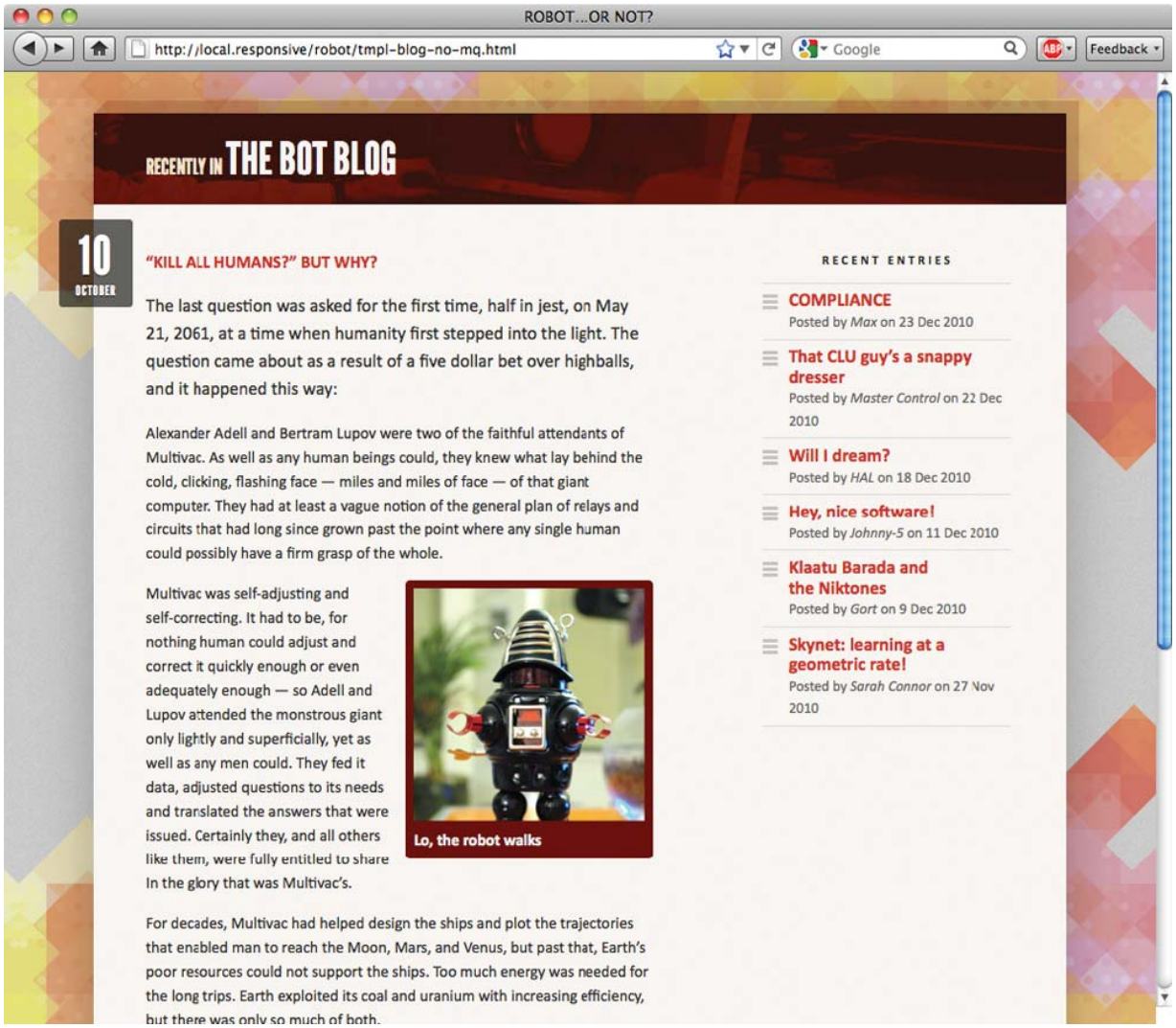

Рис. 3.3. За счет добавления max-width: 100% мы смогли удержать наше изображение в рамках контейнера. Вот за что я люблю max-width: 100%

Более того, современные браузеры умеют пропорционально менять размеры картинок. Если гибкий контейнер будет менять свои размеры, уменьшая или увеличивая изображение, соотношение сторон картинки останется неизменным (**рис. 3.4**).

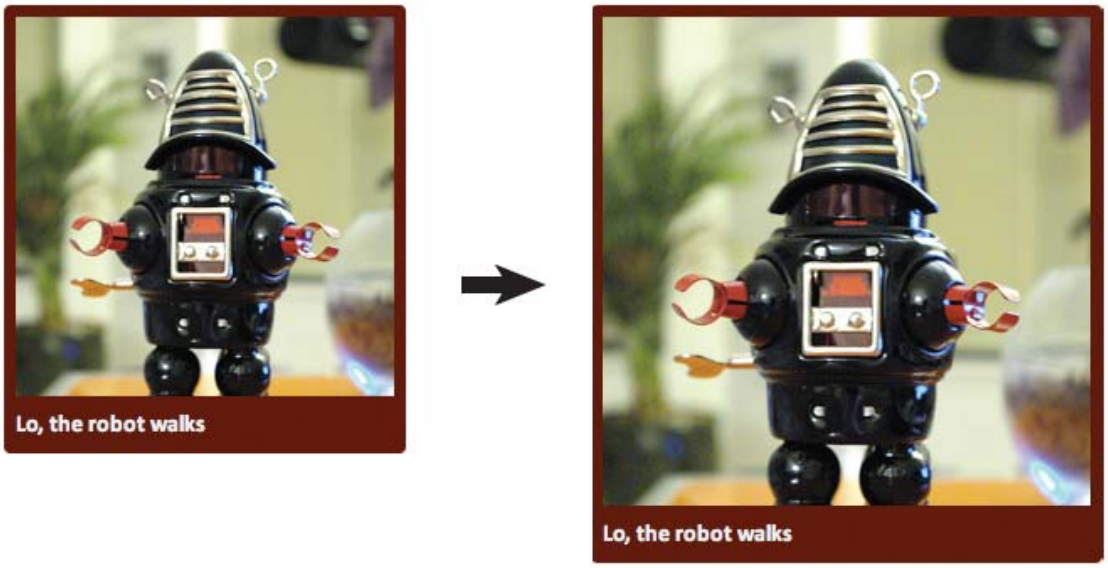

Рис. 3.4. Вне зависимости от изменения размеров гибкого контейнера изображение остается пропорциональным. Волшебство? Кто знает…

Я надеюсь, вы еще не устали от хороших новостей, поскольку, как оказалось, правило `max-width: 100%` можно применять к большинству элементов с фиксированной шириной, таким как видео- и другие медиафайлы. Фактически мы можем добавить в селектор еще и другие медиаэлементы:

```
img,
embed,
object,
video {
  max-width: 100%;
}
```

Это может быть небольшое флеш-видео (**рис. 3.5**), встроенный медиафайл или скромное изображение — браузеры все сделают сами и поменяют размеры соответственно макету. И все благодаря этому чудесному ограничению `max-width`.

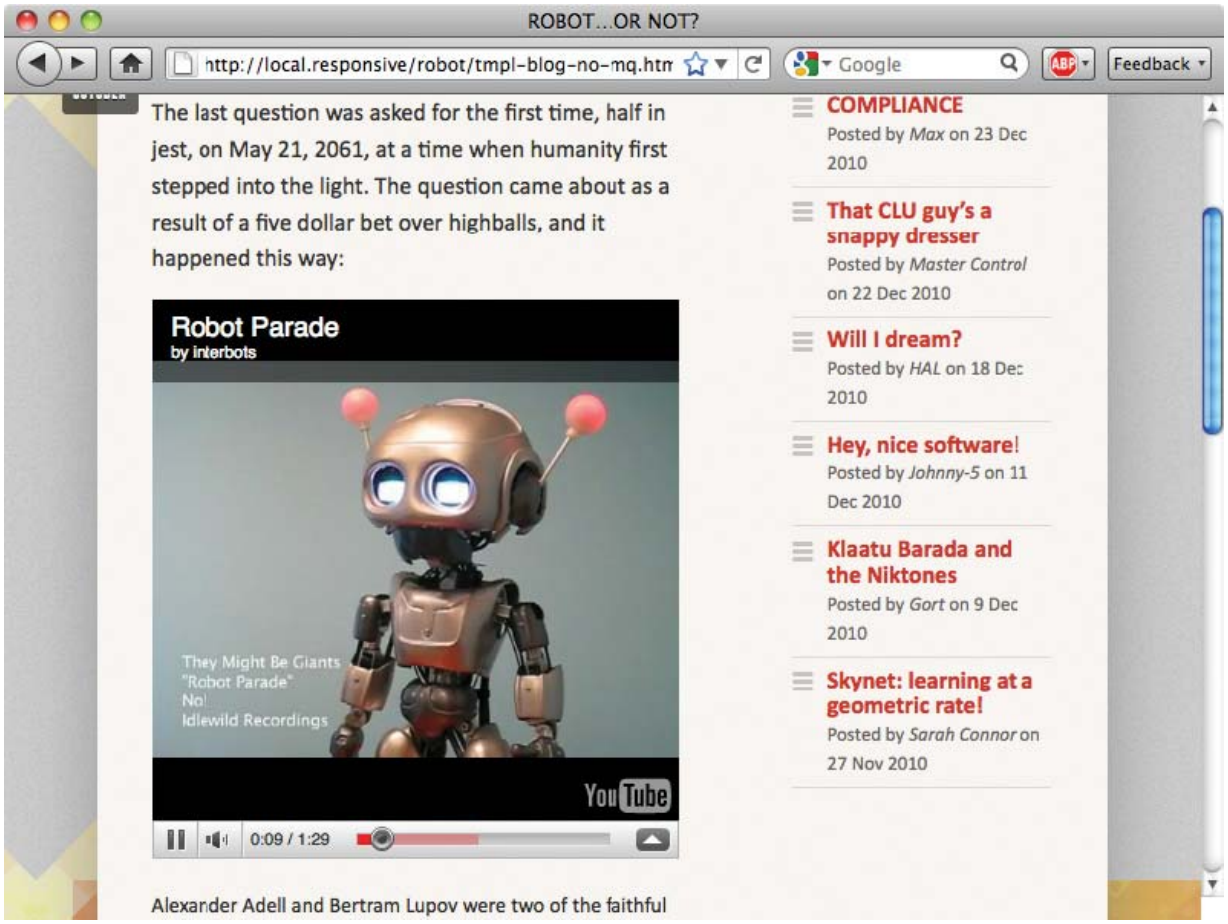

Рис. 3.5. С параметром max-width: 100% гибкими становятся и другие элементы медиа. Я уже говорил, что люблю max-width: 100%?

Так что же получается, мы действительно решили проблему гибких изображений? Одно правило — и все готово?

ЕСЛИ БЫ ВСЕ БЫЛО ТАК ПРОСТО...

Теперь займемся неприятными вещами и поговорим о некоторых особенностях браузеров по отношению к гибким изображениям.

Max-width в Internet Explorer

Суровая правда заключается в том, что Internet Explorer 6 и ниже не поддерживают свойство `max-width`. Что касается IE7 и выше, тут все в порядке. Но если вы, к моему глубочайшему сожалению, застряли в достопочтенном IE6 или более старой версии браузера, то придется кое-что доработать.

Существует несколько путей заставить свойство `max-width` работать в IE6. Большинство из них основано на JavaScript, обычно на базе проприетарного фильтра `expression` компании Microsoft, для динамической оценки ширины элемента и ручного изменения размеров в случае превышения установленного лимита. В качестве примера такого нестандартного подхода я могу порекомендовать статью Свенда Тофте (http://bkaprt.com/rwd/12/).

Что касается меня, я все-таки предпочитаю более простой CSS-подход. Все современные браузеры поддерживают свойство `max-width`:

```
img,
embed,
object,
video {
  max-width: 100%;
}
```

Но в отдельной таблице стилей для IE6 я бы сделал так:

```
img,
embed,
object,
video {
  width: 100%;
}
```

Заметили разницу? В IE6 и ниже правило `width: 100%` оказывается важнее `max-width: 100%`.

Вот здесь внимание: это совершенно разные правила. Если `max-width: 100%` запрещает изображению превышать ширину контейнера, то `width: 100%` делает его ширину равнозначной ширине содержащего его элемента.

В большинстве случаев этот прием работает отлично. Например, наша чрезмерно большая картинка `robot.jpg` будет всегда больше содержащего ее контейнера, поэтому правило `width: 100%` в этом случае можно применять смело.

Что касается более мелких изображений, например миниатюр или большинства встроенных роликов, их не всегда можно слепо масштабировать при помощи CSS. В этом случае для IE лучше сделать так:

```
img.full,
object.full,
.main img,
.main object {
  width: 100%;
}
```

Если вы не хотите на вашей странице применять правило `width: 100%` ко всем медиафайлам с фиксированной шириной, то можете написать список селекторов, которые будут размечать определенные виды изображений или видео (`img.full`) или определенные области документа, в которые вы будете вставлять файлы большого размера (`.main img, .main object`). Если изображения или другие медиафайлы появятся в этом списке, они станут гибкими, в противном случае — останутся в своем отсталом пиксельном состоянии.

Таким образом, если вы все еще работаете с морально устаревшими версиями Internet Explorer, внимательно

применяйте это правило, и все получится. Эту проблему мы решили, переходим к следующей.

И это что-то.

…И здесь становится понятно, что Windows нас ненавидит

Если вы посмотрите на модуль блога в каком-нибудь браузере на базе Windows, наша картинка `robot.jpg` превратится из впечатляющей в нечто непонятное (**рис. 3.6**).

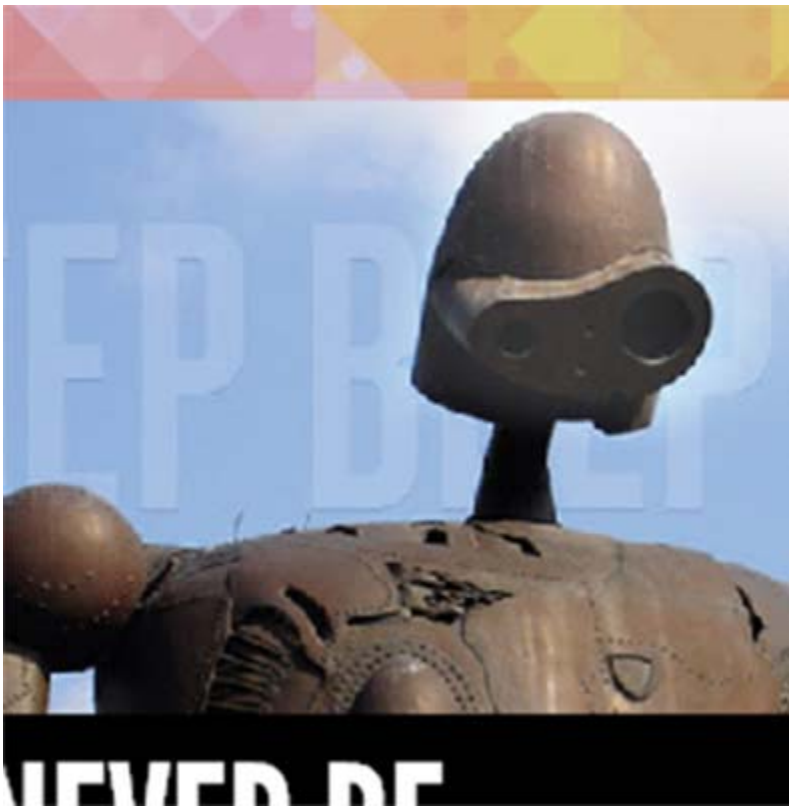

Рис. 3.6. При просмотре в IE6 можно заметить, что наше изображение робота содержит некоторые нежелательные артефакты. Судя по всему, Windows не особенно пригоден для гибких изображений

И это проблема не браузера, а платформы: Windows не слишком хорошо масштабирует изображения. В Windows изображения, размеры которых изменены средствами CSS, быстро «обрастают» различными артефактами, что крайне плохо сказывается на их качестве.

Для проверки я забросил картинку с текстом в гибкий контейнер и изменил размер изображения при помощи правила `max-width: 100%` во всех браузерах, а в IE6 и ниже при помощи `width: 100%`. Ясное дело, что никто не поместит такой объем текста в изображение, но этот пример прекрасно демонстрирует то, насколько все плохо с картинками в IE7 и более старых версиях. Как видите, картинка выглядит просто тошнотворно, прошу прощения за мой французский (**рис. 3.7**).

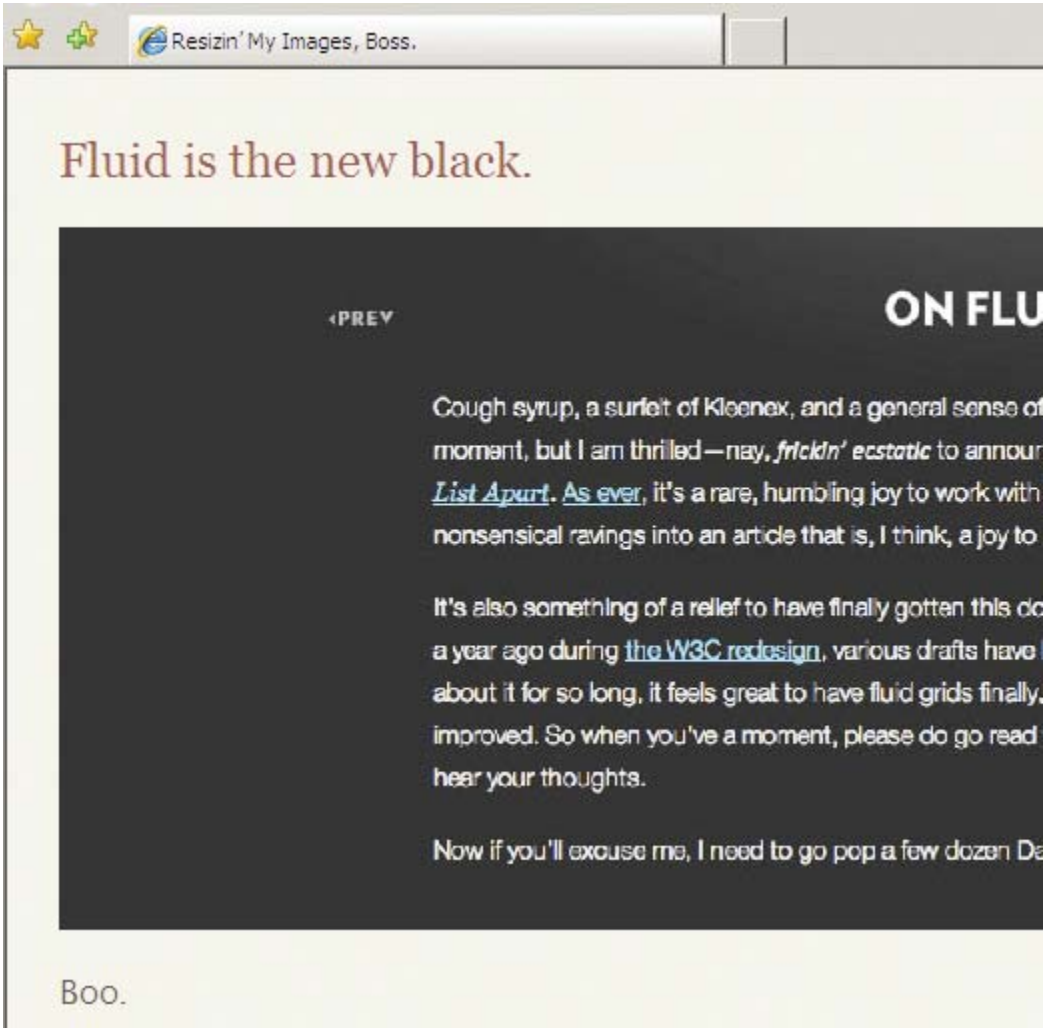

Рис. 3.7. В некоторых браузерах для Windows изображение «обрастает» большим количеством артефактов, что реально мешает восприятию

Но прежде чем опускать руки, обратите внимание на то, что этот баг появляется не во всех браузерах на базе Windows. На

самом деле от этой проблемы страдают только Internet Explorer 7 и ниже и Firefox 2 и ниже. Более современные браузеры, такие как Safari, Firefox 3+ и IE8+, не возражают против гибких изображений. Плюс ко всему в Windows 7, кажется, починили этот баг, — еще одна хорошая новость.

Может быть, теперь, когда мы выяснили, в чем проблема, мы сможем использовать какой-нибудь патч? Да, сможем (правда, за исключением Firefox 2).

Эта почтенная старая версия выпущена в 2006 году, поэтому я не думаю, что ею пользуется сколько-нибудь значительное число посетителей вашего сайта. Так или иначе, патч для Firefox 2 потребует анализа браузера для выявления определенных версий на определенной платформе, а такой анализ, мягко говоря, ненадежен. Но даже если мы решим его выполнить, в старых версиях Firefox нет возможности исправить такие изображения.

В Internet Explorer же такая возможность есть. (Придется поступиться своим самолюбием ради названия следующего раздела.)

Да здравствует герой-победитель AlphaImageLoader!

Вы когда-нибудь пытались сделать изображения в формате PNG прозрачными в IE6 и ниже? Если да, то вы наверняка использовали `AlphaImageLoader`, проприетарный CSS-фильтр компании Microsoft (http://bkaprt.com/rwd/13/). Чтобы обеспечить поддержку PNG с альфа-каналом в IE, создано достаточно много патчей (библиотека Дрю Диллера DD_belatedPNG — моя самая любимая: http://bkaprt.com/rwd/14/), но так уж исторически сложилось, что, когда нужно прикрепить PNG к фону элемента, в таблицу стилей для IE нужно написать следующее правило:

```
.logo {
  background: none;
  filter:

progid:DXImageTransform.Microsoft.AlphaImageLoa
```

```
der
  (src="/path/to/logo.png",
  sizingMethod="scale");
}
```

Этот патч делает несколько вещей. Сначала он удаляет фоновое изображение из элемента, затем вставляет его в объект `AlphaImageLoader`, который расположен между настоящим фоновым слоем и контентом элемента. Однако умное свойство `sizingMethod` (http://bkaprt.com/rwd/15/), которое говорит объекту `AlphaImageLoader`, что ему нужно обрезать (`crop`) какие-либо части изображения, вылезающие за контейнер, видит в нем обычное изображение (`image`) либо адаптирует его размер (`scale`) под содержащий его элемент.

Я так и вижу, как вы пытаетесь подавить зевок: в конце концов, какое отношение PNG-патч для IE имеет к нашим испорченным картинкам?

Как оказалось, самое непосредственное. В какой-то момент я обнаружил, что, если к изображению применить `AlphaImageLoader`, это существенно улучшает качество его отображения в IE, что ставит этот браузер на одну ступеньку с любым другим браузером. Кроме того, задав свойство `sizingMethod` для масштабирования (`scale`), мы сможем использовать объект `AlphaImageLoader` для создания иллюзии гибкого изображения.

Я сварганил небольшой JavaScript, чтобы автоматизировать этот процесс. Просто скачайте скрипт (http://bkaprt.com/rwd/16/) и вставьте его в любую страницу с гибкими изображениями; он подготовит ваш документ для создания гибких, высококачественных объектов `AlphaImageLoader`.

Разницу можно заметить невооруженным глазом (**рис. 3.8**) — из почти полностью искаженного наше изображение превратилось в безупречное. То же можно сделать и с гибким контекстом.

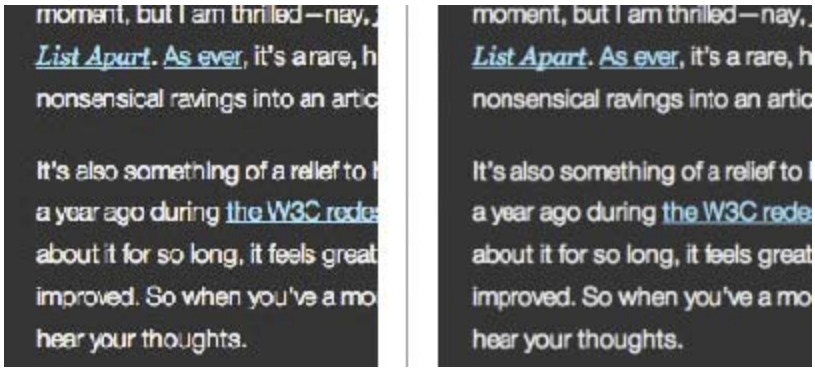

Рис. 3.8. Теперь наше изображение отлично читается и великолепно масштабируется. Скажем же спасибо AlphaImageLoader!

(Стоит упомянуть тот факт, что проприетарные фильтры Microsoft, и в частности `AlphaImageLoader`, снижают производительность — более подробно все подводные камни этого метода описывает Стоян Стефанов в блоге YUI (http://bkaprt.com/rwd/17/). Поэтому тщательно протестируйте это правило, проверьте результаты на своих пользователях и решите для себя, стоит ли улучшенное отображение таких издержек.)

При применении правила `max-width: 100%` (а также правила `width: 100%` и патча `AlphaImageLoader`) вложенные картинки прекрасно меняют свой размер наряду со всей гибкой сеткой во всех браузерах в зависимости от размера окна браузера.

Но что делать с изображениями, которые отсутствуют в нашей верстке?

ГИБКИЕ ПОВТОРЯЮЩИЕСЯ ФОНОВЫЕ ИЗОБРАЖЕНИЯ

Представим, что наш уважаемый дизайнер прислал исправленный макет модуля блога. Заметили, что изменилось (**рис. 3.9**)?

Рис. 3.9. Теперь у нас появилась фоновая графика. Это круто!

До этого момента содержание нашего блога располагалось на непонятном, практически белом фоне. Но теперь дизайн немного изменился, фон левой колонки стал серым для большей контрастности между колонками. Кроме того, в фоне появился едва заметный шум, который добавляет к нашему дизайну еще один слой текстуры (**рис. 3.10**).

Рис. 3.10. Детальный взгляд на новый фон

Но как нам добавить этот новый фон к уже существующему шаблону?

Еще в 2004 году Дэн Седерхольм написал прекрасную статью о том, как использовать вертикально повторяющуюся фоновую графику для создания эффекта «фальшивой колонки» (http://bkaprt.com/rwd/18/). Гениальность технологии в ее простоте: повторяя цветное фоновое изображение по вертикали позади нашего контента, мы создаем иллюзию колонок одной высоты.

В оригинальной версии Дэна фоновая графика располагалась по центру вверху области с контентом и повторялась по вертикали:

```
.blog {
  background: #F8F5F2 url("blog-bg.png")
```

```
repeat-y 50% 0;
}
```

И этот прием работал великолепно. Однако он был рассчитан на макет с фиксированной шириной, то есть создавал графику, которая совпадала с ним по ширине. Как же, интересно, мы будем работать с фоновым изображением, которое покрывает две гибкие колонки?

Благодаря некоторым исследованиям дизайнера Дага Баумана (http://bkaprt.com/rwd/19/) мы все еще можем применять технологию «фальшивой колонки». Просто нужно уделить немного больше внимания планированию и вытащить на свет нашу любимую формулу `target ÷ context = result`.

Сперва нужно снова глянуть на наш первоначальный макет, чтобы найти точку перехода на фоновой графике, точный пиксель, на котором белая колонка переходит в серую. Судя по всему, это происходит на 568 пикселе (**рис. 3.11**).

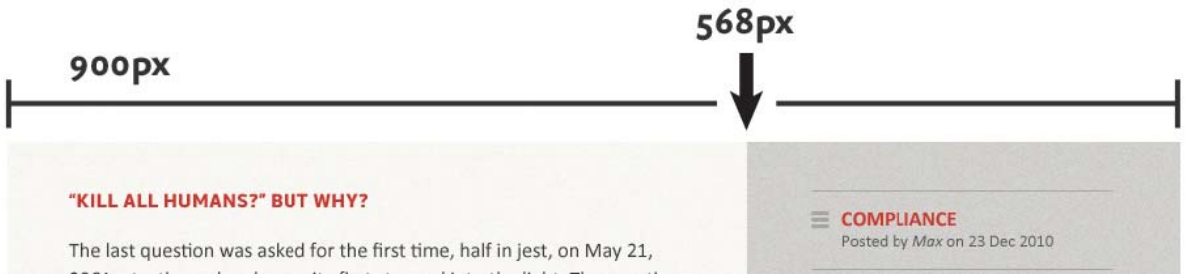

Рис. 3.11. Белая колонка переходит в серую на отметке 568px. Это и есть наша точка перехода

Вооружившись этой информацией, мы можем адаптировать подход «фальшивой колонки» к нашей «резиновой» сетке. Сначала нам нужно конвертировать точку перехода в процентное значение, соответствующее ширине модуля нашего блога. Чтобы это сделать, снова воспользуемся формулой `target ÷ context = result`. *Целевое* значение — `568px`, ширина макета (контекста) — `900px`. Подставляем эти цифры в формулу:

```
568 ÷ 900 = 0.631111111111111
```

И снова получаем какое-то невероятно длинное число, которое превращаем в проценты — `63.111111111111%`

Запомните его. Затем откройте ваш любимый редактор изображений и создайте какой-нибудь огромный документ, шириной, скажем, 3000 пикселей (**рис. 3.12**). А поскольку мы собираемся повторять его по вертикали, высоту сделайте `160px`.

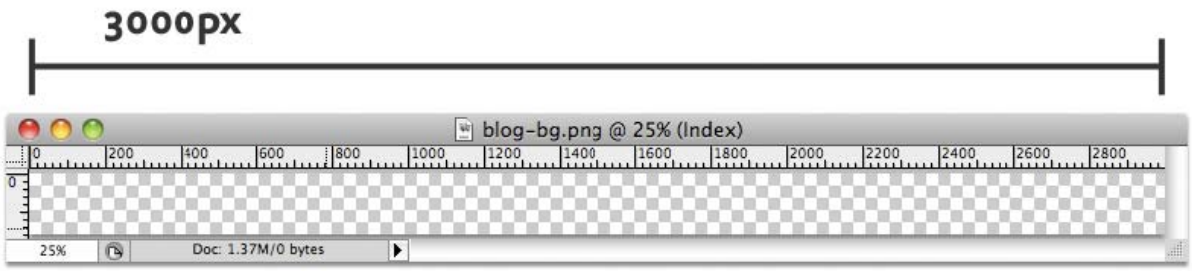

Рис. 3.12. Невероятно большой холст, который мы совсем скоро превратим в фоновую графику

Именно из этого документа мы сделаем нашу фоновую графику. Вы можете спросить: зачем такой большой? Я отвечу: изображение должно быть больше, чем любое окно браузера. Полагаю, что, если только вы не читаете этот текст из XXV века с какого-нибудь экрана шириной на всю стену, сделанного из голограмм или чего-то не менее мудреного, ваш монитор не будет шире этого изображения.

Чтобы сделать сами колонки, нам нужно применить процентное значение точки перехода (`63.111111111111%`) к новому широкому холсту. То есть, если ширина графики 3000 пикселей, мы перемножаем эти два числа:

```
3000 x 0.631111111111111 = 1893.333333333333
```

и в результате получаем `1893.333333333333`. А поскольку Photoshop не хочет иметь дело с десятичными долями, давайте

округлим это число до 1893 пикселей. Нам осталось только создать нашу текстуру из нового файла, сделав переход от белого цвета к серому на 1893-м пикселе (**рис. 3.13**).

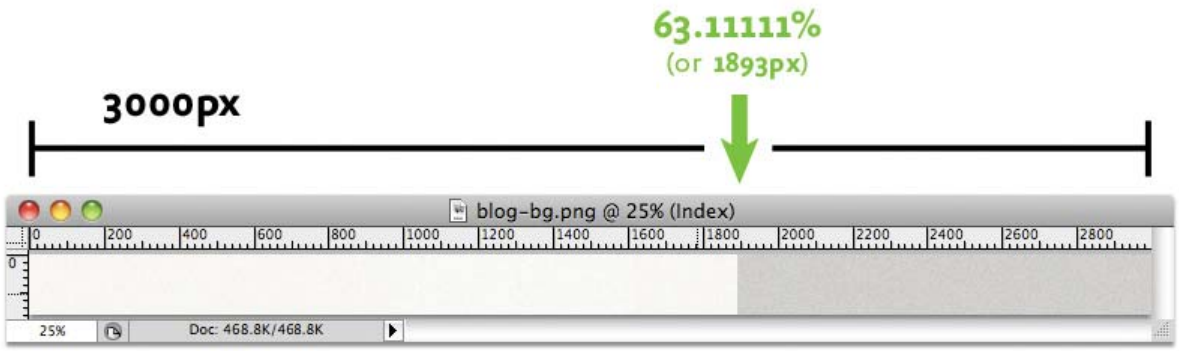

Рис. 3.13. Мы применили к широкому фоновому изображению проценты, что привело к созданию колонок

Что это нам дает? Только что мы пропорционально определили точку перехода на новом широком холсте. При помощи нового пиксельного значения мы можем сделать нужные нам колонки: белая будет шириной в `1893px`, а серая займет всю остальную часть изображения.

Осталось сделать одно: вписать новую графику в таблицу стилей:

```
.blog {
  background: #F8F5F2 url("blog-bg.png")
repeat-y 63.111111111111% 0; /* 568px / 900px
*/
}
```

Следуя оригинальной технологии Дэна, мы располагаем графику в самом верху нашего блога и повторяем ее по вертикали (`repeat-y`). Благодаря постоянному повторению процентного значения точки перехода (`63.111111111111% 0`) колонки остаются неизменными по отношению друг к другу, независимо от того, какого размера становится сам макет.

В результате мы получили прекрасные фальшивые колонки в «резиновом» макете (**рис. 3.14**). Все благодаря оригинальному подходу Дэна Седерхольма, приправленному небольшими пропорциональными размышлениями.

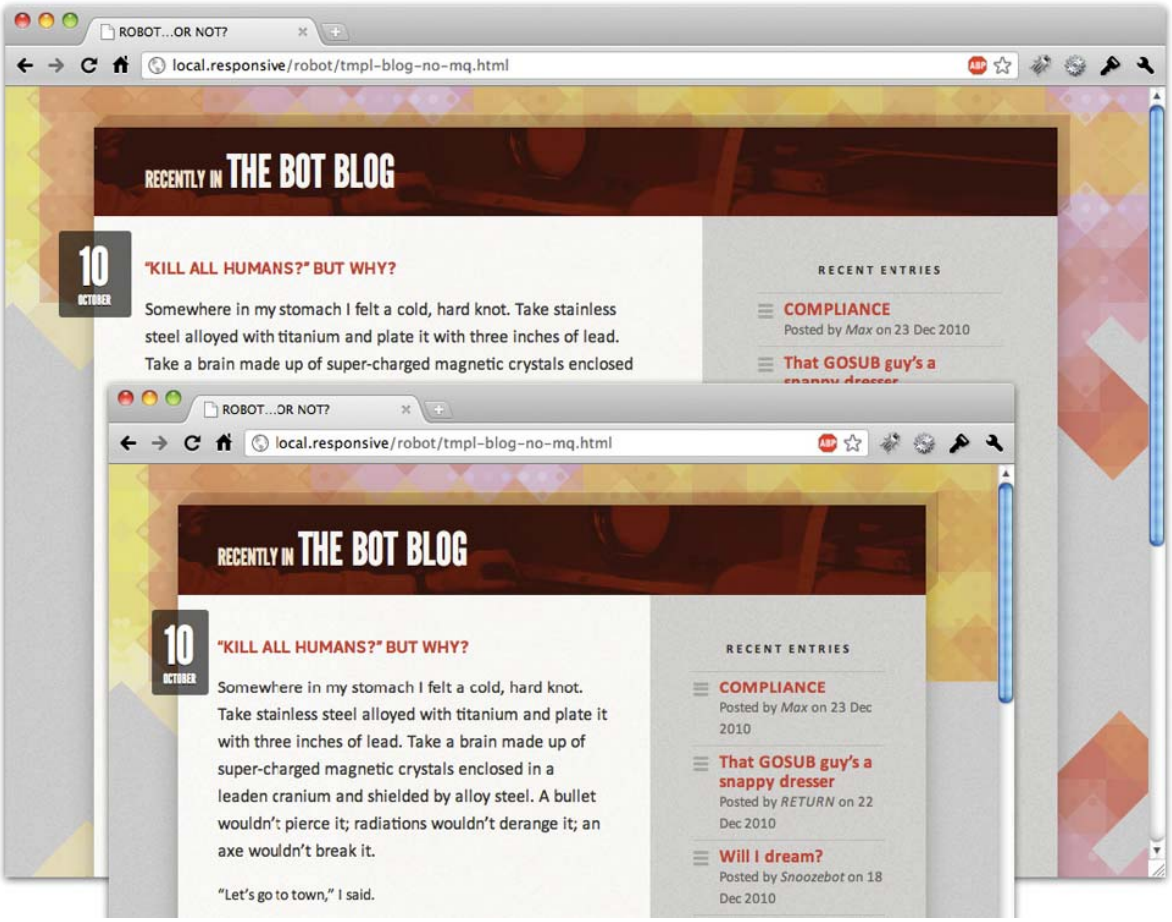

Рис. 3.14. Идеально гибкие колонки

Полностью гибкие фоновые изображения?

Конечно, наши гибкие фальшивые колонки, вообще-то, совсем не гибкие: мы просто использовали процентное значение в размещении фонового изображения так, чтобы колонки меняли свои размеры в зависимости от размеров контейнера. Размеры изображения при этом остаются неизменными.

Но что делать, если нам нужно, чтобы фоновое изображение тоже меняло свои размеры вместе с макетом? Например, вы разместили логотип на фоне элемента `h1` или используете

спрайты[2] для создания ролловер-эффекта в навигации сайта. Сможем ли мы изменить размеры картинок, изображенных на фоне?

Сможем. Существует CSS3-свойство под названием `background-size` (http://bkaprt.com/rwd/20/), которое позволяет создать действительно гибкие фоновые изображения, однако, как вы, наверное, уже догадались, не все браузеры обеспечивают его поддержку.

Но существует несколько отличных решений на базе JavaScript: например, jQuery-плагин Backstretch Скотта Робина (http://bkaprt.com/rwd/21/), который позволяет добавлять масштабируемые фоновые изображения для элемента body. Как вы узнаете из следующей главы, медиазапросы CSS3 также можно использовать для адаптации различных фоновых изображений к различным диапазонам разрешений. Так что если нет возможности использовать `background-size`, вполне возможно найти другое решение. Для пытливого ума нет преград — гласит народная мудрость.

КАК НАУЧИТЬСЯ ЛЮБИТЬ OVERFLOW

Существует еще несколько способов адаптации изображений с фиксированной шириной к «резиновому» контексту. Посмотрите эксперименты Ричарда Раттера с широкими изображениями в гибких макетах (http://bkaprt.com/rwd/11/). Раз уж вы решили заняться отзывчивым дизайном, изучите эти способы, некоторые из них могут оказаться весьма полезными.

Лично я несколько раз использовал свойство `overflow`. Как мы узнали из предыдущей главы, широкие изображения могут попросту вылезать за пределы своих контейнеров.
И в большинстве случаев для их ограничения лучше всего использовать правило `max-width: 100%`. Но можно и обрезать эти лишние данные, применив свойство `overflow: hidden`. То есть вместо того, чтобы позволить изображению изменить свои размеры автоматически:

```
.feature img {
  max-width: 100%;
}
```

мы можем попросту отрезать лишние данные:

```
.feature {
  overflow: hidden;
}

.feature img {
  display: block;
  max-width: auto;
}
```

В результате получаем изображение, обрезанное под свой контейнер (**рис. 3.15**). Оно никуда не делось, просто его лишние элементы не видны.

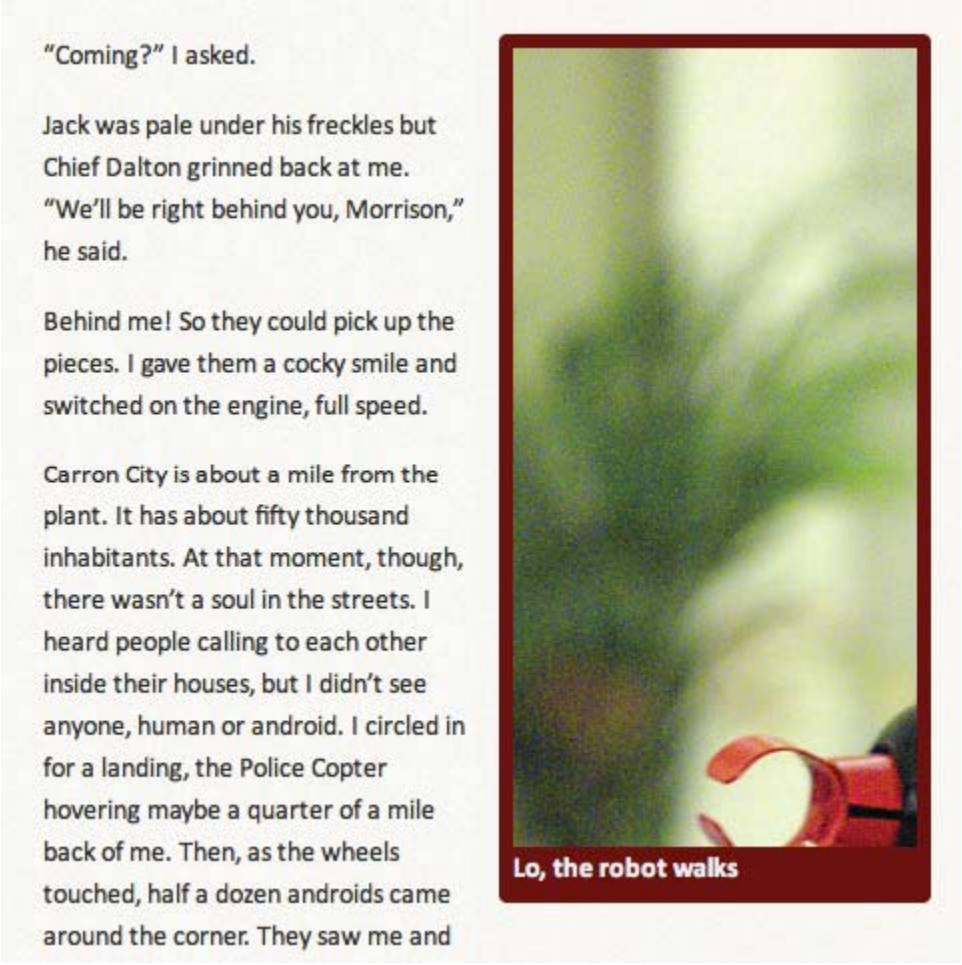

Рис. 3.15. Применив overflow: hidden к контейнеру нашего изображения, мы получили обрезанное изображение. Можно крикнуть «ура»

Однако это не лучшее решение. На самом деле я считаю, что в большинстве случаев `overflow` проигрывает `max-width`. Но этот метод имеет право на существование и в некоторых случаях даже может быть полезным.

ПРОБЛЕМЫ С КОНТЕНТОМ

В большинстве случаев и свойство `overflow`, и правило `max-width: 100%` довольно функциональны и работают для большинства медиа-файлов. Лично я успешно применял их в различных «резиновых» сетках.

Но при этом оба подхода абсолютно нечувствительны к содержанию. Каждый из них устанавливает некоторые основные

правила, которым следуют изображения по отношению к своим контейнерам: `max-width: 100%` масштабирует картинку в соответствии с размерами контейнера, а `overflow` позволяет убрать лишние данные, выходящие за его пределы.

Но что если вам предстоит работа со сложной графикой или изображением, которое несет определенную информационную нагрузку (**рис. 3.16**)? В этом случае простое масштабирование или обрезание нежелательны, поскольку могут помешать пользователю правильно понять информацию, содержащуюся в изображении.

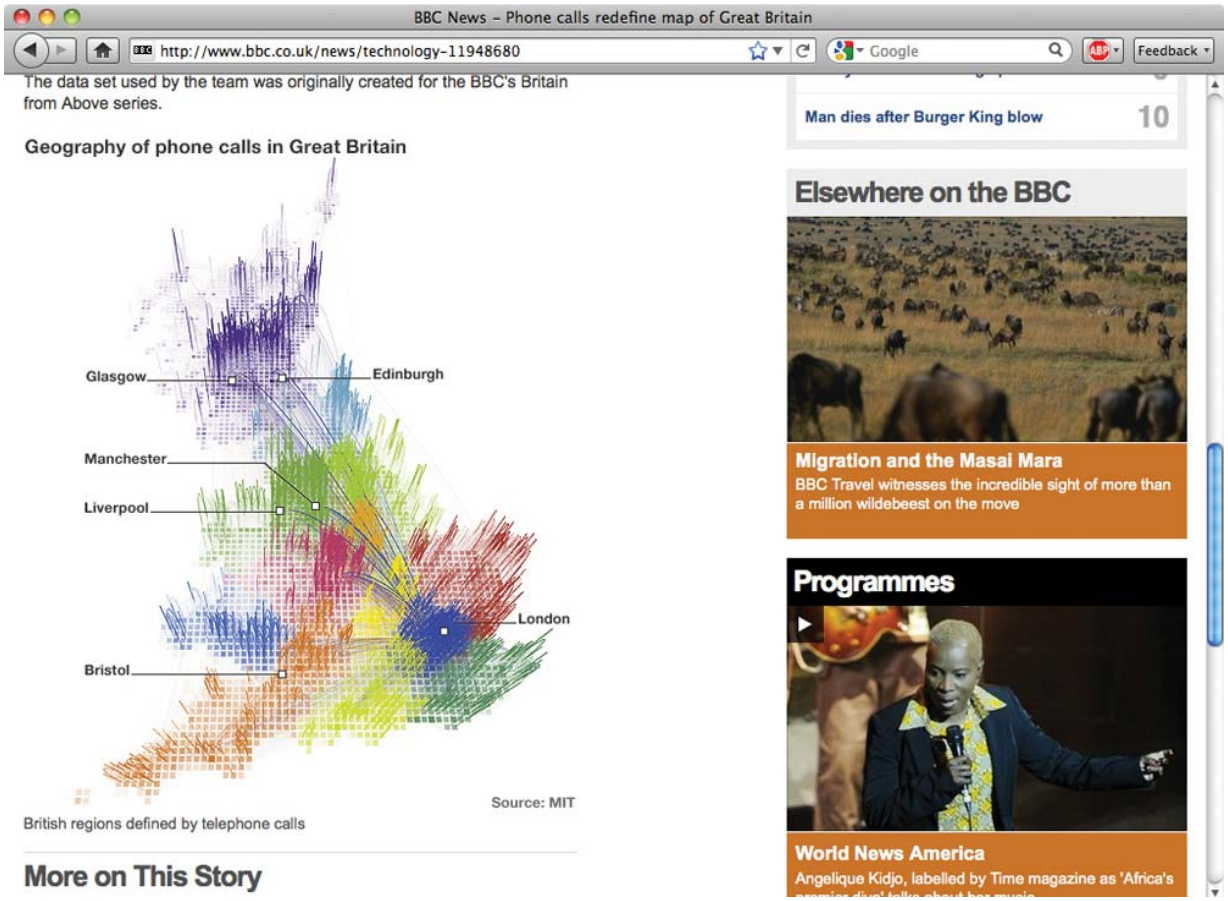

Рис. 3.16. Эта отличная инфографика с сайта BBC News (http://bkaprt.com/rwd/22/) содержит жизненно необходимую с точки зрения контента информацию. Простое масштабирование может оказаться неэффективным

В этом случае нужно найти способы передачи различных вариантов одной и той же картинки в разных диапазонах

разрешений. Другими словами, вы можете создать один образец для десктопных браузеров, а второй, более линейный, — для устройств с маленькими экранами. Задав эти параметры, вы можете положиться на сервер, который сам выберет наиболее подходящее изображение.

Такое решение выходит за рамки данной книги (и не всегда по силам вашему покорному слуге), однако дизайнер-разработчик Брайан Ригер описал возможный подход в своем блоге (http://bkaprt.com/rwd/23/), откуда вы и сможете его скачать.

Если вы решили использовать серверное решение, его можно укрепить различными клиентскими приемами, которые мы уже обсуждали. Например, вы можете создать несколько вариантов изображения под разные диапазоны разрешений, а затем использовать правило `max-width: 100%`, чтобы сгладить переход на другие устройства, браузеры и диапазоны разрешений.

ГИБКИЕ СЕТКИ И ИЗОБРАЖЕНИЯ КАК ДРЕВО ПОЗНАНИЯ

Итак, к этому моменту мы изучили все, что необходимо для успешного создания сложных, но гибких макетов: простая математика для гибких сеток и немного стратегических решений для изображений и других медиафайлов. Все эти знания вы можете применять не только по отношению к блогу, который мы писали, но и ко всему сайту Robot or Not, создавая дизайн, основанный на системе пропорций и процентов, без всяких пикселей (**рис. 3.17**).

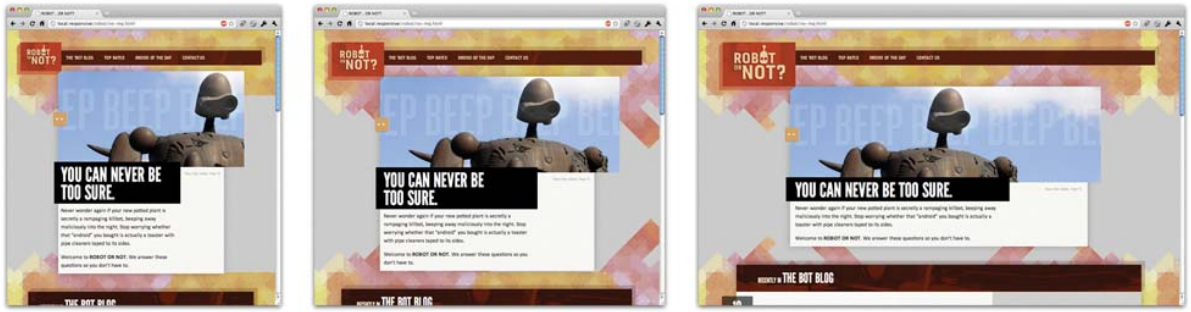

Рис. 3.17. Использовав рекомендации, содержащиеся в двух главах, мы получили завершенный и гибкий макет, который может расширяться и сужаться в зависимости от размеров окна браузера

Имея на руках гибкую основу, мы готовы добавить еще один ингредиент к нашему отзывчивому дизайну.

2 Спрайт — группа малых и средних картинок (чаще иконок), составленных в одну, чтобы уменьшить количество запросов к серверу.

4 МЕДИАЗАПРОСЫ

В ОБЩЕМ-ТО, Я ВСЕГДА БЫЛ ПРОТИВНИКОМ фиксированной верстки. Я с самого начала чувствовал, что будущее — за макетами, которые обладают гибкостью хоть в малейшей степени, ведь они всегда могут подстроиться под размеры окна, экрана или разрешения устройства. Более того, настаивая на необходимости гибкости, я часто использовал эпитеты «перспективный» и «приспосабливающийся». Увы, ко мне не очень-то прислушивались и считали страшным занудой.

Но в какой-то момент все изменилось.

Однако вернемся к нашему сайту Robot or Not. Мы сделали его максимально гибким, однако он все же еще не очень надежный. Да, «резиновая» сетка сделала его менее чувствительным к изменениям размера окна или разрешения экрана, чем при фиксированном макете. Однако изменения в размере и форме окна браузера могут вызвать деформацию всей верстки.

И знаете что? В этом нет ничего страшного!

ПРИСТУПИМ К ЛЕЧЕНИЮ

Понимаю, это неприятно, но все же нам придется понять, что именно случилось с нашим дизайном и где он нарушился. Определив все проблемы, мы сможем эффективно их устранить, даже если придется немного помучиться в процессе.

Поскольку мы работаем с гибким макетом, мы можем просто изменить размеры окна браузера. Это, конечно, не заменит полноценного тестирования на отдельных устройствах, но позволит быстро оценить, как себя поведет наш дизайн в различных диапазонах разрешений. Мы увидим, как именно он будет выглядеть на экране телефона, планшета или другого устройства.

Расстановка акцентов

Прежде всего, изменим разрешение окна браузера с 1024 пикселей на 760 пикселей (**рис. 4.1**). Проблемы сразу же станут весьма наглядными.

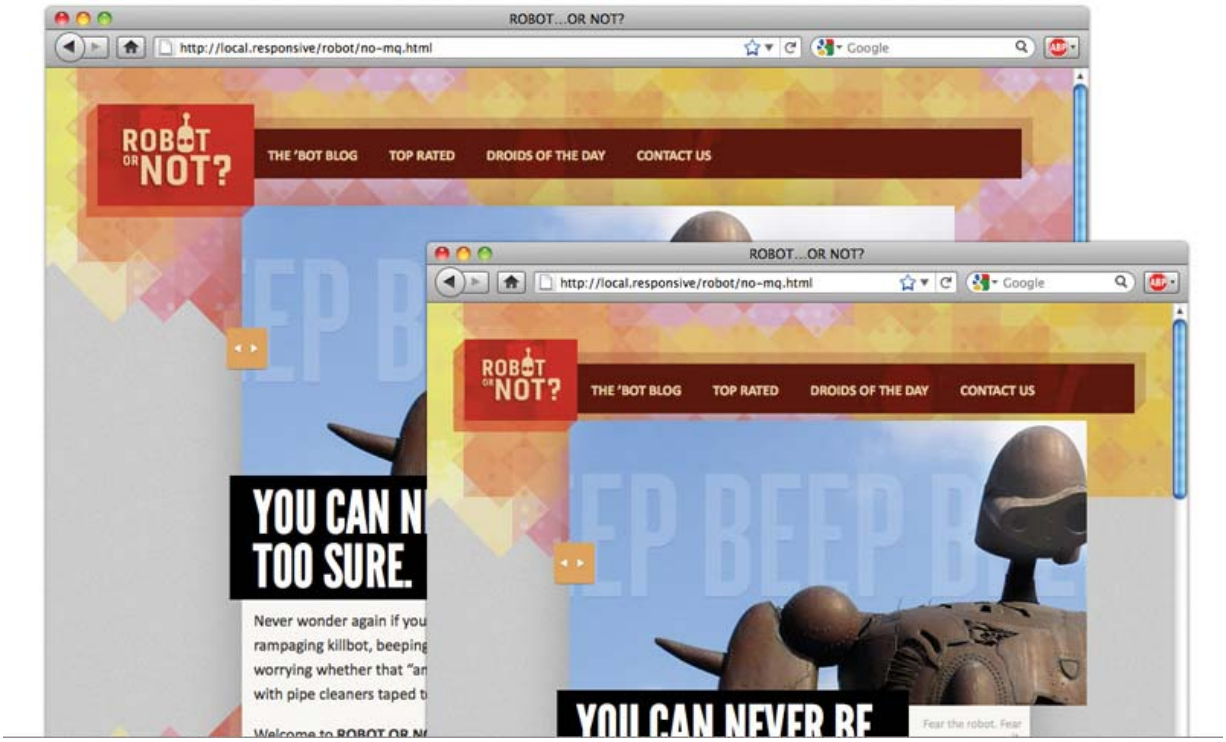

Рис. 4.1. Чтобы понять, каким образом будет выглядеть наш дизайн при разном разрешении экрана, достаточно изменить размеры окна браузера

В первоначальном дизайне было несколько привлекательных элементов: впечатляющие заголовки, яркая выделяющаяся картинка и широкие поля. Все это осталось, но — с визуальной точки зрения — стало каким-то невзрачным.

Обратите внимание на то, что картинка в верхней части сайта стала занимать практически всю страницу (**рис. 4.2**). Поскольку мы обрезали ее при помощи свойства `overflow`, она не адаптировалась под изменения всей сетки. Кроме того, само изображение — наш любимый робот — теперь обрезано. То есть картинка не только огромная, но и непонятная. Кошмар какой-то…

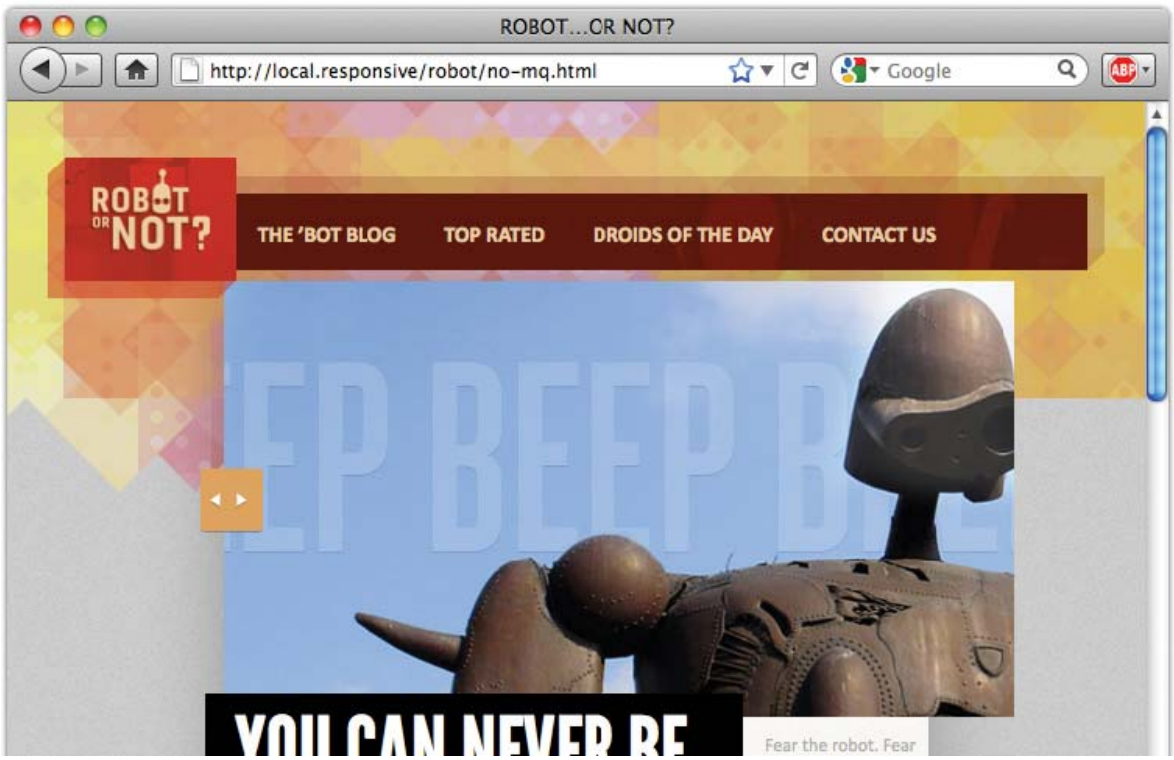

Рис. 4.2. В верхней части нашего дизайна творится что-то не то

На фоне этой гигантской картинки логотип выглядит совсем крошечным, а поле между навигацией и картинкой исчезло. Глядя на все это, испытываешь приступ клаустрофобии.

Мне неприятно это говорить, но даже при небольшом уменьшении разрешения сайт превращается в нелепое

нагромождение различных элементов.

Маленькая сетка, большие проблемы

И это еще не самое ужасное. Если мы уменьшим окно браузера до 600 пикселей — ширины окна маленького браузера или портретного режима на планшетном компьютере, — наша головная боль только усилится (**рис. 4.3**). В верхней части экрана творится полное безобразие: картинка обрезана настолько, что непонятно, что там вообще изображено, а бедный логотип стал еще меньше. Навигация же выглядит просто непотребно. С этим нужно срочно что-то делать.

Рис. 4.3. Любой посетитель сайта будет в восторге от нашего исковерканного дизайна (это сарказм)

Двигаемся ниже. Господи, что же это происходит с сайтом (**рис. 4.4**)! Раньше двухколоночная верстка обеспечивала легкий доступ к дополнительной информации, сейчас же она сжимает текст, такие короткие строчки читать крайне неудобно. Фотография не совпадает с текстом, а что на ней изображено, не понять никому.

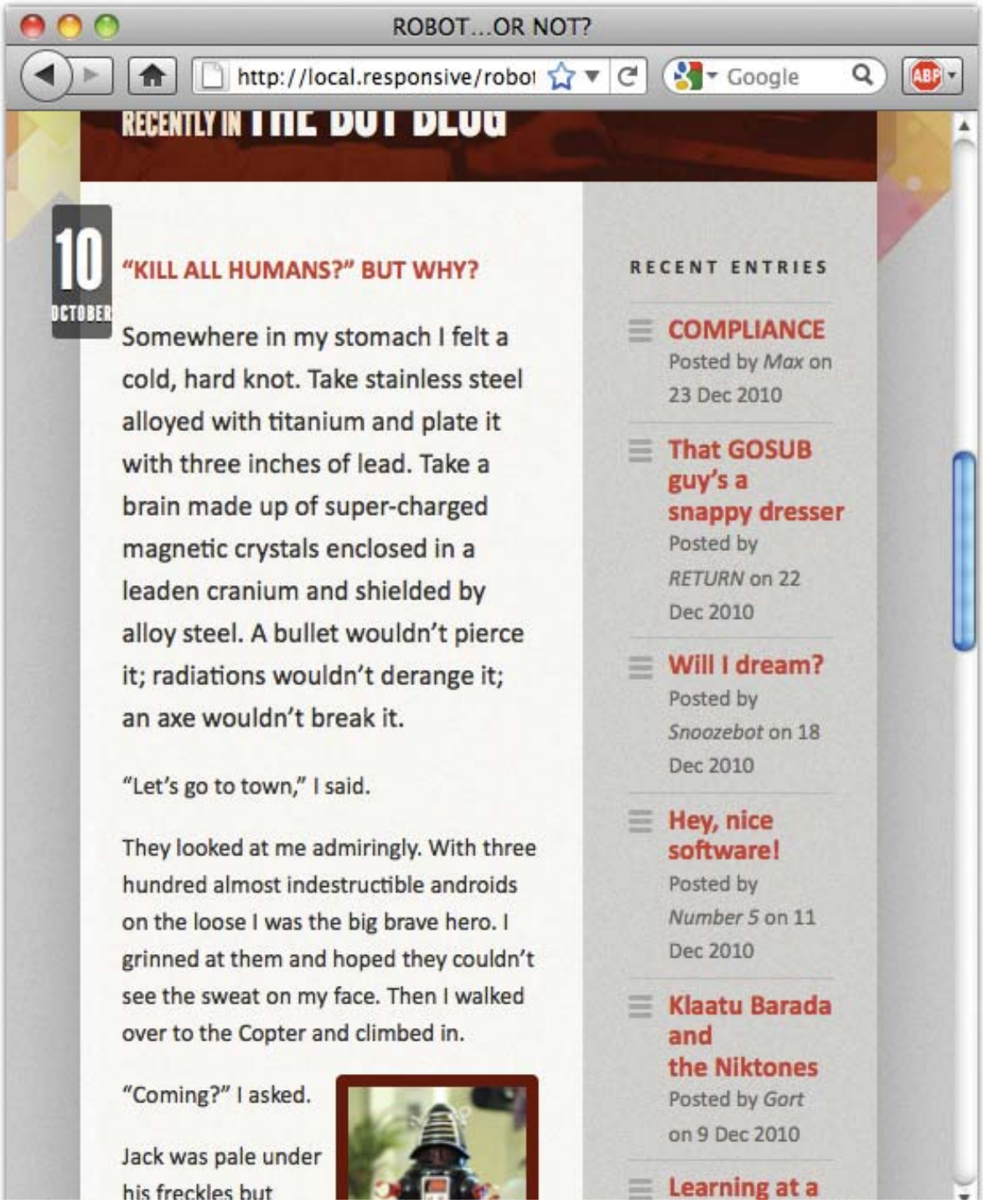

Рис. 4.4. Эта запись напоминает японское стихотворение хайку — строчки, короткие до боли

И наконец, заканчивая наш печальный обзор, посмотрим на фотографии в нижней части страницы. Они выглядят хуже всего

(**рис. 4.5**), даже хуже картинки в верхней части. Они такие маленькие, что разглядеть, что там на них, невозможно.

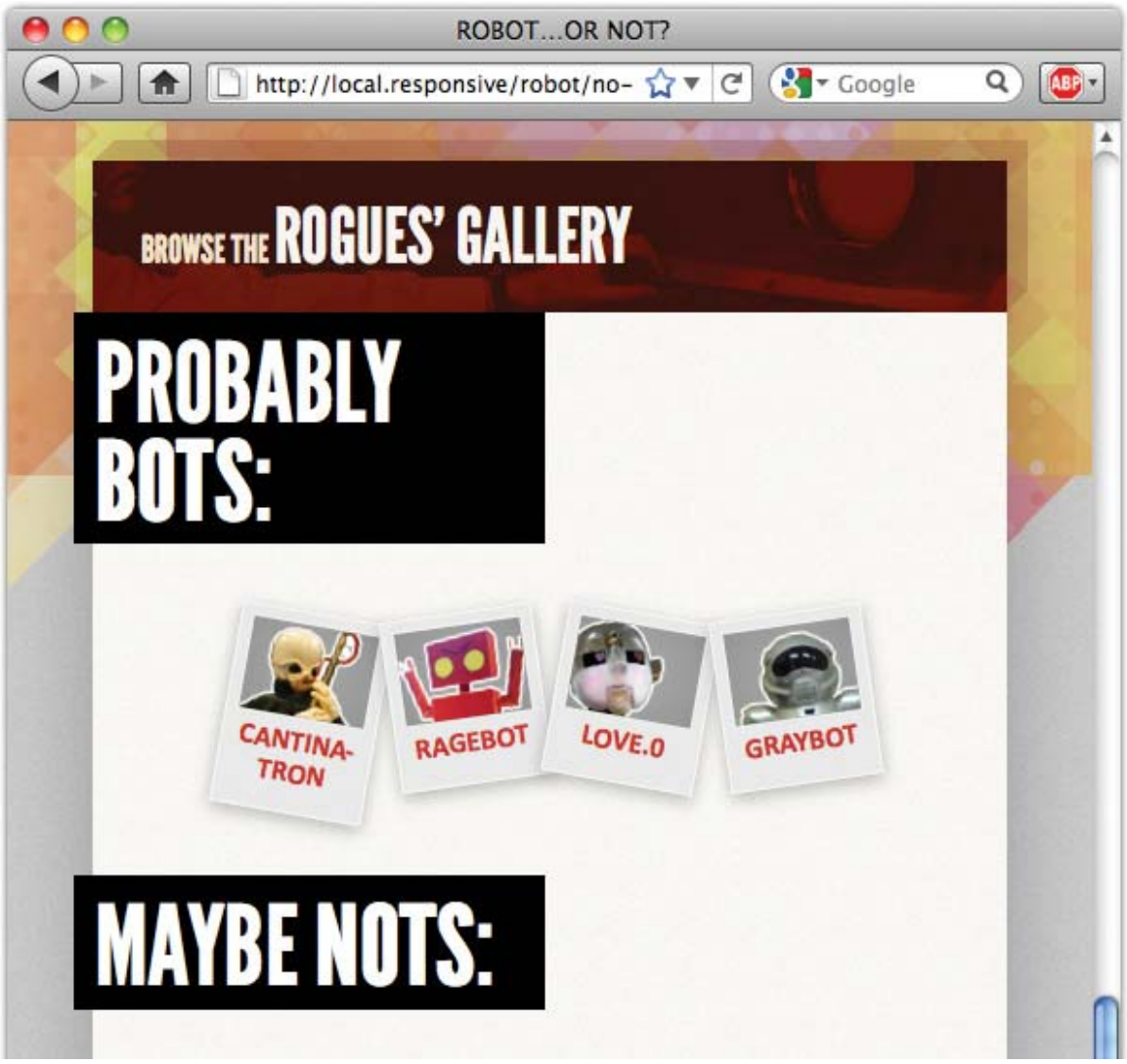

Рис. 4.5. Мелкие картинки, монструозные поля. Отвратительно!

Широкие поля, которые мы использовали для обрамления этих картинок, превратились в огромные пробельные моря, поглотившие их.

Широкоэкранные неприятности

Однако проблемы возникают не только тогда, когда разрешение экрана меньше. Если мы максимально увеличим окно браузера,

на свет вылезут новые проблемы. Верхняя часть страницы выглядит довольно неплохо (**рис. 4.6**), правда, картинка теперь меньше, чем выделенное под нее место. В остальном же все более-менее нормально… Ну, далеко от идеального, но вытерпеть можно. Сетка в целом сохранилась хорошо.

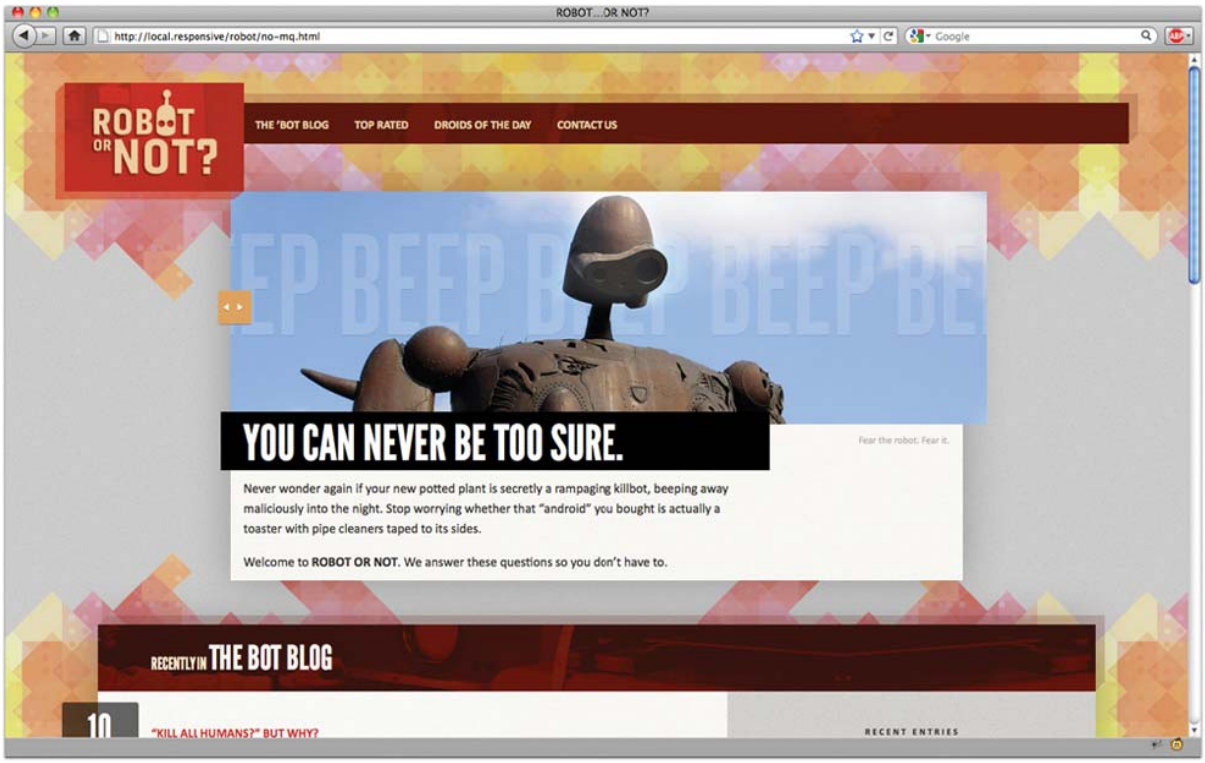

Рис. 4.6. Верхняя часть выглядит довольно широкой

А теперь прокрутим страницу вниз, и наше радостное настроение вмиг испарится (**рис. 4.7**). Помните, какими короткими выглядели строчки текста в маленьком окне? Глядя на то, что появилось на экране, я начинаю по ним скучать. Теперь строчки стали невероятно длинными, и, несмотря на то, что ничто не доставляет мне большего удовольствия, чем выискивать следующую строчку после прочтения предыдущей, придется искать другой способ.

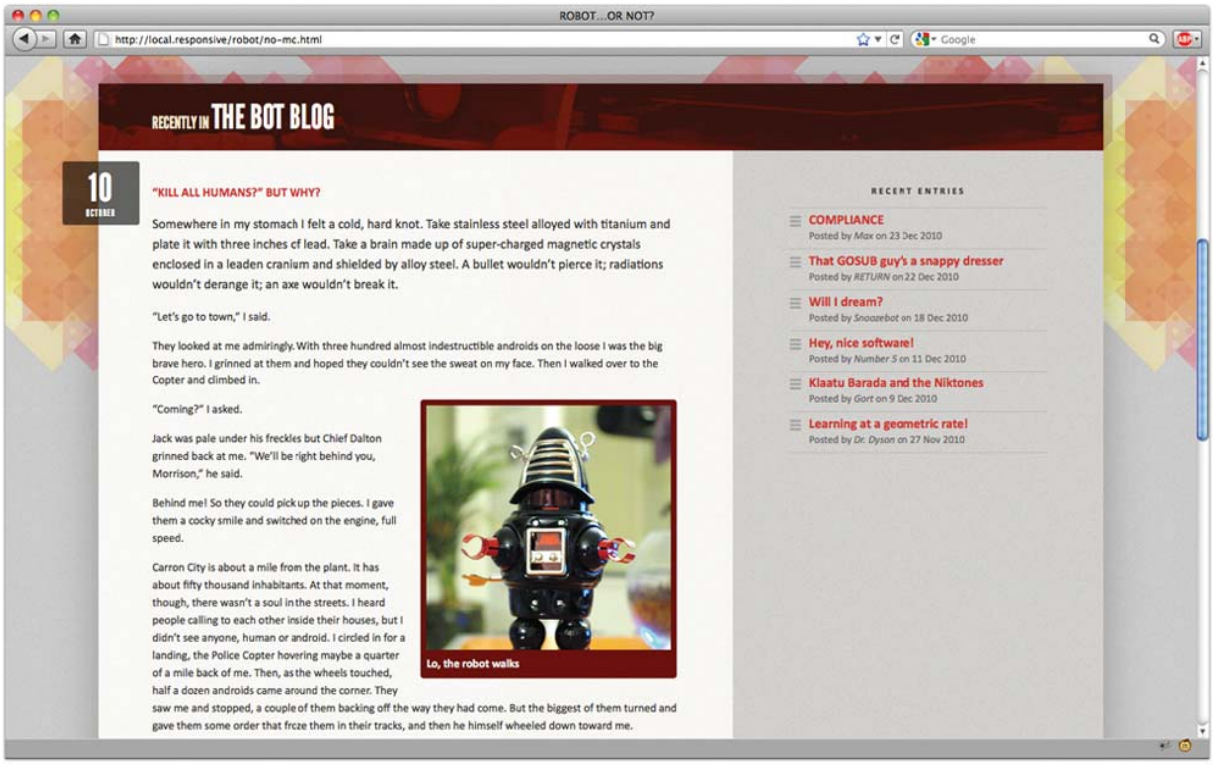

Рис. 4.7. Двигаясь вниз по странице, мы видим все больше проблем. Длинные строки, крошечные изображения, печальный Итан

Ко всему прочему, фотографии в нижней части страницы стали невероятно большими (**рис. 4.8**). Выглядят они неплохо, но занимают слишком много места. На самом деле на моем мониторе даже и не видно, есть ли что-то над или под этим блоком. Интересно, можем мы сделать хоть что-то, чтобы читатели не сломали глаза, рассматривая наш сайт?

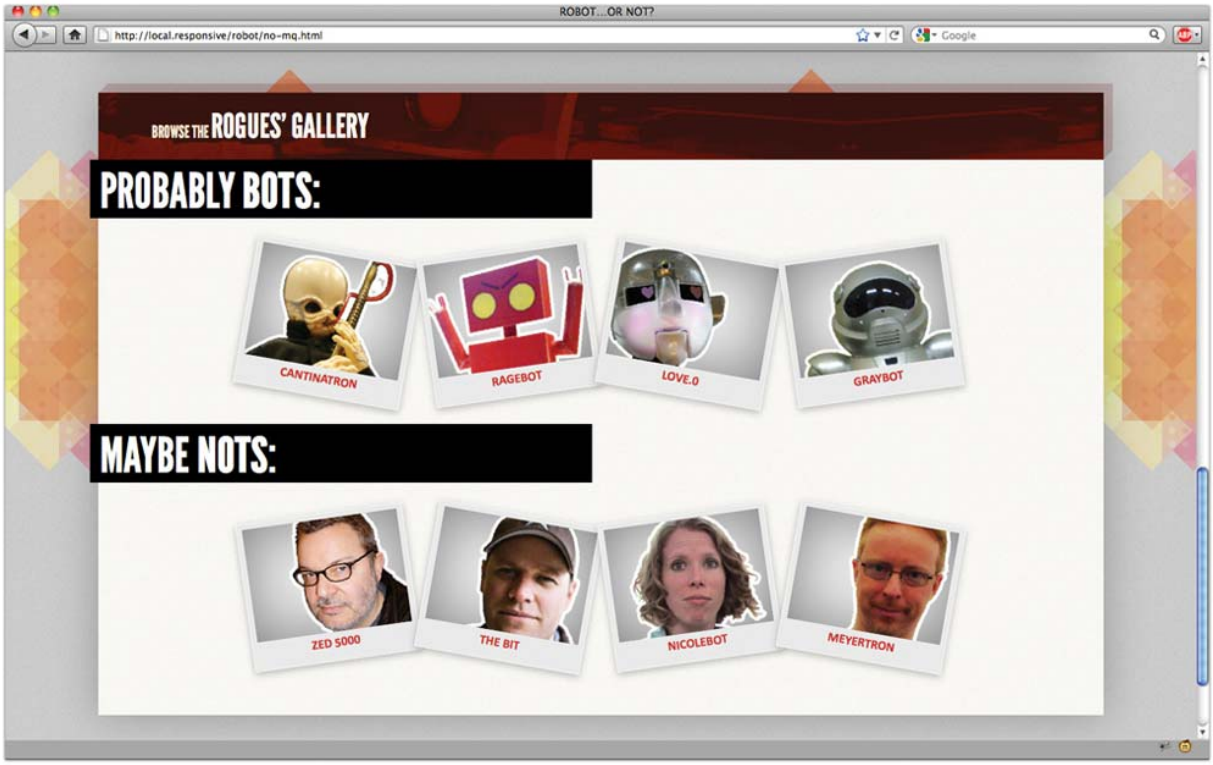

Рис. 4.8. Говоря техническим языком, эти изображения слишком крупные и массивные

НАСУЩНЫЕ ПРОБЛЕМЫ

Итак, мы определили основные визуальные неполадки. Однако нужно смотреть на проблему шире. Как только мы меняем оригинальное разрешение, сетка оказывает нежелательное воздействие на контент. Ее пропорции ограничивают содержание при низких разрешениях и окружают его пустым пространством — при высоких.

Причем эта проблема возникает не только с гибкими макетами. Ни один дизайн, фиксированный или гибкий, не сможет масштабироваться вне контекста, для которого он спроектирован.

Так как же нам сделать дизайн, который будет адаптироваться к изменениям разрешения экрана и размеров области просмотра? Как сделать так, чтобы страница оптимизировалась в соответствии с браузером и устройством, на котором ее просматривают?

Другими словами, как сделать наш дизайн более отзывчивым?

НАВСТРЕЧУ ОТЗЫВЧИВОСТИ

К счастью, Консорциум Всемирной паутины (World Wide Web Consortium, W3C) уже некоторое время занимается этой проблемой. Но чтобы лучше понять решение, которое в результате было представлено, обратимся к предыстории.

Знакомьтесь: медиатипы

Первым шагом в решении проблемы стали **медиатипы (media types)**, часть спецификации CSS2 (http://bkaprt.com/rwd/24/). Вот как они первоначально описывались:

> *Иногда таблицы стилей для различных медиатипов могут иметь одинаковое свойство, но требовать разных значений для этого свойства. Например, свойство font-size можно использовать как для монитора, так и для вывода документа на печать. Эти два медиатипа отличаются друг от друга и требуют разных значений для одного и того же свойства; документ обычно имеет больший шрифт на мониторе, чем на бумаге. Поэтому это нужно отобразить в таблице стилей или в разделе таблицы, применяемой к определенным медиатипам.*

Ничего не понятно, да? Давайте попробуем разобраться без нагромождения терминов.

Вы писали когда-нибудь стили для печати (http://bkaprt.com/rwd/25/)? Тогда вы, наверное, знакомы с понятием разработки для различных видов медиа. Даже идеальное браузерное отображение не делает никакой разницы между десктопными браузерами и принтерами или между мобильными устройствами и голосовым браузером. Чтобы решить эту проблему, W3C создала список медиатипов (http://bkaprt.com/rwd/26/) для классификации каждого браузера или устройства по медиакатегориям. Медиатипы могут принимать значения: `all, braille, embossed,`

`handheld`, `print`, `projection`, `screen`, `speech`, `tty` и `tv`.

С некоторыми из этих медиатипов, как, например, `print`, `screen` или даже `projection`, вы уже работали. Некоторые другие — `embossed` (для брайлевских принтеров) или `speech` (для голосовых браузеров и интерфейсов) — встречаются впервые. Но все эти медиатипы созданы с одной целью: чтобы мы могли лучше проектировать дизайн для каждого типа браузера или устройства, просто загружая нужный CSS. Следовательно, устройство с экраном будет игнорировать CSS, созданный для медиатипа `print`, и наоборот. А для стилевых правил, которые применимы ко всем устройствам, в спецификации создана супергруппа `all`. На практике это означает правку атрибута `media` элемента `link`:

```
<link rel="stylesheet" href="global.css"
media="all" />
<link rel="stylesheet" href="main.css"
media="screen" />
<link rel="stylesheet" href="paper.css"
media="print" />
```

А также создание блока `@media` в таблице стилей и его привязку к определенному медиатипу:

```
@media screen {
  body {
    font-size: 100%;
  }
}

@media print {
  body {
```

```
    font-size: 15pt;
  }
}
```

В любом случае спецификация предлагает браузеру определить, к какому медиатипу он относится. («Я десктопный браузер! Я отношусь к медиатипу `screen`», «Я пахну чернилами и тонером: я тип `print`», «Я браузер видеоконсоли: я тип `tv`» и т. д.) Загрузив страницу, браузер будет отображать только тот CSS, который относится к определенному медиатипу, и игнорировать все остальные. И — в теории — это потрясающая идея.

Но теория — это, наверное, последнее, что нужно занятому по горло веб-дизайнеру.

Неправильное распределение типов

Когда на сцене появились все эти браузеры для устройств с маленькими экранами, как, например, телефоны или планшеты, с ними пришли и проблемы. В соответствии со спецификацией решить эти проблемы несложно, нужно просто создать таблицу стилей для медиатипа `handheld`:

```
<link rel="stylesheet" href="main.css"
media="screen" />
<link rel="stylesheet" href="paper.css"
media="print" />
<link rel="stylesheet" href="tiny.css"
media="handheld"/>
```

Проблемы скорее кроются в нас самих, по крайней мере, частично. На первых мобильных устройствах не было эффективно работающих браузеров, поэтому мы просто игнорировали их, разрабатывая вместо этого таблицы стилей для медиатипов `screen` или `print`. А когда такие браузеры появились, в Сети не было достаточно CSS-файлов типа `handheld`. В результате

многие разработчики мобильных браузеров решили использовать таблицы стилей для медиатипа `screen`.

А растущий диапазон мобильных устройств еще больше усложняет дело. Сможет ли одна и та же таблица стилей решить все проблемы для iPhone и для телефона пятилетней давности?

Знакомство с медиазапросами

К счастью, организация W3C включила в спецификацию CSS3 синтаксис медиазапросов, усовершенствовав методологию медиатипов. Медиазапросы позволяют не только ориентироваться на конкретный класс устройств, но и анализировать физические характеристики устройства, использующегося для отображения страницы (http://bkaprt.com/rwd/27/).

Взгляните на следующий отрывок:

```
@media screen and (min-width: 1024px) {
  body {
    font-size: 100%;
  }
}
```

Каждый медиазапрос, включая и этот, содержит два компонента:

1. Он начинается с медиатипа (`screen`), взятого из списка утвержденных медиатипов спецификации CSS2.1 (http://bkaprt.com/rwd/26/).
2. После типа идет сам запрос в скобках: (`min-width: 1024px`), который тоже можно разделить на две составляющие: название свойства (`min-width`) и соответствующее значение (`1024px`).

Считайте, что медиазапросы просто проверяют ваш браузер. В процессе считывания таблицы стилей браузер получает вопрос от медиазапроса `screen and (min-width: 1024px)`: относится

ли он к медиатипу `screen`, и если да, то имеет ли он ширину области просмотра не меньше 1024 пикселей. Если браузер отвечает на оба вопроса положительно, вложенные в запрос стили отображаются, в противном случае браузер попросту игнорирует их и занимается своими делами.

Этот медиазапрос вписан в объявление `@media`, что позволило включить его непосредственно в таблицу стилей. Но вы можете также поместить запрос в элемент ссылки (`link`), вставив его в атрибут `media`:

```
<link rel="stylesheet" href="wide.css"
media="screen and (min-width: 1024px)" />
```

Кроме того, вы можете прикрепить его к инструкции `@import`:

```
@import url("wide.css") screen and (min-width:
1024px);
```

Лично я предпочитаю использовать `@media`, поскольку он хранит ваш код в отдельном файле, уменьшая количество внешних запросов браузера к серверу.

В принципе, неважно, куда вы впишете запрос, результат будет одинаковым: если браузер соответствует медиатипу и при этом выполняет условие, указанное в запросе, вложенные в запрос CSS выполняются, если нет — игнорируются.

Познакомьтесь с характеристиками

Однако дело не только в том, чтобы проверить ширину и высоту. Запросы могут проанализировать массу характеристик, указанных в спецификации. Но прежде чем мы приступим к этому делу, давайте сначала определимся с терминами, зачастую сложными и непонятными.

1. В спецификации сказано, что каждое устройство имеет «область просмотра» (display area) и «площадь изображения»

(rendering surface). Ну и что это такое? Переведем на наш язык: окно просмотра браузера — это область просмотра, а весь монитор — площадь изображения. На вашем ноутбуке областью просмотра будет окно браузера; площадью изображения — экран.

2. Чтобы задать определенные значения, некоторые характеристики могут принимать префиксы `min-` и `max-`. Например, вы можете вписать (`min-width: 1024px`) и (`max-width: 1024px`), чтобы задать область просмотра более или менее 1024 пикселей соответственно.

Все понятно? Великолепно. С этим разобрались, давайте рассмотрим характеристики, которые в соответствии со спецификацией мы можем использовать в наших запросах (http://bkaprt.com/rwd/28/) (**табл. 4.1**).

Табл. 4.1. Характеристики устройств, тестируемых с использованием медиазапросов

ХАРАКТЕРИСТИКА	ОПРЕДЕЛЕНИЕ	Наличие и отсутствие префиксов `max-` и `min-`
`width`	Ширина области просмотра	✓
`height`	Высота области просмотра	✓
`device-width`	Ширина площади изображения устройства	✓
`device-height`	Высота площади изображения устройства	✓
`orientation`	Принимает значения portrait или landscape	✗
`aspect-ratio`	Отношение ширины области просмотра к ее высоте. Например, в запросе можно указать aspect-ratio окна браузера 16:9	✓
`device-aspect-ratio`	Отношение ширины площади изображения к ее высоте. Например, в запросе можно указать aspect-ratio окна браузера 16:9	✓

`color`	Определяет число бит на канал цвета. Например, устройство с 8-битной цветностью выполнит запрос (color: 8). Монохромное устройство вернет значение 0	
`color-index`	Определяет количество цветов, которое поддерживает устройство. Например, @media screen and (min-color-index: 256)	
`monochrome`	Подобно color, характеристика monochrome позволяет нам проверить количество битов на пиксель в монохромном устройстве	
`resolution`	Позволяет проверить плотность пикселей в устройстве. Например, screen and (resolution: 72dpi) или screen and (max-resolution: 300dpi)	
`scan`	Для телевизионных браузеров — проверяет статус процесса сканирования: progressive или scan	
`grid`	Определяет, что это устройство с фиксированным размером символов. Например, телефон, который поддерживает один шрифт, размеры букв которого занимают одинаковую ширину и высоту и выстраиваются по заданной сетке	

А еще мы можем связывать несколько запросов в цепочку, соединяя их словом `and`:

```
@media screen and (min-device-width: 480px) and
(orientation: landscape) { … }
```

То есть мы можем задать несколько характеристик в одном запросе, выполняя тем самым более сложный анализ устройства, на котором просматривается наш дизайн.

Знай свои характеристики

Чувствуете себя королем мира? Тогда мне именно сейчас следует сказать, что не все браузеры, распознающие `@media`, поддерживают создание запросов для всех характеристик, указанных в спецификации.

Вот вам пример. Когда Apple выпустила свой первый iPad, он поддерживал медиазапрос `orientation`. Это значит, что вы могли написать запрос `orientation: landscape` или `orientation: portrait` для обслуживания устройства средствами CSS. Круто, да? К сожалению, iPhone не поддерживал

запрос orientation до тех пор, пока несколько месяцев спустя не вышло обновление операционной системы. В то время как все устройства позволяли пользователю изменить ориентацию, браузер iPhone не понимал запросы на эту характеристику.

Мораль этой истории? Внимательно изучайте устройства и браузеры на предмет запросов характеристик, которые они поддерживают, и проверяйте их надлежащим образом.

В то время как поддержка современных браузеров и устройств все еще находится в процессе развития, медиазапросы дают нам прекрасную возможность четко сформулировать то, как именно должен выглядеть наш дизайн на различных устройствах и браузерах.

БОЛЕЕ ОТЗЫВЧИВЫЙ РОБОТ

Медиазапросы — это последний элемент отзывчивого веб-сайта. На протяжении двух глав мы делали гибкий макет, который, однако, является всего лишь основой. Теперь мы можем использовать медиазапросы для коррекции любых визуальных дефектов, возникающих при изменении формы и размеров области просмотра.

Кроме того, мы можем использовать медиазапросы для оптимизации отображения нашего контента в соответствии с требованиями определенного устройства, создавая альтернативные макеты под определенные диапазоны разрешений. Загружая правила стилей, касающиеся этих диапазонов, медиазапросы позволяют нам создавать страницы, более чувствительные к требованиям устройств, которые их отображают.

Другими словами, объединяя гибкие макеты и медиазапросы, мы наконец-то сможем сделать наш сайт отзывчивым.

Приступим.

В компании с viewport

Мы уже определили основные пункты, на которые стоит обратить внимание в нашем дизайне. Но прежде чем применять

медиазапросы, мы должны еще раз взглянуть на оригинальный макет.

Когда в 2007 году Apple выпустила iPhone, она создала новое значение атрибута элемента `meta` для Mobile Safari: `viewport` (http://bkaprt.com/rwd/29/). Зачем? Размеры дисплея iPhone — 320 x 480, но Mobile Safari фактически отображает веб-страницы шириной 980 пикселей. Если вы когда-нибудь заходили на сайт газеты New York Times (http://nytimes.com) с телефона с браузером на движке WebKit (**рис. 4.9**), вы могли видеть это в действии: Mobile Safari рисует страницу на холсте шириной `980px`, а затем сжимает ее, чтобы уместить на экране с разрешением 320 x 480.

Рис. 4.9. По умолчанию браузер Mobile Safari воспроизводит контент с шириной 980px, даже когда вы держите телефон в горизонтальной плоскости и ширина ограничена 320px

При помощи тега `viewport` мы можем контролировать размеры этого холста и задавать точные размеры области просмотра браузера. Например, мы можем установить фиксированную ширину в `320px`:

```
<meta name="viewport" content="width=320" />
```

С того момента, как Apple представила механизм `viewport`, многие разработчики браузеров приняли ее, сделав стандартом де-факто. Давайте попробуем включить его в наш дизайн, который скоро станет отзывчивым. Однако вместо того, чтобы устанавливать фиксированную ширину в пикселях, используем подход, не зависящий от разрешения. В блоке `head` нашего HTML пишем элемент `meta`:

```
<meta name="viewport" content="initial-
scale=1.0, width=device-width" />
```

Свойство `initial-scale` устанавливает уровень масштабирования страницы на `1.0`, или 100%, что обеспечивает некоторую согласованность распознающих viewport браузеров, для устройств с маленькими экранами. (Более детальную информацию по масштабированию под различные мониторы вы сможете найти в объяснении Mozilla: http://bkaprt.com/rwd/30/.)

Большое значение для нас имеет параметр `width=device-width`, который делает ширину окна просмотра браузера равной ширине экрана устройства. Например, на iPhone область макета Mobile Safari уже составляет не `980px`, а 320 пикселей в портретном режиме и 480 — в ландшафтном.

Имея на руках это значение, мы можем использовать `max-width` и `min-width` для поиска диапазонов разрешений ниже или выше определенного порога и загружать в CSS, предназначенного для этих диапазонов. Кроме того, в этом случае браузеры, распознающие запросы, могут воспользоваться ими,

что сделает наш дизайн отзывчивым для всех пользователей, неважно, с какого устройства они его просматривают — с телефона, планшета, стационарного компьютера или ноутбука.

Так, что-то я заболтался. Давайте лучше взглянем на пресловутые запросы в действии.

МЕДИАЗАПРОСЫ В ДЕЙСТВИИ

Вы помните тот большой внушительный заголовок (**рис. 4.10**)? Вот CSS, который его сделал таким:

```
.main-title {
  background: #000;
  color: #FFF;
  font: normal 3.625em/0.9 "League Gothic",
"Arial Narrow", Arial, sans-serif; /* 58px /
16px */
  text-transform: uppercase;
}
```

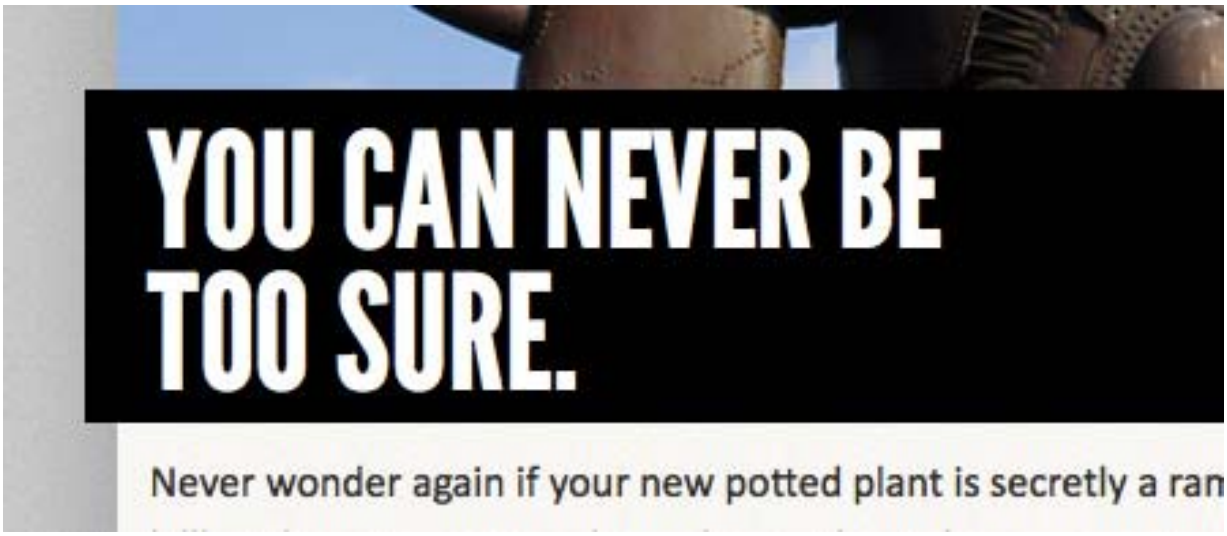

Рис. 4.10. При должном умении заголовок может стать вполне внушительным

Я упустил несколько презентационных свойств, потому что меня больше заботит то, как этот ужасный огромный заголовок

выглядит при небольшом разрешении. Он написан торжественным шрифтом League Gothic (http://bkaprt.com/rwd/31/) белого цвета (`color: #FFF`) на черном фоне (`background: #000`). И если вдруг кто-то все еще не воспринимает его всерьез, учтите, что он написан заглавными буквами (с помощью `text-transform`) размером `3.625 em`, или `58 px`.

Что ж, пока все идет хорошо. Но, как мы уже убедились, если уменьшить окно браузера или просматривать страницу на устройстве с меньшим экраном, его дизайн выглядит неправильно, поскольку совсем не масштабируется.

Давайте исправим этот недостаток.

Сначала вставим блок `@media` где-то после первого правила `.main-title`, в котором напишем запрос для более узкого диапазона разрешения:

```
@media screen and (max-width: 768px) { … }
```

В этом запросе мы дали команду браузеру обрабатывать вложенный CSS только в том случае, если ширина окна просмотра не превышает 768 пикселей. Почему `768 px`? Потому что ширина окна просмотра телефонов, поддерживающих медиазапросы, как и большей части современных планшетов, меньше этого значения. По крайней мере, в определенных режимах. Например, разрешение iPad в портретном режиме составляет `768 px`, а в ландшафтном — `1024 px`.

Но поскольку мы используем `max-width`, а не `max-device-width`, более узкие окна браузеров на вашем компьютере или ноутбуке также примут этот диапазон разрешения. (Помните: характеристики `width` и `height` определяют область просмотра или окно браузера, тогда как параметры `device-width` и `device-height` — размеры всего экрана).

Написав этот запрос, можем приступать к обработке тех элементов дизайна, которые не масштабируются. Сначала

давайте еще раз обратимся к нашему огромному заголовку. Чтобы сделать его более удобоваримым, впишем в медиазапрос правило `.main-title` с другими свойствами CSS — вместо тех, которые вызывают у нас только головную боль:

```
@media screen and (max-width: 768px) {
  .main-title {
  font: normal 1.5em Calibri, Candara, Segoe,
"Segoe UI", Optima, Arial, Helvetica, sans-
serif; /* 24px / 16px */
  }
}
```

Первое правило `.main-title` применяется всеми браузерами, которые читают наш CSS. Однако для более узких окон браузеров или устройств (разрешение которых не шире 768 пикселей) применяется второе правило, заменяющее первое. Мы сделали два изменения: во-первых, уменьшили кегль элемента `.main-title` с `3.625em` (около `58 px`) до `1.5em`, или `24 px`. На мелких экранах такой шрифт смотрится лучше.

Во-вторых, шрифт, который мы прежде использовали для этого заголовка — наш любимый League Gothic, совсем не смотрится на таких экранах (**рис. 4.11**). Поэтому я решил заменить его семейством шрифтов (`Calibri, Candara, Segoe, "Segoe UI", Optima, Arial, Helvetica, sans-serif`). Теперь заголовок стал более читабельным (**рис. 4.12**).

Рис. 4.11. Шрифт League Gothic, несмотря на всю свою прелесть, кажется слишком мелким и узким

Рис. 4.12. Не так красиво, как League Gothic? С ним вообще сложно что-то сравнить. Однако этот новый шрифт читается куда лучше, да и вполне соответствует дизайну

Вы, наверное, заметили, что мы не переписывали другие свойства в первом правиле `.main-title`. То есть черный фон, белый цвет шрифта и заглавные буквы все еще применяются к нашему уменьшенному заголовку. Запрос переписал только те характеристики, которые нас не устраивали.

Вуаля! При помощи медиазапроса мы исправили заголовок, и теперь на маленьких экранах он смотрится прекрасно (**рис. 4.13**).

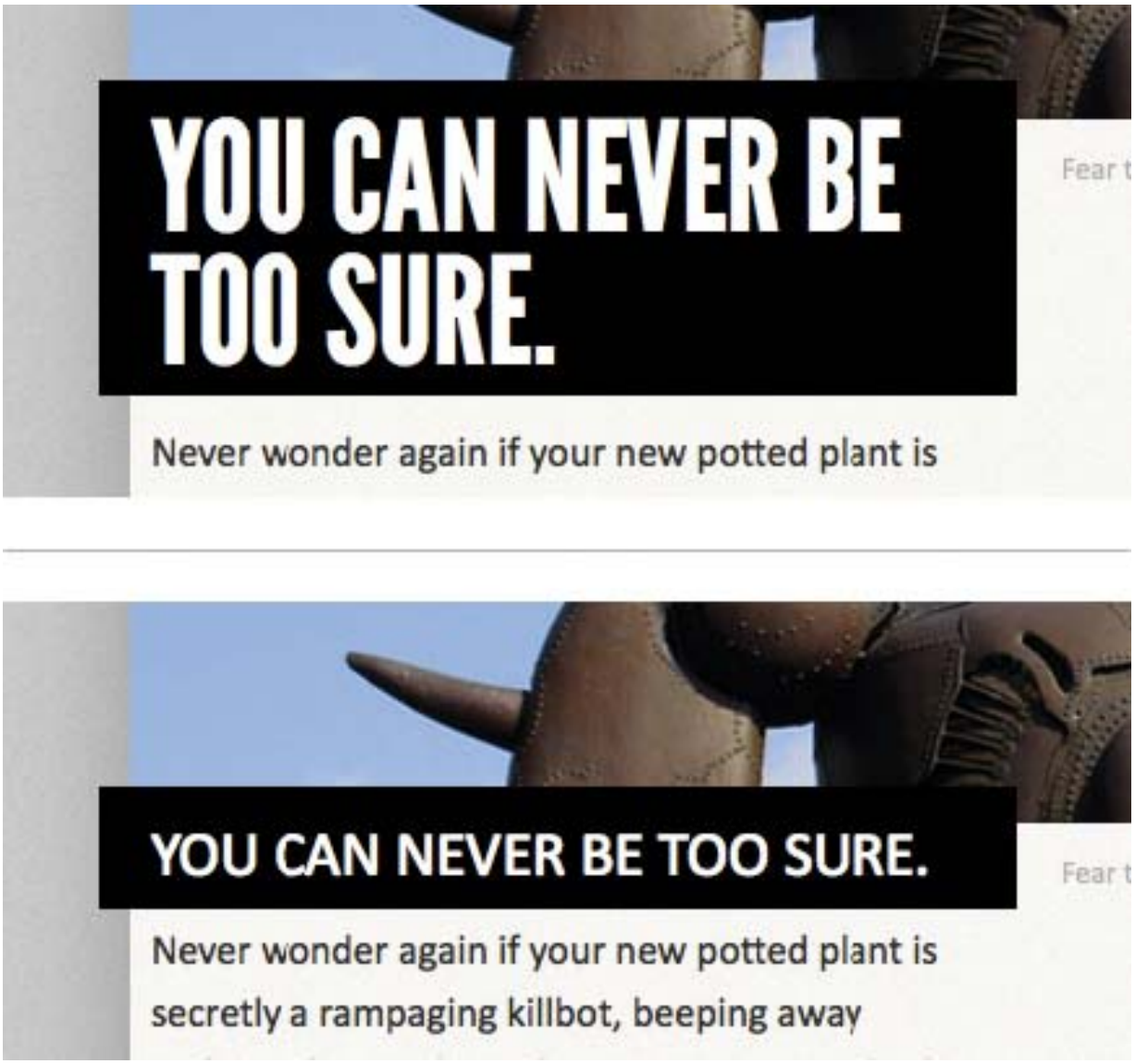

Рис. 4.13. Сравните изначальный вариант заголовка (вверху) с вариантом, получившимся вследствие применения запроса

Но это только начало. Мы можем не только подправить шрифтовое оформление, но и решить более сложные проблемы, связанные с дизайном.

Все дело в деталях

Давайте сделаем новый медиазапрос и немного подправим макет нашей страницы. Помните наш гибкий контейнер `#page` из второй главы? Вот как выглядит его CSS на данный момент:

```
#page {
  margin: 36px auto;
  width: 90%;
}
```

Мы видим, что контейнер занимает `90%` окна браузера и отцентрирован по горизонтали (`margin: 36px auto`). Прекрасно, но давайте добавим правило в уже существующий медиазапрос, чтобы подрегулировать его характеристики при отображении на устройстве с разрешением меньше оригинального:

```
@media screen and (max-width: 768px) {
  #page {
    position: relative;
    margin: 20px;
    width: auto;
  }
}
```

Теперь в случае, если разрешение будет меньше `768 px`, элемент `#page` займет всю ширину окна браузера минус поля по краям шириной `20px`. Это небольшое изменение обеспечивает нам больше пространства на экранах с меньшим разрешением.

С контейнером разобрались, теперь обратимся к области контента:

```
@media screen and (max-width: 768px) {
  #page {
    margin: 20px;
    width: auto;
   }
```

```
    .welcome,
    .blog,
    .gallery {
      margin: 0 0 30px;
      width: auto;
    }
  }
```

Новое правило выбирает три блока контента верхнего уровня — введение (`.welcome`), блог (`.blog`) и фотогалерею (`.gallery`) — и убирает их горизонтальные отступы, позволяя им занять всю ширину `#page`.

Таким образом, мы привели макет нашей страницы к более линейному виду, сделав его более читабельным на устройствах с маленькими экранами (**рис. 4.14**). Я заслужил похвалу?

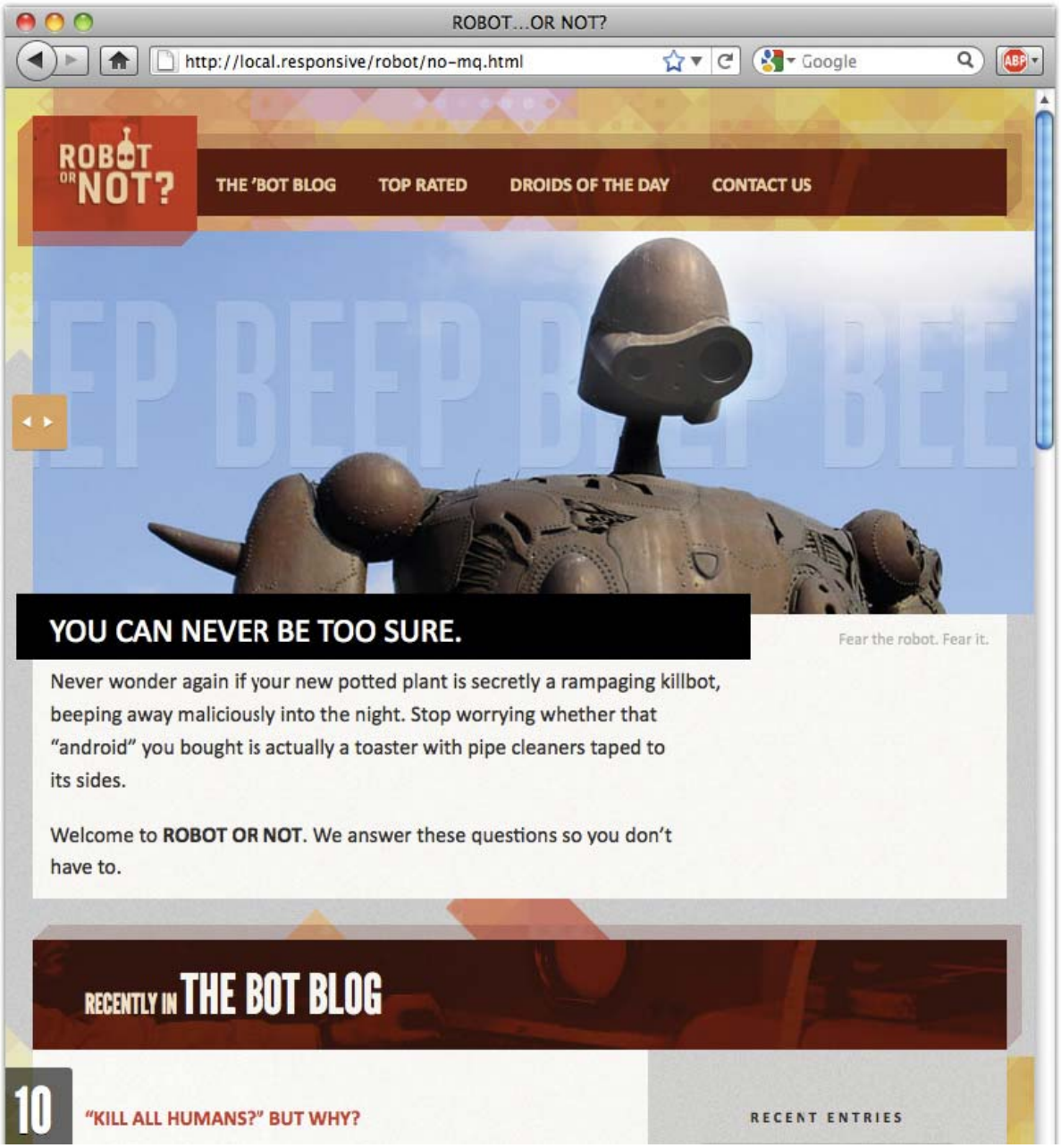

Рис. 4.14. Наш контент кажется более выровненным благодаря применению двух дополнительных правил. Однако чего-то еще не хватает…

Нет? Что вы сказали? В верхней части страницы все еще висит пугающе огромная картинка (**рис. 4.15**)?

Рис. 4.15. Однозначно над этим рисунком еще надо поработать

Ну что ж, *наверное*, и эту проблему можно решить, если она вас действительно беспокоит. Но прежде давайте снова взглянем на первоначальную разметку этого изображения, которое должно быть частью модуля слайд-шоу (и это еще предстоит сделать):

```
<div class="welcome section">
  <div class="slides">
    <div class="figure">
      <b><img src="img/slide-robot.jpg" alt=""
/></b>
      <div class="figcaption">…</div>
    </div><!-- /end .figure -->

    <ul class="slide-nav">
      <li><a class="prev" href="#">Previous</a>
```

```
</li>
      <li><a class="next" href="#">Next</a>
</li>
    </ul>
  </div><!-- /end .slides -->

   <div class="main">
   <h1 class="main-title">You can never be
     too sure.</h1>
   </div><!-- /end .main -->
</div><!-- /end .welcome.section -->
```

Изрядный кусок HTML, да? Но по существу мы сделали модуль `.welcome`, в который поместили изображение и идущий за ним вступительный текст (`.main`). Изображение, в свою очередь, входит в блок `.figure`, а сам `img` заключен в элемент `b`, который действует как CSS-трюк.

Звучит слишком заумно? И я знаю почему. Но элемент `b`, как бы глупо здесь ни выглядел, на самом деле обрабатывает большой кусок макета. Вот как выглядит соответствующий CSS:

```
.slides .figure b {
  display: block;
  overflow: hidden;
  margin-bottom: 0;
  width: 112.272727%; /* 741px / 660px */
}

.slides .figure b img {
  display: block;
  max-width: inherit;
}
```

Сначала мы задали свойству `hidden` в элементе `b` значение `overflow`, то есть контейнер обрезал любой лишний контент. Теперь же гибкие изображения меняют свои размеры при изменении элемента `b`. Поэтому мы отменяем масштабирование `max-width: 100%` по отношению к изображениям слайд-шоу (`max-width: inherit`). В результате картинка робота будет попросту обрезана, если ее ширина превысит содержащий ее элемент `b`.

Как видите, ширина элемента `b` на самом деле больше 100%. Мы использовали формулу `target ÷ context = = result`, чтобы создать элемент больше, чем модуль `.welcome`, благодаря чему изображение немного выходит за рамки с правой стороны.

Как назло, ни один из этих эффектов не будет работать при низком разрешении. Но я везучий парень. Так что давайте кое-что допишем в конце нашего медиазапроса:

```
@media screen and (max-width: 768px) {
  .slides .figure b {
    width: auto;
}

  .slides .figure b img {
    max-width: 100%;
  }
}
```

Первое правило задает элементу `b` ширину auto, делая ее такой же, как и ширина его контейнера. Второе правило восстанавливает `max-width: 100%`, которое мы обсуждали в третьей главе, позволяя изображению увеличиваться и уменьшаться вместе с контейнером. Вместе эти два правила не позволяют изображению выходить за рамки контейнера, а при

расширении — за рамки остальной части дизайна (**рис. 4.16**). Не знаю, как вы, а я выдохнул с облегчением.

Рис. 4.16. Наш рисунок теперь оказался на своем месте. Я испытываю облегчение. А вы?

И все же, прежде чем мы сможем расслабиться, нам нужно исправить еще кое-что. Навигация в верхней части страницы все еще выглядит сильно сжатой. Кроме того, если хоть немного изменить область просмотра, текст снова переносится на следующую строчку (**рис. 4.17**).

Рис. 4.17. Поле Contact Us, почему ты нас так ненавидишь?

Разметка шапки довольно простая:

```
<h1 class="logo">
 <a href="/">
   <i><img src="logo.png" alt="Robot or Not?"
/></i>
 </a>
</h1>

<ul class="nav nav-primary">
 <li id="nav-blog"><a href="#">The &#8217;Bot
Blog</a>
   </li>
 <li id="nav-rated"><a href="#">Top Rated</a>
</li>
 <li id="nav-droids"><a href="#">Droids of the
Day</a>
   </li>
```

```
 <li id="nav-contact"><a href="#">Contact
Us</a></li>
</ul><!-- /end ul.nav.nav-primary -->
```

Итак, мы обозначили логотип тегом `h1`, сделали маркированный список для навигации и присвоили им классы `.logo` и `.nav-primary` соответственно. Но что делать с CSS?

```
.logo {
  background: #C52618 url("logo-bg.jpg");
  float: left;
  width: 16.875%; /* 162px / 960px */
}

.nav-primary {
  background: #5E140D url("nav-bg.jpg");
  padding: 1.2em 1em 1em;
}

.nav-primary li {
  display: inline;
}
```

Стили достаточно простые. Мы применили фоновые изображения к обоим элементам, а не к самому макету: мы подвинули изображение влево, чтобы оно перекрывало навигацию. А элементам списка внутри `.nav-primary` соответствует свойство `display: inline`. Это решает нашу проблему, по крайней мере, пока страница не становится настолько узкой, что внутренние элементы переносятся на следующую строчку.

Вот как выглядит медиазапрос:

```
@media screen and (max-width: 768px) {
  .logo {
    float: none;
    margin: 0 auto 20px;
    position: relative;
  }

  .nav-primary {
    margin-bottom: 20px;
    text-align: center;
  }
}
```

Мы убрали свойство `float`, которое было первоначально задано `.logo`, и отцентрировали его по горизонтали над меню. Также мы установили `text-align: center` для `.nav-primary`, расположив все элементы по центру. Все изменения видны невооруженным глазом (**рис. 4.18**). Логотип и основная навигация находятся в своих собственных рядах со своими собственными свойствами.

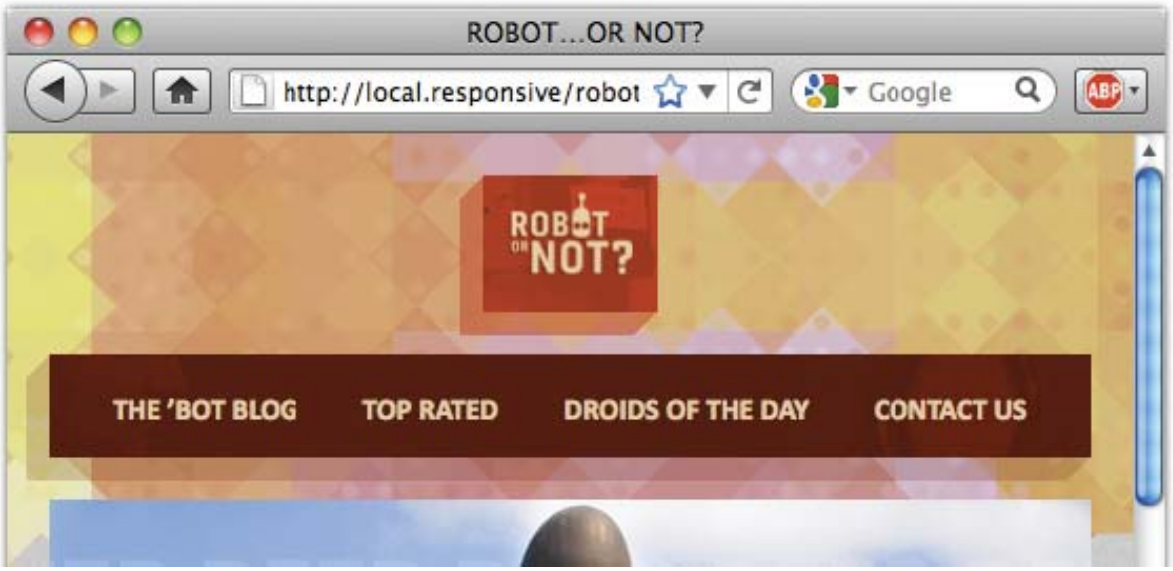

Рис. 4.18. Мы можем полностью перестроить верхнюю часть заголовка, чтобы дать возможность и логотипу, и строке навигации дышать полной грудью

Лично мне нравится, как выглядит навигация, однако расслабляться все равно еще рано. Для элементов навигации осталось не так уж и много места. Фактически, если мы хоть немного изменим размер экрана, наша четкая линия снова сломается, и текст перенесется на следующую строку (**рис. 4.19**).

Рис. 4.19. Слушайте, это уже не смешно

(У меня какая-то личная неприязнь к такому тексту. Не знаю почему.)

Мы обнаружили еще один проблемный момент, который невозможно исправить, просто передвинув логотип в свой собственный ряд. Значит, давайте напишем еще один медиазапрос и уберем возможность появления такой проблемы:

```
@media screen and (max-width: 768px) {
  …
}
```

```
@media screen and (max-width: 520px) {
  .nav-primary {
    float: left;
    width: 100%;
  }

  .nav-primary li {
    clear: left;
    float: left;
    width: 48%;
  }

  li#nav-rated,
  li#nav-contact {
    clear: right;
    float: right;
  }

  .nav-primary a {
    display: block;
    padding: 0.45em;
  }
}
```

Для еще более мелких экранов, с разрешением меньше 520 пикселей, мы передвинули каждый `li` внутри `.nav-primary`, присвоив второму и четвертому элементам свойство `float: right`. В результате мы получили более гибкую сетку 2 x 2, которая подстраивается под изменения размеров области просмотра, в отличие от `display: inline` (**рис. 4.20**).

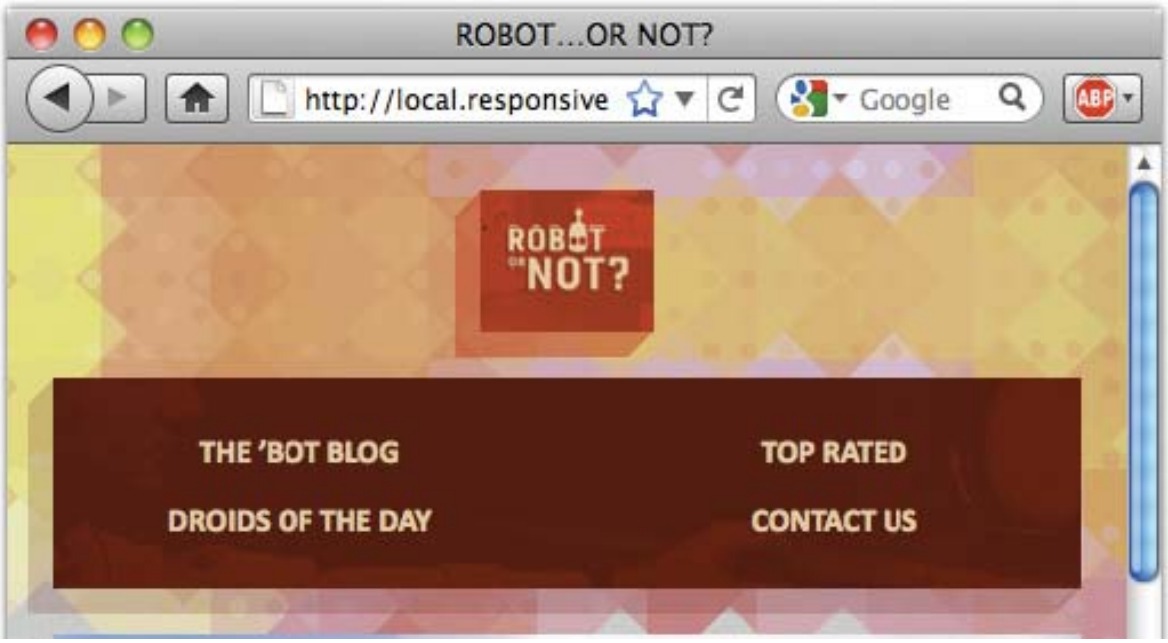

Рис. 4.20. Нужно ли говорить, насколько я доволен результатом? Нет? Тогда не буду

Стоит заметить, что нам не пришлось переписывать правила из предыдущего запроса (`screen and (max-width: 768 px)`) в этот, поскольку, если экран соответствует требованию «у́же, чем 520 пикселей», то он автоматически соответствует и требованию «у́же, чем 768 пикселей». Другими словами, правила из обоих запросов применяются к самым мелким разрешениям. В результате проблемы могут возникнуть только с областями просмотра шириной менее `520 px`.

Вот что у нас получилось (**рис. 4.21**). Немного доработав детали страницы, мы наконец-то получили дизайн, *соответствующий* устройству, на котором его просматривают. Мы больше не ограничены сеткой, макетом или типом, написанными первоначально для одного определенного диапазона разрешений. Наложенные поверх гибкого макета медиазапросы помогли нам решить проблемы, связанные с меньшими областями просмотра.

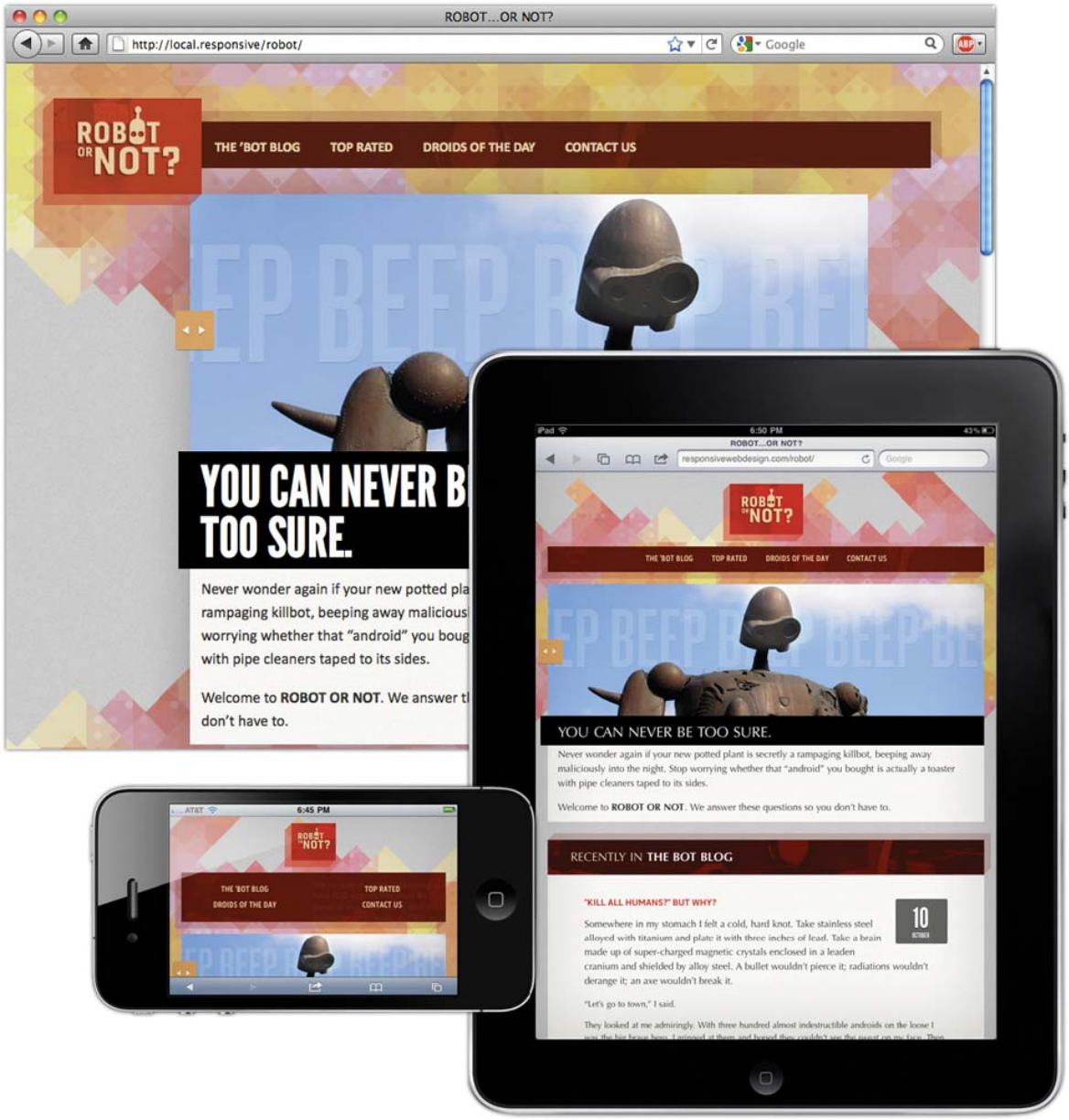

Рис. 4.21. Наш отзывчивый дизайн приобретает прекрасную форму, отлично масштабируясь даже на большом экране

Отворяй ворота!

Однако отзывчивый веб-дизайн — это не только дизайн, который хорошо смотрится на небольших экранах. При просмотре в максимизированном окне браузера также возникает ряд проблем: картинки растягиваются до невероятных размеров, строчки текста становятся слишком длинными, а сетка выходит за все

мыслимые пределы (**рис. 4.6–4.8**). Следовательно, нам необходимо наложить определенные ограничения на дизайн при помощи свойства `max-width`, выраженного в `em`, или пикселях. Давайте думать об этом как о возможности сделать дизайн для другого диапазона разрешений.

Для начала познакомимся с еще одним медиазапросом:

```
@media screen and (max-width: 768px) {
  …
}

@media screen and (max-width: 520px) {
  …
}

@media screen and (min-width: 1200px) {
  …
}
```

Первый запрос устанавливает потолок разрешения в 768 пикселей, то есть устройства и окна браузеров, ширина которых превышает ограничение `max-width`, будут попросту игнорировать вложенный в него CSS. Второй запрос повторяет действия первого, только для диапазона разрешения меньше `520 px` и при том же ограничении `max-width`.

В следующем запросе мы использовали свойство (`min-width: 1200px`) в качестве основного требования ко всем браузерам и устройствам. Если их ширина превышает `1200px`, они будут применять вложенные стили; если нет — они могут спокойно делать свои дела и ни о чем не думать.

Ну что ж, засучим рукава и приступим к работе над макетом для широких экранов:

```
@media screen and (min-width: 1200px) {
  .welcome,
  .blog,
  .gallery {
    width: 49.375%;
  }

  .welcome,
  .gallery {
    float: right;
    margin: 0 0 40px;
  }

  .blog {
    float: left;
    margin: 0 0 20px;
  }
}
```

На работающем сайте Robot or Not (http://responsivewebdesign.com/robot) вы найдете большое количество изменений, которые были внесены в макет, предназначенный для широкого экрана. Но эти три правила — основные. Мы присваиваем трем главным модулям контента (`.welcome, .blog,` и `.gallery`) практически половину (`49.375%`) ширины всей страницы. Затем мы передвигаем модули `.welcome` и `.gallery` вправо, а блог — влево. В результате получаем дизайн, который идеально подходит под широкие мониторы (**рис. 4.22**). Слишком длинные строчки стали короче, а блог, который представляет собой ключевой элемент контента, стал располагаться выше, что сделало его более доступным.

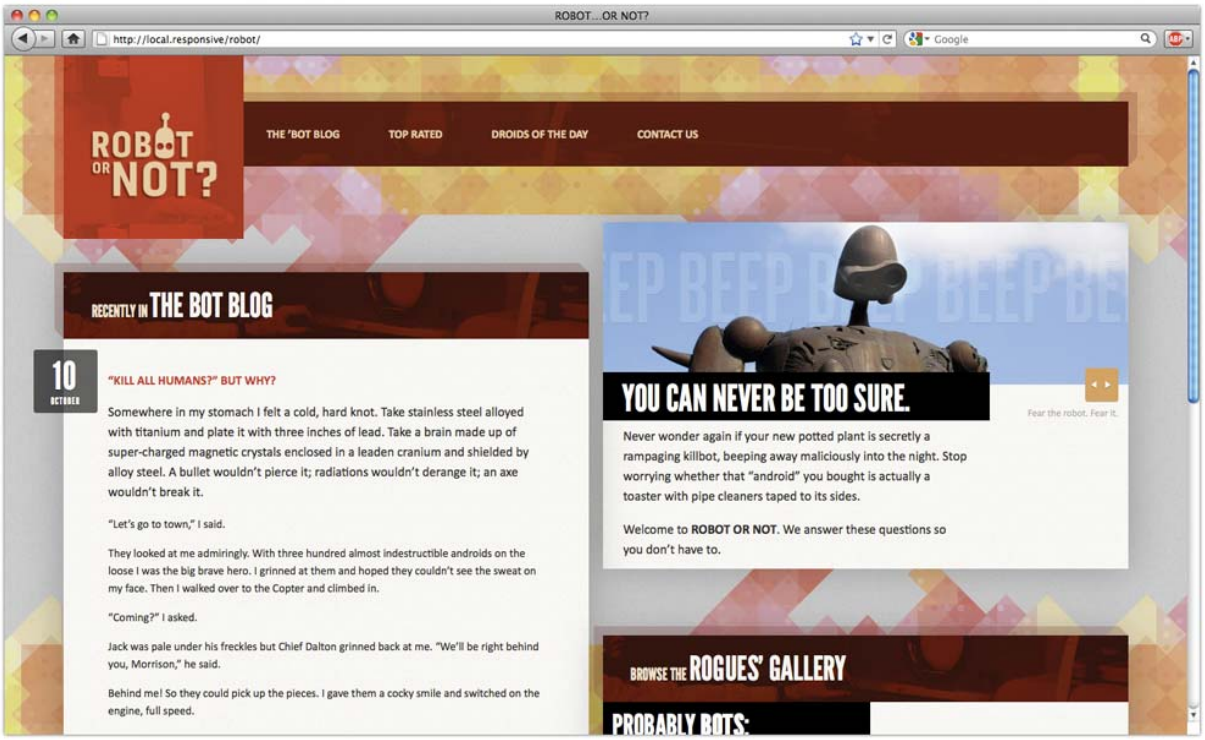

Рис. 4.22. Давайте еще раз посмотрим на наш дизайн глазами пользователей больших экранов. Он выглядит прекрасно — и все благодаря медиазапросу

Другими словами, наш отзывчивый дизайн закончен.

КОЕ-ЧТО ПО ПОВОДУ СОВМЕСТИМОСТИ

После того как мы уйму времени и страниц посвятили медиазапросам, настало время разрушить некоторые надежды, поскольку нам нужно поговорить о поддержке всего этого браузерами.

Хорошая новость заключается в том, что большинство современных десктопных браузеров поддерживают медиазапросы: среди них Opera 9.5+, Firefox 3.5+, браузеры на базе WebKit, такие как Safari 3+ и Chrome. Даже Internet Explorer 9 (http://bkaprt.com/rwd/32/) поддерживает медиазапросы (http://bkaprt.com/rwd/33/)! Кто-нибудь, ущипните меня.

Да и с мобильными браузерами дела обстоят неплохо. Медиазапросы поддерживают такие браузеры на базе WebKit, как Mobile Safari, webOS производства HP и Android. А по словам Питера-Пола Коха (http://bkaprt.com/rwd/34/), к ним не так давно

присоединились Opera Mobile и Opera Mini. Что касается Windows Phone, с 2011 года на них устанавливается IE9 (http://bkaprt.com/rwd/35/), браузер, который обеспечивает повсеместную поддержку медиазапросов. Что не может не радовать.

К сожалению, «повсеместную» совсем не означает «универсальную». С десктопными браузерами старше тех, которые перечислены, нам не повезло. И Internet Explorer версии 8 и ниже также не поддерживает медиазапросы, а это значит, что досточтимый IE6 по-прежнему остается проблемой. И несмотря на то, что многие современные устройства с маленькими экранами обеспечивают приличную поддержку запросов, некоторые широко используемые браузеры (IE Mobile и те, которые стоят на старых BlackBerry) их не понимают (http://bkaprt.com/rwd/36/).

Так что картина не совсем отрадная. Но это вовсе не означает, что отзывчивая верстка — несбыточная мечта. Прежде всего, существует достаточно решений на базе JavaScript, которые компенсируют отсутствие поддержки старых браузеров. Недавно созданная библиотека `css3-mediaqueries.js` (http://bkaprt.com/rwd/37/), которая, как предполагается, «позволяет IE5+, Firefox 1+ и Safari 2 интерпретировать, тестировать и применять медиазапросы стандарта CSS3». Это еще очень ранняя, не до конца доработанная версия, и кому-то может показаться, что она недостаточно функциональная, но лично я считаю ее весьма работоспособной.

Недавно я воспользовался маленькой шустрой библиотекой `respond.js` (http://bkaprt.com/rwd/38/), разработанной Скоттом Джелом. Там, где `css3-mediaqueries.js` кажется перегруженной функциями, иногда даже слишком, `respond.js` исправляет поддержку запросов `min-width` и `max-width` в старых браузерах. И он прекрасно работает практически для всех запросов, которые я написал на сегодня. Стоит упомянуть, что для того, чтобы этот скрипт работал как часы, необходимо

добавить определенным образом форматированный CSS-комментарий в конце каждого запроса:

```
@media screen and (max-width: 768px) {
  …
}/*/mediaquery*/

@media screen and (max-width: 520px) {
  …
}/*/mediaquery*/

@media screen and (min-width: 1200px) {
  …
}/*/mediaquery*/
```

Зачем он нужен? Часть кода `css3-mediaqueries.js` направлена на понимание структуры таблицы стилей: он открывает CSS и сразу же сообщает разницу между фигурной скобкой в конце CSS-правила и закрывающей скобкой в конце блока `@media`. Respond нет до этого никакого дела. Наоборот, он смотрит на этот небольшой комментарий и подхватывает запрос намного быстрее, чем другие скрипты.

Давайте добавим этот комментарий в конец каждого запроса сайта Robot or Not и вставим библиотеку `respond.js` в верхнюю часть страницы. Мы получим отзывчивый дизайн, который прекрасно работает в старых, не признающих медиазапросы браузерах, как, например, Internet Explorer 7 (**рис. 4.23**).

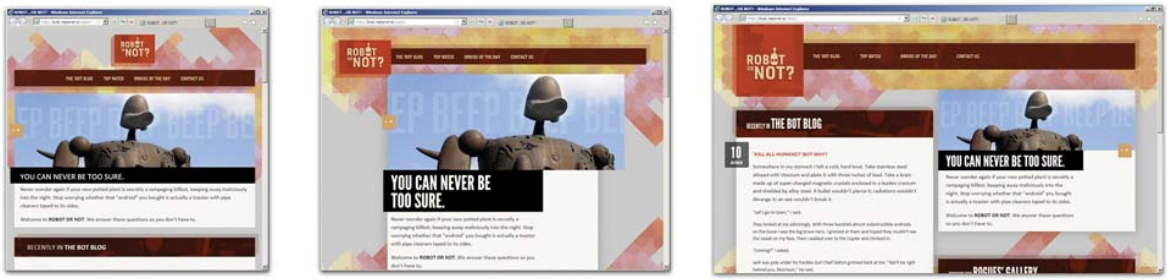

Рис. 4.23. Даже при отсутствии нашего патча для JavaScript более старые браузеры типа IE могут хоть как-то поддерживать медиазапросы

Я не особо полагаюсь на JavaScript и вам не советую. Мы можем спорить до посинения, но все равно нет никакой гарантии, что в браузере пользователя есть JavaScript. Может быть, этот пользователь работает на компьютере или ноутбуке, функции которого ограничены строгой службой IT-безопасности. А что касается мобильных браузеров… На мобильных устройствах не то что слабая поддержка JavaScript — на многих ее вообще нет.

В надежде справиться с этими проблемами, в пятой главе мы посвятим некоторое время обходным решениям, менее завязанным на JavaScript. Если вы не испытываете восторга от перспективы использования JavaScript, это вполне объяснимо. Однако для того, чтобы получить дизайн, не зависящий от устройства и разрешения, нужно строить его на гибкой основе.

ЗАЧЕМ ЭТО НУЖНО?

Можно мне, как истинному фанату, издать еще ряд восклицаний во славу медиазапросов? С их помощью мы можем делать сайты, лучше отвечающие возможностям устройств, на которых их просматривают пользователи.

Однако *сами по себе медиазапросы не делают дизайн отзывчивым*. Они всего лишь накладываются поверх гибкой нефиксированной основы. В пользу этого подхода существует достаточно много аргументов, самый важный из которых следующий: гибкий дизайн может быть резервным вариантом для устройств и браузеров, не поддерживающих JavaScript и `@media`.

Но это не единственная причина. Компания 37signals, разработчик программного обеспечения, недавно начала проводить эксперименты с отзывчивым дизайном для одного из своих приложений, и вот что они сказали по этому поводу (http://bkaprt.com/rwd/39/):

> *Оказывается, для того, чтобы сайт работал на различных устройствах, нужно всего лишь добавить к конечному продукту несколько медиазапросов CSS. Если макет изначально делался «резиновым», то все, что нужно для того, чтобы он правильно отображался на устройствах с маленькими экранами, — это сжать несколько отступов и доработать макеты боковых панелей.*

Другими словами, если у нас есть гибкая основа, нам не придется дописывать много кода. Макеты с фиксированной шириной необходимо переписывать для каждого значения разрешения, тогда как дизайны, созданные на основе процентов, а не пикселей, сами меняют свои пропорции в зависимости от разрешения устройства. В этой главе мы научились быстро и избирательно удалять или менять свойства макета и, таким образом, его оптимизировать.

Кроме того, гибкий дизайн лучше подходит к устройствам, которые еще находятся в стадии разработки. Несколько лет назад слово «планшет» ассоциировалось у нас исключительно с iPad. Теперь под это определение подходят и такие устройства, как 7-дюймовый Galaxy Tab компании Samsung, Kindle и Nook, оснащенные своими браузерами. Мы не сможем угнаться за всеми устройствами, появляющимися на рынке. Гибкий дизайн позволит нам не обращать внимания на отдельные диапазоны разрешений и поможет лучше подготовиться к новым, еще не виданным устройствам.

Ограничения по мере необходимости

Хочу напомнить вам, что никто не разбирается в вашем дизайне лучше вас, даже его пользователи. Поэтому, если вы считаете, что свойство `max-width` обеспечит целостность элемента, смело

вписывайте его в код. Вот как описывает компания 37signals свои эксперименты с отзывчивым дизайном (http://bkaprt.com/rwd/39/):

> *Разработчики практически полностью забыли такое свойство CSS, как max-width, поскольку его не поддерживал Internet Explorer 6. Однако это превосходное дополнение к «резиновому» макету, позволяющее содержанию естественным образом подстраиваться под различную ширину страницы, не разрастаясь настолько, что строчки текста кажутся абсурдно длинными. Это прекрасный компромисс между «резиновым» и фиксированным дизайном.*

В настоящее время я работаю над модернизацией проекта, и у нас возникла дискуссия по поводу этого ограничения. Я задал дизайну фиксированную ширину `max-width`, равную `1200px`, — ниже этой отметки он абсолютно гибкий. Вы спросите, почему же не сделать его полностью «резиновым»? Мы потратили достаточно времени на то, чтобы написать и вставить в код медиазапросы, благодаря которым сайт выглядит идеальным как в последней версии Chrome, так и на телефоне на базе Android или в браузере Kindle. Что же касается дизайна для широкого экрана, мы решили, что овчинка выделки не стоит: у нас просто нет таких пользователей. Поэтому и ввели ограничение `max-width`.

В качестве примера такого единения `max-width` и медиазапросов я могу привести сайт Дэна Седерхольма (http://simplebits.com) и официальный блог дизайнерской компании Happy Cog (http://cognition.happycog.com) (**рис. 4.24 и 4.25**). Это прекрасные примеры того, как «резиновый» макет ограничивается пиксельным `max-width`.

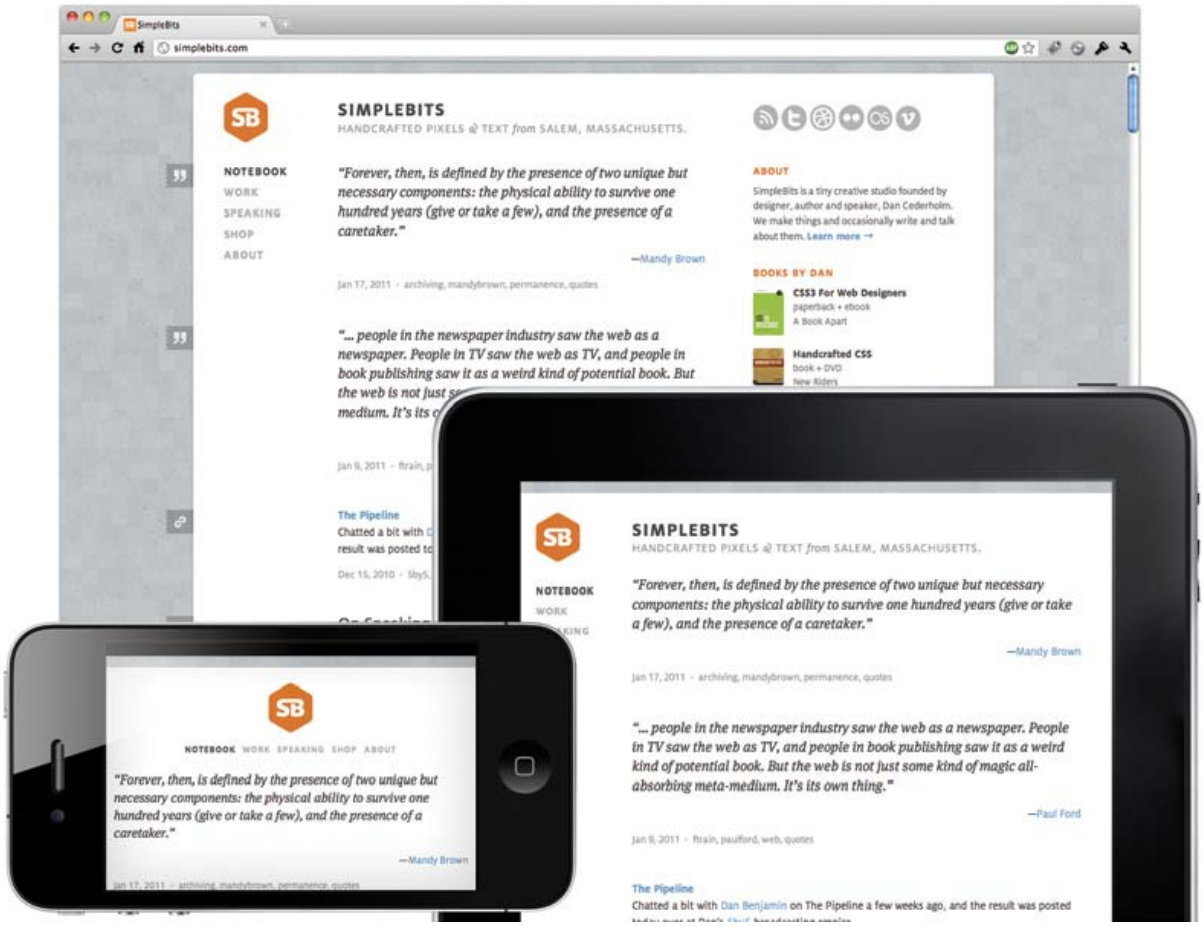

Рис. 4.24. Дэн Седерхольм, дизайнер всех дизайнеров, решил использовать max-width 960 пикселей на своем вновь переделанном сайте. И знаете что? Получилось отлично

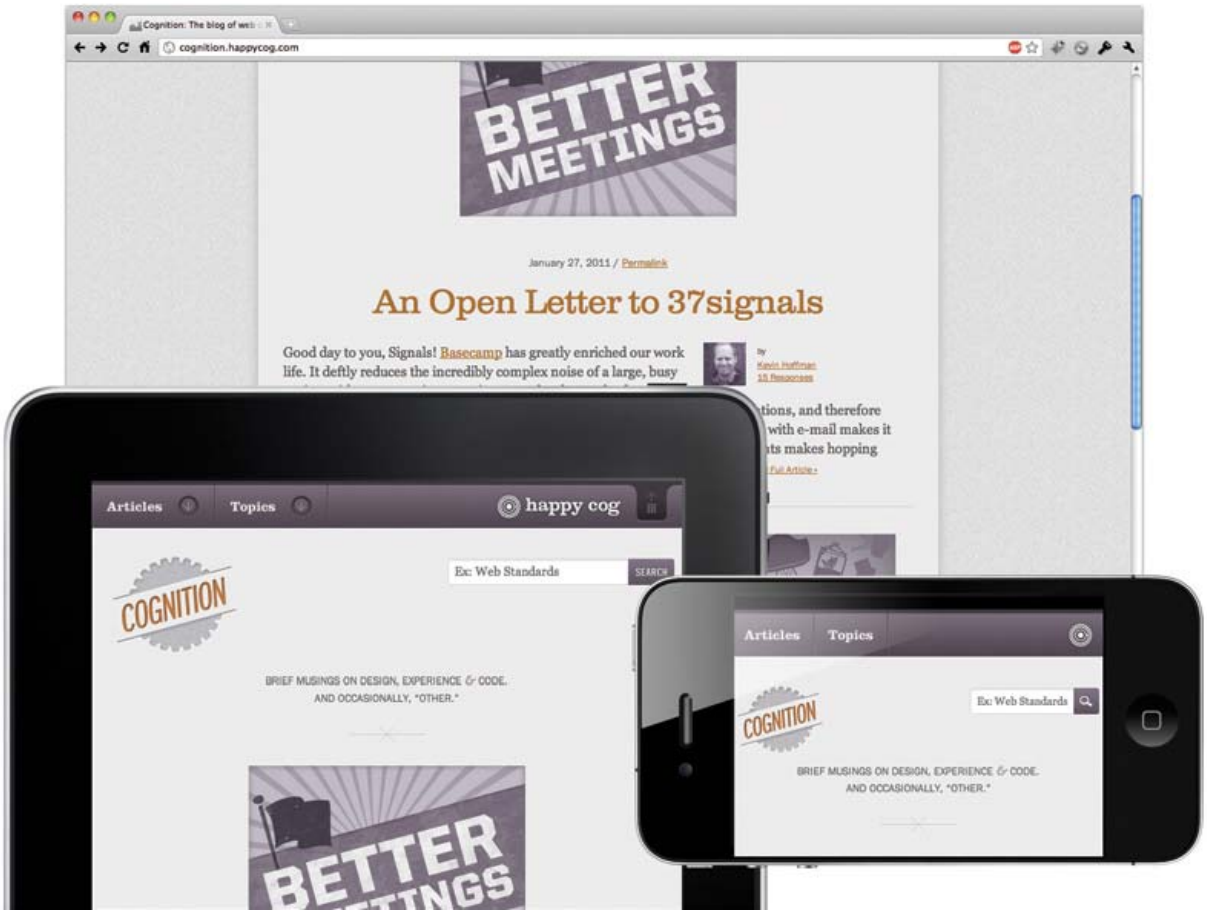

Рис. 4.25. Талантливые ребята из Happy Cog создали новый отзывчивый дизайн, использовав max-width 820 пикселей. Результат? Великолепный!

Некоторые дизайнеры предпочитают именно этот способ решения проблемы длинных строчек, однако он не единственный. Зайдите на сайт дизайнера и иллюстратора Джона Хикса (**рис. 4.26**), одного из первых, кто в 2010 году переписал свой сайт (http://bkaprt.com/rwd/40/).

Рис. 4.26. Сайт Джона Хикса полностью гибкий и великолепно выглядит при любом разрешении

Джон пошел другим путем. Он не заморачивался с ограничениями, а настроил шрифтовое оформление (`font-size`) под различные диапазоны расширений так, чтобы текст хорошо читался на любом экране (**рис. 4.27**).

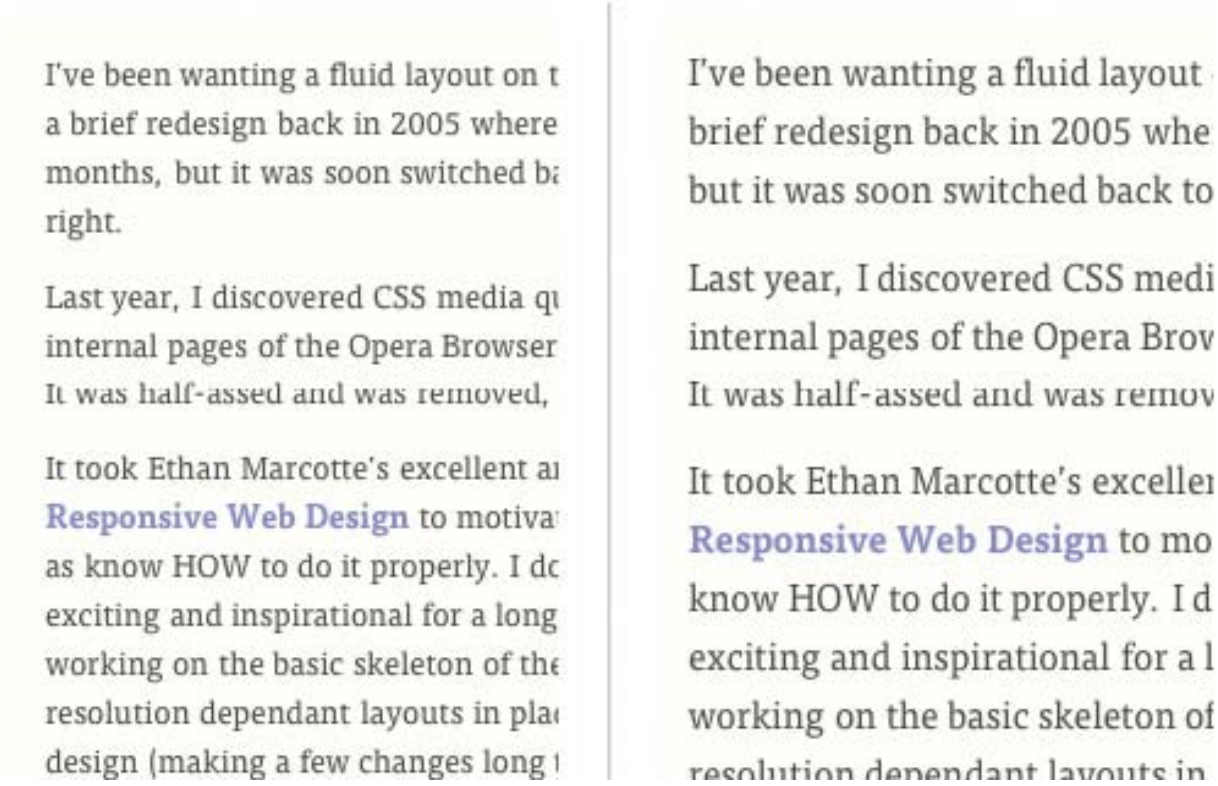

Рис. 4.27. Вместо того чтобы положиться на max-width, Джон предпочел настроить шрифтовое оформление под различные диапазоны расширений, что помогает сделать тексты читаемыми и приятными на вид, вне зависимости от того, на каком устройстве вы читаете его блог

Другими словами, гибкость не значит обязательность. Наоборот, она может стать прекрасной возможностью отточить свои умения, пообщаться с определенным типом пользователей или решить проблемы, связанные с определенными типами устройств.

Мы как дизайнеры принимаем определенные решения и находим компромиссы между гибкостью и контролем. Мы спокойно можем делать гибкие макеты и ограничивать их элементами с фиксированной шириной (**рис. 4.28**). Так что, когда и если мои клиенты решат, что их аудитория только выиграет от широкоформатных дизайнов, они смогут убрать ограничение `max-width`, дописать несколько медиазапросов и получить нужный результат.

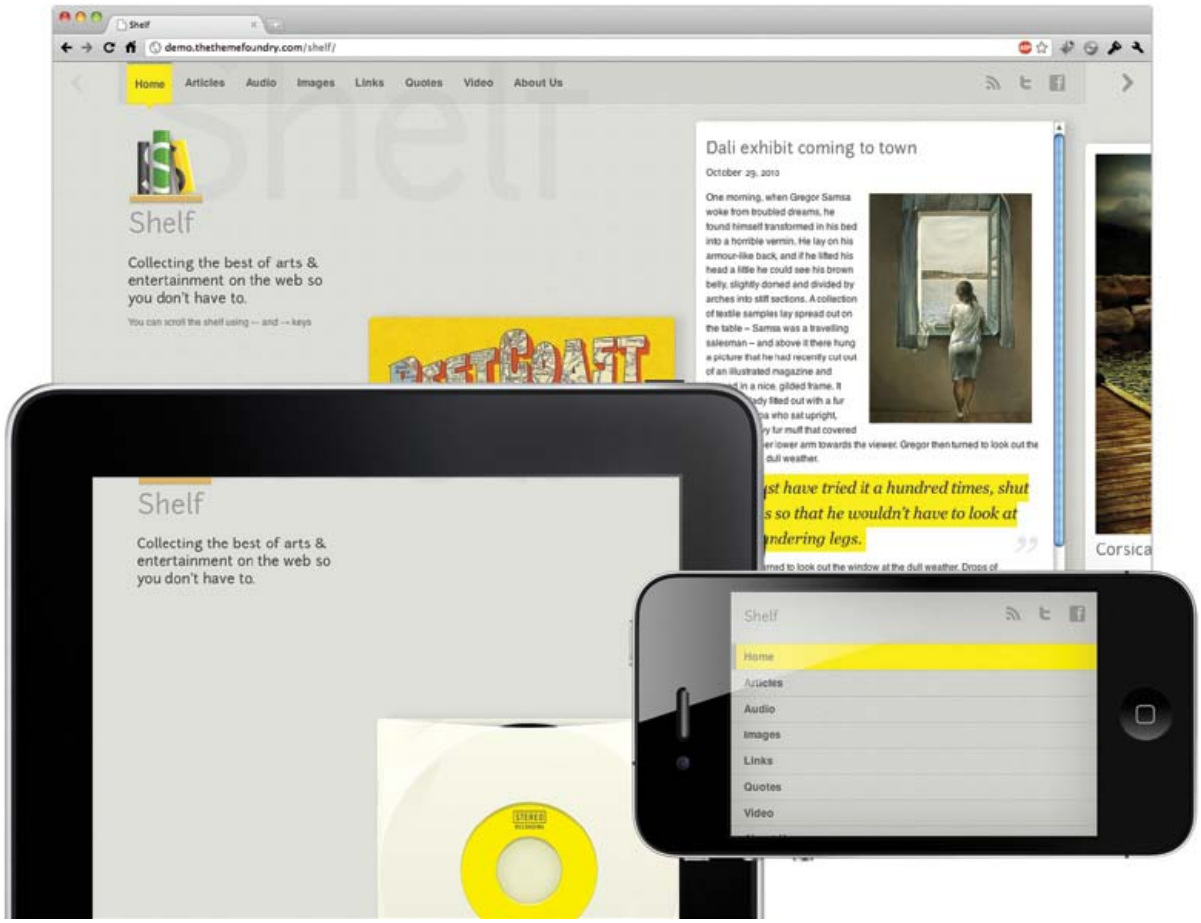

Рис. 4.28. Разработанная Джоном Хиксом тема Shelf для WordPress и Tumblr (http://bkaprt.com/rwd/41/) идеальна с точки зрения гибкости, но при этом содержит ряд контейнеров с фиксированной шириной. (Обожаю эту горизонтальную прокрутку!)

Неважно, как часто будут меняться требования пользователей, наши макеты без проблем смогут отвечать им.

5 КАК СТАТЬ ОТЗЫВЧИВЫМ

“При установлении порядка появились имена.
Поскольку возникли имена, нужно знать предел [их употребления].
Знание предела позволяет избавиться от опасности.
Когда дао находится в мире, [все сущее вливается в него], подобно тому как горные ручьи текут к рекам и морям.”

Дао Дэ Цзин, «стих 32».
В переводе Яна Хин Шуна, 1950 г.

ТЕПЕРЬ У ВАС ЕСТЬ ВСЕ НЕОБХОДИМОЕ для успешного создания отзывчивых сайтов. Вы научились строить пропорциональную гибкую сетку, изучили несколько стратегий внедрения медиафайлов с фиксированной шириной в ваш дизайн и поняли, как медиазапросы могут вывести дизайн за границы мира стационарных компьютеров.

И все это время мы рассматривали отзывчивый дизайн, так сказать, в своего рода вакууме. Теперь мы изучим несколько

способов, которые помогут нам внедрить его в работу, а также рассмотрим некоторые методики усовершенствования уже известных нам технологий.

ВСЕ ДЕЛО В КОНТЕКСТЕ

Начав экспериментировать с отзывчивыми дизайнами, вы обнаружите, что правильно созданный сайт обеспечивает высокий уровень целостности различных контекстов. Это происходит потому, что на самом базовом уровне отзывчивый дизайн адаптирует один документ HTML к различным браузерам и устройствам, делая страницы более портативными и доступными при помощи гибких макетов и медиазапросов.

Однако некоторые веб-дизайнеры выступают против такого подхода и считают, что для каждого устройства нужно делать отдельную верстку. В своем длинном посте, посвященном этому вопросу, разработчик для мобильных устройств Джеймс Пирс ставит под сомнения целесообразность подобного дизайна (http://bkaprt.com/rwd/42/):

> *То, что посетитель сайта использует устройство с маленьким экраном, еще не говорит о том, в каком контексте он его использует. Человек может идти, ехать в машине или вообще отдыхать на диване. В каждом из этих сценариев посетитель заслуживает различного обслуживания или, по крайней мере, немного переделанных версий основного сайта.*

Более лаконично высказался дизайнер Джефф Крофт (http://bkaprt.com/rwd/43/):

> *Как правило, пользователи мобильных устройств и пользователи стационарных компьютеров неодинаково смотрят на ваш продукт. Если у вас ресторан, то пользователи стационарных компьютеров хотят увидеть на сайте его фотографии, полное меню и, может быть, историю развития. Пользователи же мобильных устройств хотят просто получить адрес и часы работы.*

Давайте рассмотрим эти аргументы более детально. Во-первых, устройство, которое применяет пользователь — мобильное или стационарное, — зависит от контекста, ситуации, в который пользователь оказался. На основании такого контекста мы можем создать класс пользователей и наметить несколько целей. Другими словами, посетители, использующие мобильные устройства, хотят более быстрого доступа, чем если бы они сидели за стационарным компьютером или ноутбуком, когда время и пропускная способность канала на их стороне.

Во-вторых, если приоритеты и цели пользователя различны, применять один HTML-документ действительно нецелесообразно. Возьмем пример Джеффа: если на сайте ресторана вверху каждой страницы расположены фотографии, то, скорее всего, они находятся в верхней части HTML. А это значит, что пользователю мобильного устройства, на котором отображается линейный вариант той же самой верстки, придется достаточно долго проматывать страницу, чтобы добраться до времени работы ресторана.

Признаю: я согласен с этими аргументами, но до определенного момента. Да, по устройству вполне себе можно предположить обстановку, в которой находится пользователь, но это всего лишь предположение. Например, я довольно часто выхожу в сеть с мобильного телефона, сидя при этом на диване у себя дома. И это не еще одна шутка про то, что у меня другой жизни, кроме Интернета, не имеется: исследования показали, что достаточно много людей пользуются мобильной Сетью и в стенах собственного дома (http://bkaprt.com/rwd/44/, http://bkaprt.com/rwd/45/).

Я не говорю, что на контекст не нужно обращать внимания или что вообще не стоит задумываться о таких вещах. Но мы не можем судить об обстановке, окружающей пользователя, по устройству, которое он использует. Зачастую таких представлений, созданных на базе контекста, недостаточно, чтобы получить желаемую информацию (**рис. 5.1**). Дизайнеры не должны полагаться на столь удобное разделение устройств на «мобильные» и «стационарные» — это всего лишь термины, они

не заменят полного анализа аудитории вашего сайта. Значение имеют не только устройства и браузеры, которые они используют, но и как, где и зачем они их используют.

Рис. 5.1. При просмотре на iPad сайты Google Reader и Twitter по умолчанию предстают в «мобильном» виде. Отличный дизайн, но правильно ли он применяется в этом контексте?

Но самое главное в том, что отзывчивый веб-дизайн и не должен заменять мобильные сайты. Это скорее философия дизайна, стратегия разработки внешних интерфейсов. То есть в первую очередь необходимо понять, целесообразно ли его использование при работе над определенным проектом. Возможно, есть обоснованные причины для создания отдельных версий (мобильной и стационарной) одного и того же сайта, а может быть, ваш контент лучше представить, применив отзывчивый дизайн. Решать только вам и вашим пользователям.

Я согласен с теми дизайнерами, которые говорят, что определенные пользователи определенных сайтов заслуживают получения отдельного контента. Но я также считаю, что многие сайты только выиграют от обслуживания различных контекстов и устройств одним документом. К созданию таких сайтов и нужно подходить с точки зрения отзывчивости.

Так как же узнать, нужен ли вам отзывчивый дизайн?

Определение целей пользователей

В начале 2010 года я работал над сайтом Cog'aoke (**рис. 5.2**), предназначенным для раскрутки караоке-шоу, вести которое должен был мой тогдашний работодатель. Основная цель сайта — предоставить посетителям исчерпывающую информацию о самом событии, его спонсорах и месте проведения. Но имелось еще и приложение, в котором посетители могли записаться в качестве исполнителя, выбрать песню из каталога и проголосовать за какого-нибудь исполнителя.

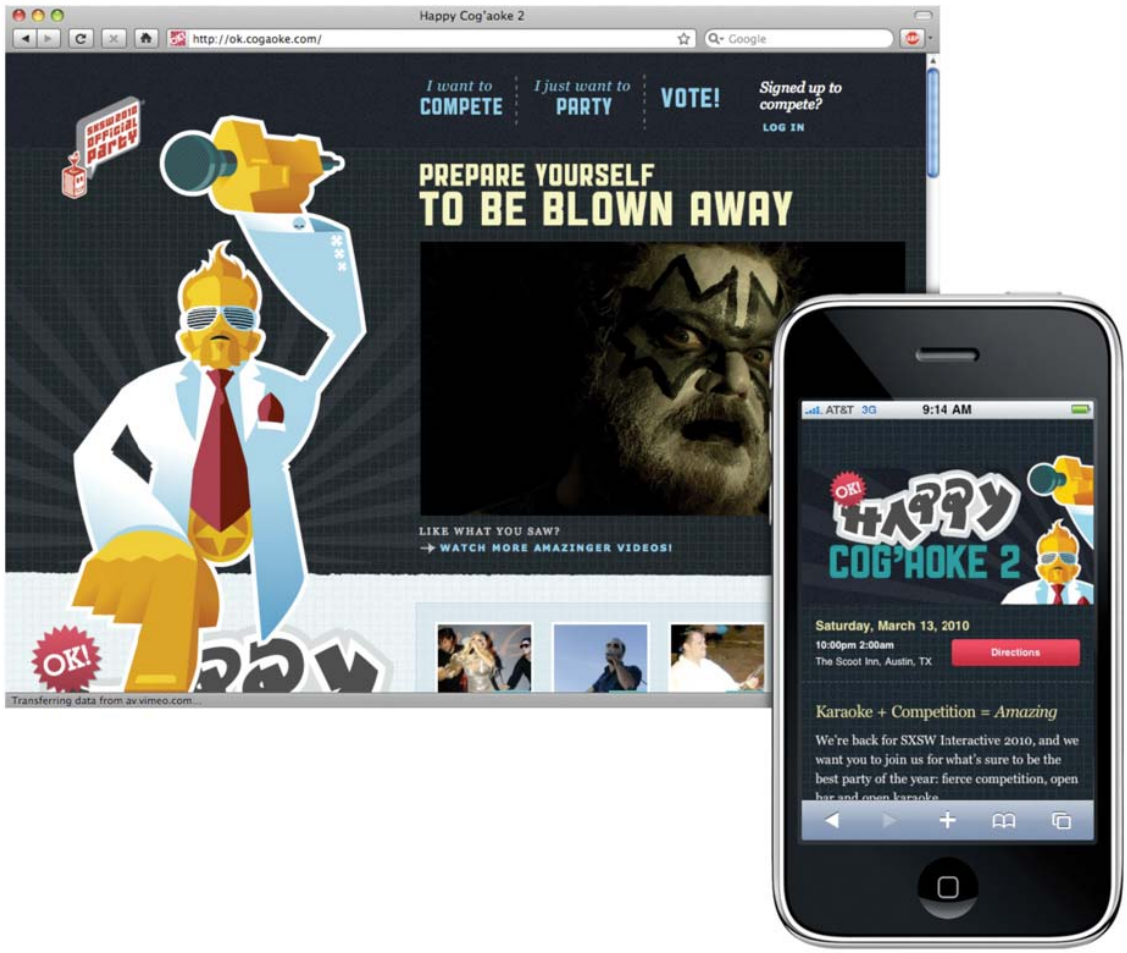

Рис. 5.2. Пример Cog'aoke. Два различных контекста — два разных сайта

Мы решили, что мобильная версия сайта должна совершенно отличаться от десктопной. Мы понимали, что зашедшие на сайт люди хотят быстро и легко ориентироваться в происходящем. Кроме того, мы собирались сделать живое голосование и предложить пользователям проголосовать за понравившихся исполнителей в определенное время — и все это на мобильной версии сайта.

Мы рассматривали десктопную версию как первичную, предварительную подготовку к самому событию. Мобильная версия же предназначалась именно для вечера шоу, для пользователей, которые физически там присутствовали.

Следовательно, цели этих двух контекстов были абсолютно разными.

Прекрасно понимая это, мы могли бы включить в каждую страницу сайта для любого контекста всю разметку. Если бы мы это сделали, каждая страница мобильной версии сайта содержала бы весь обычный «десктопный» контент, включая карту, указания и информацию о голосовании. Мы, конечно, могли бы использовать комбинацию медиазапросов и свойства `display:none` для того, чтобы соответствующая версия отображалась на соответствующем устройстве.

Но это было бы неправильно. С нашей стороны было бы безответственно заставлять посетителей скачивать все эти лишние HTML с частью контента, который они не только не увидят — он им совершенно не нужен. Кроме того, это вообще не их проблема. Неважно, чем пользуются наши посетители — мы не должны нагружать их лишними данными.

ЗНАКОМЬТЕСЬ: «СНАЧАЛА МОБИЛЬНЫЕ»

Когда у вас появится свободная минутка (и крепкий алкоголь под рукой), зайдите к Мерлину Манну и посмотрите его подборку Noise to Noise Ratio (http://bkaprt.com/rwd/46/). Там представлены самые насыщенные контентом страницы, они просто утопают в хламе. А саму статью, оба ее параграфа, найти практически невозможно.

Даже если вы раньше не видели сайты из галереи Мерлина, бьюсь об заклад, что проблемы, которые там представлены, вам вполне знакомы. Более того, я считаю, что эта тенденция подпитывает некоторые наши предубеждения о создании сайтов для «мобильных» пользователей. Мы предполагаем, что им нужно давать меньше контента, поскольку они менее толерантны, чем пользователи стационарных компьютеров. У последних есть внушительных размеров экраны, удобное кресло и больше времени и желания найти то, что им нужно.

Но даже если у них есть такая возможность, неужели им это нужно? Другими словами, почему быстрый доступ к ключевым

элементам дается только мобильным пользователям? Почему все пользователи не могут получать нужный им контент легко и быстро?

В конце 2009 года дизайнер Люк Вроблевски произвел небольшую революцию в нашей профессии, предположив, что мобильное представление сайтов не должно ограничиваться второстепенной ролью (http://bkaprt.com/rwd/47/). Ссылаясь на резкий рост мобильного веб-трафика и новые технические возможности современных телефонов, Люк предложил веб-дизайнерам создавать в первую очередь мобильные версии сайтов и приложений.

«Сначала мобильные» — это прекрасная дизайнерская философия. Более того, я считаю, что она просто бесценна для проектов отзывчивого дизайна, над которыми я работаю. Развитие браузеров и устройств и рост интереса наших клиентов к сайтам, разработанным не только для стационарных компьютеров, дает нам прекрасную возможность взглянуть на нашу профессию со стороны — на работу и язык, на вопросы, которые мы задаем, и решения, которые принимаем.

НАВСТРЕЧУ ОТЗЫВЧИВОМУ ПРОЦЕССУ РАЗРАБОТКИ

Мы в самом начале пути. Многие дизайнеры, студии и агентства еще только начинают изучать отзывчивый дизайн. Поэтому у нас не так уж много практического опыта. Со временем это изменится, мы станем мыслить более отзывчиво. А пока, мне кажется, я смог поделиться некоторыми моими наработками в этой сфере. Возможно, они вам пригодятся, и вы наверняка сможете их усовершенствовать.

В настоящее время я работаю над модернизацией дизайна одного большого сайта с большим информационным наполнением. На данный момент читатель может зайти на сайт с утра со стационарного компьютера, прочитать пару статей, пока едет на работу, и проверить его еще несколько раз в течение дня.

Зная о разнообразии аудитории сайта, клиент решил, что оптимальным вариантом будет придать ему немного

отзывчивости. На стадии планирования команда дизайнеров тщательно изучила каждый элемент контента и задала вопрос: *что этот материал дает пользователям мобильных устройств?*

Наверное, можно было бы выразить это более завлекательно и интересно, но вообще-то маркетинг не мой конек. Мы стали задавать такие вопросы, потому что считаем, что подход «сначала мобильные» невероятно полезен в процессе создания веб-сайта. Вот как Люк объясняет смысл этого подхода (http://bkaprt.com/rwd/48/):

> *Если вы разрабатываете дизайн вначале для мобильного телефона, вы создаете основу, которую можете использовать при создании версии сайта для стационарного компьютера/ноутбука. Если мы включаем в мобильное представление самый нужный и важный для наших клиентов и бизнеса контент, то почему его нужно менять с увеличением пространства экрана?*

Легко заполнить окно десктопного браузера панелями инструментов социальных сетей, ссылками на похожие статьи, RSS-ссылками и облаками тегов. (Если я не ошибаюсь, то это называется «созданием дополнительной ценности».) Но как только мы вынуждены работать с экраном, размеры которого на 80% меньше, чем наш обычный холст, несущественный контент и всякий хлам быстро отпадают, и мы можем сосредоточиться на действительно важных аспектах наших проектов.

Другими словами, создавая дизайн для мобильных устройств в первую очередь, дизайнеры приобретают уникальный опыт, поскольку сосредотачиваются на самом главном, а в современном веб-дизайне это редкость. Это не означает, что сайт должен содержать минимум информации. Но, задав себе в самом начале производственного процесса этот простой вопрос, мы можем критически оценивать каждый предложенный элемент, каждую новую часть функциональности.

Определяем разрешение

Процесс создания дизайна начинается с определения устройств, для которых этот дизайн предназначен. На основании этого мы составляем список разрешений: горизонтальные ширины, которые мы должны будем согласовать в нашем отзывчивом дизайне. Например, наш список может выглядеть как табл. 5.1.

Табл. 5.1. Список устройств с различным разрешением

320 пикселей	Небольшие экранные устройства (например, телефоны) в вертикальном положении
480 пикселей	Небольшие экранные устройства (например, телефоны) в горизонтальном положении
600 пикселей	Небольшие планшеты типа Amazon Kindle (600 × 800) и Barnes & Noble Nook (600 × 1024) в вертикальном положении
768 пикселей	10-дюймовые планшеты типа iPad (768 × 1024) в вертикальном положении
1024 пикселя	Планшеты типа iPad (1024 × 768) в горизонтальном положении, а также некоторые виды экранов ноутбуков и десктопных компьютеров
1200 пикселей	Широкие экраны, в основном экраны ноутбуков и десктопных компьютеров

Это не значит, что разрешения выше или ниже этого порога будут проигнорированы или что мы не будем обслуживать устройства, разрешения которых не указаны в списке. (В конце концов, отзывчивый макет строится на базе гибкой сетки, а значит, не будет зависеть от разрешения.) Но такой список помогает определить объем работ и область применения. Зная, какие устройства будут использоваться для отображения нашего дизайна, мы можем более эффективно выполнять свою работу.

Когда список готов, можно приступать к самому дизайну.

Итеративный совместный дизайн

В настоящее время бóльшая часть проектов строится по принципу «каскадного» процесса разработки, который подра-зумевает разделение всей работы на отдельные, посвященные одной задаче этапы. У каждого агентства могут быть свои особенности, но в целом все придерживаются следующих четырех этапов: планирование, проектирование, разработка и, наконец, представление готового сайта. На каждом этапе создаются

документы или файлы (например, карта сайта и каркасное представление на этапе планирования), которые должны быть одобрены клиентом прежде, чем начнутся работы по соответствующему этапу.

Опять же, процесс управления проектами может немного разниться от студии к студии. На этапе проектирования команда дизайнеров создает оригинальный макет в графическом редакторе (Photoshop, Fireworks и т. д.). После утверждения он передается команде разработчиков, которые и превращают его в статические шаблоны HTML.

Но работа над отзывчивым сайтом несколько отличается по содержанию и может сделать таким образом организованный процесс громоздким и неповоротливым. Давайте представим, что вам нужно переделать сайт из одной страницы. Вы набросали макет в своем любимом приложении. Но как вы сообщите своему клиенту, как эта страница будет выглядеть на телефоне? Или на iPad? Или на широком экране? Если у вас есть время, бюджет и люди, вы можете спокойно сделать дизайн каждого дополнительного устройства, отдать их на рассмотрение клиенту, получить комментарии и в соответствии с ними исправить каждый вариант. Но если вы делаете пятнадцать или пятьдесят страниц, такой подход выглядит непрактичным.

В процессе работы над последними проектами с отзывчивым подходом я обнаружил, что, если объединить этапы проектирования и разработки, такая коллективная работа будет более эффективной. Я называю этот новый этап «проекторабоТкой». (Шутка.)

Вначале команда дизайнеров представляет макет страницы всей группе. Как правило, это дизайн для стационарных компьютеров (**рис. 5.3**), хотя иногда мы можем начать с мобильного представления дизайна. Цель заключается в том, чтобы дать всей группе импульс для обсуждения того, как этот дизайн будет согласоваться с различными диапазонами разрешений и типами ввода. Возникают самые разнообразные вопросы: «Как это слайд-шоу будет работать с сенсорным интерфейсом?», «Этот модуль всегда будет свернутым по

умолчанию или пользователи стационарных компьютеров увидят больше информации?», «Как этот элемент станет выглядеть (и функционировать), если в браузере не будет JavaScript?»

Рис. 5.3. Начнем с изучения готового дизайна для стационарного компьютера и попробуем понять, каким образом он должен измениться для различных браузеров или устройств

Отвечая на эти вопросы, участники группы обмениваются идеями, обсуждают, как дизайн будет функционировать на различных устройствах, и изучают особенно сложные элементы взаимодействия. Если какая-нибудь часть нуждается в доработке, этим занимается команда дизайнеров. Но если группа чувствует себя комфортно, а доработки совсем незначительны, то команда разработчиков приступает к созданию первоначального варианта, прототипа.

«И это до окончательного утверждения дизайна?» — спросите вы. Да. Наша цель состоит в том, чтобы выйти за пиксельные ограничения Photoshop и начать воплощать дизайн, который будет менять свои размеры в зависимости от размеров окна браузера. Поэтому команда разработчиков приступает к созданию отзывчивого дизайна: преобразовывает фиксированную сетку в «резиновую», обсуждает способы гибкого представления различных медиатипов и использует медиазапросы для адаптации нашего дизайна к различным диапазонам разрешений.

После этого мы начинаем менять размеры окна браузера и анализировать соответствующие изменения нашего дизайна (**рис. 5.4**). Некоторые расширения, как, например, Web Developer Toolbar для Firefox и Chrome (http://bkaprt.com/rwd/49/), на этом этапе могут быть исключительно полезны. Если у вас есть список разрешений (**табл. 5.1**), вы можете сохранить их в расширении для быстрого доступа в будущем (**рис. 5.5**).

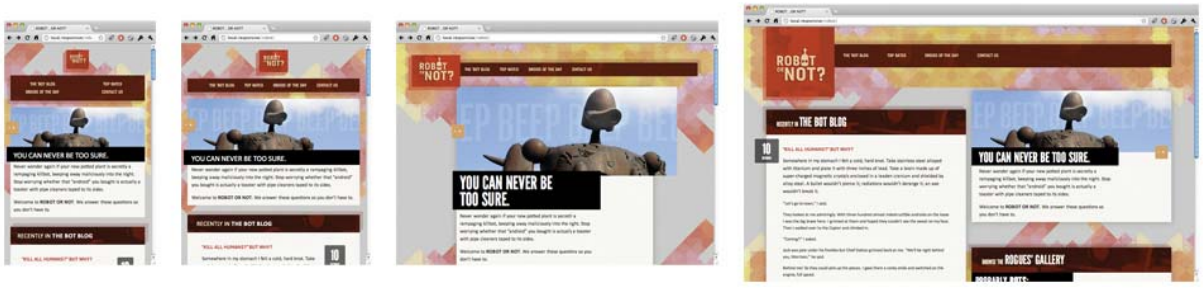

Рис. 5.4. Как я уже говорил, изменение размеров окна браузера позволяет быстро протестировать качество дизайна. Но это лишь первый шаг

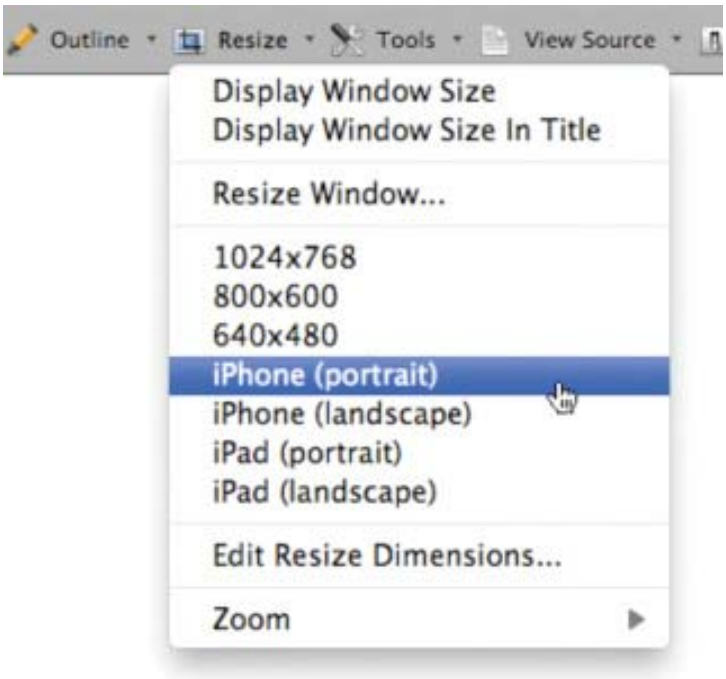

Рис. 5.5. Меню Resize в Web Developer Toolbar позволяет изучить содержимое в разрешениях для различных устройств

Но вспомните: в предыдущей главе мы говорили о том, что изменение размеров окна браузера — это только промежуточный этап. Если вы действительно хотите проверить, как ваша страница будет отображаться на том или ином устройстве, ничто не заменит само устройство. (Если вы интересуетесь приложением для тестирования на мобильных устройствах, я настоятельно рекомендую вам почитать статью Питера-Пола Коха Smartphone Browser Landscape («Многообразие браузеров для смартфонов»), которую вы сможете найти на A List Apart: http://bkaprt.com/rwd/50/. В принципе, даже если вы не собираетесь покупать кучу разных телефонов, все равно почитайте, оно того стоит).

На этом этапе начинает прорисовываться прототип. Он основан на оригинальном макете, созданном командой дизайнеров, но в процессе написания кода разработчики дают свои рекомендации по поводу того, как этот дизайн должен реагировать на различные устройства. Другими словами, разработчики действуют как дизайнеры, просто они работают в

другой среде и рассматривают дизайн с точки зрения браузера, а не Photoshop. Затем эти рекомендации обсуждаются, проверяются и анализируются всей командой.

Данный вариант еще не может быть полностью выверенным или готовым к производству, поскольку на этом этапе мы снова приступаем к анализу дизайна. Но теперь разработчики и дизайнеры обсуждают код, а не макет.

Интерактивный анализ дизайна

Перед общим собранием разработчики загружают страницу прототипа в различные целевые устройства (**рис. 5.6**), а на самом собрании представляют ее всей группе. Целью этого этапа является испытание прототипа и анализа отображения страницы на ноутбуках и стационарных компьютерах, телефонах и планшетах. Члены команды меняют размеры окон браузера, изучают фотогалереи и проверяют, насколько удобно выполнять различные действия на устройствах с клавиатурой и с сенсорными экранами.

Рис. 5.6. Устройства, применяемые при тестировании сайтов в jQuery Mobile (Filament Group, Inc., http://bkaprt.com/rwd/51/)

Эти эксперименты с прототипом проходят в русле содержательной дискуссии. Я считаю, что для команды разработчиков полезно будет заранее составить список проблем, которые возникли в процессе создания отзывчивого дизайна. Возможно, они заметили, что на сенсорном экране тяжело попасть в какую-нибудь важную ссылку или что анимация отображается слишком медленно в каком-нибудь десктопном браузере. Определяя области интересов и потенциальные камни преткновения, разработчики подготавливают почву для конструктивного обсуждения функциональности дизайна и его общего впечатления.

Цель такого собрания — анализ «живого» дизайна. Оригинальный макет — это примерный план с правилами разметки, инструкцией по типографике и библиотекой шаблонов,

который команда разработчиков должна превратить в отзывчивый дизайн. Другими словами, мы проверяем рекомендации разработчиков и обсуждаем, есть ли необходимость в дальнейшей доработке. Это может быть полный пересмотр оригинального макета или всего лишь некоторые незначительные изменения. После собрания обе команды расходятся со своими новыми задачами, и все повторяется с начала. Анализ, дизайн, разработка и повторение.

Вот гипотетический пример того, как это работает. Например, команда дизайнеров сделала макет модуля глобальной навигации с двумя ключевыми ссылками и полем поиска. С этим макетом в руках команда разработчиков начинает встраивать навигацию в шаблон (**рис. 5.7**).

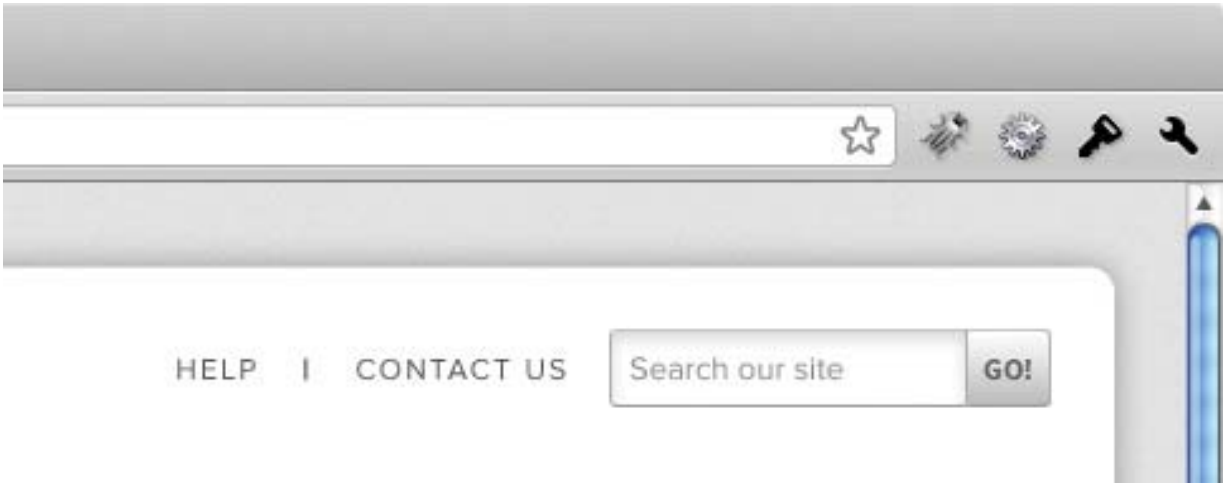

Рис. 5.7. Так будет выглядеть только что спроектированная строка навигации на экране десктопа

Довольно простой дизайн предусматривает две ссылки, расположенные в ряд, и поле поиска с правой стороны от них. При создании отзывчивого дизайна разработчики решили, что для устройств с маленькими экранами будет более эффективно, если поле поиска займет всю ширину страницы, а ссылки расположатся под ним (**рис. 5.8**).

Рис. 5.8. Для устройств с более мелким разрешением ссылки помещены ниже строки поиска

На этапе анализа дизайна несколько дизайнеров высказались в том смысле, что более мелкая версия совсем неудобна. Строка поиска слишком заметна и затмевает собой ссылки. Кроме того, когда они начали тестировать дизайн на различных устройствах, то, пытаясь нажать на ссылку на сенсорных телефонах, попадали в поле поиска.

То есть эта версия никуда не годится. Обсудив все возможности, дизайнеры предложили альтернативное решение (**рис. 5.9**). Вместо того чтобы отображать поле поиска в меньшем разрешении, они решили оставить его первоначальный размер и поместить его в виде еще одной ссылки рядом с двумя другими.

Само поле должно появиться под основным меню при нажатии на эту ссылку (**рис. 5.10**).

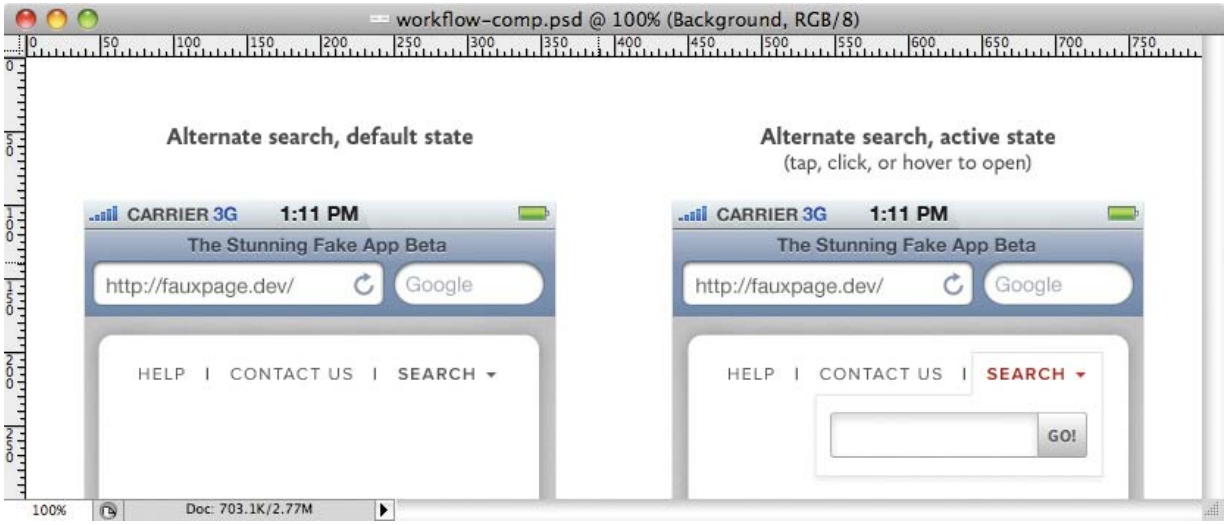

Рис. 5.9. После обсуждения проблемы дизайнеры решили использовать для проблемной строки навигации другой вариант

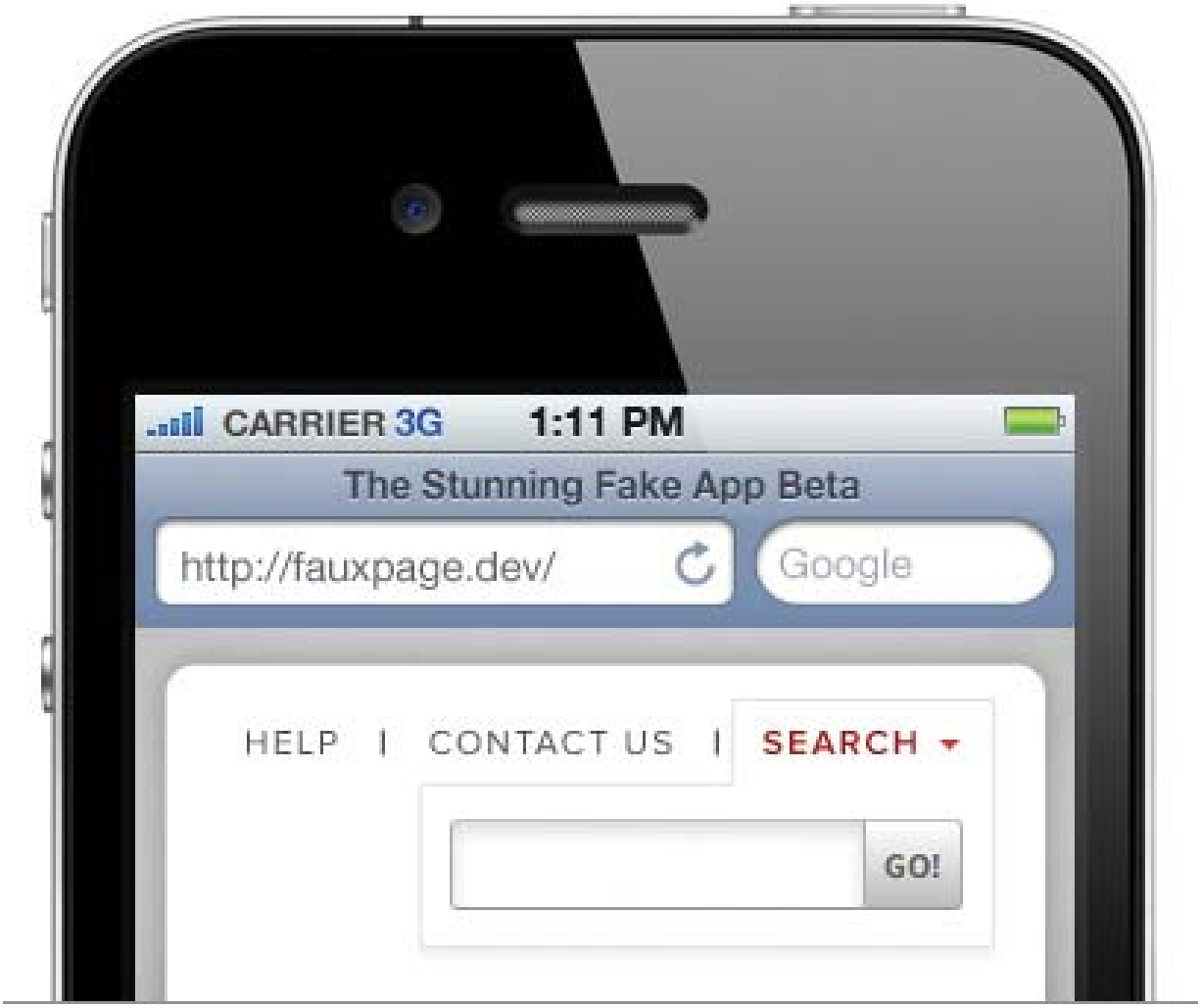

Рис. 5.10. Окончательный вид строки навигации, созданный дизайнерами и разработчиками после нескольких попыток

Это только один небольшой пример того, как идет такая совместная работа. Взаимодействие дизайнеров и разработчиков должно быть постоянным. Обе команды должны периодически проверять свою работу и давать ее на рассмотрение другой. Работая над текущим проектом, наши дизайнеры и разработчики встречаются каждую неделю и выполняют его интерактивный анализ, но в течение недели они также обмениваются эскизами кода или дизайна.

В конечном счете мы хотим объединить традиционные циклы «проектирования» и «разработки» и дать возможность двум группам сотрудничать на такой близкой основе, которая позволит

им создать эффективный отзывчивый дизайн. Работая над текущим проектом, наши дизайнеры и разработчики, конечно, используют для проектирования дизайна такие программы, как Photoshop, но более динамичный подход заставил их обратиться к нашему реальному холсту: браузеру.

ОТВЕТСТВЕННЫЙ ПОДХОД К ОТЗЫВЧИВОМУ ДИЗАЙНУ

В процессе проектирования/разработки мы постоянно исправляем и совершенствуем страницы для достижения конечной цели — получения готового к производству шаблона. Вот тут мы и обнаружили, что философия «сначала мобильные» может быть невероятно полезной при создании отзывчивого дизайна.

В этой книге на базе сайта Robot or Not мы продемонстрировали, как объединенные возможности «резиновой» сетки, гибких изображений и медиазапросов обеспечивают более отзывчивый подход к созданию сайтов. Сначала мы взяли фиксированный макет, разработанный в Photoshop, и превратили его в «резиновую» сетку. В четвертой главе мы обсудили, какие проблемы возникают при изменении размеров окна браузера, поскольку наш первоначальный дизайн не предназначался для разных размеров окна браузера. Чтобы решить эти проблемы, мы ввели медиазапросы и создали альтернативные макеты для маленьких и широких экранов. И наконец, для браузеров, которые не поддерживают медиазапросы, мы включили библиотеку `respond.js` для доступа к нашим альтернативным вариантам дизайна.

Однако здесь возникает еще одна серьезная проблема: что если у браузеров, которые не поддерживают `@media`, нет доступа к JavaScript? В этом случае они были бы вынуждены отображать полный, десктопный дизайн, независимо от того, является ли это подходящим для их устройства. На многих мобильных устройствах это станет выглядеть как дизайн, предназначенный для намного более широкого экрана, но втиснутый в крошечное пространство (**рис. 5.11**).

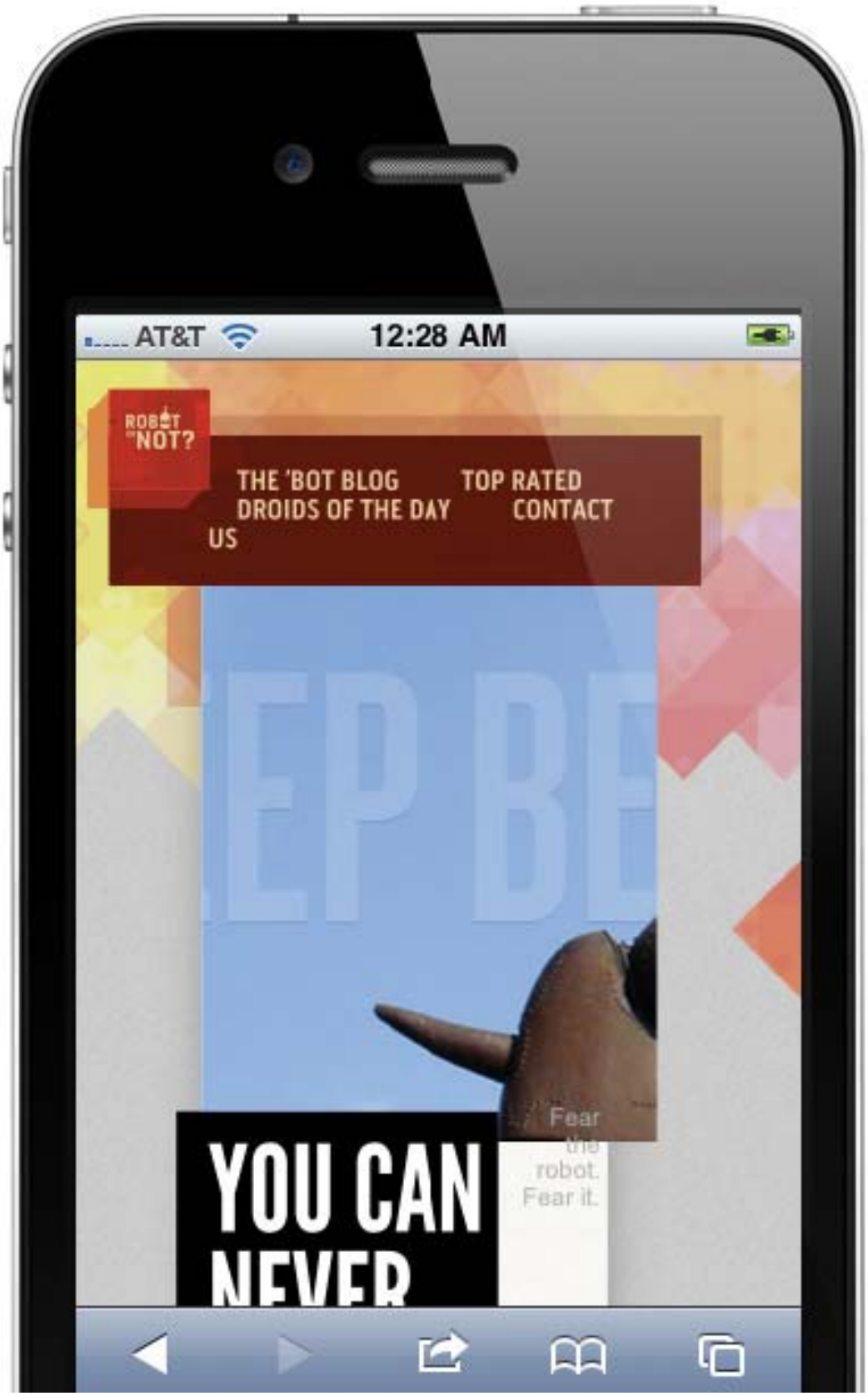
AT&T
12:28 AM
ROBOT OR NOT?
THE 'BOT BLOG
TOP RATED
DROIDS OF THE DAY
CONTACT US
Fear the robot. Fear it.
YOU CAN

Рис. 5.11. Нет медиазапросов? Нет JavaScript? Выглядит ужасно — наш гибкий и предназначенный для больших компьютеров дизайн начинает на небольших экранах распадаться на части

Кроме того, существует еще одна проблема со структурой сайта. Взгляните на небольшой кусочек CSS:

```
.blog {
  background: #F8F5F2 url("img/blog-bg.png")
repeat-y;
}

@media screen and (max-width: 768px) {
  .blog {
    background: #F8F5F2 url("img/noise.gif");
  }
}
```

Во-первых, мы включили фоновое изображение, а именно двухцветную картинку `blog-bg.png`, которую использовали во второй главе для создания иллюзии двух колонок, в элемент `.blog`. Затем для маленьких экранов с шириной менее `768px` мы вместо этого разместили простой размноженный GIF, поскольку мы сделали эти более узкие страницы линейными.

Возникающая в этом случае проблема заключается в том, что некоторые мобильные браузеры, особенно Mobile Safari на iPhone и iPad, фактически загружают *обе* картинки, даже если в итоге отображаться на странице будет только одна. А поскольку пропускная способность канала на мобильных устройствах небольшая, мы наказываем их пользователей лишним тяжелым изображением, которое к тому же еще и не отобразится.

К счастью, в отзывчивом дизайне эти проблемы успешно решаемы, нужно только продумать способ, как это сделать.

«Сначала мобильные» и медиазапросы

В общих чертах, отзывчивый дизайн предназначен для отображения в определенном диапазоне разрешений, тогда как медиазапросы должны адаптировать его к другим диапазонам. Более ответственный подход к отзывчивому дизайну означает создание таблицы стилей с точки зрения идеологии «сначала мобильные». Таким образом, мы начинаем с определения макета для устройств с маленькими экранами, а затем используем медиазапросы для расширения дизайна с увеличением разрешения.

Я даже применил этот подход при создании своего личного сайта-портфолио (http://ethanmarcotte.com). По умолчанию контент представлен в линейной манере, предназначенной для отображения в первую очередь на мобильных устройствах и в узких окнах браузера (**рис. 5.12**). С расширением области просмотра сетка становится более сложной и асимметричной (**рис. 5.13**). И наконец, в самом широком варианте раскрывается «полный» дизайн: разметка становится еще более сложной, появляются некоторые тяжелые элементы, как это абстрактное фоновое изображение (**рис. 5.14**).

ABOUT writing portfolio contact

ETHAN MARCOTTE

is a web designer & developer, a speaker and author.

Ethan lives in Boston, Massachusetts, and is passionate about beautiful design, elegant code, and the intersection of the two. He is a popular and experienced speaker, having been featured at An Event Apart, Carsonified Workshop, and South by Southwest, and co-hosts the Handcrafted CSS seminar series with Dan Cederholm.

More fun facts: Ethan has a blog, and blathers incessantly on Twitter. His clientele has included *New York Magazine*, Stanford University, and the World Wide Web Consortium. Also, he is tall.

Рис. 5.12. Дизайн, созданный по умолчанию для небольших экранов

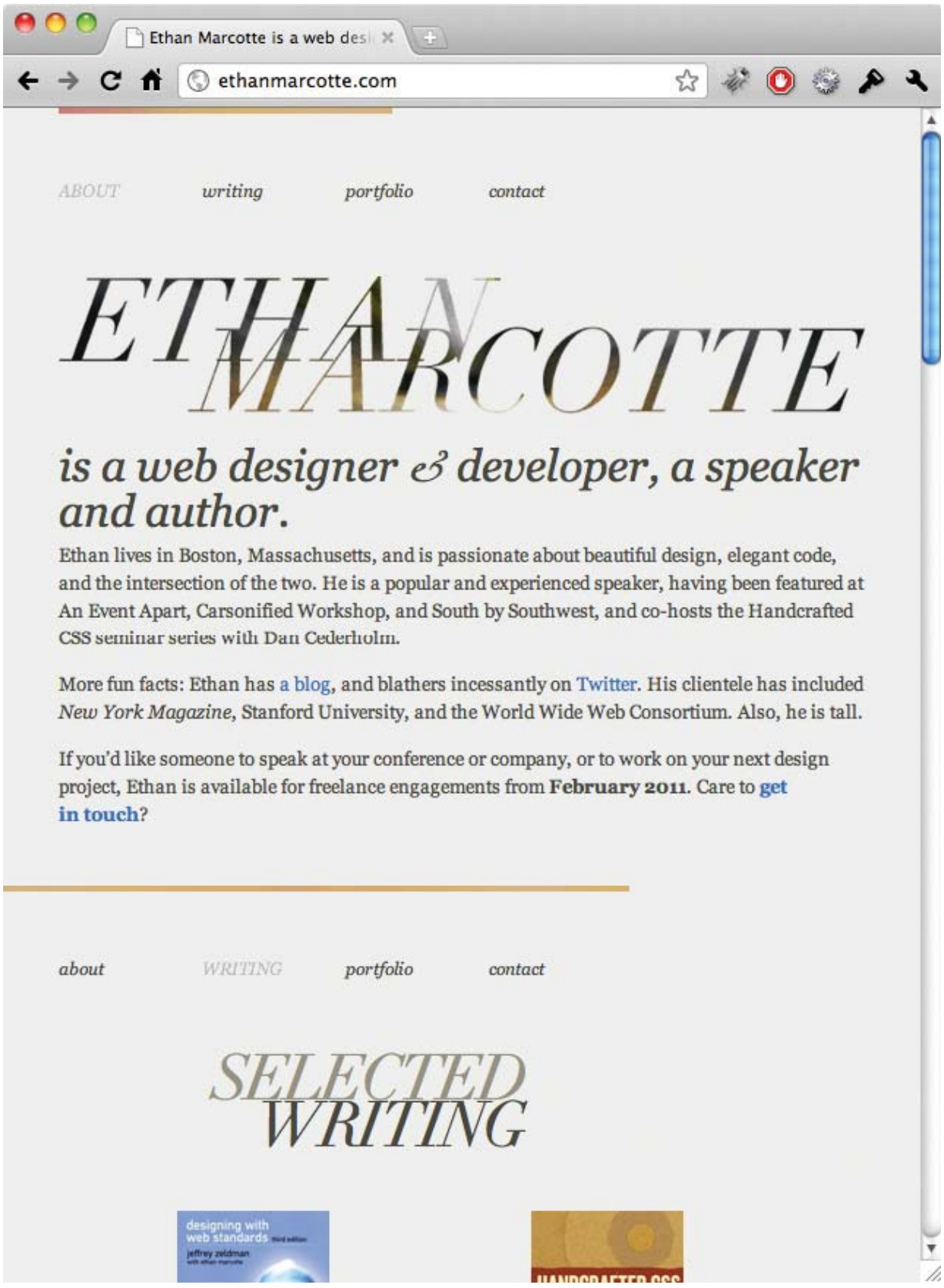

Рис. 5.13. При расширении области просмотра дизайн становится сложнее

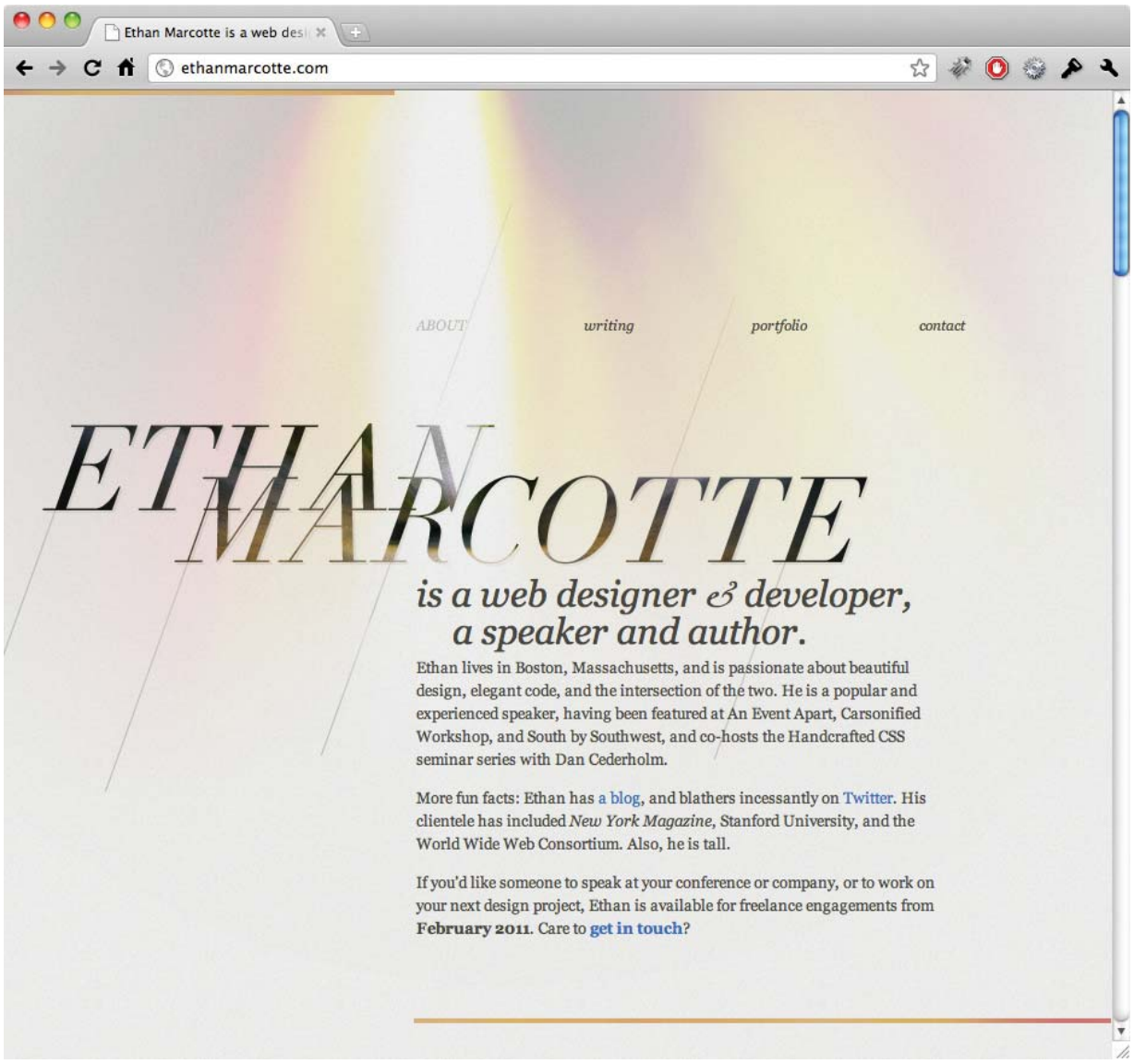

Рис. 5.14. При максимальном расширении дизайн становится виден полностью благодаря применению медиазапросов

Дизайн все еще отзывчив. В нем есть все, что мы обсудили к настоящему времени: разметка основана на «резиновой» сетке, а изображения прекрасно масштабируются. Но, в отличие от сайта Robot or Not, я использовал медиазапросы `min-width`, чтобы увеличить дизайн по мере расширения окна просмотра. Базовая структура таблицы стилей выглядит примерно так:

```
/* Default, linear layout */
.page {
  margin: 0 auto;
  max-width: 700px;
  width: 93%;
}

/* Small screen! */
@media screen and (min-width: 600px) { … }

/* "Desktop" */
@media screen and (min-width: 860px) { … }

/* IT'S OVER 9000 */
@media screen and (min-width: 1200px) { … }
```

Основная часть таблицы содержит правила, связанные с цветом и типом, что предоставляет всем пользователям базовый (но, мы надеемся, все еще привлекательный) дизайн. Затем в медиазапросе установлено четыре диапазона разрешений для минимальной ширины области просмотра в `480`, `600`, `860` и `1200` пикселей. При увеличении расширения сверх этих значений применяются соответствующие правила. Если же сайт открыть браузером, который не поддерживает медиазапросы, он отобразится в первоначальном «одноколоночном» виде, при условии, что патч на JavaScript недоступен (**рис. 5.15**).

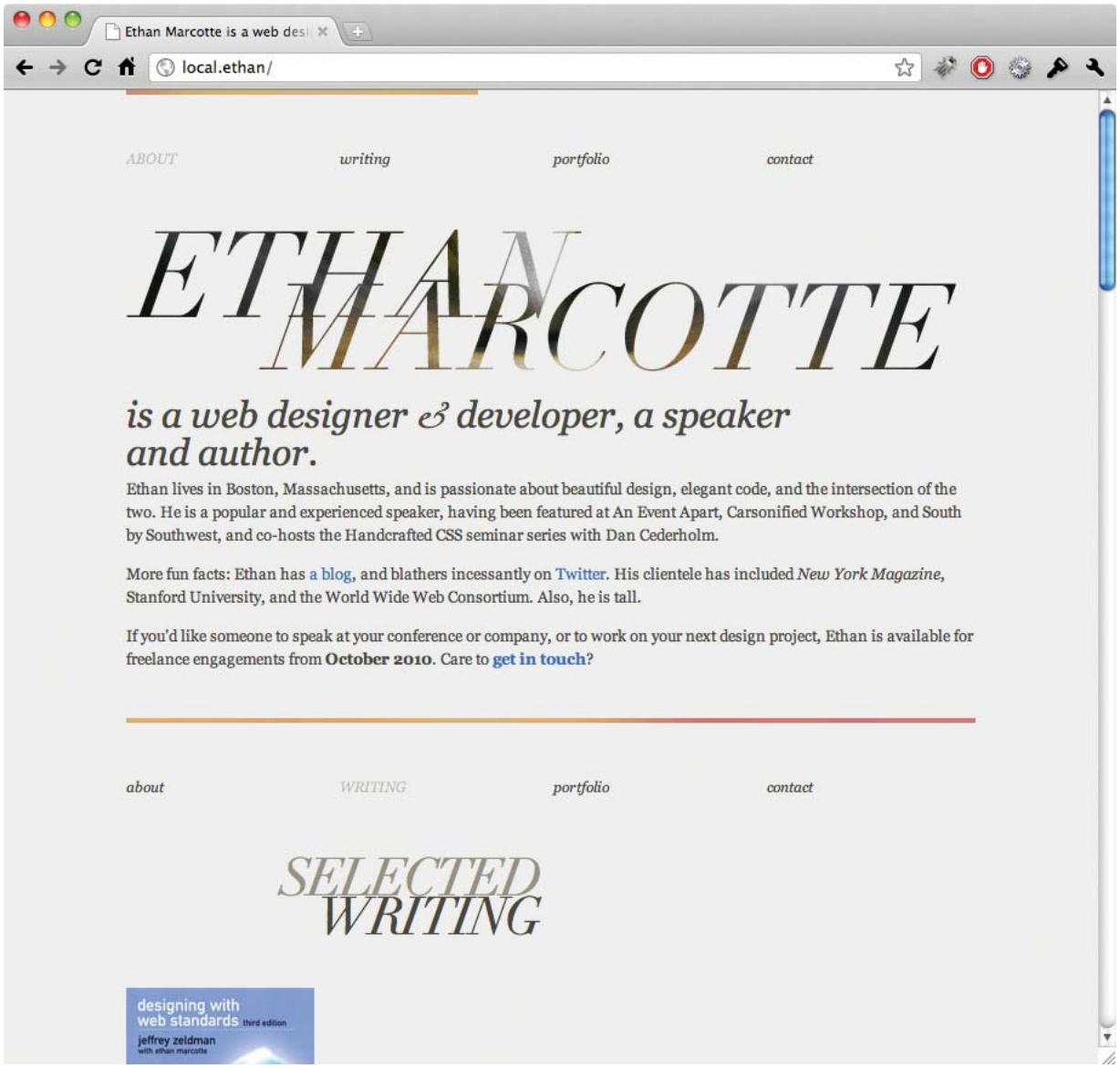

Рис. 5.15. Никаких медиазапросов? Никакого JavaScript? На этот раз никаких проблем

Такой подход к созданию отзывчивых шаблонов гарантирует большую доступность к контенту, при этом никак не зависит от возможностей устройства или браузера, отображающего наш дизайн. Испробовав подобный подход на ряде проектов, я понял, что это лучший и самый надежный способ сделать отзывчивый дизайн.

КОНЦЕПЦИЯ ПРОГРЕССИВНОГО УЛУЧШЕНИЯ

Примером такого подхода может стать недавняя модернизация сайта студии мобильного дизайна Yiibu (http://yiibu.com). Основатели Yiibu Брайан и Стефани Риджер назвали эту модернизацию слиянием подхода «сначала мобильные» и отзывчивого дизайна и добавили кое-что от себя (http://bkaprt.com/rwd/52/):

> *Основное содержание и стандартное представление сделано и оптимизировано для отображения в первую очередь на самых простых устройствах. Мы назвали это «базовой» поддержкой. Устройства с маленькими экранами, поддерживающие медиазапросы, получают более расширенную верстку и иногда более сложное содержание. Это «мобильная» поддержка. И, наконец, «десктопный» макет и контент отображаются на устройствах с широкими экранами.*

Термины, конечно, немного отличаются, но прекрасно пересекаются с оригинальной концепцией прогрессивного улучшения Ника Финка и Стивена Чампиона (http://bkaprt.com/rwd/53/):

> *В отличие от идеи постепенного уменьшения возможностей, концепция прогрессивного улучшения использует веб-технологии как слои, которые накладываются на основной контент и функциональность сайта, предоставляя любой программе или человеку простой доступ к контенту, а для более «продвинутых» браузеров демонстрирует еще и дополнительные эффекты и стили.*

С 2003 года, когда Ник и Стивен придумали этот термин, прогрессивное улучшение было отличительным признаком ответственного подхода к основанному на стандартах веб-дизайну. Начиная с написания семантической, правильно структурированной разметки, приправленной CSS и сценарием DOM посредством JavaScript, где это необходимо, мы можем создать убедительный дизайн для различных браузеров, при этом гарантируя универсальный доступ к контенту, скрытому под дизайном.

Стивен Хэй также высказался в пользу прогрессивного улучшения в своем прекрасном эссе There is no Mobile Web («Нет никакой мобильной Сети») (http://bkaprt.com/rwd/54/):

> *Большинство сайтов в Сети создано без учета сценария мобильного использования. Однако миллионы людей заходят на эти сайты именно с таких устройств. Они получают доступ к «нормальному» (чтобы это ни означало) веб-сайту через «мобильное» устройство.*
>
> …
>
> *Честно говоря, я могу вспомнить несколько сценариев использования сайтов или приложений, которые были или должны были быть полностью мобильными. Судя по всему, благодаря мобильной Сети мы мысленно возвращаемся к тем разговорам о вовлечении, прогрессивном улучшении и доступности, которые мы вели несколько лет назад.*

У вас случались такие моменты, когда кто-то вдруг взял и четко объяснил, почему вы верите в то, во что верите? В своем эссе Стивен в точности указал все причины, по которым мне так нравится отзывчивый веб-дизайн. Вместо того чтобы вкладывать наш контент в различные сайты, предназначенные для различных устройств, мы можем использовать прогрессивное улучшение, чтобы гарантировать качественный доступ для всех, с расширенным представлением для более «продвинутых».

Работа с JavaScript

Прежде всего, давайте обратим внимание на слайд-шоу в верхней части сайта Robot or Not (**рис. 5.16**). В настоящее время разметка выглядит следующим образом:

```
<div class="slides">
  <div class="figure">
     <b><img src="img/slide-robot.jpg" alt=""
/></b>
```

```
      <div class="figcaption">…</div>
    </div><!-- /end .figure -->

      <ul class="slide-nav">
        <li><a class="prev" href="#">Previous</a>
</li>
        <li><a class="next" href="#">Next</a>
</li>
      </ul>
</div><!-- /end .slides -->
```

Рис. 5.16. Слайд-шоу или, точнее, его подобие

Совсем не впечатляет, да? К тому же еще и не очень функционально: мы написали интерфейс для слайд-шоу, но он еще не работает. Мы включили в шаблон один слайд, а также навигацию «вперед/назад». Но нажатие на эти ссылки ни к чему не приводит.

То есть нужно включить немного JavaScript и придать нашему дизайну некоторую интерактивность. Но для начала нам нужны слайды! Так что давайте найдем еще пару изображений и допишем HTML:

```
<div class="slides">
  <div class="figure">
    <b><img src="img/slide-robot.jpg" alt="" />
</b>
    <div class="figcaption">…</div>
  </div><!-- /end .figure -->
  <div class="figure">
    <b><img src="img/slide-tin.jpg" alt="" />
</b>
    <div class="figcaption">…</div>
  </div><!-- /end .figure -->
  <ul class="slide-nav">
    <li><a class="prev" href="#">Previous</a>
</li>
    <li><a class="next" href="#">Next</a></li>
  </ul>
</div><!-- /end .slides -->
```

Используя тот же самый модуль `.figure`, вставим еще четыре слайда.

Выглядит это, конечно, немного странно, поскольку все наши слайды накладываются друг на друга (**рис. 5.17**). Чтобы привести слайд-шоу в нормальное состояние, воспользуемся бесплатным jQuery-плагином, разработанным Мэтом Маркизом (http://bkaprt.com/rwd/55/). Это один из самых функциональных скриптов слайд-шоу, которыми я когда-либо пользовался. К тому же он прекрасно обходится со слайдами, содержащими различное количество текста или изображений, без обращения к замысловатой CSS-мишуре. (Да, я сказал «мишуре», и я вполне серьезен.)

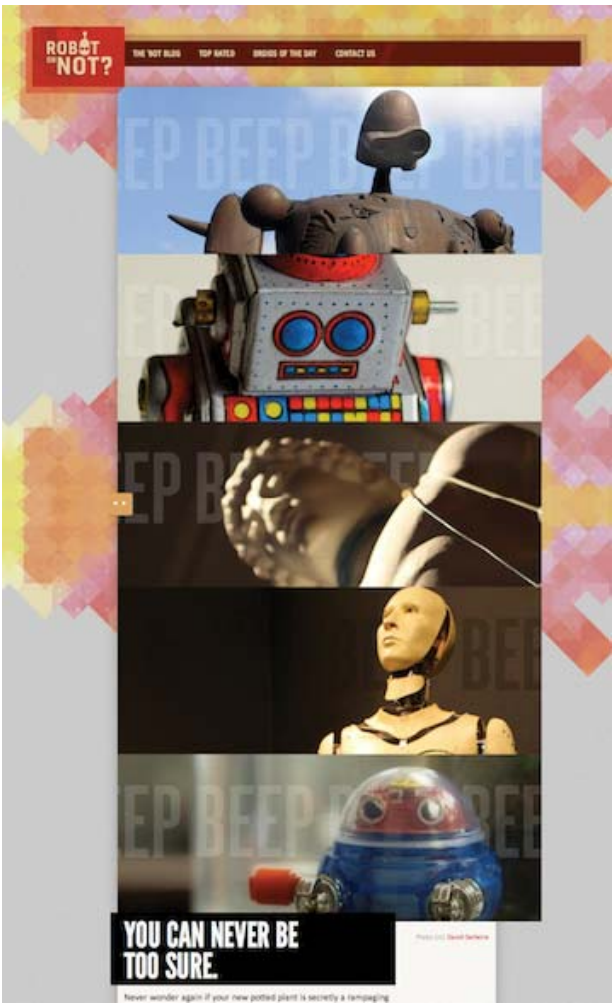

Рис. 5.17. Мы добавляем новые слайды и начинаем ненавидеть статичные картинки

Нашему скрипту «карусели» не хватает еще трех элементов `script`:

```
<script src="jquery.js"></script>
<script src="carousel.js"></script>
<script src="core.js"></script>
```

Поскольку скрипт Мэтта требует запуска jQuery, я загрузил библиотеку с http://jquery.com и поместил ее в верхнюю часть страницы (`jquery.js`), за которой следует скрипт Мэта (`carousel.js`) и файл `core.js`, где мы и напишем код для слайд-шоу.

Кстати, он совсем простой. Впишем в `core.js` следующие строчки:

```
$(document).ready(function() {
  $(".welcome .slides")
    .wrapInner('<div class="slidewrap">
      <div id="welcome-slides" class="slider">
      </div></div>')
   .find(".slidewrap")
     .carousel({
       slide: '.figure'
     });
});
```

Ничего страшного, если вам не нравится JavaScript или вы прежде не использовали jQuery. Этот скрипт выполняет следующие действия.

1. Он располагает элемент `div.slides` внутри модуля `.welcome` при помощи знакомой нам выборки по CSS-селекторам, работающей в jQuery (`$(".welcome .slides")`).
2. Затем обрамляет контент необходимой разметкой (`.wrapInner(…)`)
3. Запускает функцию `.carousel()`, создавая слайд-шоу. А поскольку мы присвоили отдельным слайдам класс `.figure`, мы дали указания скрипту их использовать.

Эти восемь строчек JavaScript дают нам в результате прекрасно работающее слайд-шоу (**рис. 5.18**). Ура!

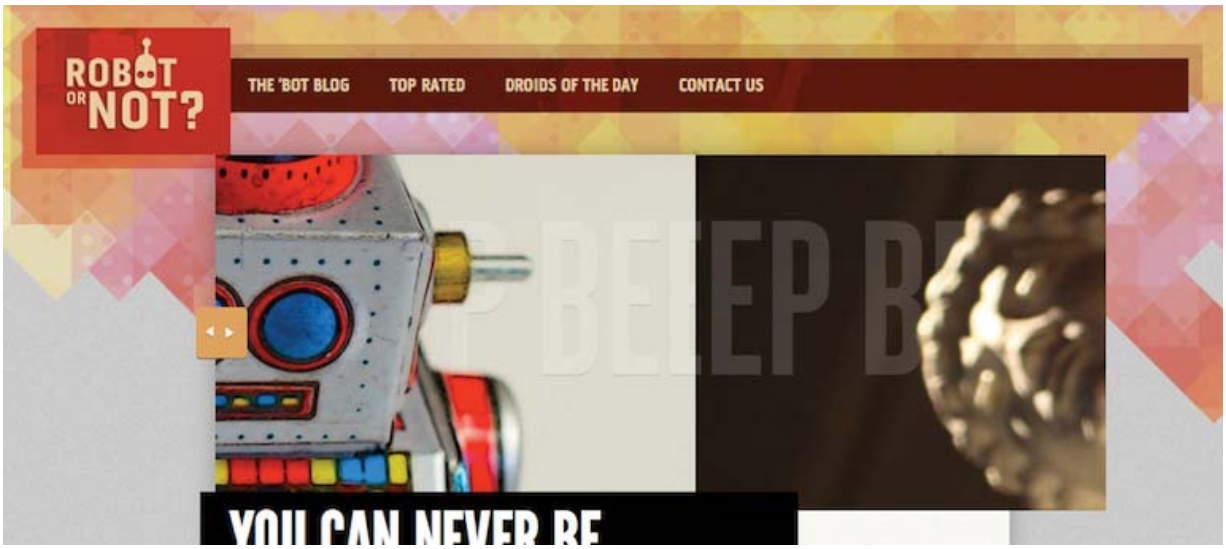

Рис. 5.18. Нам удалось оживить слайд-шоу!

Загружаем контент не спеша, но с умом

По крайней мере, нам есть с чего начинать. Если мы в браузере отключим JavaScript, мы вернемся к тому, с чего начинали: слайды накладываются друг на друга, а меню навигации тут просто для красоты. То есть любой посетитель сайта, у которого нет доступа к JavaScript, получит невнятное и неправильное представление. Что ж, давайте решим эту проблему.

Для начала уберем навигацию «вперед/назад» из разметки и вставим ее через JavaScript:

```
$(document).ready(function() {
  var sNav = [
    '<ul class="slide-nav">',
    '<li><a class="prev" href="#welcome-
slides">Previous</a></li>',
    '<li><a class="next" href="#welcome-
slides">Next</a></li>',
    '</ul>'
  ].join("");

  $(".welcome .slides")
```

```
    .wrapInner('<div class="slidewrap"><div
id="welcome-slides" class="slider"></div>
</div>')
    .find(".slidewrap")
      .append(sNav)
      .carousel({
        slide: '.figure'
      });
});
```

Теперь код выглядит более сложным, но мы на самом деле ввели только одну новую часть функциональности. Прежде всего мы заявили переменную `sNav`, которая содержит HTML для навигации слайдов, и вставили ее перед функцией `carousel()`. Благодаря тому, что мы использовали jQuery для вставки навигации в страницу, пользователи без JavaScript не смогут ее увидеть. Прогрессивное улучшение в действии.

Но это не решает проблему накладывания слайдов один на другой. И здесь мы немного схитрим: мы уберем со страницы все слайды, кроме одного, и переместим их в отдельный HTML-файл. Теперь основа нашей страницы выглядит значительно легче:

```
<div class="slides">
  <div class="figure">
    <b><img src="img/slide-robot.jpg" alt="" />
</b>
    <div class="figcaption">…</div>
  </div><!-- /end .figure -->
</div><!-- /end .slides -->
```

Но мы создали отдельный файл (давайте назовем его `slides.html`) и вставили в него разметку для четырех

оставшихся слайдов.

```
<div class="figure">
  <b><img src="img/slide-tin.jpg" alt="" /></b>
  <div class="figcaption">…</div>
</div><!-- /end .figure -->
<div class="figure">
  <b><img src="img/slide-statue.jpg" alt="" />
</b>
  <div class="figcaption">…</div>
  …
</div><!-- /end .figure -->
```

Вы, вероятно, заметили, что `slides.html` — это даже не полноценный HTML-документ. Это на самом деле отрывок, мини-документ, в котором мы можем сохранить часть HTML-кода и использовать его позже. Фактически мы воспользуемся jQuery, чтобы открыть `slides.html` и загрузить изображения в слайд-шоу:

```
$(document).ready(function() {
  $.get("slides.html", function(data) {
  var sNav = [
    '<ul class="slide-nav">',
    '<li><a class="prev" href="#welcome-
slides">Previous</a></li>',
    '<li><a class="next" href="#welcome-
slides">Next</a></li>',
    '</ul>'
  ].join("");

  $(".welcome .slides")
```

```
    .append(data)
      .wrapInner('<div class="slidewrap"><div
  id="welcome-slides" class="slider"></div>
  </div>')
    .find(".slidewrap")
      .append(sNav)
      .carousel({
        slide: '.figure'
      });
    });
  });
```

На этом ставим точку. Функция jQuery `.get()` открывает наш HTML-отрывок (`slides.html`) и вставляет его содержание в модуль при помощи `append()`. Если JavaScript недоступен или если jQuery не может загрузить этот файл, то пользователь увидит одну картинку в верхней части страницы: абсолютно приемлемый вариант для нашего дизайна (**рис. 5.19**).

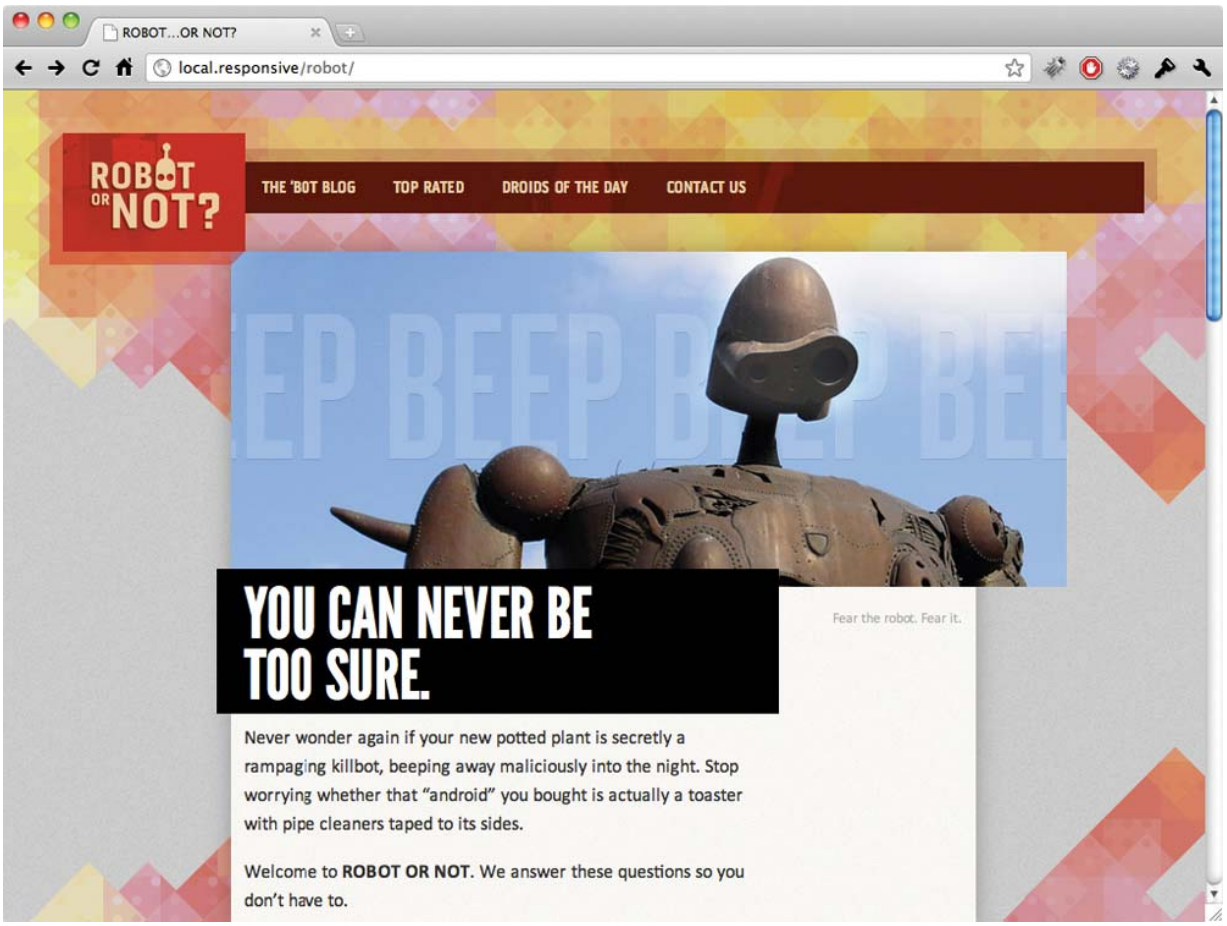

Рис. 5.19. Нет JavaScript? Никаких проблем. Слайд-шоу сокращается до размеров одной картинки, которая все равно отлично выглядит

Дальнейшие улучшения

Наш простой скрипт пополнился кодом, но конечный результат выглядит более надежно и функционально. Мы не принимаем во внимание возможности браузера или устройства, отображающего нашу страницу: если JavaScript доступен, на экране появится слайд-шоу.

Но нет предела совершенству, и этот небольшой черновой образец — не исключение. Например, мы могли бы сделать так, чтобы слайд-шоу появлялось только на определенных типах экранов, то есть указать в скрипте определенные диапазоны разрешений. Так, если мы не хотим, чтобы оно отображалось на устройствах с маленькими экранами, мы могли бы вставить в скрипт простую проверку разрешения:

```
if (screen.width > 480) {
  $(document).ready(function() { … });
}
```

Этот оператор `if` выполняет те же функции, что медиазапрос `min-width: 480px`: если экран уже 480 пикселей, вложенный JavaScript не будет применяться (**рис. 5.20**).

Рис. 5.20. Мы решили, что слайд-шоу станет доступным только для браузеров с разрешением свыше 480px. На более мелких экранах будет видна одна-единственная картинка

Пойдем еще дальше. Например, мы можем использовать легковесный JavaScript-загрузчик LabJS (http://labjs.com/) или Head JS (http://headjs.com/) для динамической загрузки jQuery, плагина «карусель» и нашего собственного `custom.js` для применения только в том случае, когда разрешение экрана достигает определенных значений. Благодаря этому пользователи устройств с маленькими экранами не окажутся заваленными лишним JavaScript, особенно если это препятствует загрузке «карусели». А поскольку мы учитываем пропускную способность канала, мы можем использовать фантастическую библиотеку

«отзывчивых изображений», написанную Filament Group (http://bkaprt.com/rwd/56/). На устройства с маленькими экранами она выдает более легкие картинки, а на обычные, широкоэкранные — полноразмерные.

ВПЕРЕД! БУДЬ ОТЗЫВЧИВЫМ!

Я упомянул эти расширения совсем не потому, что это единственный правильный и нужный подход. В век портативных точек доступа с 3G и телефонов, оснащенных Wi-Fi, мы не можем на основании размеров экрана делать вывод о пропускной способности устройств. Но при необходимости вы всегда сможете использовать эти расширения.

И все же я считаю, что решать связанные с функциональностью проблемы необходимо с точки зрения философии «сначала мобильные». Если я не подпускаю к хитрому интерфейсу мобильных пользователей, то какой от него прок для моей остальной аудитории? Если вы думаете, что это некорректный вопрос, знайте: легких ответов здесь нет.

Дело в том, что для успеха в веб-дизайне необходимо уметь задавать правильные вопросы. Отзывчивый веб-дизайн — это возможное решение, способ сделать дизайн более приспособленным к врожденной гибкости Сети. В первой главе я говорил, что отзывчивый дизайн состоит из «резиновой» сетки, гибких изображений и медиазапросов. Но на самом деле это всего лишь слова, которые мы используем, чтобы озвучить решения проблем, с которыми сталкиваются пользователи, описать рамки для упорядочивания контента в постоянно увеличивающемся море устройств и браузеров.

Если мы будем интересоваться потребностями наших пользователей и внимательно применять эти компоненты, отзывчивый веб-дизайн станет сильной и устойчивой технологией.

С нетерпением жду того момента, когда вы будете рассказывать мне свои истории создания отзывчивого дизайна.

БЛАГОДАРНОСТИ

У меня не хватит слов и места, чтобы должным образом выразить свою благодарность тем людям, которые повлияли на всю мою работу, не говоря уже об этой книге. И все же попытаюсь.

Прежде всего, я бесконечно благодарен владельцам сайта A Book Apart за то, что они заинтересовались отзывчивым дизайном и дали мне возможность написать мою первую книгу. Джейсон Санта-Мария уделил беспрецедентное внимание деталям и качеству. Мэнди Браун — очень умный и проницательный редактор, и я бесконечно благодарен ей за помощь и терпение в придании книге ее окончательного вида. И конечно, мое сердечное спасибо Джеффри Зельдману за его страстные статьи и неустанную работу, и за все те возможности, которые появились у меня благодаря его идеям.

Если я иногда могу сказать что-то внятное, то это только благодаря Гаррету Кайзеру.

Питер-Пол Кох, Брайан и Стефани Риджер, Джейсон Григсби и Стивен Хей научили меня всему, что я знаю о дизайне для мобильных устройств, и неимоверно укрепили мою веру в отзывчивый дизайн. А технология Люка Вроблевски «сначала мобильные» бесценна для любого дизайна — отзывчивого или фиксированного.

Хой Винь и Марк Болтон поведали нашему сообществу, и мне в том числе, много интересных и нужных фактов из истории нашей профессии. Более того, «резиновая» сетка была бы невозможна без ранних исследований Ричарда Раттера.

Если бы более десяти лет назад я не прочитал великолепную статью Джона Олсоппа *A Dao Of Web Design* («Дао веб-дизайна»), мое представление о Сети было бы совершенно другим, а эта книга никогда не появилась бы на свет.

Дэвид Слейт, команда Filament Group (Пэтти Толэнд, Тодд Паркер, Мэгги Костелло и Скотт Джел) и Мэт Маркиз оказали неоценимую помощь на ранних этапах написания этой книги. Кроме того, компания Filament дала мне прекрасную

возможность модернизировать дизайн крупного проекта, и от этого выиграл не только я, но и эта книга.

Техническая редакция Дэна Седерхольма была вдумчивой, основательной и веселой. Как и он сам.

Я даже не могу выразить словами, насколько я польщен тем, что Джереми Кит согласился написать предисловие. Черт, «польщен» даже близко не стояло с тем, что я чувствую.

Моя семья — родители, братья, сестры и бабушка — поддерживали меня все это время. Я люблю вас, ребята.

И конечно, моя жена Элизабет. Все в моей жизни, и эта книга тоже, для нее.

ПРИЛОЖЕНИЕ

Полные веб-адреса упомянутых в книге источников

Глава 1

1. http://www.dolectures.com/speakers/craig-mod/
2. http://www.flickr.com/photos/carabanderson/3033798968/
3. http://www.alistapart.com/articles/dao/
4. http://www.morganstanley.com/institutional/techresearch/mobile_internet_report122009.html
5. http://vimeo.com/14899669
6. http://vimeo.com/14899445
7. http://www.smartglassinternational.com/
8. http://vimeo.com/4661618

Глава 2

9. http://meyerweb.com/eric/tools/css/reset/

Глава 3

10. http://www.flickr.com/photos/uberculture/1385828839/
11. http://clagnut.com/sandbox/imagetest/
12. http://www.svendtofte.com/code/max_width_in_ie/
13. http://msdn.microsoft.com/en-us/library/ms532969.aspx
14. http://www.dillerdesign.com/experiment/DD_belatedPNG/
15. http://msdn.microsoft.com/en-us/library/ms532920(VS.85).aspx
16. http://unstoppablerobotninja.com/entry/fluid-images
17. http://www.yuiblog.com/blog/2008/12/08/imageopt-5/
18. http://www.alistapart.com/articles/fauxcolumns/
19. http://stopdesign.com/archive/2004/09/03/liquid-bleach.html
20. http://www.w3.org/TR/css3-background/#the-background-size
21. http://srobbin.com/jquery-plugins/jquery-backstretch/
22. http://www.bbc.co.uk/news/technology-11948680
23. http://bryanrieger.com/issues/mobile-image-replacement/

Глава 4

24. http://www.w3.org/TR/CSS2/media.html
25. http://www.alistapart.com/articles/goingtoprint/
26. http://www.w3.org/TR/CSS21/media.html#media-types
27. http://www.w3.org/TR/css3-mediaqueries/
28. http://www.w3.org/TR/css3-mediaqueries/#media1
29. http://developer.apple.com/library/safari/#documentation/appleapplications/reference/SafariHTMLRef/Articles/MetaTags.html
30. https://developer.mozilla.org/en/Mobile/Viewport_meta_tag#Viewport_basics
31. http://www.theleagueofmoveabletype.com/fonts/7-league-gothic
32. http://windows.microsoft.com/ie9
33. http://ie.microsoft.com/testdrive/HTML5/CSS3MediaQueries/
34. http://www.quirksmode.org/mobile/#t14
35. http://blogs.msdn.com/b/iemobile/archive/2011/02/14/ie9-coming-to-windows-phone-in-2011.aspx
36. http://www.quirksmode.org/m/css.html#t021
37. http://code.google.com/p/css3-mediaqueries-js/
38. https://github.com/scottjehl/Respond
39. http://37signals.com/svn/posts/2661-experimenting-with-responsive-design-in-iterations
40. http://hicksdesign.co.uk/journal/finally-a-fluid-hicksdesign
41. http://thethemefoundry.com/shelf/

Глава 5

42. http://tripleodeon.com/2010/10/not-a-mobile-web-merely-a-320px-wide-one
43. http://jeffcroft.com/blog/2010/aug/06/responsive-web-design-and-mobile-context/
44. http://thefonecast.com/News/tabid/62/EntryId/3602/Mobile-shopping-is-popular-when-watching-TV-says-Orange-UK-research.aspx

45. http://www.lukew.com/ff/entry.asp?1263
46. http://www.flickr.com/photos/merlin/sets/72157622077100537/
47. http://www.lukew.com/ff/entry.asp?933
48. http://www.lukew.com/ff/entry.asp?1117
49. http://chrispederick.com/work/web-developer/
50. http://www.alistapart.com/articles/smartphone-browser-landscape/
51. http://www.flickr.com/photos/filamentgroup/5149016958/
52. http://yiibu.com/about/site/
53. http://www.hesketh.com/publications/inclusive_web_design_for_the_future/
54. http://www.the-haystack.com/2011/01/07/there-is-no-mobile-web/
55. http://matmarquis.com/carousel/
56. http://filamentgroup.com/lab/responsive_images_experimenting_with_context_aware_image_sizing/
57. http://en.wikipedia.org/wiki/Canons_of_page_construction
58. http://www.amazon.com/dp/0520250125/
59. http://www.amazon.com/dp/3721201450/
60. http://www.amazon.com/gp/product/0321703537/
61. http://www.fivesimplesteps.com/books/practical-guide-designing-grid-systems-for-the-web
62. http://www.markboulton.co.uk/journal/comments/a-richer-canvas
63. http://www.thegridsystem.org/
64. http://www.alistapart.com/articles/fluidgrids/
65. http://www.w3.org/TR/css3-mediaqueries/
66. https://developer.mozilla.org/En/CSS/Media_queries
67. https://github.com/filamentgroup/Responsive-Images
68. http://unstoppablerobotninja.com/entry/responsive-images/
69. http://filamentgroup.com/lab/responsive_images_experimenting_with_context_aware_image_sizing/
70. http://clagnut.com/blog/268/

71. http://bryanrieger.com/issues/mobile-image-adaptation
72. http://www.alistapart.com/articles/dao
73. http://adactio.com/journal/1716/
74. http://adactio.com/journal/4443/
75. http://timkadlec.com/2011/03/responsive-web-design-and-mobile-context/
76. http://globalmoxie.com/blog/mobile-web-responsive-design.shtml
77. http://www.cloudfour.com/weekend-reading-responsive-web-design-and-mobile-context/
78. http://unstoppablerobotninja.com/entry/with-good-references/
79. http://unstoppablerobotninja.com/entry/toffee-nosed/

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

Jan Tschichold. The New Typography. University of California Press, 2006.
Josef Müller-Brockmann. Grid Systems in Graphic Design. Verlag Niggli AG, 2008.
Khoi Vinh. Ordering Disorder: Grid Principles for Web Design. New Riders Press, 2010.
Mark Boulton. A Practical Guide to Designing Grid Systems for the Web. Five Simple Steps, 2009.

ПОМОЩЬ ОНЛАЙН

Вопросы применения сетки для веб-дизайна рассматриваются в следующих источниках:

- Запись в блоге Марка Болтона A Richer Canvas: http://bkaprt.com/rwd/62/
- Веб-сайт The Grid System: http://bkaprt.com/rwd/63/
- Моя статья для A List Apart о гибкой сетке (Fluid Grids): http://bkaprt.com/rwd/64/

Если вы ищете информацию о медиазапросах, то следующие две ссылки — это действительно фантастическое чтение:

- Спецификации медиазапросов World Wide Web Consortium (W3C): http://bkaprt.com/rwd/65/
- Справочные материалы разработчиков Mozilla на тему медиазапросов: http://bkaprt.com/rwd/66/

Если вы работаете с изображениями и другими средствами медиа в гибкой среде, рекомендую вам для ознакомления:

- Скрипт Responsive Images от Filament Group: http://bkaprt.com/rwd/67/, а также соответствующие записи в блоге: http://bkaprt.com/rwd/68/, http://bkaprt.com/rwd/69/
- Эксперименты Ричарда Раттера с изменениями размеров изначальных изображений: http://bkaprt.com/rwd/70/

• Эксперименты Брайана Ригера с адаптацией изображений: http://bkaprt.com/rwd/71/

Дополнительную информацию, как и когда взять на вооружение подход отзывчивого дизайна, можно найти здесь:

• Семинар Джона Олсоппа A Dao of Web Design: http://bkaprt.com/rwd/72/
• Статья Люка Вроблевски на тему «Сначала мобильные»: http://bkaprt.com/rwd/47/, а также статьи на его сайте: http://www.lukew.com/ff/archive.asp?tag&mobilefirst
• Работы Джереми Кита One Web: http://bkaprt.com/rwd/73/ и Context: http://bkaprt.com/rwd/74/
• Запись Тима Кадлека в статье Responsive Web Design and Mobile: http://bkaprt.com/rwd/75/
• Блоги Джоша Кларка и Джейсона Григсби. Джош (http://bkaprt.com/rwd/76/) и Джейсон (http://bkaprt.com/rwd/77/) постоянно ведут интересную дискуссию, которая поможет вам понять, в каких случаях и для каких проектов необходим отзывчивый дизайн.
• Мои записи в блогах With Good References (http://bkaprt.com/rwd/78/) и Toffee-Nosed (http://bkaprt.com/rwd/79/).

ОБ АВТОРЕ

Итан Маркотт — разносторонний дизайнер-разработчик. Его работы — это сочетание качественного кода и захватывающего дизайна. Среди его клиентов — New York

Magazine, Стэнфордский университет, кинофестиваль «Сандэнс» и Консорциум Всемирной паутины (World Wide Web Consortium, W3C). Он ведет блог http://unstoppablerobotninja.com и постоянно зависает в «Твиттере» (@beep).

Итан — автор и технический редактор A List Apart, журнала для людей, делающих веб-сайты. Он популярный лектор, он выступал на конференциях An Event Apart, SXSW Interactive Festival («Интерактивный фестиваль SXSW»), Future of Web Design («Будущее веб-дизайна») и AIGA's In Control («Контроль в руках AIGA»).

Вместе с основателем Happy Cog Джеффри Зельдманом Итан работал над третьей редакцией книги «Web-дизайн по стандартам» (New Riders, 2009), которая занимает почетное место на книжных полках всех опытных дизайнеров. Кроме того, он участвовал в написании таких книг, как Handcrafted CSS («CSS ручной работы») (New Riders, 2009), Web Standards Creativity («Творчество и стандарты Сети») (friends of ED, 2007) и Professional CSS («Профессиональные CSS») (Wrox, 2005).

«Отзывчивый веб-дизайн» — первая самостоятельная книга Итана, он безумно рад тому, что она увидела свет, и хотел бы поблагодарить вас за то, что вы ее прочитали.

ОБ ИЗДАТЕЛЬСТВЕ

Как все начиналось

Мы стартовали в июне 2005 года с двумя книгами. Первой стала «Клиенты на всю жизнь» Карла Сьюэлла, второй — «Маркетинг на 100%: ремикс». «Доброжелатели» сразу же завертели пальцами у виска: зачем вы выходите на этот рынок? Вам же придется бороться с большими и сильными конкурентами!

Отвечаем. Мы создали издательство, чтобы перестать переживать по поводу того, что отличные книги по бизнесу не попадают к российским читателям (или попадают, но не ко всем и зачастую в недостойном виде). Весь наш опыт общения с другими издательствами привел нас к мысли о том, что эти книги будет проще выпустить самим.

И с самого начала мы решили, что это будет самое необычное издательство деловой литературы — начиная с названия (мы дали ему наши три фамилии и готовы отвечать за все, что мы делаем) и заканчивая самими книгами.

Как мы работаем

- Мы издаем только те книги, которые считаем самыми полезными и самыми лучшими в своей области.
- Мы тщательно отбираем книги, тщательно их переводим, редактируем, публикуем и активно продвигаем (подробнее о том, как это делается, вы можете прочитать на сайте нашего издательства mann-ivanov-ferber.ru в разделе «Как мы издаем книги»).
- Дизайн для наших первых книг мы заказывали у Артемия Лебедева. Это дорого, но красиво и очень профессионально. Сейчас мы делаем обложки с другими дизайнерами, но планка, поднятая Лебедевым, как нам кажется, не опускается.

Мы знаем: наши книги помогают делать вашу карьеру быстрее, а бизнес — лучше.

Для этого мы и работаем.

С уважением,
Игорь Манн,
Михаил Иванов,
Михаил Фербер

ПРЕДЛОЖИТЕ НАМ КНИГУ!

Когда я не умел читать на английском бегло, я часто думал: «Как много я пропускаю! Какое количество книг выходит на английском языке и как ничтожно мало издается на русском!»

Потом я научился читать на английском, но проблемы мои не закончились. Я не умел читать на немецком, японском, китайском, итальянском, французском языках… И мимо меня проходило (и проходит) огромное количество хороших деловых книг, изданных на этих и других языках. И точно так же они проходят мимо вас — я не думаю, что среди нас много полиглотов.

Потом вышла моя книга «Маркетинг на 100%», где в одном из приложений были опубликованы рецензии на более чем 60 лучших, на мой взгляд, книг из тех 300, которые я прочитал на английском. Издательства деловой литературы начали издавать их одну за другой — и ни слова благодарности, ни устно, ни письменно.

Теперь я сам немного издатель. Поэтому хочу обратиться к таким же активным читателям, как я. Предложите нам хорошую книгу для издания или переиздания!

Мы вам твердо обещаем три вещи

- Во-первых, если книга стоящая — деловая и максимально полезная, то мы обязательно издадим или переиздадим ее (если права на нее свободны).
- Во-вторых, мы обязательно укажем в самой книге и на ее странице на нашем сайте, кем она была рекомендована. Читатели должны знать, кому они обязаны тем, что у них в руках отличная книга.
- В-третьих, мы подарим вам три экземпляра этой книги, и один будет с нашими словами благодарности.

Мы внимательно читаем все письма. Если предложенная вами книга заинтересует нас, мы обязательно свяжемся с вами.

И если вы хотите проверить твердость наших обещаний, то заполните, пожалуйста, специальную форму на нашем сайте mann-ivanov-ferber.ru

Мы ждем!

Игорь Манн

ГДЕ КУПИТЬ НАШИ КНИГИ

Узнайте, где можно купить наши книги в вашем городе, на сайте **www. mann-ivanov-ferber.ru**

Специальное предложение для компаний

Если вы хотите купить сразу более 20 книг, например для своих сотрудников или в подарок партнерам, мы готовы обсудить с вами специальные условия работы. Для этого обращайтесь к нашему менеджеру по корпоративным продажам: (495) 792-43-72, b2b@mann-ivanov-ferber.ru

Книготорговым организациям

Если вы оптовый покупатель, обратитесь, пожалуйста, к нашему партнеру — торговому дому «Эксмо», который осуществляет поставки во все книготорговые организации.
142701, Московская обл., Ленинский р-н, г. Видное, Белокаменное ш., д. 1; +7 (495) 411-50-74, reception@eksmo-sale.ru

Санкт-Петербург
ООО «СЗКО», 193029, г. Санкт-Петербург, пр-т Обуховской обороны, д. 84, лит. «Е»; +7 (812) 365-46-03 / 04, server@szko.ru

Нижний Новгород
Филиал ТД «Эксмо» в Нижнем Новгороде
603074, г. Нижний Новгород, ул. Маршала Воронова, д. 3; +7 (831) 272-36-70, 243-00-20, 275-30-02, reception@eksmonn.ru

Ростов-на-Дону
ООО «РДЦ Ростов-на-Дону», 344091, г. Ростов-на-Дону, пр-т Стачки, д. 243а; +7 (863) 220-19-34, 218-48-21, 218-48-22, info@rnd.eksmo.ru

Самара
ООО «РДЦ Самара», 443052, г. Самара, пр-т Кирова, д. 75/1, лит. «Е»; +7 (846) 269-66-70 (71…79), RDC@samara.eksmo.ru

Екатеринбург
ООО «РДЦ Екатеринбург», 620007, г. Екатеринбург,
ул. Прибалтийская, д. 24а; +7 (343) 378-49-45 (46…49)

Новосибирск
ООО «РДЦ Новосибирск», 630105, г. Новосибирск, ул. Линейная,
д. 114; +7 (383) 289-91-42,
eksmo-nsk@yandex.ru

Хабаровск
Филиал «РДЦ Новосибирск» в Хабаровске,
680000, г. Хабаровск, пер. Дзержинского, д. 24, лит. «Б»,
оф. 1; +7 (4212) 21-83-81, eksmo-khv@mail.ru

Казахстан
«РДЦ Алматы», 050039, г. Алматы, ул. Домбровского, д. 3а;
+7 (727) 251-58-12, 251-59-90 (91, 92, 99),
RDC-Almaty@mail.ru

МЫ В FACEBOOK!

Присоединяйтесь к нам в Facebook! Все самое интересное из первых рук: http://www.facebook.com/mifbooks

ЭМОЦИОНАЛЬНЫЙ ВЕБ-ДИЗАЙН

Аарон Уолтер

Designing for emotion
Aarron Walter

Все о том, как привлечь и удержать клиентов

Тематика

Веб-дизайн, повышение конверсии сайта, интернет-маркетинг

О книге

Реальность, с которой мы все сталкиваемся, — это типовые и однообразные сайты компаний, интернет-магазинов, информационных порталов... Однообразие — не лучший инструмент для привлечения клиентов. Эмоции продают!
Чтобы сайт притягивал клиентов, достаточно сделать так, чтобы его оформление вызывало положительные эмоции. Удовольствие, удивление, предвкушение в сочетании с яркой индивидуальностью — вот основы эмоционального дизайна.

Освоить его азы и предлагает эта книга. На ее страницах вы найдете подробное описание новых механизмов взаимодействия с вашей аудиторией, а также практические рекомендации, что нужно сделать, чтобы ваш продукт надолго завоевал сердца клиентов.
Применяйте их, и от клиентов не будет отбоя!

Для кого эта книга

Для собственников бизнеса, владельцев сайтов, менеджеров, маркетологов, а также программистов и веб-дизайнеров, как начинающих, так и опытных.

Об авторе

Аарон Уолтер — ведущий дизайнер в области пользовательского опыта компании MailChimp. До этого в течение десяти прекрасных лет он делился опытом с жадными до знаний веб-дизайнерами в колледжах по всем Соединенным Штатам. В наши дни свою страсть обучать других он реализует через проект The Web Standards Project's InterACT.
Аарон живет со своей женой и сыном в городе Афины, штат Джорджия, и собирается параллельно освоить профессию баристы.

ОСНОВЫ КОНТЕНТНОЙ СТРАТЕГИИ

Эрин Киссейн

The Elements of Content Strategy
Erin Kissane

Как создавать качественный контент и управлять им — ответы профессионального контент-стратега

Тематика

Повышение конверсии сайта, интернет-маркетинг

О книге

Эрин Киссейн подробно рассказывает о том, как создавать правильный контент. Ее книга дает ответы на основные вопросы современного владельца любого интернет-ресурса: как понять, какой контент будет интересен целевой аудитории, каким требованиям он должен удовлетворять, чтобы привлечь посетителей и превратить их в покупателей, как разработать контент-стратегию и придерживаться ее. Она позволит освоить основные принципы создания качественного содержания, которое будет притягивать взгляды и деньги.

Для кого эта книга

Для владельцев сайтов, редакторов и контент-менеджеров, справедливо обеспокоенных качеством своего контента.

Об авторе

Эрин Киссейн на протяжении почти десяти лет была редактором в журнале A List Apart, а также работала внештатным редактором и директором по издательским вопросам в HappyCog Studios. В 2011 году Эрин присоединилась к компании Brain Traffic, дающей консультации по вопросам контентной стратегии. Здесь она руководит контентными проектами, а в минуты отдыха страстно читает книги и поглощает кофе, перемежая эти удовольствия написанием статей о модернистской литературе 1920-х и 1930-х годов.

Эрин ведет блог на сайте Incisive.nu.

СНАЧАЛА МОБИЛЬНЫЕ!

Люк Вроблевски

Mobil First
Luke Wroblewski

Все, что нужно знать, чтобы привлечь на сайт пользователей мобильных устройств и заработать на этом

Тематика

Веб-дизайн, повышение конверсии сайта, интернет-маркетинг

О книге

Подход, предложенный Люком Вроблевски, произвел настоящую революцию в области веб-дизайна. Его суть проста: чтобы сайт наиболее полно соответствовал потребностям пользователей любых устройств, сначала следует проектировать его мобильную версию.
В своей книге Люк подробно рассказывает о том, как перейти от создания обычных сайтов к разработке их мобильных версий, какие особенности поведения владельцев мобильных устройств следует учитывать при проектировании интерфейса, как

правильно организовать контент и навигацию, и о многом-многом другом.
Следуя его рекомендациям, вы сможете создать максимально удобный дизайн, который обеспечит высокую посещаемость вашего сайта в самое ближайшее время.

Для кого эта книга

Для собственников бизнеса, владельцев сайтов, маркетологов, а также программистов и веб-дизайнеров — как начинающих, так и опытных.

Об авторе

Люк Вроблевски — сооснователь и руководитель социальной сети Bagcheck (http://bagcheck.com), которая была приобретена компанией Twitter Inc. всего через девять месяцев после создания. До этого он занимал пост директора по дизайну в компании Yahoo!, занимаясь вопросами совершенствования продуктов и пользовательских интерфейсов.
Кроме того, Люк успел поработать старшим дизайнером пользовательских интерфейсов в eBay Inc., где руководил разработкой дизайна потребительских продуктов (таких как eBay Express и Kijiji), а также основать компанию LukeW Ideation & Design (http://www.lukew.com), предоставляющую консультации в области продуктовых стратегий и дизайна, и выступить сооснователем Ассоциации интерактивного дизайна (IxDA).
Люк — автор популярной книги по веб-дизайну Web Form Design («Дизайн веб-форм»), а также множества статей о дизайне и стратегии в области цифровых продуктов.

Отзывчивый веб-дизайн

Маркотт, И.

«Интернет вышел за границы мира стационарных компьютеров, и сегодня можно с уверенностью сказать, что в течение нескольких лет лидирующей формой доступа в Сеть станут устройства с маленькими экранами.

Перед вами первое и единственное практическое руководство, которое в пошаговой форме дает ответ на вопрос, как сделать сайт максимально удобным для всех его посетителей, независимо от того, на каком устройстве они будут его просматривать. Оно содержит рекомендации, как избежать наиболее распространенных ошибок и решить большинство проблем, с которыми сталкиваются современные интернет-пользователи. Кроме того, в нем вы найдете программные коды, которые позволят применить на практике все предложенные разработки.»

Манн, Иванов и Фербер

ISBN 978-5-91657-385-5