Использование HTML5 для разработки приложений для планшетов и смартфонов

jQuery Mobile

Разработка приложений для смартфонов и планшетов

Максимилиано Фиртман

O'REILLY®

bhv®

jQuery Mobile: Up and Running

Maximiliano Firtman

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

Максимилиано Фиртман

jQuery Mobile

Разработка приложений для смартфонов и планшетов

Санкт-Петербург
«БХВ-Петербург»
2013

Фиртман М.

jQuery Mobile: разработка приложений для смартфонов и планшетов: Пер. с англ. — СПб.: БХВ-Петербург, 2013. — 256 с.: ил.

Рассмотрено использование фреймворка jQuery Mobile для создания гибких мультиплатформенных приложений для различных мобильных устройств (iPad, Kindle Fire, iPhone, Android и др.). Описано использование основных компонентов пользовательского интерфейса, а также его оформление и настройка внешнего вида с помощью JavaScript, AJAX и CSS3. Показано создание динамического содержимого с помощью JavaScript, AJAX и библиотеки jQuery. Уделено внимание распространению приложений и созданию приложений с возможностью автономной работы off-line.

Для программистов

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Екатерина Капалыгина*
Перевод с английского	*Сергея Иноземцева*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Оформление обложки	*Марины Дамбиевой*

Authorized translation of the English edition of jQuery Mobile: Up and Running by Maximiliano Firtman, ISBN: 978-1-449-39765-4, Copyright © 2012 O'Reilly Media, Inc. This translation is published and sold by permission of O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

Авторизованный перевод английской редакции книги: ISBN: 978-1-449-39765-4, Copyright © 2012 O'Reilly Media, Inc. Перевод опубликован и продается с разрешения O'Reilly Media, Inc., собственника всех прав на публикацию и продажу издания.

Оглавление

Об авторе **11**

Предисловие **13**
Соглашения, принятые в тексте 13
Использование примеров кода 13
Как связаться 14

Глава 1. Мобильная платформа **15**
Для чего нужна технология jQuery Mobile? 15
Мифы о мобильной Всемирной паутине 15
Мобильной Всемирной паутины не существует, а есть все та же Всемирная паутина 15
Вам не придется делать что-то особенное при разработке приложений для Всемирной паутины 16
Один веб-сайт должен быть пригоден для всех устройств (настольных, мобильных, ТВ) 16
Для получения мобильного веб-сайта достаточно создать HTML-страничку шириной 240 пикселов 17
Мобильные веб-приложения 17
Так зачем нам нужна технология jQuery Mobile? 18
Что такое jQuery Mobile? 19
Чем не является jQuery Mobile? 19
Платформа 20
Мир мобильных и планшетных устройств 21
Категории устройств 21
Мобильные телефоны 21
Бюджетные мобильные устройства 22
Мобильные устройства промежуточного и имиджевого уровня 22
Смартфоны 23
Планшеты 24
Операционные системы и браузеры 24
Совместимость jQuery Mobile 26
Поддержка мобильных браузеров по категориям 27
HTML5 и CSS3 28
Основные функциональные возможности 30
Поддержка ненавязчивой семантики HTML5 30
Прогрессивное улучшение 32
Доступность для пользователей с ограниченными возможностями 33

Тестирование веб-приложений ... 33
Эмуляторы и симуляторы ... 33
Удаленные лаборатории ... 37

Глава 2. Приступаем к работе ... 38
Подготовка документа ... 38
Требования ... 38
Хранение файлов ... 38
Использование CDN ... 40
Самые свежие сборки ... 41
Основной шаблон для HTML5 ... 42
Окно просмотра ... 43
Производительность кода JavaScript ... 44
Поддержка со стороны Adobe Dreamweaver ... 44
Предварительный просмотр файлов ... 46
Архитектура ... 46
Роли ... 47
Темы ... 48
Страница ... 49
Заголовок и нижний колонтитул ... 51
Содержимое ... 51
Навигация ... 52
Кнопка *Назад* ... 53
Внутренние ссылки ... 54
Ссылки на внешние страницы ... 57
Абсолютные внешние ссылки ... 61
Мобильные специальные ссылки ... 62
Переход между страницами ... 62
Обратные переходы ... 64
Диалоговые страницы ... 64
Закрывать страницу или возвращаться к предыдущей? ... 66
Открытие страниц из диалоговых страниц ... 69
Интеграция с телефоном ... 70
Телефонный звонок ... 70
Видеозвонки и звонки VoIP ... 71
Отправка сообщения по электронной почте ... 72
Отправка SMS-сообщения ... 73
Прочие URI-схемы ... 73
Подведение итогов ... 74

Глава 3. Компоненты пользовательского интерфейса ... 76
Панели инструментов ... 76
Размещение ... 77
Подлинно фиксированные панели инструментов ... 79
Добавление содержимого в заголовок ... 79
Добавление кнопок ... 79
Добавление логотипа ... 81
Настройка внешнего вида заголовка ... 82

Добавление содержимого в нижний колонтитул 82
Панели навигации 83
Применение пиктограмм 85
Выделенный элемент 86
Постоянный нижний колонтитул 87
Форматирование содержимого 89
Сворачиваемое содержимое 90
Вложенное сворачиваемое содержимое 92
Аккордеон 94
Столбцы 95
Кнопки 97
Встроенные кнопки 98
Сгруппированные кнопки 98
Эффекты 100
Значки 100
Создание нестандартных значков 101
Расположение значков 103
Кнопки, содержащие только значки 104
Тени на значках 104

Глава 4. Списки 105
Полностраничные и вставленные списки 108
Визуальные разделители 109
Интерактивные строки 111
Вложенные списки 114
Списки с разделенными кнопками 117
Как указать степень важности строк 120
Упорядоченные интерактивные списки 120
Применение изображений 120
Значки строк 121
Миниатюры 121
Дополнительное содержимое 122
Название и описание 123
Применение счетчиков 123
Фильтрация данных с помощью поиска 124
Шпаргалка по представлениям списка 125

Глава 5. Компоненты формы 127
Действие формы 127
Форма без использования AJAX 128
Элементы формы 128
Метки 129
Контейнеры полей 129
Текстовые поля 130
Автоматически увеличивающаяся текстовая область 132
Новые атрибуты HTML5 133
Поля с датами 134
Ползунковый регулятор 135
Двухпозиционный переключатель 136

Меню ... 137
Группирование элементов меню ... 139
Нестандартные меню ... 142
Переключатели ... 144
Флажки ... 146
Выгрузка файлов ... 148

Глава 6. Платформа и JavaScript ... 149
События документов ... 149
Конфигурация ... 151
Глобальная конфигурация ... 152
Пользовательский интерфейс ... 152
Функциональность ядра и AJAX ... 153
Локализуемые строки ... 154
Сенсорное переполнение ... 156
Конфигурация страницы ... 157
Загрузка страницы ... 157
Конфигурация виджетов ... 158
Утилиты ... 160
Утилиты *data-** ... 160
Утилиты страниц ... 161
Опции перехода между страницами ... 161
Утилиты платформы ... 163
Утилиты пути ... 164
Утилиты пользовательского интерфейса ... 164
Нестандартные переходы ... 165
Динамическое содержимое ... 166
Создание страниц ... 166
Создание виджетов ... 169
Обновление виджетов ... 170
Создание сеток ... 171
Изменение содержимого страницы ... 171
Обработка событий ... 171
События страницы ... 171
События создания страницы ... 172
События загрузки страницы ... 172
События показа страницы ... 173
События виджетов ... 174
Событие смены ориентации ... 174
События жестов ... 175
События виртуальных щелчков ... 175

Глава 7. Создание тем ... 177
Приложение ThemeRoller ... 178
Глобальные настройки ... 179
Настройки образцов цвета ... 179
Инспектор свойств ... 180
Виджет Adobe Kuler ... 180
Экспорт темы ... 181

Редактор тем Fireworks ... 182
Редактирование тем ... 187
Нестандартные переходы ... 188

Глава 8. Установка и автономная работа ... 190
Определение пакета ... 190
Манифест HTML ... 191
Процедура загрузки ... 192
Обращение к ресурсам в Интернете ... 193
Обновление ресурсов ... 194
Объект JavaScript ... 195
События ... 196
Установка значка ... 198
Предложение по установке ... 198
Имя значка ... 199
Определение значка ... 201
Полноэкранный режим ... 203
Распознавание полноэкранного режима ... 203
Применение стилей к приложению ... 204
Подведем итоги ... 206
Хранение данных в автономном режиме ... 207

Глава 9. Законченное веб-приложение ... 209
Структура приложения ... 209
Манифест автономной работы ... 210
Страницы ... 211
Таблица стилей ... 219
Данные ... 220
Сценарий ... 220

Глава 10. Расширение возможностей платформы ... 226
Создание дополнительного модуля ... 226
Базовый шаблон ... 227
Создание собственного дополнительного модуля ... 228
Использование ... 228
Виджет ... 229
Автоинициализация ... 231
Применение нашего модуля ... 231
Полный код модуля ... 231
Замечательные модули ... 233
Модуль Pagination ... 233
Модуль Bartender ... 234
Модуль DateBox ... 235
Модуль Simple Dialog ... 237
Модуль Action Sheet ... 239
Дополнительные модули для планшетов ... 239
Модуль SplitView ... 239
Модуль MultiView ... 241
Совместимые модули ... 242

Глава 11. Упаковка приложения для продажи 244
Распространение через магазин 245
Распространение своими силами 245
Подготовка пакета 246
Упаковка с помощью PhoneGap 247
Сервис PhoneGap Build 248

Предметный указатель 249

Об авторе

Максимилиано Фиртман (Maximiliano Firtman), @firt, специализируется на разработке мобильных приложений и приложений в кодах HTML5. Он преподает мобильные технологии, основав тренинговую компанию ITMaster Professional Training. Он автор многих книг, в том числе "Programming the Mobile Web"[1]. Он выступал на международных конференциях OSCON, Velocity, Breaking Development, GOTO Europe, Campus Party, QCon и Adobe en Vivo.

Он имеет звания Adobe Community Professional с 2011 года и Nokia Developer Champion с 2006 года. Принимал участие во многих проектах, связанных с мобильными разработками, например, MobileHTML5.org, MobileTinyURL.com и iWebInspector.com. Он ведет блог **http://www.mobilexweb.com/**, относящийся к мобильным разработкам.

Он эксперт в низкоуровневом программировании и программировании на HTML5 с применением технологий iOS, Android, PhoneGap и jQuery.

[1] Фиртман М. Веб-программирование для мобильных устройств. — М.: Рид групп, 2011.

Предисловие

Эта книга будет идеальным спутником веб-дизайнеров и веб-разработчиков, желающих создавать мобильные приложения с помощью технологии jQuery Mobile.

Платформа jQuery Mobile появилась на рынке для решения одной задачи: создания совместимых интерфейсов для десятков мобильных платформ и браузеров.

Эта книга потребует от читателя лишь знания основ HTML (любой версии), а знание основ JavaScript облегчит чтение последних глав. Читателю необязательно знать HTML5, JavaScript и jQuery, чтобы работать на платформе jQuery Mobile и понимать материал этой книги.

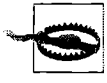
Соглашения, принятые в тексте

Курсив обозначает новые термины, а также текст, который должен быть заменен пользовательскими значениями или значениями, вытекающими из контекста.

`Моноширинный шрифт` применяется для листингов, а также в тексте для обозначения программных элементов, таких как имена переменных и функций, баз данных, типов данных, переменных окружения, операторов и ключевых слов.

`Моноширинный полужирный шрифт` выделяет команды или другой текст, который должен быть набран пользователем буквально.

Так выделяются советы, предложения и общие замечания.



Так выделяются предупреждения и предостережения.

Использование примеров кода

Эта книга написана специально для того, чтобы вы лучше справлялись со своей работой. Вообще говоря, вы можете пользоваться кодом из этой книги в своих программах и документации. Если вы не воспроизводите наш код в значительном объеме, то не обязаны связываться с нами для получения разрешения. Например, при написании программы, в которой использовано несколько фрагментов кода из этой книги, разрешение не требуется. Однако для продажи или распространения дисков

с примерами кода из книг издательства O'Reilly разрешение необходимо. Если, отвечая на чей-то вопрос, вы процитируете эту книгу и приведете пример кода из нее, вам не нужно получать разрешение. Если же вы захотите включить большой объем кода из этой книги в документацию к своему продукту, то должны будете получить разрешение на это.

Мы будем благодарны за указание источника кода, но не настаиваем на этом. Сведения об источнике, как правило, включают в себя название книги, автора, название издательства и стандартный международный номер книги (ISBN). Например, "jQuery Mobile: Up and Running, Maximiliano Firtman (O'Reilly). Copyright 2012 Maximiliano Firtman, 978-1-449-39765-4". Если вам кажется, что при использовании нашего кода вы вышли за рамки, обозначенные выше, не стесняйтесь написать нам по адресу **permissions@oreilly.com**.

Как связаться

Для этой книги была создана веб-страница с перечислением замеченных опечаток, а также с примерами и дополнительной информацией. Она доступна по адресу:

http://shop.oreilly.com/product/0636920014607.do

ГЛАВА 1

Мобильная платформа

Если вы читаете эту книгу, значит, вы, скорее всего, веб-дизайнер или веб-разработчик, возможно, поклонник технологии jQuery или разработчик приложений для Всемирной паутины. Прежде чем приступить к кодированию, мы должны разобраться с мобильной экосистемой и с местом, которое занимает в ней технология jQuery Mobile. Так сделаем это.

Для чего нужна технология jQuery Mobile?

Первый вопрос, который вы должны себе задать: *почему существует jQuery Mobile*? Почему нам нужна специальная технология для мобильных устройств, когда имеется множество мобильных браузеров, способных отображать стандартные веб-сайты, созданные для настольных компьютеров?

Чтобы ответить на эти вопросы, я, с вашего разрешения, скопирую отрывки из другой моей книги, "Programming the Mobile Web"[1].

Мифы о мобильной Всемирной паутине

С тех пор как Всемирная паутина стала доступна мобильным устройствам, разработчики много говорят о значении этого факта для своей работы. Некоторые из высказываемых мыслей справедливы, в то время как другие уводят в сторону, сбивают с толку и даже представляют некоторую опасность.

Мобильной Всемирной паутины не существует, а есть все та же Всемирная паутина

За последние годы я слышал это высказывание много раз, и это правда. Мы действительно имеем дело с одной Всемирной паутиной. Подумайте о своей повседневной жизни. Вы же не создаете отдельный электронный ящик для своего мобильного устройства. (Да, я знаю некоторых людей, делающих это, но я не думаю, что такое явление типично.)

Вы узнаете результаты баскетбольных игр на своем любимом сайте, возможно, ESPN, но у вас нет отдельных источников информации для настольного компьютера и для мобильного устройства. Вы не вступаете в отдельную социальную сеть для

[1] Фиртман М. Веб-программирование для мобильных устройств. — М.: Рид групп, 2011.

мобильного устройства, а пользуетесь той же учетной записью Facebook или Twitter, что и на настольном компьютере. Вы потратили уйму сил на создание списка друзей и списка игнорирования и не хотите проделывать ту же работу на мобильном устройстве.

Итак, это та же самая Всемирная паутина. Однако при разработке приложений для мобильной паутины мы имеем дело с очень специфичными устройствами. Их самая заметная особенность — размер экрана, и это наша первая проблема. Есть и менее очевидные особенности. Одна из них — существенное отличие контекста, в котором мы используем мобильные устройства, от условий применения настольных компьютеров и даже ноутбуков и нетбуков. Не поймите меня превратно, это не означает, что мы, разработчики, вынуждены создавать две, три или десять версий одной программы. Здесь как раз тот случай, когда на помощь приходит технология jQuery Mobile.

Вам не придется делать что-то особенное при разработке приложений для Всемирной паутины

Почти любой смартфон, имеющийся сегодня на рынке, например iPhone или устройства на базе Android, может читать и воспроизводить на экране страницы полноценных "настольных" веб-сайтов. Да, это так. Пользователи хотят, чтобы в мобильной паутине было все то же самое, к чему они привыкли на настольных компьютерах. И это тоже правда. Имеется статистика, показывающая, что владельцы смартфонов предпочитают обычные версии веб-сайтов мобильным версиям. Интересно, это потому, что нам нравится постоянно масштабировать и прокручивать странички на экране в поисках нужной информации, или потому, что мобильные версии действительно ужасны и не проявляют привычного пользователям поведения? Я видел множество мобильных сайтов, не содержащих ничего, кроме логотипа и пары текстовых ссылок. Мой смартфон хочет большего!

Один веб-сайт должен быть пригоден для всех устройств (настольных, мобильных, ТВ)

Как мы увидим далее, существуют технологии, позволяющие нам создавать один файл, который будет по-разному выглядеть на самых разных устройствах, включая настольные компьютеры, мобильные устройства, телевизоры и игровые приставки. Эта концепция называется "Одна паутина". Сегодня существует много мобильных устройств (не смартфонов), имеющих очень низкие скорости обмена данными и ограниченные аппаратные ресурсы. Теоретически они способны прочитать и интерпретировать любой файл, но не обеспечивают оптимальный пользовательский опыт, а при передаче им документа, предназначенного для настольного компьютера, могут возникнуть проблемы с совместимостью и производительностью. Поэтому "Одна паутина" остается целью на будущее. Для обеспечения оптимального пользовательского опыта требуется некоторая дополнительная работа для каждого мобильного устройства, однако существуют технологии, позволяющие минимизировать эту работу и избежать дублирования кода и данных.

Для получения мобильного веб-сайта достаточно создать HTML-страничку шириной 240 пикселов

Это еще один взгляд на мобильную Всемирную паутину с позиций "фастфуда". Сейчас на рынке имеется свыше 3000 моделей мобильных устройств и почти 50 разных браузеров (фактически, больше 500 браузеров, если различать их по номеру версии). Создание одного HTML-файла для мобильного веб-сайта будет заведомо провальным проектом. Кроме того, такой проект подкрепит позиции тех, кто верит, что в мобильной паутине нет толку.

Мобильные веб-приложения

Я не собираюсь устраивать дискуссию на тему "разработка приложений для мобильной паутины против обычной разработки". На самом деле, я считаю такую постановку вопроса некорректной. Как правило, необходимо сравнивать код в командах устройства с кодом JavaScript или браузерные приложения с устанавливаемыми приложениями. Однако в подобных дискуссиях обычно упускается из виду, что создание многоплатформенных приложений является очень сложной задачей в исходной среде разработки, потому что для каждой платформы требуется отдельный комплект SDK. Следовательно, поскольку нашей заботой в действительности является простота разработки и развертывания приложений на многих мобильных устройствах, разработка для мобильной паутины является оптимальным решением в большинстве ситуаций. Термин *"веб-приложение"* имеет массу синонимов или сходных терминов. Это мобильные веб-приложения, виджеты, гибридные приложения, приложения HTML5 и т. д.

В частности, мобильное веб-приложение отличается от типичных мобильных веб-сайтов своим предназначением. Веб-приложение, как правило, ведет себя более "транзакционно" в части пользовательского интерфейса, эмулируя собственные приложения мобильного устройства. Оно создается с помощью веб-технологий (HTML, CSS, JavaScript, AJAX), но ведет себя перед пользователем аналогично собственным приложениям устройства.

В мобильных приложениях часто используются функциональные возможности HTML5, такие как автономный или геопозиционный доступ, что обеспечивает более качественный пользовательский опыт. Геопозиционирование (определение географических координат) не является официальной составляющей спецификации HTML5; это самостоятельный API-интерфейс W3C. Тем не менее, оно часто упоминается в непосредственной связи с HTML5.

Веб-приложение может быть реализовано самыми разными способами (рис. 1.1), в том числе:

- запущено из веб-браузера;
- установлено в качестве полноэкранного приложения;
- установлено как веб-приложение с помощью пакета, официально реализованного производителем (иногда такие приложения называются *виджетами*);

- установлено как приложение, встроенное в приложение на языке устройства (обычно такие приложения называются *гибридными*).

Далее в этой книге мы обсудим, как создавать такие веб-приложения. Более подробную информацию можно найти в моей книге "Programming the Mobile Web".

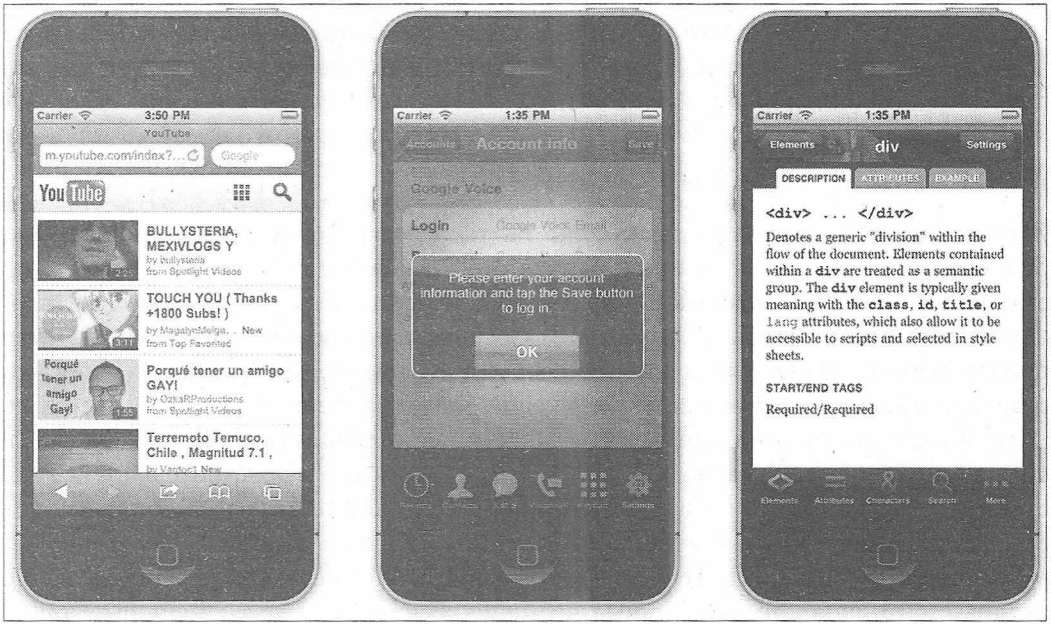


Рис. 1.1. Веб-приложение, которое (слева направо) работает в браузере, является полноэкранным приложением и встроено в собственное приложение устройства (так называемый гибрид)

Как правило, веб-приложение ставит сложные задачи перед веб-дизайнерами и разработчиками. Например, возникает необходимость загружать представления, а не страницы, поддерживать двухстороннюю навигацию между представлениями и создавать мощные элементы управления специально для сенсорных устройств.

Так зачем нам нужна технология jQuery Mobile?

Если вы прочитали начало этой главы (а я уверен, что это так), то отдаете себе отчет в том, что дизайн и разработка приложений для Всемирной паутины являются сложной задачей. Нам необходимо создавать веб-приложения, представляющие собой нечто большее, чем упрощенные сайты. На свете существует великое множество устройств, в разной степени совместимых с имеющимися браузерами, а также масса библиотек для решения этой проблемы при поддержке производителей и сообщества разработчиков.

Вот почему появилась технология jQuery Mobile. Она должна помочь дизайнерам и разработчикам в создании многоплатформенных настраиваемых приложений, имеющих "ненавязчивый" код и обеспечивающих качественную работу пользователя во Всемирной паутине.

Наличие многочисленного всемирного сообщества разработчиков, использующих jQuery, обеспечивает этой технологии хорошее будущее.

Среда jQuery Mobile пользуется официальным спонсорством и поддержкой со стороны многих крупнейших компаний, работающих в этой области, среди которых:

- Adobe;
- Mozilla Corporation;
- HP Palm;
- BlackBerry/RIM;
- Nokia;
- DeviceAtlas and dotMobi.

Что такое jQuery Mobile?

Согласно официальному описанию на сайте **http://www.jquerymobile.com**, jQuery Mobile представляет собой систему унифицированного пользовательского интерфейса для всех платформ, построенную на надежном основании jQuery и jQuery UI. Ее облегченный код включает в себя прогрессивные нововведения и обеспечивает гибкий дизайн с легко сменяемыми темами.

Чем не является jQuery Mobile?

Чтобы разобраться в технологии jQuery Mobile, важно понимать, чем она не является.

- jQuery Mobile не является jQuery-альтернативой мобильным браузерам.

  Чтобы пользоваться технологией jQuery Mobile, необходимо подключить обычную среду jQuery. Это не замена; это слой пользовательского интерфейса поверх jQuery.

- jQuery Mobile не является SDK-комплектом для веб-приложений.

  С помощью jQuery Mobile вы можете приобрести полноценный мобильный пользовательский опыт, но вам придется потратить некоторые усилия для компиляции своей работы в собственный код мобильного устройства. В последующих главах мы обсудим, как, зачем и когда это нужно делать.

- jQuery Mobile не является средой для поклонников JavaScript.

  За исключением некоторых особо сложных случаев, для работы jQuery Mobile не требуется код на языке JavaScript. Это должно обрадовать веб-дизайнеров, ненавидящих все эти скобки и точки с запятой.

- jQuery Mobile не является решением для всех мобильных приложений, сайтов или игр.

  Однако эта технология подходит для большинства из них. Что касается остальных... я, пожалуй, предложу вам почитать другую мою книгу.

Платформа

Если вы не знаете, что такое jQuery, то вы, по всей вероятности, являетесь путешественником во времени и прибыли к нам из прошлого, отстоящего более чем на 10 лет. Итак, если вы новый Марти Макфлай[1], направьте свой браузер на сайт **http://jquery.com** и почитайте об этой чрезвычайно мощной JavaScript-платформе, самой популярной у веб-разработчиков с 2007 года.

Технология jQuery Mobile является платформой, делающей веб-приложения доступными для мобильных и планшетных устройств, преимущественно ориентированных на касания пальцем. Эти приложения создаются без особых усилий для разных мобильных платформ и используют только стандартный код HTML5. Типичное приложение jQuery Mobile изображено на рис. 1.2.

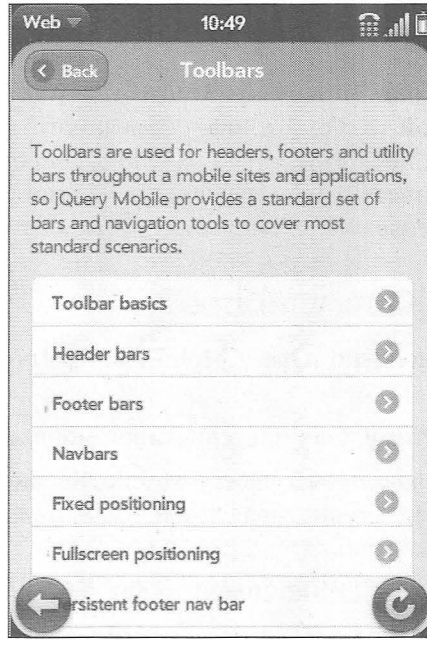


Рис. 1.2. Типичное приложение jQuery Mobile со стандартной установкой тем в смартфонах, в данном случае, в устройстве с webOS

Платформа использует "ядро" jQuery, библиотеку JavaScript, таблицу стилей CSS3 и некоторое количество изображений-ресурсов.

Платформу jQuery Mobile можно сравнить с jQuery UI для настольных компьютеров — это всего лишь платформа для создания пользовательского интерфейса. Название (без аббревиатуры UI, означающей "пользовательский интерфейс") наводит на мысль, что это ядро, однако, я полагаю, что оно было выбрано из маркетинговых соображений с учетом популярности бренда jQuery у дизайнеров и разработчиков.

---

[1] Намек автора на главного героя кинотрилогии "Назад в будущее". — *Ред.*

Платформа была создана тем же коллективом, что и jQuery, под руководством Джона Ресига (John Resig), создателя JavaScript Tool для Mozilla Corporation (его аккаунт в Twitter — @jeresig).

Эта новая платформа, как jQuery и jQuery UI, выпущена как проект с открытым исходным кодом под двойной лицензией MIT или GPL версии 2.

Если вы хотите поучаствовать в разработке, то можете создать "заплаты", исправить ошибки, принять участие в обсуждении и разработать рабочий код на сайте **http://jquerymobile.com/contribute**.

Мир мобильных и планшетных устройств

Люди выходят во Всемирную паутину, используя не только настольные компьютеры. Сейчас в нашем распоряжении имеются очень разные устройства с разными размерами экранов, механизмами ввода и даже с новыми функциональными возможностями от "старых добрых" HTML, JavaScript и CSS.

Мобильные устройства повсюду. Можно уверенно сказать, что во всем мире их более пяти миллиардов, и это число увеличивается. Планшеты тоже получают все более широкое распространение, и на рынке их уже миллионы.

Категории устройств

На данный момент мы можем разделить мобильные устройства на следующие категории:

- мобильные телефоны;
- бюджетные мобильные устройства;
- мобильные устройства промежуточного и имиджевого уровня, также известные как социальные;
- смартфоны;
- планшеты.

Мобильные телефоны

Мобильные телефоны по-прежнему продаются. Это телефоны, позволяющие делать звонки и отправлять SMS-сообщения. В них нет браузеров и подключения к Интернету, и на них невозможно устанавливать приложения. Они не представляют для нас интереса, поскольку мы не можем создавать для них приложения.

Через пару лет благодаря программам утилизации и расширению спектра услуг, предлагаемых операторами и производителями, такие телефоны, вероятно, исчезнут с рынка.

Бюджетные мобильные устройства

Бюджетные мобильные устройства имеют громадное преимущество над сотовыми телефонами: они позволяют выходить во Всемирную паутину. Как правило, они имеют очень примитивный браузер, но их рынок огромен. Возможно, они сейчас не являются основными устройствами для выхода в Интернет, но ситуация может быстро измениться с развитием социальных сетей и сервисов Web 2.0. Если ваши друзья могут обмениваться картинками с помощью мобильных устройств, такое желание, вероятно, возникнет и у вас, и вы откажетесь от старого сотового телефона при первой возможности.

Nokia, Motorola, Kyocera, LG, Samsung и Sony Ericsson поставляют устройства на этот рынок. Как правило, у них отсутствует сенсорный экран, память невелика, а фотокамера и плеер самые простые.

Мобильные устройства промежуточного и имиджевого уровня

Это самый доступный на массовом рынке вариант для тех, кому нужен мобильный выход во Всемирную паутину.

Устройства промежуточного и имиджевого уровня обеспечивают баланс между хорошими потребительскими свойствами и умеренной ценой. В последние годы за этой категорией закрепилось название "социальные устройства", отражающее тот

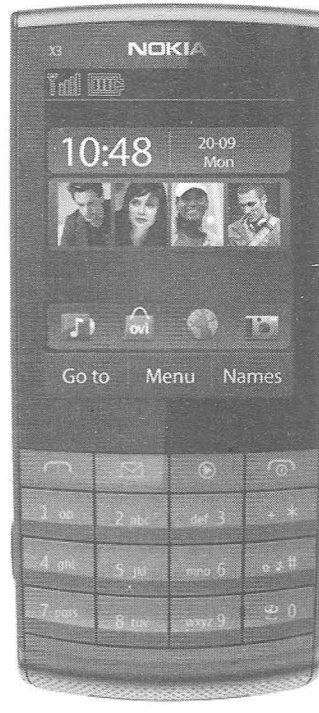


Рис. 1.3. Nokia X3-02 Touch and Type: сенсорное устройство промежуточного класса с цифровой клавиатурой и Wi-Fi

факт, что с их помощью пользователи выходят в социальные сети, такие как Facebook и Twitter.

Устройства из этой категории, как правило, имеют экран среднего размера, простой HTML-браузер, в некоторых случаях 3G, неплохую камеру, плеер, набор игр, а иногда сенсорный экран и возможность установки приложений. Огромная разница между этими устройствами и смартфонами состоит в том, что при продаже имиджевых устройств обычно не предлагается безлимитный тарифный план для пользования Интернетом. Покупатель может найти для себя безлимитный тарифный план, но должен делать это самостоятельно. Начиная с 2011 года, многие из этих устройств имеют WLAN (Wi-Fi), что видно из рис. 1.3.

Смартфоны

На рынке существуют десятки моделей смартфонов, таких как iPhone, устройства на базе Android, webOS, Symbian, BlackBerry, а также Windows Phone (рис. 1.4). Эта категория труднее всего поддается определению. Почему некоторые устройства промежуточного и имиджевого уровня считаются недостаточно "интеллектуальными", чтобы попасть в смартфоны? Само определение "интеллектуальности" устройства меняется каждый год. Даже самое простое современное мобильное устройство показалось бы "интеллектуальным" 10 лет назад.

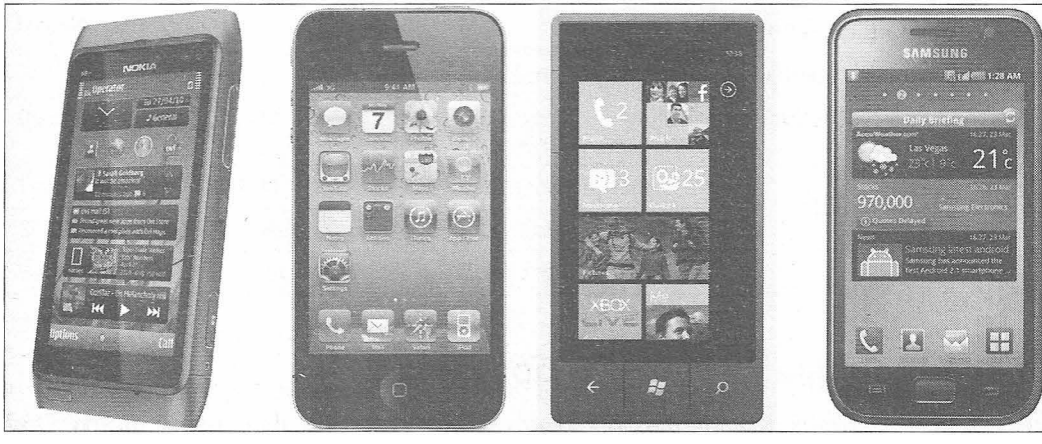


Рис. 1.4. Примеры смартфонов

Обычно при покупке смартфона вы заключаете одно- или двухгодичный контракт с безлимитным тарифным планом для передачи данных. Согласно современному определению смартфона, он имеет многозадачную легко опознаваемую операционную систему, современный HTML5-браузер, возможность подключения к беспроводной сети LAN (WLAN, также известный как Wi-Fi) и 3G, аудиоплеер и какие-либо из следующих функциональных возможностей:

- систему глобального позиционирования GPS или A-GPS;
- цифровой компас;
- камеру с возможностью видеосъемки;

- выход на телевизор;
- Bluetooth;
- сенсорный экран;
- ускорение 3D-видео;
- акселерометр.

Некоторые мультимедийные устройства для нас, разработчиков веб-приложений, мало отличаются от смартфонов. Однако они не имеют функций телефона. В качестве примеров можно назвать Apple iPod Touch и Sony PlayStation Portable (PSP). От планшетов они отличаются лишь размером экрана.

Планшеты

Планшет — это устройство с большим экраном (от 6 до 11 дюймов), полнофункциональным HTML5-браузером, возможностью подключаться к WLAN (Wi-Fi) и, в некоторых случаях 3G, сенсорным управлением и всеми функциональными возможностями смартфона.

В эту категорию входят многие устройства, в том числе:

- Apple iPad;
- Samsung Galaxy Tab;
- BlackBerry PlayBook;
- Barnes and Noble Nook Color;
- Motorola Xoom;
- LG Optimus Pad;
- Amazon Fire;
- Sony S1 и S2.

Операционные системы и браузеры

Глубокое погружение в мобильную экосистему не является целью этой книги. Подробный список операционных систем, платформ и браузеров можно найти в книге "Programming the Mobile Web". Однако если мы собираемся создавать мобильные веб-приложения, то должны хотя бы знать, о чем идет речь.

В мобильном мире операционные системы можно разбить на две основные категории: опознаваемые и собственные (патентованные) операционные системы. Во вторую группу входят системы, на которых работают телефоны, а также бюджетные и промежуточные устройства.

Что касается опознаваемых операционных систем, нас будет больше интересовать, какая ОС установлена в устройстве, чем производитель и модель самого устройства. Иными словами, мы будем разрабатывать веб-приложение не для Samsung Galaxy, а для устройства с системой Android. Исключением из этого правила, воз-

можно, будет iPhone, поскольку он имеет собственную платформу, на которой в момент написания этой книги работало только одно устройство: iPhone. (Различные версии устройства положения не меняют; с точки зрения веб-разработчика нет существенных отличий iPhone 4 от iPhone 3GS.)

В табл. 1.1 перечислены операционные системы смартфонов и планшетов, имеющихся сегодня на рынке.

Таблица 1.1. *Операционные системы и браузеры на смартфонах, социальных устройствах и планшетах*

Операционная система	Производитель	Установленный браузер	Прочие браузеры
iOS	Apple	Safari	Opera Mini и псевдобраузеры
Android	Google	Android Browser	Firefox Opera Mini, Amazon Silk, Opera Mobile
Symbian	Nokia	Symbian Browser	Opera Mini, Opera Mobile
webOS/Open webOS	HP (бывший Palm)	webOS Browser	
Windows Phone	Microsoft	Internet Explorer	
Windows Mobile	Microsoft	Internet Explorer	Opera Mobile
MeeGo	Nokia	Micro Browser/Nokia Browser	Firefox
BlackBerry OS	RIM	BlackBerry Browser	Opera Mini
Tablet OS	RIM	Tablet OS Browser	
S40	Nokia	Nokia Browser	
Bada	Samsung	Samsung Browser	

У каждой операционной системы есть разные версии, и некоторые из них позволяют пользователю перейти к более свежей версии. Каждая ОС имеет установленный браузер, но пользователь может установить альтернативный. В некоторых случаях изготовитель или оператор, у которого пользователь купил устройство, устанавливает альтернативный браузер, например Opera Mobile, или даже заменяет им основной.

Если мы расширим наш интерес к браузерам до бюджетных и промежуточных устройств, то обнаружим более 20 новых обозревателей, в том числе Ovi Browser, NetFront Browser и Phantom Browser от LG. Однако в данный момент это не является целью jQuery Mobile.

Что такое псевдобраузер?

Псевдобраузер — это приложение, написанное в кодах устройства, которое может быть на этом устройстве установлено. Псевдобраузер использует тот же движок, что и браузер, установленный по умолчанию, но обладает дополнительными функциональ-

ными возможностями. Для iOS существует большое количество псевдобраузеров, например SkyFire или Perfect Browser. Все они используют Safari в качестве механизма отображения, поэтому не являются разными браузерами с точки зрения jQuery Mobile.

В книге "Programming the Mobile Web" целых 20 страниц посвящено подробной информации о типах браузеров и характеристиках каждого из них.

Совместимость jQuery Mobile

Платформа jQuery Mobile создана специально для устройств с сенсорным экраном, таких как смартфоны, планшеты и мультимедийные устройства. Список совместимости будет постепенно расширяться по мере развития платформы, так что привести здесь полный перечень весьма затруднительно.

Платформа jQuery Mobile 1.0 совместима со следующими браузерами, устанавливаемыми по умолчанию:

- iOS: Safari for iPhone, iPod Touch и iPad from iOS 3.2;
- Android OS: Android Browser для телефонов и планшетов;
- BlackBerry OS: BlackBerry Browser для смартфонов от 5.0 и планшетов;
- Symbian: Nokia Browser для сенсорных устройств;
- webOS: webOS Browser из webOS 1.4;
- Bada: Bada Browser;
- MeeGo: Micro Browser и Nokia Browser (поставляемый с Nokia N9);
- Windows Phone: Internet Explorer из Windows Phone/Mobile 6.5 и Windows Phone 7.0;
- Kindle: браузер из Kindle 3.

Платформа jQuery Mobile также совместима с браузерами от сторонних производителей:

- Opera Mini, полностью поддерживаемый на большинстве устройств, начиная с версии 5.0;
- Opera Mobile, полностью поддерживаемый на большинстве устройств, начиная с версии 10.0;
- Firefox Mobile.

Эти сведения о совместимости дают вам лишь самую начальную информацию. Вопрос о совместимости гораздо сложнее, поскольку сочетание множества операционных систем с множеством версий браузеров дает очень разные результаты. А новые устройства, не перечисленные здесь, будут совместимы с библиотекой, если они поддерживают минимум функциональных возможностей, необходимых платформе.

Проще говоря, jQuery Mobile будет работать на любом браузере, который способен отображать то, что предлагает платформа. Таким образом, в список следует включить любой современный браузер.

Многие современные мобильные браузеры используют движок на базе WebKit, аналогичный Safari или Chrome для настольных компьютеров. Любой современный мобильный браузер на базе WebKit должен быть полностью совместим с jQuery Mobile. Кроме того, Chrome, Firefox, Safari, Opera и Internet Explorer для настольных компьютеров совместимы с jQuery Mobile.

Поддержка мобильных браузеров по категориям

В библиотеке jQuery Mobile используется таблица, определяющая совместимость каждого устройства с этой библиотекой (рис. 1.5). На месте читателя я не стал бы углубляться в вопрос о разбивке на категории. Однако если вам интересно, вы можете получить подробную информацию по адресу **http://jquerymobile.com/gbs/**.

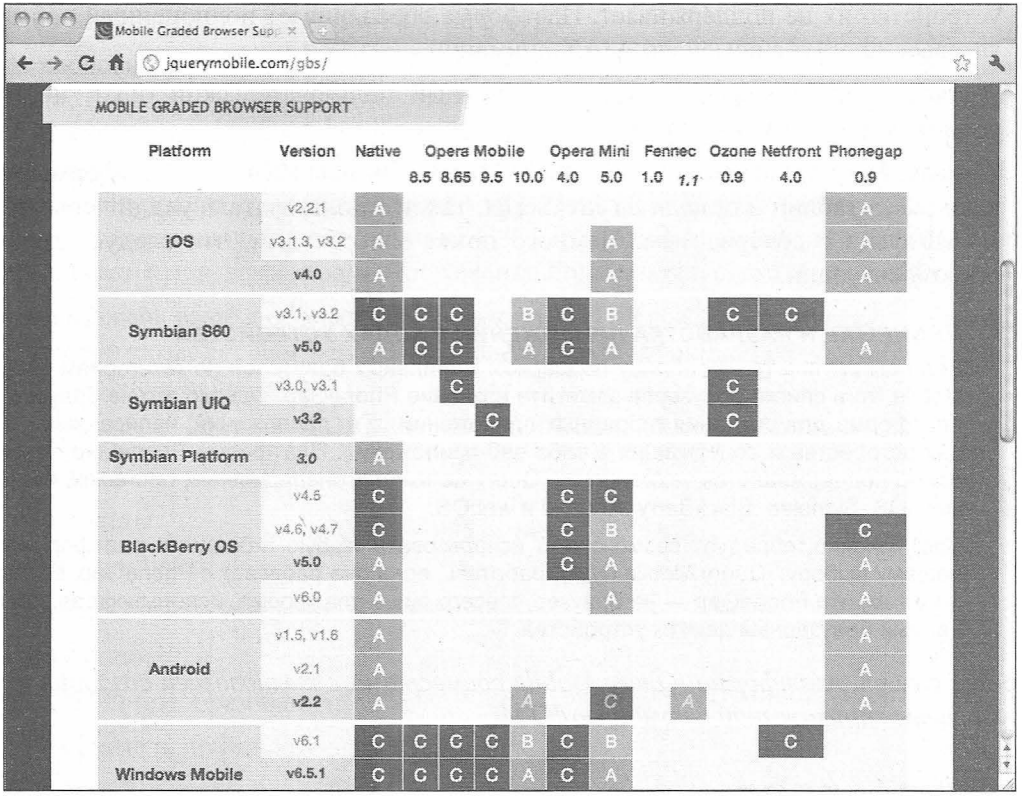


Рис. 1.5. На сайте jQuery Mobile имеется список совместимых браузеров

Многие современные браузеры для настольных компьютеров, такие как Firefox, Google Chrome, Safari и Internet Explorer, тоже совместимы с jQuery Mobile. Хотя сама технология и не предназначена для создания настольных приложений, это обстоятельство полезно для тестирования. Однако, как мы убедимся впоследствии, установка эмулирующей среды еще полезнее.

Я думаю, что тема совместимости гораздо сложнее, чем показано в этой таблице, и что "простому" веб-дизайнеру или разработчику невозможно разобраться в ней до

конца. Существуют более надежные способы выяснения, доступна ли та или иная функциональная возможность в мобильном браузере, чем попытки отнести ее к какой-то категории в таблице. Один из этих способов имеется в вашем распоряжении: *воспользуйтесь технологией jQuery Mobile.*

Согласно классификации GBS (Graded Browser Support, поддержка браузеров по категориям) мобильные браузеры разделены на три категории: A-grade (категория A), B-grade (категория B) и C-grade (категория C). Для jQuery Mobile эта классификация имеет следующий смысл.

- A-grade. Это браузер с поддержкой медийных запросов CSS3. Такие браузеры всесторонне тестируются командой разработчиков jQuery. Однако некоторые функциональные возможности автоматически отключаются, если конкретное устройство их не поддерживает. Платформа обеспечивает полноценный пользовательский опыт с помощью AJAX-анимации.
- B-grade. Браузер обеспечивает качественный пользовательский опыт, но без возможностей AJAX-навигации.
- C-grade. Браузер несовместим с jQuery Mobile. Он не получит от платформы никаких CSS-таблиц или кода на JavaScript, так что пользователь увидит обычный HTML-файл с содержимым. Немного позже мы обсудим, что следует делать в такой ситуации.

> ***PhoneGap и разработка приложения в кодах устройства***
>
> Если вы взглянете на таблицу поддержки мобильных браузеров по категориям jQuery Mobile, то в списке браузеров заметите название PhoneGap. Однако это не браузер, а платформа для создания гибридных приложений, т. е. приложений, написанных в кодах устройства и содержащих в себе веб-приложения. Платформа PhoneGap официально поддерживается технологией jQuery во многих операционных системах, например iOS, Symbian, BlackBerry, Android и webOS.
>
> Вас, конечно, обрадует возможность использовать любую гибридную платформу по вашему выбору: jQuery Mobile будет работать, если она работает с PhoneGap. Все дело в том, что PhoneGap — не браузер, а всего лишь платформа, использующая собственный браузерный движок устройства.

Проще говоря, *платформа jQuery Mobile совместима с технологией создания низкоуровневых приложений с помощью HTML.*

HTML5 и CSS3

Мне известно, что большинство веб-дизайнеров и разработчиков панически боится HTML5 и CSS3. Для начала скажу, что у вас нет повода волноваться, поскольку jQuery Mobile все сделает за вас. Таким образом, вам не придется изучать HTML5 и CSS3 для работы на этой платформе. И, тем не менее, я настоятельно советую вам сделать это. Зная эти стандарты, вы сможете сделать гораздо больше, однако эту тему мы обсудим позже.

Это книга не является учебником по HTML5 или CSS3, но вам важно понимать некоторые моменты, связанные с ними. Большинство мобильных браузеров, как пра-

вило, в смартфонах и планшетах, поддерживает HTML5, CSS3 и другие API-интерфейсы.

Про HTML5 я могу говорить часами, рассказывая историю его возникновения и описывая возможности, представляемые этим стандартом мобильной экосистеме.

Упрощенно говоря, HTML5 — это развивающийся стандарт, который включает в себя изменения в языке разметки HTML и большое количество новых API-интерфейсов на JavaScript (да, в HTML5 очень много внимания уделено API-интерфейсам JavaScript). Выражаясь образно, HTML5 — это зонтик над многими современными возможностями браузеров, включая формальный стандарт HTML5 от консорциума W3C, другие API-интерфейсы от W3C, CSS3, а также нестандартные расширения. Информацию о совместимости HTML5 с мобильными браузерами можно найти на сайте **http://mobilehtml5.org**.

В технологии jQuery Mobile используются многие возможности HTML5, позволяющие обеспечить качественный пользовательский опыт на мобильных браузерах. Это не означает, что сам браузер должен целиком поддерживать HTML5. На самом деле, многие старые браузеры поддерживают некоторые теги разметки HTML5, даже не подозревая о существовании этого стандарта. В технологии jQuery Mobile также активно используются таблицы CSS3, где это возможно, для создания анимаций, градиентов, эффектов и отображения пользовательского интерфейса.

Чтобы разжечь ваше любопытство, скажу, что HTML5, CSS3 и другие современные технологии позволят вам реализовать следующую функциональность (с помощью jQuery Mobile или без нее):

- автономный доступ к данным;
- автономное хранение информации;
- веб-сокеты;
- геопозиционирование;
- поддержку акселерометра и гироскопа;
- анимацию;
- 2D- и 3D-преобразования;
- градиенты и визуальные эффекты;
- управление окном просмотра (для поддержки масштабирования в браузере);
- установочные метаданные веб-приложения;
- интеграцию с приложениями на собственном коде устройства;
- поддержку мультимедийных возможностей;
- рисование графики (векторной и растровой);
- поддержку нестандартных шрифтов.

В моем блоге (**http://www.mobilexweb.com/**) имеется несколько примеров и ссылок по этой теме.

Основные функциональные возможности

Технология jQuery Mobile возникла в августе 2010 года в качестве современной платформы, включающей в себя лучшие образцы и технические приемы разработки многоплатформенных приложений. Среди ее основных достоинств и функциональных возможностей следует назвать:

- независимость от платформы, устройства и браузера;
- интерфейс, оптимизированный для сенсорных устройств;
- настраиваемый дизайн со сменными темами;
- применение только кода HTML5, имеющего "ненавязчивую" семантику, отсутствие требования знать JavaScript, CSS или API;
- автоматические вызовы AJAX для загрузки динамического содержимого;
- происхождение от хорошо известного и поддерживаемого ядра jQuery;
- небольшой размер — 12 Кбайт в сжатом виде;
- прогрессивное улучшение;
- поддержка пользователей с ограниченными возможностями.

Некоторые из этих пунктов мы уже обсудили. Давайте подробно рассмотрим другие.

Поддержка ненавязчивой семантики HTML5

Я знаю: вам не терпится увидеть конкретный код. Сейчас я его приведу. Технология jQuery Mobile позволяет использовать стандартный и семантический HTML5 для создания веб-приложений, отлично подходящих для SEO- и WPO-оптимизации (оптимизации для поисковых механизмов и оптимизации производительности соответственно):

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>My first jQuery Mobile code</title>
  <link rel="stylesheet"
    href="http://code.jquery.com/mobile/1.0/jquery.mobile-1.0.min.css" />
  <script src="http://code.jquery.com/jquery-1.6.4.min.js"></script>
  <script type="text/javascript"
    src="http://code.jquery.com/mobile/1.0/
                jquery.mobile-1.0.min.js"></script>
  <meta name="viewport" content="width=device-width, initial-scale=1">
</head>
<body>
  <div data-role="page" data-theme="a">
    <div data-role="header">
```

```
      <h1>jQuery Mobile</h1>
    </div>
    <div data-role="content">
      <ul data-role="listview"
          data-inset="true" data-divider-theme="b">
        <li data-role="list-divider">Summary</li>
        <li><a href="ch1.html">The Platform</a></li>
        <li><a href="cap2.html">The Page</a></li>
        <li><a href="cap3.html">Lists</a></li>
        <li><a href="cap4.html">Components</a></li>
      </ul>
      <ul data-role="listview"
          data-inset="true" data-divider-theme="d">
        <li data-role="list-divider">Links</li>
        <li><a href="http://www.mobilexweb.com">Mobile Web Blog</a></li>
        <li><a href="http://www.oreilly.com">O'Reilly Media</a></li>
      </ul>
    </div>
    <div data-role="footer">
      <h4>&copy; 2011 Maximiliano Firtman @firt</h4>
    </div>
  </div>
</body>
</html>
```

На рис. 1.6 показано, как этот код отображается разными мобильными браузерами, а на рис. 1.7 — как он отображается браузером, несовместимым с jQuery Mobile.

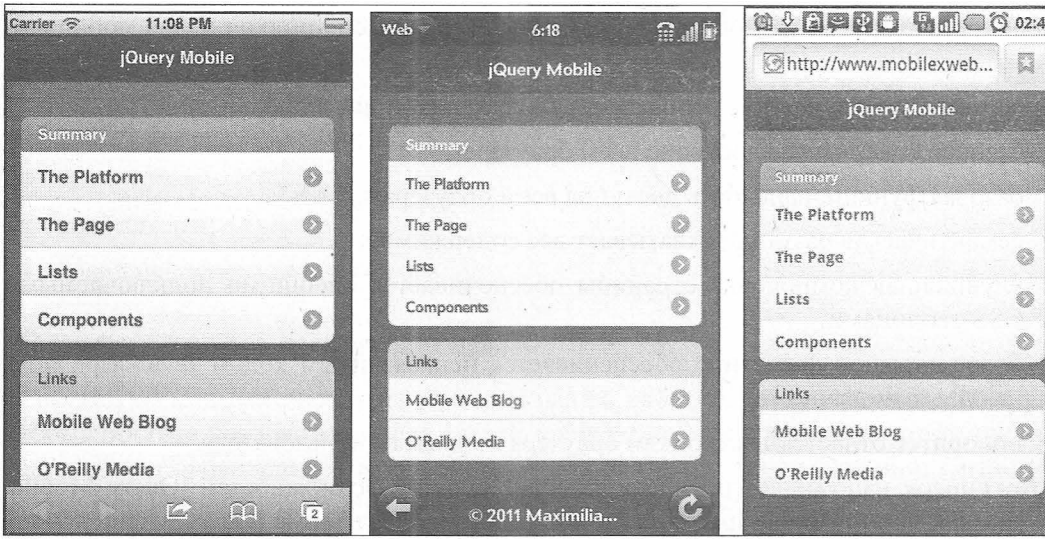


Рис. 1.6. Отображение нашего простого кода jQuery Mobile в разных системах: iOS, webOS и Android

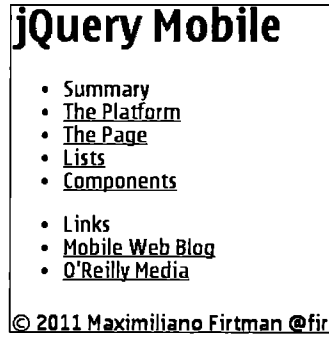


Рис. 1.7. Отображение кода jQuery Mobile несовместимым браузером: простой, полнофункциональный HTML-файл

Нетрудно заметить, что для инициализации не используется код JavaScript, и фрагмент включает в себя лишь незначительные вставки JavaScript.

Наберитесь терпения; анализ кода jQuery будет произведен в последующих главах.

Прогрессивное улучшение

Прогрессивное улучшение — это простая, но очень эффективная техника веб-дизайна, определяющая уровни совместимости, которые позволяют любому пользователю обращаться к базовому содержимому, сервисам и функциональности сайта, предоставляя при этом больше возможностей браузерам, в большей степени соответствующим стандартам. Платформа jQuery Mobile целиком построена с применением этой техники.

Термин "прогрессивное улучшение" был предложен Стивеном Чампеоном (Steven Champeon) в 2003 году (**http://www.hesketh.com**). Хотя этот подход не был специально ориентирован на мобильную паутину, он прекрасно подходит для мобильного веб-дизайна.

Прогрессивное улучшение основано на следующих принципах:

- базовое содержимое доступно всем браузерам;
- базовая функциональность доступна всем браузерам;
- семантическая разметка охватывает все содержимое;
- улучшенная компоновка страницы обеспечивается внешними подключаемыми CSS-таблицами;
- более сложное поведение обеспечивается ненавязчивым кодом JavaScript, подключаемым извне;
- приоритет отдается настройкам браузера конечного пользователя.

Этот список напоминает перечень функциональных возможностей jQuery Mobile, не так ли? Именно так. Приложение, созданное на платформе jQuery Mobile, будет работать и на самом примитивном браузере, не поддерживающем CSS или JavaScript. И это огромное достоинство мобильного веб-приложения.

Доступность для пользователей с ограниченными возможностями

Приведу цитату из Википедии:

> "Доступность во Всемирной паутине означает исключительную практику создания сайтов, которыми могут пользоваться люди, имеющие или не имеющие ограничений в деятельности. На правильно спроектированных, разработанных и сопровождаемых сайтах все пользователи имеют равный доступ к данным и функциональности."

Поддержка пользователей с ограниченными возможностями находится в мобильных браузерах в зачаточном состоянии. Тем не менее, платформа jQuery Mobile уже полностью соответствует спецификации WAI-ARIA от консорциума W3C, относящейся к мобильным браузерам (**http://www.w3.org/TR/wai-aria/**). На момент работы над этой книгой только система iOS версии 4.0 и выше была совместима с этой спецификацией благодаря функциональной возможности, называемой VoiceOver.

Таким образом, веб-приложение на основе jQuery Mobile обеспечит полноценный доступ к содержимому сайтов для незрячих пользователей, имеющих iPhone, iPod и iPad.

Тестирование веб-приложений

Уже упоминалось, что веб-приложение на основе jQuery Mobile будет работать практически в любом современном "настольном" браузере. Однако будет лучше, если мы сможем тестировать свои приложения в среде, максимально точно отражающей реальность (рис. 1.8).

Чтобы протестировать мобильное веб-приложение в разном окружении, мы можем использовать:

- реальные устройства;
- удаленные лаборатории;
- эмуляторы;
- симуляторы;
- большое количество знакомых.

Эмуляторы и симуляторы

Самыми полезными инструментами для нашей работы будут эмуляторы и симуляторы. Вообще говоря, *эмулятор* — это программа, транслирующая откомпилированный код оригинальной архитектуры в код платформы, на которой она работает. Эмулятор позволяет нам запустить операционную систему и ее приложения в другой операционной системе. В контексте разработки мобильных приложений эмулятор является приложением на настольном компьютере, которое эмулирует поведение аппаратной части и операционной системы мобильного устройства, позволяя

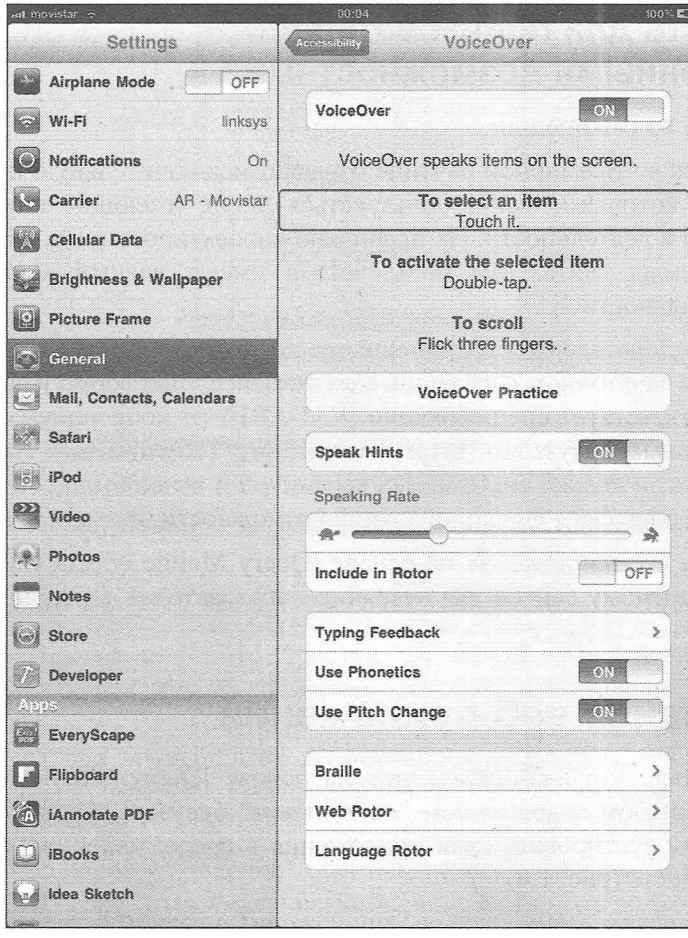


Рис. 1.8. Тестирование поддержки любого пользователя со стороны jQuery Mobile на iPad

нам тестировать и отлаживать приложения и наблюдать за их поведением. Браузер и даже сама операционная система не подозревают, что работают на эмуляторе, так что мы можем выполнять код, как будто на реальном устройстве.

Мы также должны добавить в среду разработки мобильных приложений классические инструменты управления проектом, такие как отладчик и контроль версий.

На рис. 1.8 приведен пример того, как тестируемое приложение на jQuery Mobile поддерживает пользователей с ограниченными возможностями на устройствах iPhone, iPod и iPad с операционной системой iOS версии 4.0 и выше. Выберите **Settings | General | Accessibility** (Параметры | Общие | Доступ) и активизируйте функцию VoiceOver. Теперь закройте глаза и исследуйте сайт на слух.

На рис. 1.9 показано, как приложение Android Emulator обеспечивает работу полной версии операционной системы Android OS на настольном компьютере, предлагая изображения различных устройств, в том числе планшетов Galaxy Tab и Nook Color.

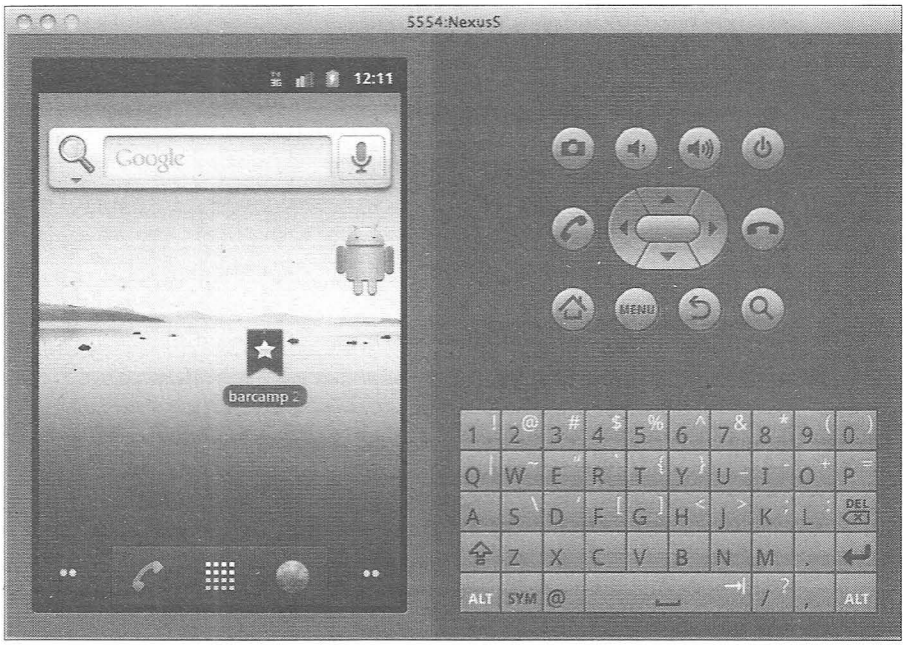


Рис. 1.9. Программа Android Emulator

Эмуляторы создаются производителями и предоставляются разработчикам бесплатно либо как самостоятельные приложения, либо в составе SDK для написания в кодах устройства.

Существуют также эмуляторы операционных систем, представляющие не аппаратную часть реального устройства, а операционную систему в целом. Такие эмуляторы имеются для Windows Mobile и Android.

Симулятор — это не менее сложное приложение, частично имитирующее поведение устройства, но не эмулирующее работу аппаратуры и реальной операционной системы. Эти инструменты проще эмуляторов, и пользы от них меньше. Симулятор может быть создан изготовителем устройства или какой-нибудь другой фирмой, создающей симулирующую среду для разработчиков. Для мобильных браузеров существуют симуляторы, имитирующие устройство на уровне пикселов, но есть и такие, которые не только не позволяют менять скины у типичного браузера (например, Firefox или Safari) с более-менее правдоподобным внешним видом, но даже и не симулируют механизм представления страниц. На рис. 1.10 показано приложение iOS simulator, бесплатно предоставляющее вам iPad в недрах вашего Mac. То же самое может произойти и с другими планшетами на компьютерах с ОС Windows или Linux.

Даже при использовании эмуляторов вы не получите точно такое же отображение содержимого и точно такую же производительность, как на реальном устройстве. Поэтому правильным решением будет тестирование веб-приложения на реальном устройстве, даже если вы собираетесь проводить тесты только на некоторых ключевых моделях.

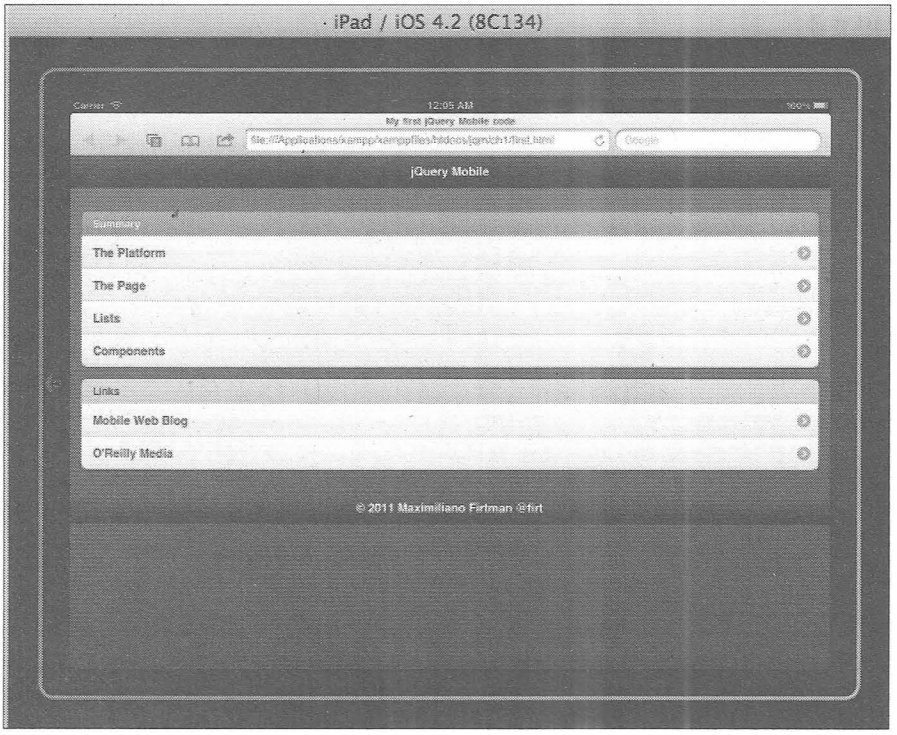


Рис. 1.10. Приложение iOS simulator

Для целей разработки мобильных приложений имеются эмуляторы от Nokia, Symbian, BlackBerry, Android, webOS и Windows Mobile и симуляторы от Apple для iPhone и iPad (впрочем, только для Mac OS X).

Также доступны эмуляторы некоторых браузеров (работающие на нескольких разных платформах), например эмулятор Opera Mobile.

В табл. 1.2 перечислены симуляторы и эмуляторы, которые можно загрузить из Интернета.

Таблица 1.2. *Эмуляторы и симуляторы мобильных устройств и планшетов, доступные для загрузки из Интернета*

Название	Платформа	Тип	Браузеры	Windows	Mac	Linux
iOS Simulator	iOS	Симулятор	Safari	Нет	Да	Нет
Android Emulator	Android	Эмулятор	Android Browser и загружаемые браузеры	Да	Да	Да
HP webOS Emulator	webOS	Эмулятор	webOS Browser	Да	Да	Да
Nokia Symbian Emulators	Symbian	Эмулятор	Внутренний и загружаемые браузеры	Да	Нет	Нет

Таблица 1.2 (окончание)

Название	Платформа	Тип	Браузеры	Windows	Mac	Linux
Windows Phone Emulator	Windows Phone	Эмулятор	Internet Explorer	Да	Нет	Нет
Nokia Series 40 Emulators	Nokia OS	Эмулятор	S40, Ovi Browser, Opera Mini	Да	Нет	Нет
BlackBerry Simulators	BlackBerry OS	Эмулятор	BB Browser и загружаемые браузеры	Да	Нет	Нет
BlackBerry PlayBook Simulator	Tablet OS	Эмулятор	Внутренний браузер	Да	Да	Да
Opera Mobile Emulator	Разные	Эмулятор браузера	Opera Mobile	Да	Да	Да
Opera Mini Simulator	Разные	Онлайн-эмулятор браузера	Opera Mini	Да	Да	Да
PhoneGap Simulator	Разные	Симулятор	Гибрид PhoneGap	Да	Да	Да
Adobe Device Central	Разные	Симулятор	Разные	Да	Да	Нет

Актуальный список адресов, по которым эмуляторы доступны для загрузки, можно найти на сайте **http://www.mobilexweb.com/emulators**.

Удаленные лаборатории

Удаленная лаборатория — это веб-служба, позволяющая использовать реальное устройство дистанционно, т. е. не находясь рядом с ним физически. Это простое, но очень эффективное решение, предоставляющее разработчику доступ к тысячам реальных устройств, объединенных в реальные сети по всему миру, всего за один щелчок мышью. К такой лаборатории можно относиться как к удаленному компьютеру для мобильных телефонов.

Самые действенные из служб, доступных на этом рынке, следующие:

- Keynote DeviceAnywhere (платная) — **http://www.deviceanywhere.com**;
- Perfecto Mobile (платная) — **http://www.perfectomobile.com**;
- Nokia Remote Device Access for Symbian and MeeGo (бесплатная) — **http://www.mobilexweb.com/go/rda**;
- Samsung Lab.Dev for Android (бесплатная) — **http://www.mobilexweb.com/go/labdev**.

Самую свежую информацию по этой теме можно найти на сайте **http://www.mobilexweb.com/go/labs**.

ГЛАВА 2

Приступаем к работе

Подготовка документа

Не побоимся черной работы и создадим шаблон типичного веб-приложения jQuery Mobile.

Требования

Наш документ в формате HTML5 должен включать в себя:

- JavaScript-файл ядра jQuery;
- JavaScript-файл ядра jQuery Mobile;
- CSS-файл ядра jQuery Mobile;
- CSS-файл темы jQuery Mobile (необязательно).

Кроме того, для jQuery Mobile требуется ряд PNG-файлов для некоторых элементов пользовательского интерфейса, но у нас нет необходимости подключать их явным образом. Также существует версия CSS-файла, включающая в себя как файл ядра, так и тему, выбираемую по умолчанию.

Прежде чем приступить к кодированию, необходимо принять решение о том, как будут храниться ресурсы. Существуют два подхода:

- хранить все файлы в рамках проекта;
- использовать CDN (Content Delivery Network, сеть доставки содержимого).

Хранение файлов

Если вы хотите хранить все файлы вместе с веб-приложением, вам нужно будет загрузить самый свежий пакет с сайта **http://jquerymobile.com/download**. Имя ZIP-файла содержит версию платформы, например, jquery.mobile-1.0.zip.

Пакет jQuery Mobile не включает в себя ядро jQuery. Вам придется загрузить его с сайта **http://jquery.com** (рекомендуется эксплуатационная версия).

Пакет jquery.mobile-*XX*.zip имеет следующую структуру:

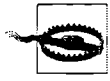
- папка demos;
- папка images;
- jquery.mobile-*XX*.css;
- jquery.mobile-*XX*.js;

- jquery.mobile-*XX*.min.css;
- jquery.mobile-*XX*.min.js;
- jquery.mobile.structure-*XX*.css;
- jquery.mobile.structure-*XX*.min.css.

Здесь *XX* обозначает номер версии, включающей в себя тип выпуска, например, 1.1b1 для 1.1 Beta 1, 1.0rc2 для 1.0 Release Candidate 2 или 1.0 для 1.0 final.

Нетрудно заметить, что пакет содержит файлы двух типов: с суффиксом min в имени и без него.

Файлы с суффиксом min рекомендуются для готовых приложений, потому что они минимизированы (сжаты и не содержат пробелов, комментариев и разбивки на строчки). Если вам нужно отлаживать приложение на платформе jQuery Mobile, вы можете использовать файлы, имена которых не содержат суффикс.



Для jQuery Mobile 1.0 необходима версия jQuery 1.6.4. Не используйте более позднюю версию, потому что могут возникнуть проблемы с совместимостью версий. Если у вас уже есть более поздняя версия мобильной платформы, изучите документацию на предмет того, какая версия ядра платформы подходит для вашего случая.

В самой типичной ситуации вы должны будете добавить в корневой каталог проекта следующие файлы:

- jquery-*XX*.js (из ядра jQuery);
- папку images;
- jquery.mobile-*XX*_min.js;
- jquery.mobile-*XX*_min.css.

Если вы создаете веб-приложение с помощью PhoneGap или другого механизма разработки автономных/гибридных приложений, то вам лучше встроить эти файлы в пакет, чтобы приложение могло работать автономно.

Файлы с суффиксом structure в имени полезны, когда вы собираетесь создать собственную тему. Об этом мы поговорим чуть позже.

Лицензия jQuery Mobile

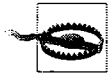
Использование платформы (и ядра jQuery) бесплатно. Это проекты с открытым исходным кодом с двойной лицензией, MIT или GPL версии 2. Лицензия MIT рекомендуется для большинства проектов, причем она от вас ничего не требует. Единственное важное условие — вы не должны удалять или изменять информацию об авторских правах в начале файлов. Если у вас возникнут сомнения, изучите документ, расположенный по адресу **http://jquery.org/license**.

Если вы храните файлы на своем сервере, убедитесь, что они упакованы, если клиент поддерживает сжатие. Тем самым вы сократите объем передаваемого и загружаемого кода JavaScript и CSS на 80%. Если у вас возникнут вопросы, проконсультируйтесь у провайдера или изучите материал по адресу **http://mobilexweb.com/go/performance**.

Использование CDN

Существует более простой механизм использования jQuery Mobile для разработки веб-приложений: это сети CDN (Content Delivery Network, сеть доставки содержимого). Сеть jQuery представляет собой публичный сервер во Всемирной паутине, на котором мы можем хранить файлы. Такой подход имеет как достоинства, так и недостатки.

Основной недостаток состоит в том, что ваше приложение будет работать только тогда, когда работает CDN. Конечно, администрация и сотрудники сети делают все возможное, чтобы находиться в режиме онлайн круглосуточно семь дней в неделю. Впрочем, некоторые проекты, например автономные приложения, не зависят в своей работе от сторонних серверов.



При создании гибридных или низкоуровневых приложений с помощью jQuery Mobile (например, приложений PhoneGap) не следует использовать CDN. Если устройство потеряет связь с оператором, вы даже не сможете выдать корректное сообщение. Создавая автономные веб-приложения на основе HTML5, вы можете использовать CDN, если сделаете соответствующую запись в манифесте приложения.

Назову основные достоинства использования CDN в разработке:

- вы можете использовать jQuery Mobile прямо сейчас, ничего не загружая;
- вашему серверу нужно будет передавать гораздо меньше файлов, что уменьшит трафик;
- ваше приложение сможет применять кэширование: если пользователь запускал другие веб-приложения, созданные с помощью jQuery Mobile и той же сети CDN, его браузер, скорее всего, уже имеет в кэше все необходимые ресурсы;
- с большинства публичных серверов хостинга ресурсы jQuery Mobile будут загружаться быстрее, если используется CDN;
- ваше приложение сможет обращаться к разным доменам с целью повышения производительности;
- некоторые сети CDN позволяют всегда использовать самую свежую версию файлов. Однако это не рекомендуется, поскольку вы не знаете, как ваше приложение поведет себя с последними версиями, пока не выполните тестирование.

Из-за недостатка места в этой книге не обсуждаются вопросы оптимизации производительности для мобильных браузеров. Если вас интересует эта тема, посетите страницу **http://www.mobilexweb.com/go/performance** и подпишитесь на блог Стива Соудерса (Steve Souders) — **http://stevesouders.com**.

Для ядра jQuery имеется несколько CDN-сетей на выбор:

- официальная CDN-сеть для jQuery;
- CDN-сеть для jQuery от Microsoft (**http://www.asp.net/ajaxlibrary/CDN.ashx**);
- API-библиотека AJAX от Google (**http://code.google.com/apis/libraries/**).

На момент работы над этой книгой официальная сеть CDN jQuery и CDN-сеть от Microsoft обеспечивали хостинг файлов jQuery Mobile.

Пользоваться сетью CDN очень просто. Достаточно скопировать URL-адрес во внешний файл JavaScript или CSS. На страничке **http://jquerymobile.com/download/** вы найдете фрагмент кода, который следует скопировать:

```
<link rel="stylesheet"
  href="http://code.jquery.com/mobile/1.0/jquery.mobile-1.0.min.css" />
<script src="http://code.jquery.com/jquery-1.6.4.min.js"></script>
<script src="http://code.jquery.com/mobile/1.0/
jquery.mobile-1.0.min.js"></script>
```

Если вы создаете собственную тему (о чем мы поговорим чуть позже), вам нужно использовать следующий код:

```
<link rel="stylesheet"
  href="http://code.jquery.com/mobile/1.0/jquery.mobile.structure-1.0.min.css" />
<-- Здесь должен быть CSS-файл темы -->
<script src="http://code.jquery.com/jquery-1.6.4.min.js"></script>
<script src="http://code.jquery.com/mobile/1.0/jquery.mobile-1.0.min.js"></script>
```

Как видите, здесь всего лишь три обращения к внешним ресурсам на сервере **code.jquery.com**, который хранит минимизированные версии каждого ресурса. Вам ничего не нужно загружать — ни изображения, ни CSS-таблицы, ни код JavaScript. Не забудьте проверить, какая последняя версия доступна на сайте.

Самые свежие сборки

Тому, кто хочет, чтобы его код использовал самую свежую версию, CDN-сеть jQuery Mobile позволяет встроить в код новые ресурсы. Однако не следует забывать, что версии обновляются автоматически, так что приложение, работающее сегодня, может завтра столкнуться с проблемами. Эту опцию следует использовать для целей разработки и тестирования. Помните, что свежие версии могут работать нестабильно.

Код, который позволяет всегда работать со свежей версией, выглядит так:

```
<link href="http://code.jquery.com/mobile/latest/jquery.mobile.min.css"
      rel="stylesheet" type="text/css" />
<script src="http://code.jquery.com/jquery-1.6.4.min.js"></script>
<script src="http://code.jquery.com/mobile/latest/jquery.mobile.min.js">
</script>
```

Достоинство свежих сборок состоит в том, что вы можете обращаться к функциональным возможностям, недоступным в последней эксплуатационной сборке.

Основной шаблон для HTML5

Для создания веб-приложения jQuery Mobile вам нужно создать пустой файл в формате HTML5. Если вы никогда не создавали HTML5-документ, не волнуйтесь. Это просто и напоминает создание файла HTML 4.01.

Тег <DOCTYPE> (первая строчка документа) выглядит так:

```
<!DOCTYPE html>
```

Вторая особенность такого документа — тег <meta charset> внутри тега <head>:

```
<meta charset="utf-8" />
```

Из-за недостатка места мы не можем подробно рассмотреть нововведения в HTML 4.01, однако достаточно заметить, что эти новшества совместимы с некоторыми старыми устройствами, не поддерживающими HTML5 в целом, так что на этих устройствах jQuery Mobile тоже работает.

Если вы работаете в приложении Dreamweaver версии не ниже CS5, то можете создать пустой шаблон HTML5, выбрав **HTML5** в раскрывающемся списке **Document Type** (Тип документа) в окне **New Document** (Новый документ). Если ваш экземпляр приложения не предоставляет такую опцию, вам следует загрузить обновление версии 11.0.3 или выше с сайта **adobe.com** или воспользоваться приложением Adobe Updater.

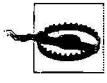
Команда разработчиков jQuery Mobile официально рекомендует включать код JavaScript и CSS-ресурсы (собственные или расположенные в сети CDN) внутрь тега <head> и добавлять тег <meta> для определения окна просмотра:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Your Title</title>
  <!-- Ваш заголовок -->
  <link rel="stylesheet"
   href="http://code.jquery.com/mobile/1.0/jquery.mobile-1.0.min.css"/>
  <script src="http://code.jquery.com/jquery-1.6.4.min.js"></script>
  <script type="text/javascript"
    src="http://code.jquery.com/mobile/1.0/jquery.mobile-1.0.min.js">
  </script>
  <meta name="viewport" content="width=device-width, initial-scale=1">
</head>
<body>
</body>
</html>
```

Готово! Мы только что создали пустой документ jQuery Mobile. Естественно, тег <body> пуст, но мы вскоре к нему вернемся.

Окно просмотра

Окно просмотра — это область, в которой находится страница. Вы можете указать ее ширину и высоту, причем она может быть больше или меньше видимой области экрана. В таком случае будут задействованы функции масштабирования в мобильном браузере. Если вы создаете сайт, дружественный мобильным устройствам, следует избегать необходимости масштабировать страницы. В оптимальном случае видимая область стартовой страницы равна по ширине экрану устройства. Тогда вы можете сообщить браузеру, что начальный масштаб должен быть 1:1 (отношение области окна просмотра к видимой области). На рис. 2.1 показано, как выглядит приложение jQuery Mobile в системе iOS при отсутствии определения окна просмотра.



В альфа-версиях jQuery Mobile окно просмотра создавалось автоматически. При переносе старых приложений, созданных с помощью альфа-версии jQuery Mobile, в новые проекты вы должны явно добавлять метатег окна просмотра в свой код.

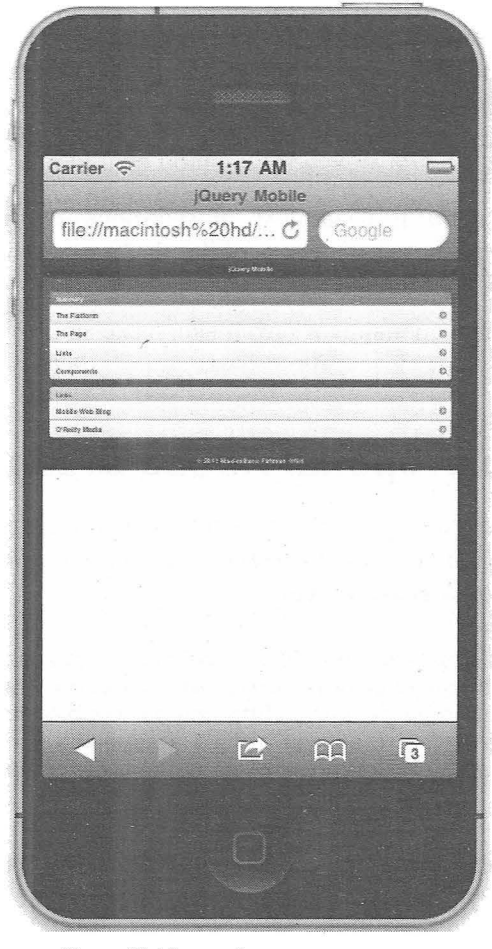


Рис. 2.1. В приложениях jQuery Mobile необходимо определять метатег окна просмотра во избежание возникновения проблем с интерфейсом при загрузке приложения

Типичный метатег окна просмотра для jQuery Mobile выглядит следующим образом:

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

Вы также можете сообщить браузеру, что не хотите, чтобы пользователь изменял масштаб страницы (нажатием на кнопки):

```
<meta name="viewport"
      content="width=device-width, initial-scale=1, user-scalable=no">
```

Производительность кода JavaScript

Прежде чем продолжить создание приложения, необходимо поговорить о производительности. В контексте оптимизации производительности веб-приложений существует хорошо известное утверждение, что вставка тегов внешних сценариев в заголовок отрицательно сказывается на производительности. И это абсолютная правда.

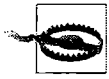
Однако с точки зрения jQuery Mobile, перенос двух сценариев (jQuery и jQuery Mobile) в конец HTML-файла может привести к нежелательному результату: в течение нескольких миллисекунд, пока не будет загружена платформа, ваше веб-приложение будет показано в виде простого HTML-файла без CSS, даже если файл CSS помещен в заголовок.

Причина заключается в так называемом прогрессивном улучшении, подходе, используемом внутри платформы jQuery Mobile. Файл CSS никак не влияет на отображение страницы без кода JavaScript.

Поэтому лучше иметь два сценария в заголовке, даже за счет небольшого снижения производительности.

Поддержка со стороны Adobe Dreamweaver

Начиная с версии CS5.5, компания Adobe официально поддерживает jQuery Mobile в своем приложении. Бесплатную пробную версию вы можете загрузить с сайта **http://www.adobe.com/go/dreamweaver**.



Первая версия Dreamweaver CS5.5 была ориентирована на jQuery Mobile версии alpha 3, а не на самый свежий выпуск. Инструкции по обновлению до последней версии вы найдете на страничке **http://mobilexweb.com/go/dwjqm**. В любом случае вы можете начать с версии alpha 3 и заменить ее вручную. Более поздние версии Dreamweaver ориентируются на последнюю стабильную версию на момент выпуска.

Поддержка со стороны Dreamweaver заключается в том, что вы можете создать страницу, использующую jQuery Mobile. Откройте приложение, выберите **File | New** (Файл | Создать), а затем **Page from Sample | Mobile Starters** (Страница из шаблона | Мобильные стартовые страницы).

У вас появится выбор из трех шаблонов (рис. 2.2):

- jQuery Mobile из сети CDN;
- jQuery Mobile с локальными файлами;
- jQuery Mobile с локальными файлами и поддержкой PhoneGap (о чем мы поговорим далее в этой книге).

Каждый шаблон начинается с документа jQuery Mobile, состоящего из четырех связанных друг с другом страниц.

Однако основное удобство работы с Dreamweaver состоит не в наличии шаблонов, а в автоматической помощи по синтаксису. Вы начинаете вводить `data-` и получаете список всех возможных значений `data-*` в jQuery Mobile. Кроме того, вы можете получить список возможных значений каждого атрибута `data-*` в jQuery Mobile (как минимум, определенных к версии alpha 3).

С помощью нового метода многоэкранного предварительного просмотра, который появился в Dreamweaver CS5.5, вы можете узнать, как приложение jQuery Mobile будет адаптироваться к различному размеру и ориентации экрана, в том числе на смартфонах и планшетах.

Кроме прочего, в меню **Insert** (Вставка) вы найдете новый пункт **jQuery Mobile**, предлагающий образцы кода для большинства компонентов пользовательского интерфейса, обсуждаемых в этой книге.

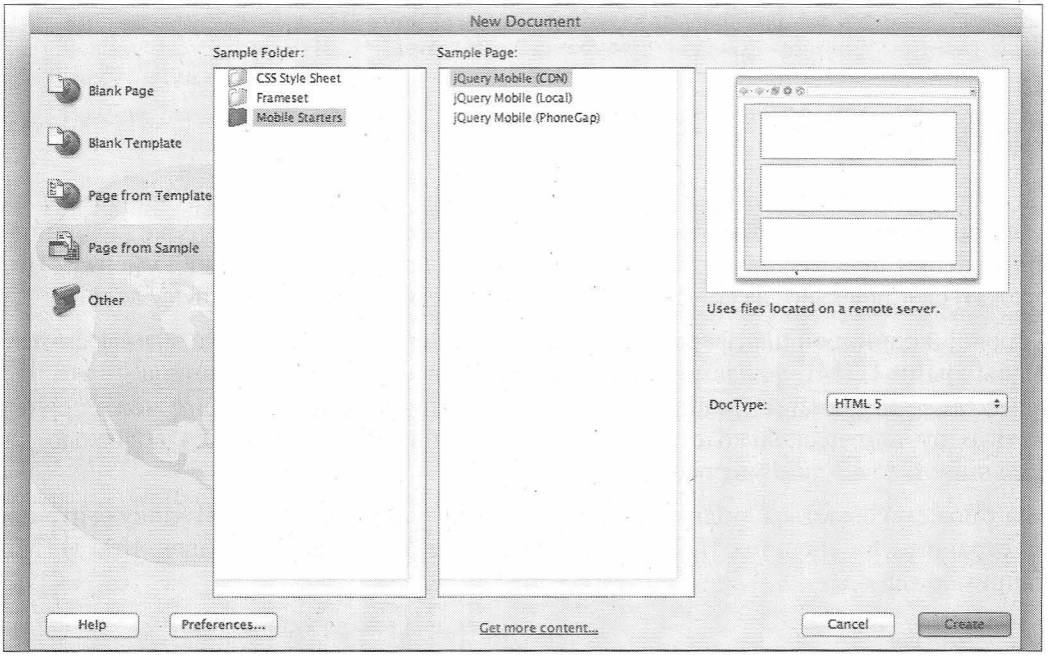


Рис. 2.2. Начиная с версии Dreamweaver CS5.5, имеется поддержка создания документов с нуля на основе шаблонов jQuery Mobile

Предварительный просмотр файлов

Чтобы увидеть работу приложения jQuery Mobile с помощью Dreamweaver, воспользуйтесь функцией Live View (рис. 2.3).

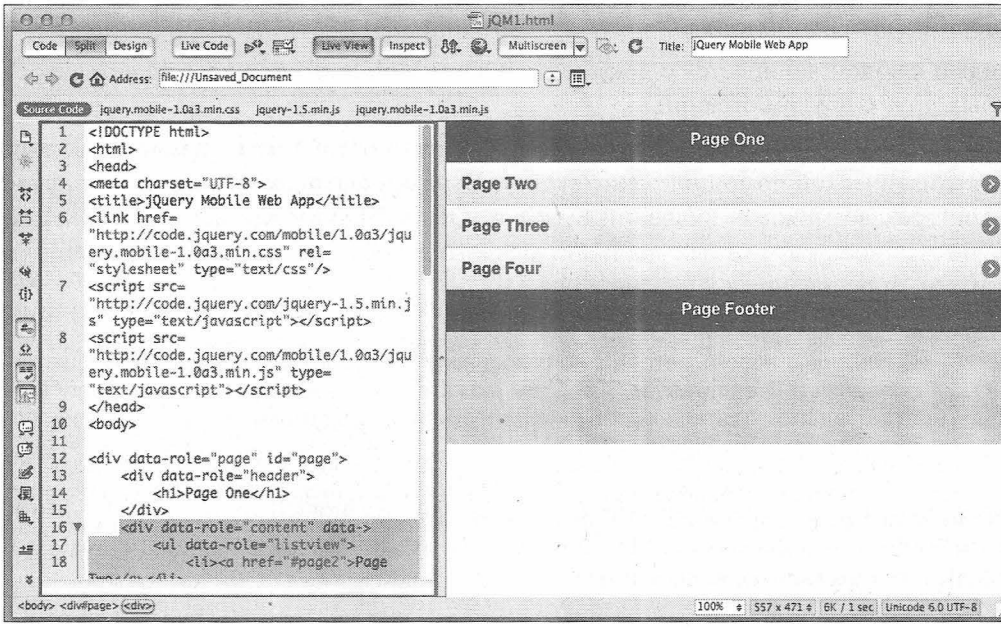


Рис. 2.3. Функция Live View позволяет увидеть работу приложения jQuery Mobile с помощью Dreamweaver

Архитектура

На платформе jQuery Mobile применяется очень простой и эффективный подход для определения содержимого веб-приложения. Вспомним, что мы уже говорили о применении ненавязчивых элементов кода. Это позволяет HTML-документам отображаться корректно, даже если возникли проблемы с загрузкой jQuery Mobile.

Основной единицей платформы является страница. Получается, что мы имеем дело с обычными HTML-файлами? Нет, и сейчас я все объясню. Страница — это элемент `<div>`, играющий специфическую роль. Один HTML-документ может содержать одну или несколько страниц в одном файле. И это является новостью для большинства веб-дизайнеров.

Мы сможем ссылаться с одной страницы на другую в одном HTML-документе или на страницы во внешних HTML-документах, используя простые элементы HTML, например тег `<a>`.

Карты

Возможность встраивать несколько *страниц* в один документ существует в программировании мобильных приложений уже более 10 лет. Морально устаревший стандарт WML (Wireless Markup Language, язык разметки для беспроводной связи) предусмат-

ривал возможность вставки нескольких визуальных страниц в один документ с целью уменьшения задержек и ускорения загрузки. В технологии jQuery Mobile та же идея реализуется с применением обычных кодов HTML и JavaScript.

В стандарте WML страницы назывались *картами*, а WML-документ — *колодой*. Для определения страницы внутри документа в WML использовался тег `<card>`, а в jQuery Mobile обычно применяется тег `<div>`, играющий специфическую роль.

Роли

На платформе jQuery Mobile используется стандартная HTML-разметка, в частности тег `<div>`. Для указания того, что платформа должна делать с тегом `<div>`, мы определяем его *роль* с помощью атрибута `data-role`. Например:

```
<div data-role="page">
```

Основные роли, доступные в jQuery Mobile 1.0, перечислены в табл. 2.1, и мы будем обсуждать их по мере изложения материала.

НЕСТАНДАРТНЫЕ АТРИБУТЫ DATA-*

Применение атрибутов `data-<некоторое значение>` или `data-*` в HTML-теге является особенностью HTML5. Такие атрибуты называются *нестандартными атрибутами данных* (согласно спецификации W3C). Эта функциональная возможность HTML5 позволяет нам определять любые атрибуты, которые мы пожелаем добавить к тегу, без нарушения корректности HTML-документа. Добавление нестандартных метаданных в теги с сохранением корректности разметки является очень полезной возможностью.

Она активно используется в технологии jQuery Mobile для определения нестандартных атрибутов платформы. Однако следует внести ясность: атрибут `data-role` не является новым атрибутом HTML5. Его применение — это неявный контракт, заключенный между нами и платформой.

Важная особенность нестандартных атрибутов состоит в том, что они работают в браузерах, не поддерживающих HTML5, и не вызывают никаких проблем.

Если вы работаете с приложением Adobe Dreamweaver версии CS5.5 и выше, оно будет вам подсказывать возможные значения атрибутов jQuery Mobile при наборе `data-` внутри HTML-элемента.

Таблица 2.1. *Основные роли, доступные в jQuery Mobile 1.0*

Роль	Описание
`page`	Определяет страницу, единицу показа содержимого в jQuery Mobile
`header`	Заголовок страницы
`content`	Содержимое страницы
`footer`	Нижний колонтитул страницы
`navbar`	Определяет строку навигации, как правило, внутри заголовка
`button`	Отображает визуальную кнопку
`controlgroup`	Отображает компонент
`collapsible`	Сворачиваемая панель с содержимым внутри страницы

Таблица 2.1 (окончание)

Роль	Описание
`collapsible-set`	Группа сворачиваемых панелей (аккордеон)
`fieldcontain`	Контейнер для полей формы
`listview`	Содержимое нескольких элементов списка
`dialog`	Страница диалога
`slider`	Визуальный бегунок для булевых значений
`nojs`	Элемент, который будет скрыт в браузерах, совместимых с jQuery Mobile

Темы

jQuery Mobile использует мощный механизм создания тем, позволяющий определять внешний вид пользовательского интерфейса. Мы коснемся вопроса смены тем и персонализации далее в этой книге, а сейчас нам важно знать, что каждый элемент пользовательского интерфейса (страница, кнопка или компонент) может иметь свой образец цвета в рамках темы.

Вплоть до версии 1.0 платформа имеет только одну тему по умолчанию. По адресу **http://jquerymobile.com/themeroller** доступно приложение Theme Roller (веб-приложение для создания тем), позволяющее нам определить собственную тему, не занимаясь непосредственным кодированием CSS-файлов.

Тема — это набор определений для компоновки, стилей и цветов. Каждая тема включает себя набор образцов цвета, который мы можем изменить в любом месте приложения. Образцы цвета создаются для того, чтобы можно было по-разному представлять элементы. Образец цвета обозначается буквой, от `a` до `z`. В тему, принимаемую по умолчанию, включены образцы от `a` до `e` с возможностью добавлять новые.

В табл. 2.2 приводятся соглашения относительно образцов цвета, изображенных на рис. 2.4.

***Таблица 2.2.** Соглашения относительно образцов цвета*

Буква	Описание	Цвет в теме, принимаемой по умолчанию
a	Наивысший уровень визуального приоритета (по умолчанию на панелях инструментов)	Черный
b	Второй уровень визуального приоритета	Синий
c	Базовый уровень (образец по умолчанию в большинстве ситуаций)	Серебристый
d	Альтернативный второй уровень	Серый
e	Выделение	Желтый

Образец цвета определяется для каждого HTML-элемента jQuery Mobile с помощью атрибута `data-theme` и буквенного обозначения, например: `data-theme="e"` обозначает цвет выделения.

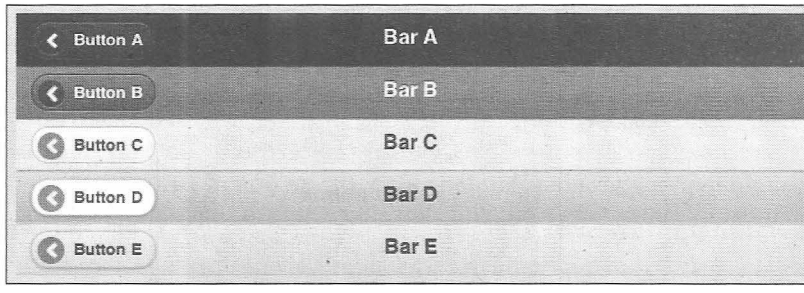


Рис. 2.4. Темы по умолчанию от a до e

Как правило, мы можем менять образцы цвета у многих элементов веб-приложения, таких как страницы, списки, кнопки, элементы, элементы форм и панели инструментов. При этом нет необходимости менять их сразу у всех одноименных элементов.

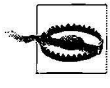
Образцы цвета подчиняются каскадному принципу. То есть, если в элементе-контейнере определен образец цвета, этот образец применяется ко всем его потомкам, пока не будет явно определен другой образец.

Страница

Мы знаем, что страница является основной единицей содержимого в jQuery Mobile. Типичная страница разделяется на три части: *заголовок (верхний колонтитул)*, *содержимое* и *нижний колонтитул*. Обязательной частью является только содержимое. Каждая часть объявляется с помощью тега `<div>` с указанием его роли:

```
<div data-role="page">
  <div data-role="header">
  </div>
  <div data-role="content">
  </div>
  <div data-role="footer">
  </div>
</div>
```

Любой элемент, т. е. страница, заголовок, нижний колонтитул и содержимое, может иметь собственный образец цвета из текущей темы.



Элемент `page-role` не является обязательным для одностраничного документа, и, если мы его не укажем, платформа добавит его сама. Однако хороший стиль предписывает определять этот элемент, чтобы код выглядел аккуратнее и был устойчив к последующим изменениям.

На рис. 2.5 представлена схема типичного документа jQuery Mobile. Помните, что эта страница должна содержать в теге <body> документа HTML5, содержащего файлы jQuery Mobile.

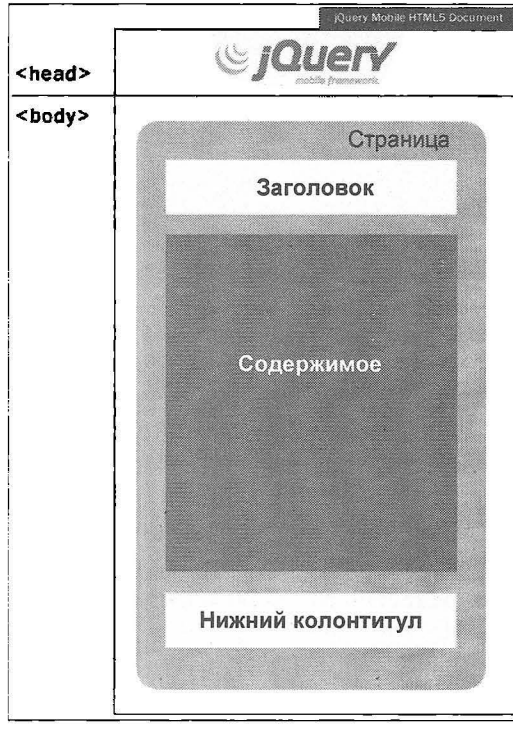


Рис. 2.5. Типичная страница документа jQuery Mobile содержит заголовок, область содержимого и необязательный нижний колонтитул

Платформа jQuery Mobile самостоятельно реагирует на смену ориентации (книжная или альбомная) на большинстве устройств, автоматически адаптируя пользовательский интерфейс к новому окну просмотра.

Если мы хотим предоставить особое содержимое браузерам, не входящим в категорию совместимости A, то можем указать роль `nojs`, например, `<div data-role="nojs">`. Это содержимое будет автоматически скрываться в браузерах категории A.

На совместимых устройствах, например на тех, которые работают под управлением Android, webOS и iOS, платформа jQuery Mobile постарается скрыть адресную строку браузера, манипулируя полосой прокрутки. Это делается, чтобы своим внешним видом и поведением приложение в большей степени походило на "родное" приложение устройства. Однако этот прием работает только в тех случаях, когда страница достаточно длинная, чтобы занять все имеющееся пространство. Если на странице мало содержимого, адресная строка браузера будет видна.

Заголовок и нижний колонтитул

В заголовок и нижний колонтитул мы можем вставить любой HTML-код. Однако стандартная таблица стилей jQuery Mobile такова, что оптимальный результат отображения пользовательского интерфейса будет достигнут с помощью тега `<h1>` в заголовке и тега `<h4>` в нижнем колонтитуле. Впоследствии мы узнаем, как настраивать пользовательский интерфейс по своему желанию.

Нижний колонтитул необязателен, а верхний обычно необходим для навигационных элементов интерфейса веб-приложения. Структура заголовка определена заранее, так что он состоит из левой части, собственно заголовка и правой части.

Левую и правую части мы обсудим позже, а пока я скажу, что они предназначены для кнопок.

Заголовок (центральная часть) будет автоматически взят из тега `<hX>`, например, из элемента `<h1>` или `<h2>`. Область для него ограничена, и он будет обрезан, если не поместится. На большинстве устройств вместо обрезанной части мы увидим многоточие (рис. 2.6). То же самое справедливо и для нижнего колонтитула.

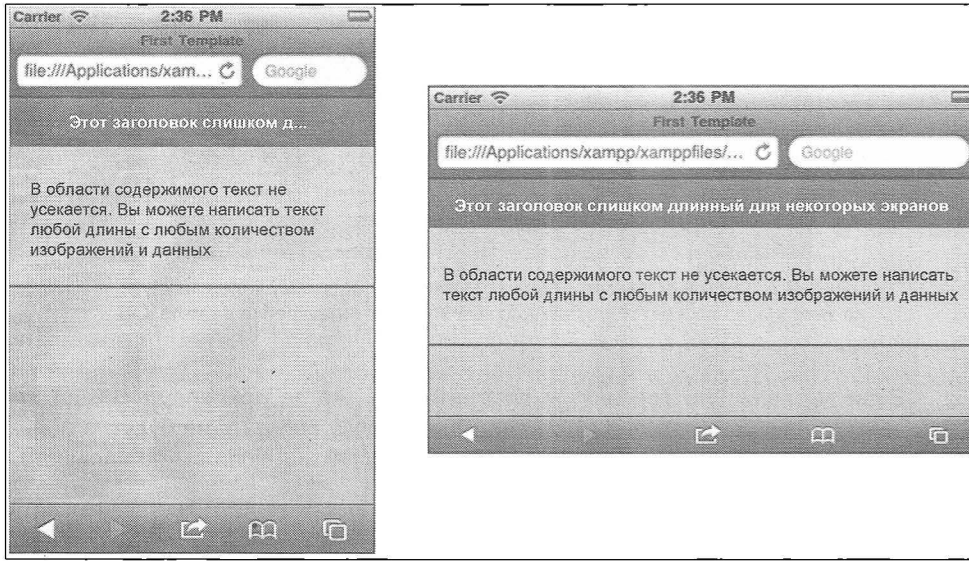


Рис. 2.6. Если заголовок не помещается, jQuery Mobile добавляет многоточие

Содержимое

В области содержимого может находиться любой HTML-код. Обычно там имеются стилизованные элементы управления, определенные платформой, — кнопки, списки или формы.

Изменим наш код в соответствии с рис. 2.7:

```
<div data-role="page">
  <div data-role="header">
    <h1>Our first webapp</h1>
```

```
    <!-- Наше первое веб-приложение -->
  </div>
  <div data-role="content">
    <p>This is the main content of the page</p>
    <!-- Основное содержимое страницы -->
  </div>
  <div data-role="footer">
    <h4>More on mobilexweb.com</h4>
    <!-- Подробности на mobilexweb.com -->
  </div>
</div>
```

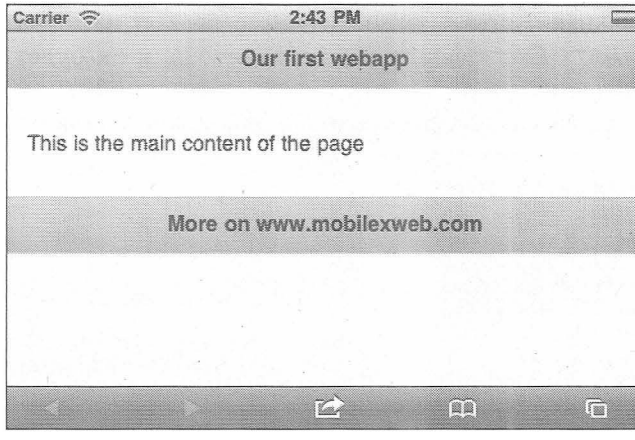
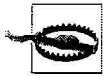


Рис. 2.7. Каждой роли (заголовку, содержимому, нижнему колонтитулу) по умолчанию соответствует аккуратный элемент интерфейса в стиле iOS



Если вы напишете код внутри элемента `page`, но за пределами секций `header`, `content` или `footer`, таблицы стилей jQuery Mobile работать не будут, а пользовательский интерфейс будет отображаться некорректно, если вы не напишете нужный код вручную.

Навигация

Для навигации между страницами используются стандартные элементы `<a>`. Платформа jQuery Mobile воспринимает их и делает всю необходимую работу.

Для начала заметим, что существует несколько типов гиперссылок:

- внутренние ссылки на другие страницы в одном (многостраничном) документе;
- внешние ссылки на страницы в других документах jQuery Mobile;
- абсолютные ссылки на документы, не имеющие отношения к jQuery Mobile;
- мобильные специальные ссылки.

Ссылки между страницами jQuery Mobile (первые два случая) вызывают специальное поведение приложения:

- на устройствах, совместимых с jQuery Mobile (например, работающих под управлением iOS или Android), воспроизводится анимация, обозначающая переход;
- при работе в браузере (а не в устанавливаемом приложении, не имеющем подсистемы Chrome) кнопка браузера **Назад** автоматически возвращает пользователя на сославшуюся страницу.

Бесспорным достоинством jQuery Mobile является умение распознать кнопку **Назад** у браузера и предоставить пользователю интуитивно понятный способ возврата на предыдущую страницу. На некоторых устройствах (с операционной системой Android или BlackBerry) присутствуют аппаратные кнопки "Назад", которые тоже выполняют навигацию на предыдущую страницу.

В многостраничных документах при первой загрузке отображается первая страница в объектной модели документа DOM.

Навигация в обратном направлении сопровождается тем же эффектом перехода, воспроизводимом также в обратном направлении.

Поведение устройства при возврате на предыдущую страницу определяется поведением пользовательского интерфейса iOS или другой мобильной операционной системы. Платформа jQuery Mobile заносит в стек каждую страницу, посещаемую пользователем, чтобы он мог в любой момент вернуться на любую просмотренную страницу.

Эта навигация целиком реализуется платформой jQuery Mobile, которая манипулирует текущим URL-адресом, добавляя хеш-значения к нему. В некоторых устройствах этот процесс приводит к странному эффекту. Если адресная строка скрыта, а пользователь переходит на другую страницу, адресная строка появляется на секунду, а затем опять исчезает. Этот эффект наблюдается только в браузерных веб-приложениях. Его можно избежать, написав полноэкранное или гибридное приложение, о которых мы будем говорить далее в этой книге.

Кнопка *Назад*

Если мы хотим добавить программную кнопку **Назад** слева от заголовка, то должны написать атрибут `data-add-back-btn="true"` на странице документа. Конкретная надпись на кнопке определяется атрибутом `data-back-btn-text`, также на странице документа, а образец цвета можно задать атрибутом `data-back-btn-theme`. Например:

```
<div data-role="page" data-add-back-btn="true"
     data-back-btn-text="Назад" data-back-btn-theme="e">
</div>
```

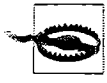
Если вы создаете браузерное приложение (а не приложение без подсистемы Chrome, устанавливаемое в устройстве), то должны знать, что у каждого браузера есть кнопка возврата, либо в виде элемента интерфейса, либо в виде аппаратной кнопки устройства. Поэтому создание специальной кнопки возврата приведет к дублированию действий и нерациональному использованию ценной площади экрана. Если же вы создаете устанавливаемое веб-приложение без Chrome, то должны обязательно предусмотреть кнопку возврата.

Внутренние ссылки

Мы уже говорили, что документ jQuery Mobile может включать в себя несколько страниц. Они должны быть оформлены как потомки элемента `<body>`. Чтобы иметь доступ к страницам, мы должны указать для каждой из них идентификационное имя (атрибут `id`), пользуясь стандартным синтаксисом HTML.

Для создания ссылки на другую страницу в пределах одного документа нам достаточно будет воспользоваться конструкцией `#id` в атрибуте `href`, где `<id>` — идентификационное имя целевой страницы. Например:

```
<a href="#next">
```



Адрес в ссылке на внешнюю страницу должен принадлежать тому же домену, в котором находится текущая страница, или тому же пакету в низкоуровневом приложении. Если вы сошлетесь на документ в другом домене, ссылка будет воспринята как абсолютная внешняя, если вы не предусмотрите кроссдоменную AJAX-загрузку в коде JavaScript.

По умолчанию платформа использует элемент `<title>` документа для вывода заголовка в браузере. Заголовок нужен в браузерных веб-приложениях, поскольку выводится в строке заголовка браузера и присутствует в закладках, устанавливаемых пользователем. Если мы хотим, чтобы заголовок менялся при переходе на ту или иную внутреннюю страницу, мы можем на каждой странице поставить необязательный атрибут `data-title`.

Рассмотрим пример (которому соответствует рис. 2.8):

```
<body>
<div data-role="page" id="page1">
  <div data-role="header">
    <h1>First page</h1>
    <!-- Первая страница -->
  </div>
  <div data-role="content">
    <p>This is the main content of the page.</p>
    <!-- Это основное содержимое страницы -->
    <p>You can go to the <a href="#page2">second page</a>.</p>
    <!-- Вы можете перейти на вторую страницу -->
  </div>
```

```
  <div data-role="footer">
    <h4>mobilexweb.com</h4>
  </div>
</div>
<div data-role="page" id="page2" data-title="This is the second page">
  <div data-role="header">
    <h1>Second page</h1>
    <!-- Вторая страница -->
  </div>
  <div data-role="content">
    <p>This is the main content of the second page</p>
    <!-- Это основное содержимое второй страницы -->
    <p>You can go back using the header's button,
       <a href="#page1">clicking here</a>
       or using your browser's back button.
    <!-- Вы можете вернуться с помощью кнопки в заголовке,
         или щелкнув здесь, или с помощью кнопки "Назад" в браузере -->
  </div>
  <div data-role="footer">
    <h4>mobilexweb.com</h4>
  </div>
</div>
</body>
```

Можем ли мы создать целое веб-приложение, написав только один HTML-документ? Если нам не требуется динамическое содержимое, то можем. Однако следует

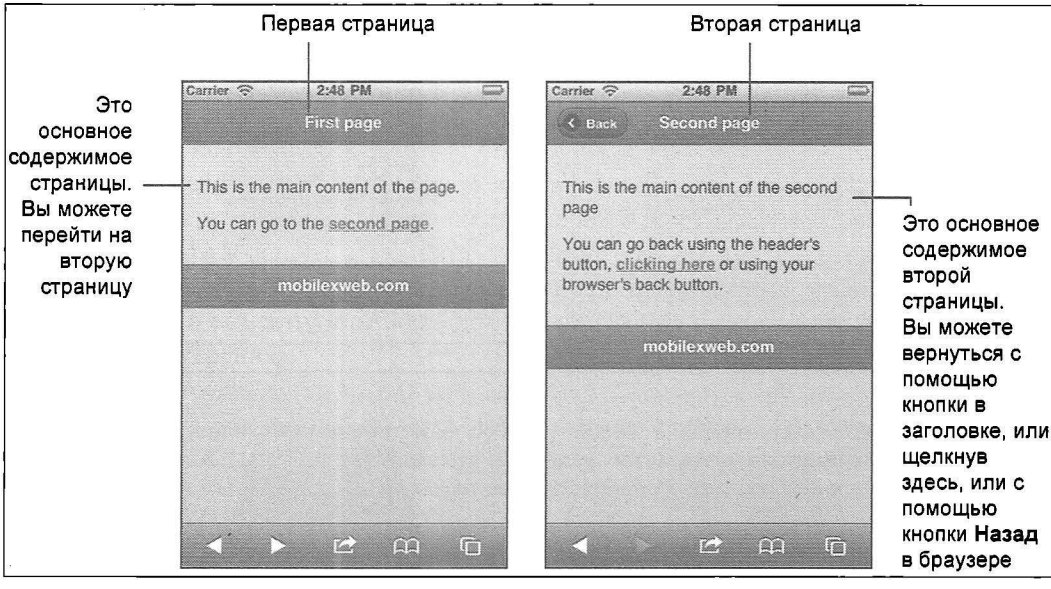


Рис. 2.8. Навигация между внутренними страницами проста и понятна благодаря стандартным тегам ссылок

помнить, что при наличии нескольких страниц в теле документа пользователю придется загрузить документ полностью, даже если он не увидит все страницы. Мы должны найти компромисс между производительностью приложения и доступностью информации и решить, сколько страниц и каких именно следует передавать в одном документе.

На рис. 2.9 показана схема навигации между внутренними страницами.

Чтобы узнать, какие страницы пользователь посетит с наибольшей вероятностью, можно собирать статистику визитов на сайте. Это позволит собрать определенные страницы в одном документе.

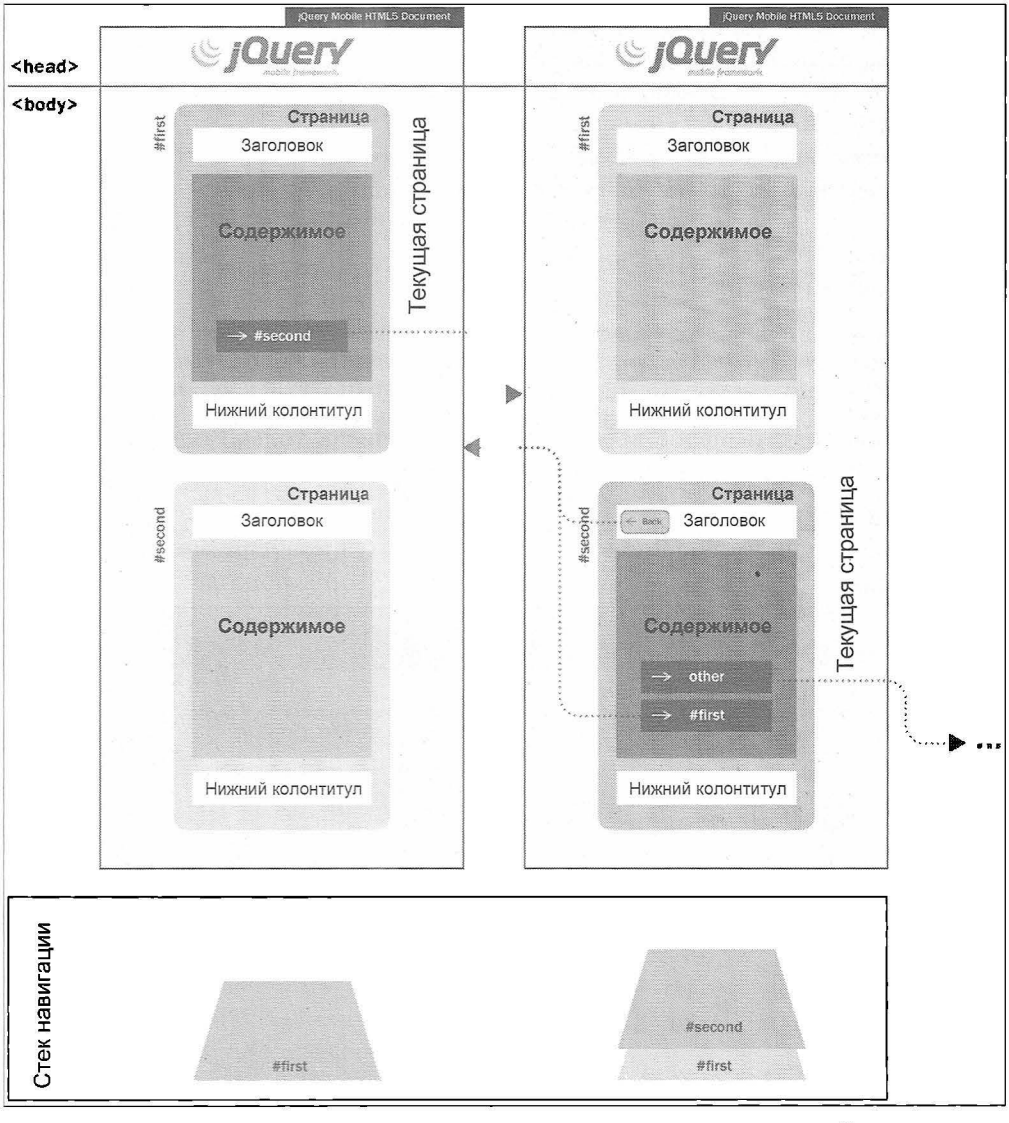


Рис. 2.9. При переходе между внутренними страницами каждая ссылка вперед добавляет запись в стек навигации, а каждое обратное действие приводит к снятию записи со стека

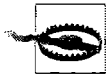
Когда платформа обнаруживает ссылку на предыдущую страницу, она автоматически конвертирует эту ссылку в ссылку обратного перехода. Поэтому анимация воспроизводится в обратном направлении, а в журнале не появляется новая запись. Если вы хотите, чтобы ссылка позволяла перейти назад, вы можете снабдить атрибутом `data-rel="back"` каждый элемент `<a>`.

Ссылки на внешние страницы

Если мы не хотим включать некоторую страницу в тот же документ, где находится первая, или собираемся создавать содержимое динамически (например, с помощью сценария PHP или серверного кода), мы можем указать ссылку на другой HTML-документ jQuery Mobile с помощью стандартных тегов:

```
<a href="otherDocument.html">На следующую страницу</a>
```

И это все? Да. Платформа автоматически распознает любую ссылку, аналогичную приведенной в предыдущем абзаце, и обеспечит практически такое же поведение приложения, как и в случае с внутренними ссылками.



Адрес в ссылке на внешнюю страницу должен принадлежать тому же домену, в котором находится текущая страница, или тому же пакету в низкоуровневом приложении. Если вы сошлетесь на документ в другом домене, ссылка будет воспринята как абсолютная внешняя.

Единственное отличие состоит в том, что платформа воспользуется технологией AJAX, чтобы запросить другой документ, выполнит разбор его содержимого, добавит страницу в текущую модель DOM и выполнит аккуратный переход на новую страницу. В то время, когда происходит запрос, пользователь видит окно с анимацией загрузки (рис. 2.10). Не забудьте обновить заголовок в браузере с помощью атрибута `data-title` после загрузки новой страницы.

Задумаемся на секунду над тем, что происходит. Когда платформа jQuery Mobile обнаруживает ссылку на страницу, она вначале распознает, является ли ссылка внутренней (начинается с символа `#`) или внешней. Если это внешняя ссылка без атрибута `rel` или `target` (как в нашем примере), jQuery Mobile перехватывает щелчок, создает AJAX-запрос документа, указанного в `href`, и выводит модальное окно с индикатором загрузки. Когда запрос выполнен, платформа добавляет загруженную страницу в объектную модель документа (DOM) и переходит на страницу так, словно она внутренняя.

В браузерах, не совместимых с jQuery Mobile, внешние ссылки будут работать как обычные, не вызывая никаких проблем, потому что определены стандартными тегами `<a>`. В этом заключается мощь прогрессивного улучшения.

Платформа jQuery Mobile автоматически добавляет загруженную внешнюю страницу в объектную модель документа, снабжая страницу новым идентификационным именем (определенным как относительный URL-адрес из оригинального до-

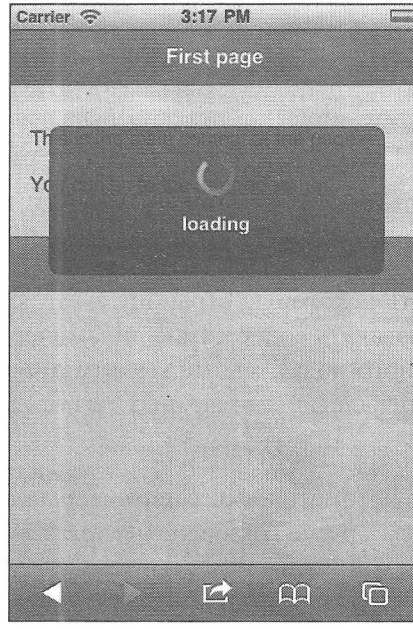


Рис. 2.10. При переходе на внешние страницы jQuery Mobile платформа автоматически выводит анимацию, пока загружается запрошенная страница (если загрузка занимает много времени)

кумента). В результате страница окажется "под рукой", если пользователь захочет снова посетить ее, пока документ jQuery Mobile остается загруженным в браузер. Эти действия проиллюстрированы на рис. 2.11.

URL-адрес и навигация jQuery Mobile

Для перехода между страницами на некоторых устройствах платформа jQuery Mobile использует хеш-навигацию (`#id`). Поэтому якорная навигация HTML не будет действовать внутри документа jQuery Mobile для перехода в пределах одной страницы.

Для эмулирования этой функциональности вы должны написать код на языке JavaScript. На устройствах, несовместимых с jQuery Mobile, внутренние страницы будут выводиться одновременно и внутренние ссылки будут работать как обычные ссылки якорной навигации HTML.

Существует несколько мобильных браузеров, поддерживающих History API (один из новых API-интерфейсов стандарта HTML5), что позволяет сайту изменять URL-адрес целиком без полной перезагрузки страницы. Платформа jQuery Mobile использует этот API-интерфейс на совместимых с ней браузерах, так что вместо выполнения хеш-навигации `#id` для внешних ссылок будет выведен целевой URL-адрес в адресной строке браузера, пока содержимое загружается в результате AJAX-запроса от первого документа.

Запрос AJAX может закончиться неудачно. На то могут быть самые разные причины: ошибка на сервере, неправильный URL-адрес, сбой связи или проблемы с выходом в сеть у устройства. Если целевую страницу загрузить не удается, пользователь получит сообщение об ошибке (рис. 2.12). Далее в этой книге мы обсудим, как можно менять это сообщение.

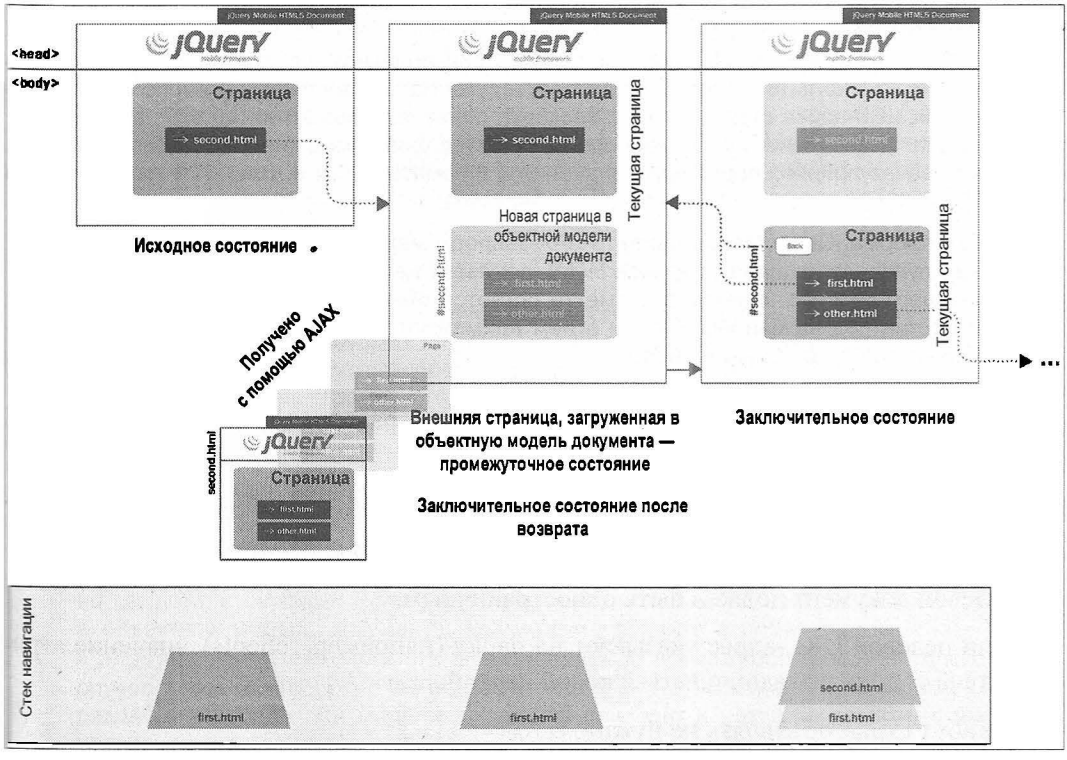


Рис. 2.11. Платформа добавляет в объектную модель документа страницу, загруженную в результате AJAX-запроса по внешней ссылке, чтобы использовать ее в будущем

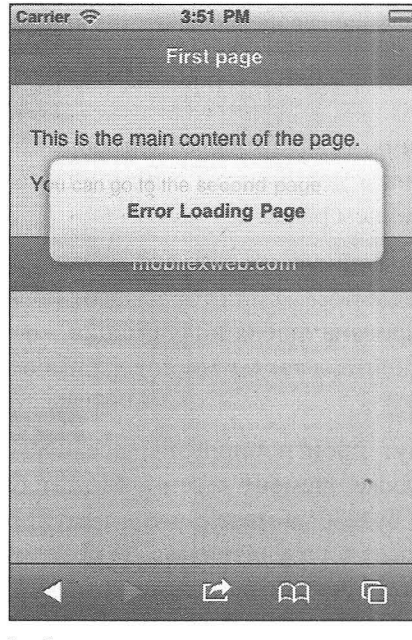


Рис. 2.12. Сообщение, появляющееся, когда целевая страница не может быть загружена по внешней ссылке с jQuery-страницы

AJAX или Hijax?

AJAX (Asynchronous JavaScript and XML, Асинхронный JavaScript и XML) — это клиентская технология, позволяющая браузеру делать "закулисный" запрос серверу, не меняя URL-адрес страницы и не блокируя пользовательский интерфейс во время загрузки. Изначально в AJAX-запросах в качестве формата данных использовался XML, но сейчас принято передавать значения в формате JSON и даже обычный текст или HTML-код.

Платформа jQuery Mobile делает AJAX-запросы между переходами с одной страницы на другую, запрашивая целевой HTML-документ целиком в текстовом виде для дальнейшего разбора, причем в разметке остается обычная HTML-ссылка. Такой подход известен под названием Hijax, и в нем применяются принципы прогрессивного улучшения для эффективной работы.

Чтобы без проблем создавать внешние ссылки на страницы, мы должны соблюдать следующие правила:

- целевой документ тоже должен быть документом jQuery Mobile;
- целевой документ должен находиться в том же домене;
- целевой документ должен быть одностраничным;
- если целевой URL-адрес указывает на папку (например, /clients), значение атрибута `href` должно заканчиваться косой чертой: `href="/clients/"`;
- атрибут `target` объявлять не нужно, `target="_blank"`.

Типичный жизненный цикл веб-приложения jQuery Mobile состоит из одного полного HTTP-запроса (первый документ), многократных загрузок внутренних страниц и многократных AJAX-запросов на внешние страницы. Другие полные HTTP-запросы не выполняются, пока не будет обнаружена абсолютная внешняя ссылка или пока не будет открыт браузер категории B или C.

Чтобы избежать внутренних конфликтов с платформой, мы должны обращаться к внешним многостраничным документам с помощью абсолютных внешних ссылок, обсуждаемых в следующем разделе.

Платформа берет из загруженного документа только первую страницу (первый элемент `<div>` с атрибутом `role="page"`) и отбрасывает прочее содержимое. Это означает, что будет проигнорирована любая информация, которую вы поместите в элемент `<head>` целевого документа, а также все содержимое за пределами первой страницы.

Отсюда следует, что будут проигнорированы все CSS-таблицы и JavaScript-код в других документах, а также элемент `<title>`. Может быть, нам следует удалить их? Ни в коем случае. Не будем забывать, что в платформе jQuery Mobile используются принципы прогрессивного улучшения, и поэтому игнорируемое содержимое может пригодиться устройствам, не совместимым с jQuery Mobile, когда они будут загружать документ в браузер целиком. В дальнейшем мы обсудим, как обойти это ограничение.

Предварительная загрузка и кэширование страниц

Иногда мы отказываемся от использования внешних страниц, потому что хотим вывести на экран следующую страницу, как только она будет затребована. Чтобы сократить время выполнения AJAX-запросов, мы можем загрузить страницу предварительно. У нас есть возможность сообщить платформе jQuery Mobile, по каким ссылкам должна произойти предварительная загрузка, чтобы соответствующие страницы были доступны в объектной модели документа. Для этого достаточно добавить в любую ссылку булев атрибут `data-prefetch`, например:

```
<a href='newpage.html' data-prefetch>На следующую страницу</a>
```

Помните, что использовать эту возможность следует только в отношении страниц, вероятность просмотра которых велика. Типичными примерами являются страницы многостраничных статей фотогалереи (в которых вы предварительно загружаете следующую картинку) или страницы, о популярности которых свидетельствует статистика. Эта функция увеличит HTTP-трафик и расходы пользователя, так что применять ее следует осторожно.

Чтобы объектная модель документа не разрасталась в памяти, jQuery Mobile автоматически удаляет из нее загруженную внешнюю страницу, когда она исчезает с экрана (в результате перехода назад или вперед к новой странице). Если эта страница понадобится снова, платформа попытается получить ее из кэша, а в случае неудачи заново загрузит с сервера. Если мы хотим принудительно оставить внешнюю страницу в объектной модели документа, мы можем добавить атрибут `datadom-cache="true"` в элемент `page`. Применять эту функциональную возможность следует только в том случае, когда мы уверены, что пользователь вернется на страницу, поскольку сильное разрастание объектной модели документа приводит к нехватке памяти и снижению производительности.

Абсолютные внешние ссылки

Иногда приходится делать ссылку на сайт или документ, не охваченный технологией jQuery Mobile. В этом случае необходимо явно определить абсолютную внешнюю ссылку, добавив атрибут `data-rel="external"` в тег `<a>`. Например:

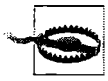
```
<a href="http://www.mobilexweb.com" data-rel="external">Check my blog</a>
<!-- Посетите мой блог -->
```

Существуют и другие ситуации, в которых ссылка интерпретируется как абсолютная внешняя. Например, это происходит при определении атрибута `target` или при ссылке на документ в другом домене, независимо от того, является ли он документом jQuery Mobile или нет:

```
<a href="http://www.mobilexweb.com" target="_blank">Check my blog</a>
<!-- Посетите мой блог -->
<a href="http://www.otherdomain.com/whatever">Check my blog</a>
<!-- Посетите мой блог -->
```

Еще один способ сделать ссылку абсолютной внешней состоит в добавлении к ней атрибута `data-ajax=false` (этот прием полезен для страниц, находящихся в том же домене):

```
<a href="otherpage.html" data-ajax="false">Other page</a>
<!-- Другая страница -->
```



Когда пользователь щелкает (или постукивает пальцем) по абсолютной ссылке, экземпляр приложения jQuery Mobile выгружается, и браузер переходит на указанную страницу. Если мы хотим, чтобы веб-приложение оставалось открытым при переходе по ссылке, мы должны указать в гиперссылке атрибут `target="_blank"`. Этот прием сработает на большинстве смартфонов и планшетов с поддержкой многостраничной навигации.

Если вы создаете веб-приложение в собственных кодах устройства, то должны с особой осторожностью использовать абсолютные ссылки. По умолчанию целевая страница будет загружена в контейнер веб-приложения (с учетом ограничений вашего приложения), а это не всегда то, что нужно. Например, если ваше низкоуровневое приложение не имеет кнопок навигации, в частности кнопки **Назад**, то пользователь не сможет вернуться в него. В некоторых системах, например в PhoneGap, страница будет открыта в браузере, запускаемом по умолчанию, а ваше приложение будет закрыто или свернуто.

Если вы указываете в документе jQuery Mobile абсолютный внешний URL-адрес, ссылка отработает как обычно, но для целевой страницы не будет кнопки **Назад**. Целевая страница откроет новый экземпляр приложения jQuery Mobile.

В последующих главах мы обсудим разработку низкоуровневых приложений с помощью jQuery Mobile, а пока я предлагаю вам самостоятельно подумать над этой задачей.

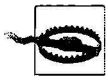
Мобильные специальные ссылки

Не будем забывать, что мы создаем приложения для мобильной Всемирной паутины. Мы должны обогатить пользовательский опыт, интегрируя наше приложение с устройством, где только возможно. С помощью URI-схем мы можем совершать звонки и отправлять SMS-сообщения. Более подробно мы обсудим эту возможность в *разд. "Интеграция с телефоном" далее в этой главе*.

Помните о необходимости во всем придерживаться принципов прогрессивного улучшения на вашем сайте. Это означает, что вы всегда должны использовать стандартные элементы-ссылки `<a>` вместо выполнения навигации в коде JavaScript. Навигация с помощью стандартных элементов HTML позволит вашему приложению работать при любом сценарии поведения мобильного браузера.

Переход между страницами

Как было сказано ранее, если пользователь переходит с одной страницы на другую, jQuery Mobile воспроизводит анимацию плавного перехода. По умолчанию во всех переходах применяется переход "справа налево". Такой принцип хорош для навигации от общего к частному, когда пользователь хочет узнать подробности. Когда вы углубляетесь в содержимое, то видите анимационный переход справа налево, а при возвращении на предыдущие страницы — переход слева направо.



При переходах используется технология CSS3, и на большинстве устройств анимация ускоряется аппаратно. На некоторых несовместимых устройствах анимация при переходе между страницами не выполняется.

Платформа jQuery Mobile позволяет изменять или явно определять анимацию перехода для каждой смены страницы. С этой целью нам следует использовать атрибут `data-transition` внутри ссылки. Для возврата на предыдущую страницу у каждого перехода есть вариант, при котором анимация движется в обратном направлении.

Доступны следующие переходы (рис. 2.13):

- `slide` — анимация "справа налево", выполняемая по умолчанию;
- `slideup` — анимация "снизу вверх", применяемая, в основном, для мобильных страниц;
- `slidedown` — анимация "сверху вниз";
- `pop` — новая страница появляется в виде точки в центре и разрастается на весь экран;
- `fade` — старая страница постепенно исчезает, в то время как "проявляется" новая;
- `flip` — двухмерное или трехмерное "переворачивание" страницы. Трехмерное переворачивание доступно только на некоторых устройствах, например, в системе iOS. На других устройствах, например, на работающих под управлением Android, визуально происходит двухмерное переворачивание, что вас, скорее всего, не устроит.

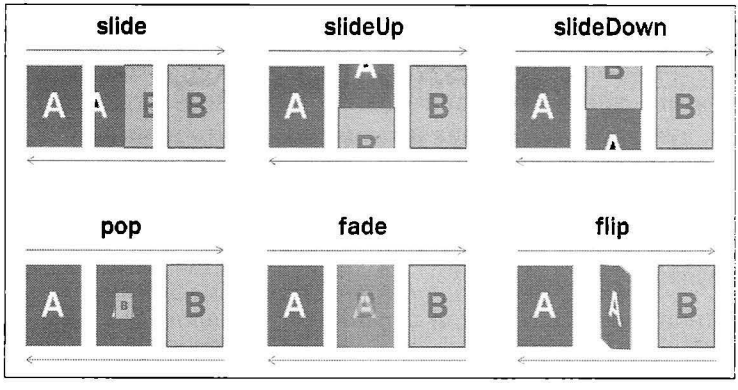
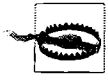


Рис. 2.13. Анимированные переходы между страницами, доступные в jQuery Mobile 1.0

Большинство эффектов перехода для jQuery Mobile 1.0 было разработано командой jQTouch (**http://jqtouch.com**), собранной в свое время Дэвидом Канедой (David Kaneda) и возглавляемой сейчас Джонатаном Старком (Jonathan Stark). В версии 1.1 существующие эффекты доработаны и добавлены два новых — `flow` и `turn`.

Для определения нового перехода в ссылке мы применяем атрибут `data-transition="`*название*`"`, например:

```
<a href="#page2" data-transition="pop">second page</a>
```



Анимированные переходы работают только в браузерах категории A во внутренних и внешних ссылках на страницы jQuery Mobile. Анимация не работает при абсолютных ссылках или в особых схемах URI, например, при отправке SMS-сообщений.

Я рекомендую использовать классические переходы между страницами и создавать стилистически непротиворечивое поведение пользовательского интерфейса. Например, при углублении в содержимое или переходе вниз по иерархической структуре всегда применяйте переход `slide`. При побочных действиях, таких как получение справки, настройка параметров или добавление новых элементов, используйте любой другой вид перехода.

Помните, что переходы являются атрибутами ссылок, а не страниц. Это означает, что переход на другую страницу может сопровождаться разными анимационными эффектами, если на нее ведут разные ссылки. Старайтесь придерживаться единого стиля в своем приложении в таких ситуациях.

Обратные переходы

Если вы хотите, чтобы анимированный переход воспроизводился в обратном направлении (при возврате на страницу), вы можете указать в соответствующей ссылке атрибут `data-direction="reverse"`. Платформа jQuery Mobile автоматически обеспечит воспроизведение анимации в обратном направлении.

Диалоговые страницы

Диалог — это всего лишь еще один способ отобразить страницу в приложении. Следовательно, диалоговая страница не является чем-то новым. Это просто страница с особой семантикой.

Диалоговая страница предназначена для отображения модальных сообщений, списков или информации, не имеющей иерархических отношений со страницей, которая на нее сослалась.

Основные различия между обычной страницей и диалоговой (рис. 2.14) заключаются в следующем:

- на диалоговой странице присутствует кнопка закрытия (в виде косого крестика) в левом верхнем углу, где у обычной страницы имеется кнопка возврата, если определен атрибут `data-add-back-btn`;
- диалоговая страница имеет рамку, поясняющую пользователю, что перед ним не полноэкранная страница, а всплывающее окно поверх исходной страницы;
- диалоговая страница не заносится в стек навигации как новая страница. (Как видно из рис. 2.15, если вы открываете новую страницу из диалоговой, она

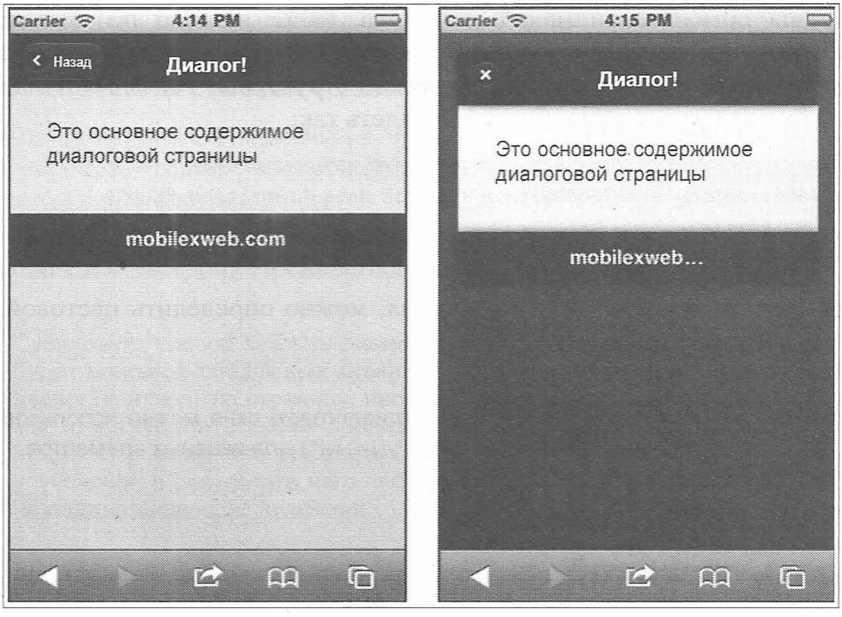


Рис. 2.14. Одна и та же страница, открытая как обычная страница и как диалоговая

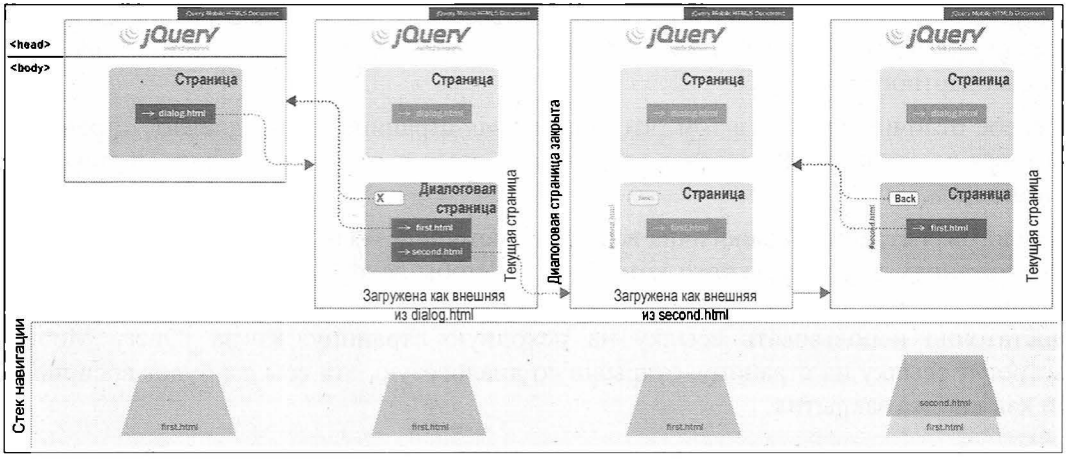


Рис. 2.15. Диалоговая страница не создает запись в стеке навигации

откроется так, словно никакой диалоговой страницы нет. То же самое происходит при внешней загрузке.)

Чтобы открыть диалоговую страницу, мы должны поставить атрибут `data-rel="dialog"` в теге `<a>`, содержащем ссылку. Здесь `rel` означает отношение между текущей страницей и той, на которую указывает ссылка (в данном случае — отношение `dialog`). Например:

```
<!-- Это ссылка на внешнюю страницу, отображаемую как диалоговая -->
<a href="./confirmation.html" data-rel="dialog">Delete this item</a>
<!-- Удалить этот элемент -->
```

Рекомендуется поменять анимацию перехода, используемую по умолчанию, на какую-нибудь другую, чтобы не возникла путаница со стандартным переходом между страницами в рамках их иерархической структуры. Тогда типичная ссылка, открывающая диалоговое окно, будет выглядеть так:

```
<!-- Это ссылка на внешнюю страницу, открываемую как диалоговая -->
<a href="./confirmation.html" data-rel="dialog" data-transition="pop">
  Delete this item</a>
  <!-- Удалить этот элемент-->
```

Когда страница открывается как диалоговая, можно определить цветовой образец перекрытия, добавив атрибут `dataoverlay-theme`.

Вместо атрибута `data-role="page"` в диалоговом окне можно использовать атрибут `datarole="dialog"` (а не `data-rel="dialog"`) для якорных элементов.

Закрывать страницу или возвращаться к предыдущей?

Как видно из рис. 2.14, со страниц, открытых как диалоговые, невозможно вернуться назад; их можно только закрыть. Это означает, что при использовании диалоговых страниц процесс навигации выполняется платформой с некоторыми отличиями от стандартного.

Первое отличие состоит в том, что диалоговая страница "принадлежит" странице, которая ее открыла. Диалоговая страница не создает отдельную запись в стеке навигации.

Диалоговая страница аналогична всплывающему или модальному окну в типичных приложениях для настольного компьютера. Чтобы закрыть диалоговое окно с помощью ссылки, кнопки или иного элемента пользовательского интерфейса, нам достаточно использовать ссылку на исходную страницу. Когда jQuery Mobile встретит ссылку на страницу, открывшую диалоговую, эта ссылка будет воспринята как кнопка закрытия.

В следующих главах мы обсудим, как настраивать визуальный дизайн страниц, диалоговых страниц и компонентов с применением тем и CSS-таблиц.

Предположим, у нас реализовано удаление элемента на странице delete.html. Соответствующая ссылка откроет диалоговое окно (confirm.html), позволяющее пользователю подтвердить операцию удаления. В диалоговом окне должны присутствовать два элемента: на удаление и на отмену.

Диалоговые страницы не добавляют записи в стек навигации. Поэтому, если пользователь перезагрузит страницу при открытой диалоговой странице, он получит исходную страницу, а диалоговая закроется.

Ссылка на отмену удаления должна закрывать диалоговую страницу, как и кнопка закрытия в левом верхнем углу. То есть эта ссылка должна находиться в обычном теге <a> и указывать на страницу confirm.html.

Чтобы этот пример был интереснее, я забегу вперед на несколько глав. С помощью атрибута data-role="button" мы можем любую ссылку преобразовать из обычной подчеркнутой гипертекстовой ссылки в стандартную кнопку во всю ширину экрана с высотой, удобной для нажатия пальцем. В последующих главах мы еще поговорим о кнопках и о настройке их внешнего вида.

Существует способ закрыть диалоговое окно с помощью кода JavaScript, который будет описан в следующих главах. Однако применение стандартных ссылок, ведущих на исходную страницу, является более предпочтительным подходом, если мы хотим охватить своим приложением и устройства, не совместимые с jQuery Mobile. Вспомним, что эта платформа придерживается принципов прогрессивного улучшения, в результате чего веб-приложения способны работать и в браузерах, не поддерживающих JavaScript.

Рассмотрим пример, изображенный на рис. 2.16. Код страницы delete.html выглядит:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Up and Running with jQuery Mobile</title>
  <!-- jQuery Mobile: разработка приложений для смартфонов и планшетов-->
  <link rel="stylesheet" href="jquery.mobile-1.0.min.css" />
  <script src="jquery-1.6.4.min.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.0.min.js">
  </script>
  <meta name="viewport" content="width=device-width, initial-scale=1">
</head>
<body>
  <div data-role="page" id="main">
    <div data-role="header">
      <h1>Book Properties</h1>
      <!-- Выходные данные книги -->
    </div>
    <div data-role="content">
      <h2>Up and Running with jQuery Mobile</h2>
      <!-- jQuery Mobile: разработка приложений для смартфонов
           и планшетов -->
      <p>Author: <b>Maximiliano Firtman</b></p>
      <!-- Автор: Максимилиано Фиртман -->
      <p>This book has no reviews yet on our database.</p>
      <!-- На эту книгу еще нет отзывов в нашей базе данных -->
      <p>
```

```
        <a href="./confirmation.html" data-rel="dialog"
          data-transition="pop" data-role="button">
          Delete this book
          <!-- Удалить эту книгу -->
        </a>
        <a href="" data-role="button">
          Modify this book
          <!-- Изменить эту запись -->
        </a>
      </p>
      </div>
      <div data-role="footer">
        <h4>O'Reilly Store</h4>
        <!-- Магазин O'Reilly -->
      </div>
    </div>
  </body>
</html>
```

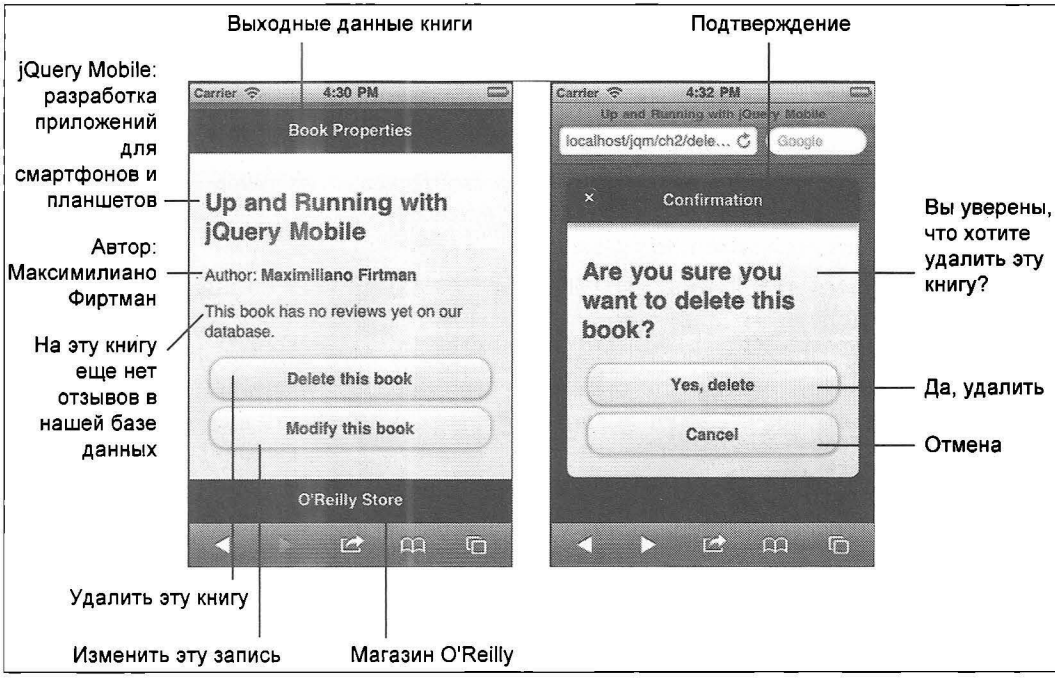


Рис. 2.16. Пример с открытием диалоговой страницы, запущенный на iPhone

Код страницы confirmation.html:

```
<!DOCTYPE html>
<html>
  <head>
```

```
    <meta charset="utf-8" />
    <title>Delete Confirmation</title>
    <!-- Подтверждение операции удаления -->
    <link rel="stylesheet" href="jquery.mobile-1.0.min.css" />
    <script src="jquery-1.6.4.min.js"></script>
    <script type="text/javascript" src="jquery.mobile-1.0.min.js">
    </script>
    <meta name="viewport" content="width=device-width, initial-scale=1">
  </head>
  <body>
    <div data-role="page" id="alert">
      <div data-role="header">
        <h1>Confirmation</h1>
        <!-- Подтверждение -->
      </div>
      <div data-role="content">
        <h2>Are you sure you want to delete this book?</h2>
        <!-- Вы уверены, что хотите удалить эту книгу? -->
        <!-- Это обычная загрузка страницы -->
        <a href="delete-yes.html" data-role="button">Yes, delete</a>
        <!-- Да, удалить -->
        <!-- Это просто закрытие страницы -->
        <a href="delete.html" data-role="button">Cancel</a>
        <!-- Отмена -->
      </div>
    </div>
  </body>
</html>
```

Если вы хотите, чтобы на кнопке закрытия был другой текст, воспользуйтесь атрибутом `data-close-btn-text`.

Открытие страниц из диалоговых страниц

Диалоговые страницы могут содержать обычные ссылки на другие страницы (или абсолютные внешние ссылки). Когда пользователь выбирает ссылку на страницу, отличную от той, которая открыла диалоговую (что мы видели в предыдущем разделе), платформа jQuery Mobile закрывает диалоговое окно, переходит на страницу, открывшую его, и затем открывает новую страницу так, словно ссылка на нее присутствует в исходной странице.

В нашем примере щелчок по кнопке **Yes, delete** (Да, удалить) приведет к закрытию диалоговой страницы и выполнению стандартного перехода на страницу delete-yes.html.

На рис. 2.15 изображена схема того, что происходит в этой ситуации.

Интеграция с телефоном

Повторяйте за мной: "Мы разрабатываем страницу для мобильного устройства, мобильного устройства". Что это означает? Мы должны выполнять интеграцию с телефоном везде, где только возможно.

Один из способов достижения этой цели заключается в использовании URI-схем, различные протоколы которых можно указывать в атрибуте `href` тега `<a>`. Я не сомневаюсь, что вам знаком протокол `mailto:`. В мобильных браузерах допустимы и другие протоколы. Некоторые из них совместимы с большинством устройств, а другие являются платформно-зависимыми (например, работают только в iOS). Последние можно использовать, когда мы создаем низкоуровневое приложение и знаем, на какой платформе оно будет работать.

Телефонный звонок

Помните: большинство мобильных устройств — это телефоны! Что мешает нам предусматривать ссылки, позволяющие позвонить? Если вы создаете деловой справочник, хотя бы и для одной-единственной модели телефона, учтите, что большинство людей предпочтет звонок и разговор с живым человеком заполнению формы на мобильном устройстве. На рис. 2.17 показано, как это может выглядеть на двух разных устройствах.

Рис. 2.17. Окно для звонка, открывающееся при активизации телефонной ссылки в системах webOS (Palm) и Android

Лучший метод (взятый из японских стандартов i-Mode) состоит в использовании схемы `tel:<`*`номер телефона`*`>`:

```
<a href="tel:+1800229933">Call us free!</a>
<!-- Позвоните нам бесплатно! -->
```

Эта URI-схема была предложена в качестве стандарта RFC 5341 (**http://tools.ietf.org/html/rfc5341**), однако при чтении этой спецификации следует быть

внимательным, поскольку большинство из предложенных параметров допустимо не для каждого устройства.

Я рекомендую указывать номер телефона в международном формате: знак "плюс" (+), код страны, код местности, местный номер телефона. Мы не знаем, где находятся посетители нашей странички. Если они сейчас в той же стране или даже в той же местности, что и мы, международный формат тоже сработает.

Если пользователь выберет ссылку на звонок, он увидит окно подтверждения звонка с полным номером и сможет принять окончательное решение. Это позволит избежать звонков на телефоны недобросовестных рекламодателей, расположенных в другой стране или требующих повышенной оплаты разговора.

На некоторых мобильных устройствах (например, iPod Touch или iPad) голосовые звонки невозможны. На их экране появляется предложение добавить номер в записную книжку (рис. 2.18).

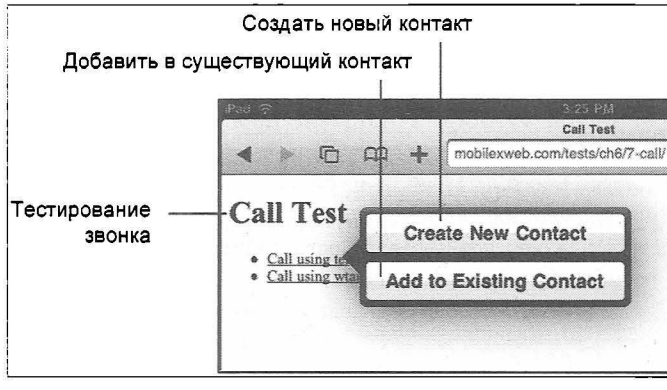


Рис. 2.18. На некоторых устройствах, например Apple iPad, нет функции телефона, и они предлагают пользователю другие действия

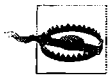
Видеозвонки и звонки VoIP

Устройства с камерами, работающие под управлением iOS (например, iPhone 4 и iPod Touch 4G), имеют приложение для видеочата, называемое FaceTime. Если вы разрабатываете приложения для этих устройств, то можете создать ссылку на видеозвонок с помощью атрибута `facetime://<имя_пользователя-или-телефон>`. На других устройствах такие ссылки приводят к ошибкам:

```
<a href="facetime:101010">Call me using Facetime</a>
<!-- Позвоните мне по Facetime -->
```

Система Skype тоже имеет свою URI-схему для ссылок. В ссылке необходимо указать имя пользователя Skype. Существует необязательный параметр `?call`, позволяющий инициировать звонок немедленно. Без него мы увидим на экране пользовательский профиль:

```
<a href="skype:skype_user?call">Call us using Skype</a>
<!-- Позвоните мне по Skype -->
```



Если на устройстве не установлено низкоуровневое приложение, на которое мы ссылаемся, такое как FaceTime или Skype, пользователь получит сообщение об ошибке. Такие ссылки лучше не показывать на устройствах, не совместимых с jQuery Mobile. Вопросы написания веб-приложений под конкретные устройства и соответствующей адаптации содержимого обсуждаются в книге "Programming the Mobile Web".

Самая свежая информация о схемах URI для мобильных браузеров находится по адресу **http://www.mobilexweb.com/go/uri**.

Отправка сообщения по электронной почте

В некоторых современных устройствах с браузерами имеются также почтовые приложения, способные работать по классическому протоколу `mailto:`. Синтаксис обращения к нему таков: *?параметры*. Список распознаваемых параметров может меняться от устройства к устройству, но он обычно включает в себя `cc`, `bcc`, `subject` и `body`. Параметры определяются в формате URL (*ключ=значение&ключ=значение*), причем значения должны быть кодированы по URI.

Вот некоторые примеры:

```
<a href="mailto:info@mobilexweb.com">Mail us</a>
<!-- Напишите нам по электронной почте -->
<a href="mailto:info@mobilexweb.com?subject=Contact%20from%20mobile">Mail us</a>
<!-- Напишите нам по электронной почте -->
<a href="mailto:info@mobilexweb.com?subject=Contact&body=This%20is%20the%20body">
  Mail us</a>
<!-- Напишите нам по электронной почте -->
```

Вы должны отдавать себе отчет в том, что механизм `mailto:` не гарантирует отправку сообщения. Обычно он всего лишь открывает почтовое приложение, а пользователь должен подтвердить отправку, возможно, внеся некоторые изменения. Если вам нужна фактическая отправка почтового сообщения, воспользуйтесь серверным механизмом.

Вообще говоря, если мы хотим добавить новую строчку в электронное письмо, то можем сделать это с помощью символов возврата каретки и перевода строки (%0D%0A). В настоящее время этот прием не работает в приложении Mail в системе iOS, но мы можем поставить HTML-теги в тело сообщения, в частности тег `<br />` для браузера Mobile Safari:

```
<a href="mailto:info@mobilexweb.com?subject=Contact&body=This%20is%20the%20
body%0D%0AThis%20is%20a%20new%20line">Mail us</a>
<!-- Напишите нам по электронной почте -->

<a href="mailto:info@mobilexweb.com?subject=Contact&body=This%20is%20the%20body<br />
This%20is%20a%20new%20line">Mail us from iPhone</a>
<!-- Напишите нам с iPhone -->
```

Отправка SMS-сообщения

Мы все любим службу SMS (Short Message Service, служба коротких сообщений), и поэтому мобильные браузеры предоставляют нам возможность открывать окна для SMS-сообщений по ссылке. В нашем распоряжении имеются две URI-схемы — `sms://` и `smsto://`. К сожалению, нет стандартного способа с уверенностью определить, какая из них совместима с браузером конкретного пользователя. Однако для смартфонов, совместимых с jQuery Mobile, мы можем без боязни применять схему `sms://`.

Схемы имеют следующий синтаксис:

```
sms[to]://[<номер адресата>][?параметры]
```

Как видите, номер адресата необязателен, так что вы можете открыть в устройстве окно для составления SMS-сообщения, не указав никаких параметров. Параметры, участвующие в определении тела сообщения, совместимы не со всеми телефонами из соображений безопасности (чтобы у пользователя была возможность избежать отправки SMS-сообщений по завышенной цене). Как и в случае с сообщениями электронной почты, SMS-сообщение не отправляется автоматически после нажатия на ссылку. Ссылка лишь открывает окно приложения для составления SMS-сообщений, и пользователь должен завершить процесс вручную. Номер, на который отправляется сообщение, должен иметь международный формат, а если он является коротким, мы должны гарантировать, что абонент находится в соответствующей стране и пользуется услугами оператора, позволяющего звонить на этот номер.

Приведу несколько примеров:

```
<a href="sms://">Send an SMS</a>
<!-- Отправить SMS-сообщение -->

<a href="sms://?body=Visit%20the%20best%20site%20at%20http://mobilexweb.com">
  Invite a friend by SMS<a>
<!-- Пригласить друга по SMS -->

<a href="sms://+3490322111">Contact us by SMS</a>
<!-- Свяжитесь с нами по SMS -->

<a href="sms://+3490322111?body=Interested%20in%20Product%20AA2">More info for product
  AA2</a>
<!-- Более подробная информация о товаре -->
```

Прочие URI-схемы

Если вы хотите глубже разобраться в интеграции HTML-кода в мобильные устройства и изучить такие технологии, как MMS, iDENnetworks сети прямого вызова, видеозвонки, сообщения BlackBerry PIN, Facebook, Twitter и интеграцию с другими низкоуровневыми приложениями, прочитайте книгу "Programming the Mobile Web".

Работа со специальными URI-схемами не является функцией jQuery Mobile. Платформа оставляет их "как есть", перекладывая всю ответственность на мобильный браузер.

Подведение итогов

Напишем простое веб-приложение jQuery Mobile, использующее различные URI-схемы:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Up and Running with jQuery Mobile</title>
  <!-- jQuery Mobile: разработка приложений для смартфонов
       и планшетов -->
  <link rel="stylesheet" href="jquery.mobile-1.0.min.css" />
  <script src="jquery-1.6.4.min.js"></script>
  <script type="text/javascript" src="jquery.mobile-1.0.min.js">
  </script>
  <meta name="viewport" content="width=device-width, initial-scale=1">
</head>
<body>
  <div data-role="page" id="main">
    <div data-role="header">
      <h1>Special Links</h1>
      <!-- Специальные ссылки -->
    </div>
    <div data-role="content">
      <p>Use the following buttons for testing special
         mobile behaviours.</p>
      <!-- Воспользуйтесь следующими кнопками для тестирования
           специальных мобильных возможностей -->
      <p>
      <a href="tel:+1800229933" data-role="button">
         Call the White House
         <!-- Позвонить в Белый дом -->
      </a>
      <a href="sms:+1800229933" data-role="button">
         SMS the White House
         <!-- Отправить SMS-сообщение в Белый дом -->
      </a>
      <a href="sms:+1800229933?body=Hello!" data-role="button">
         SMS with a body text
         <!-- SMS-сообщение со встроенным текстом -->
      </a>
```

```
      <a href="mailto:info@mobilexweb.com?subject=Sent%20from%20the%20web"
         data-role="button">
         Mail me
         <!-- Письмо автору книги -->
      </a>
      <a href="skype:maximiliano.firtman?call" data-role="button">
         Skype me
         <!-- Связаться с автором по Skype -->
      </a>
      <a href="facetime:+1800229933" data-role="button">
         Call using Facetime
         <!-- Позвонить через Facetime -->
      </a>
      (iOS with camera only)
      <!-- Только на iOS-устройствах с камерой -->
      </p>
    </div>
    <div data-role="footer">
      <h4>www.mobilexweb.com</h4>
    </div>
  </div>
</body>
</html>
```

ГЛАВА 3

Компоненты пользовательского интерфейса

Платформа jQuery Mobile предлагает нам огромный выбор компонентов пользовательского интерфейса для веб-приложений. Не будем забывать, что мы всегда можем применять обычный HTML-код и CSS-таблицы для добавления содержимого и реализации собственных идей. Однако из соображений перекрестной совместимости мы будем отдавать предпочтение компонентам платформы.

Компоненты, предоставляемые платформой jQuery Mobile, можно разделить на следующие группы:

- панели инструментов;
- компоненты форматирования;
- кнопки;
- списки;
- формы.

В этой главе мы рассмотрим первые три категории, а списки и формы оставим на будущее.

Панели инструментов

Панели инструментов — это необязательные элементы веб-приложения, определяющие верхние и нижние колонтитулы. Хотя эти элементы и необязательны, но верхние колонтитулы (заголовки) имеются почти в каждом мобильном приложении.

В предыдущей главе мы уже говорили о заголовке. Это полоса в верхней части экрана, в которой расположены текст заголовка и/или кнопки для возврата или закрытия. Вообще, заголовок определяется элементами `<div>` с ролью `header` и текстом заголовка в теге `<h1>`:

```
<div data-role="header">
  <h1>Page's title</h1>
  <!-- Заголовок страницы -->
</div>
```

Нижний колонтитул — аналогичная полоса в нижней части веб-приложения, имеющая более широкое предназначение. Она может содержать информацию об авторских правах, ссылку для звонка или ряд кнопок, образующих панель инстру-

ментов или область навигации во вкладке. Обычно нижний колонтитул определяется элементом с ролью footer:

```
<div data-role="footer">
</div>
```

Размещение

Размещение панелей инструментов кажется простым делом: верхние колонтитулы сверху, нижние — снизу. Тем не менее, у платформы jQuery Mobile имеется система размещения, позволяющая определить различные образцы поведения для обеих панелей.

Каждая панель инструментов (заголовок или нижний колонтитул) может быть размещена на страницы четырьмя способами:

- в режиме встраивания;
- в стандартном фиксированном режиме;
- в полноэкранном фиксированном режиме;
- в подлинно фиксированном режиме.

Режим встраивания устанавливается для каждой панели инструментов по умолчанию. Он означает, что заголовок и нижний колонтитул как бы встроены в страницу. Иначе говоря, если содержимое страницы не умещается на экране, нижний колонтитул будет скрыт и появится только после прокрутки, а заголовок будет виден, только когда на экране показано начало страницы.

Иногда бывает необходимо, чтобы панели инструментов были доступны все время, и для этой цели платформа jQuery Mobile предлагает полноэкранный и фиксированные режимы. Для определения режима размещения панели инструментов предусмотрен атрибут data-position.

Фиксированная (определяемая значением fixed) панель инструментов jQuery Mobile будет размещена сверху (заголовок) или снизу (нижний колонтитул). Во время прокрутки содержимого страницы панель инструментов автоматически скрывается с эффектом постепенного исчезновения. По окончании прокрутки фиксированная панель появляется в соответствующем месте, сверху или снизу.

Если пользователь прикоснется пальцем к неинтерактивной области страницы, фиксированные панели инструментов перейдут в режим встраивания, если до этого находились в фиксированном, и наоборот. Это позволяет пользователю переключать режимы прикосновением к экрану и освобождать место на нем, когда необходимо.

Для определения фиксированной панели инструментов (рис. 3.1) мы пишем атрибут data-position="fixed".

Полноэкранная панель инструментов — это всего лишь скрытая фиксированная панель, которая появляется, когда пользователь касается экрана, и исчезает, когда пользователь касается экрана во второй раз. Содержимое в этом случае занимает

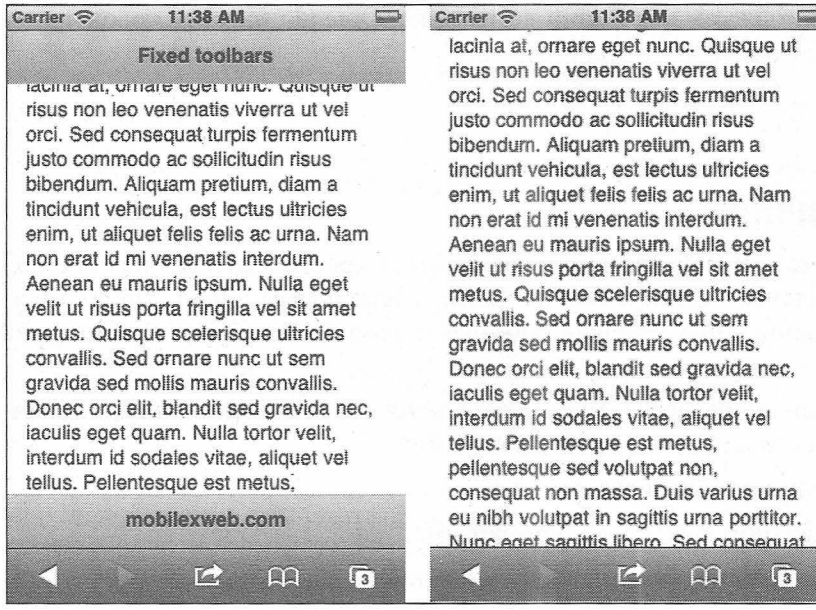


Рис. 3.1. Фиксированная панель инструментов находится в верхней (заголовок) или нижней (нижний колонтитул) части экрана независимо от объема содержимого и положения прокрутки, однако в процессе прокрутки эти панели скрываются

весь экран, а если пользователю понадобится элемент панели инструментов, он просто прикоснется к экрану.

В случае с фиксированными и полноэкранными панелями инструментов пользователь может скрывать и возвращать панели на экран прикосновением к содержимому. Разница состоит в том, что в полноэкранном режиме панели действительно скрываются, а в фиксированном режиме они становятся встроенными.

Полноэкранный режим исключительно удобен при демонстрации фотографий и выводе длинного текста или большой формы, поскольку позволяет максимально полно использовать экран.

Когда панели инструментов видны, они закрывают содержимое. В большинстве браузеров заголовки отображаются с частичной прозрачностью, чтобы было видно, что за ним находится.

Для определения этого режима мы пишем `data-position="fixed"` в коде панелей инструментов, а затем `datafullscreen="true"` в коде страницы (т. е. элемента, у которого атрибут `data-role` имеет значение `page`).

В следующем примере определяются полноэкранные панели инструментов:

```
<div data-role="page" data-fullscreen="true">
  <div data-role="header" data-position="fixed">
    <h1>Page's title</h1>
    <!-- Заголовок страницы -->
  </div>
```

```
  <div data-role="content">
  </div>
  <div data-role="footer" data-position="fixed">
  </div>
</div>
```

Подлинно фиксированные панели инструментов

На момент работы над этой книгой лишь немногие мобильные браузеры поддерживали конструкции `position: fixed` и `overflow: auto` в каскадных таблицах стилей. Фиксированное размещение и зоны прокрутки позволяют нам определять фиксированные области на экране, которые не перемещаются даже при прокрутке документа. Окно просмотра и возможное изменение масштаба обычно усложняют реализацию идеи фиксированного размещения.

Самые последние версии некоторых мобильных браузеров, в том числе Safari в iOS, Android, Black-Berry и Nokia, поддерживают конструкцию `position: fixed` и/или возможность прокрутки внутри блока с атрибутом `overflow: auto`.

Если мы хотим реализовать подлинно фиксированные панели инструментов, то можем включить функциональную возможность touch overflow (сенсорное переполнение) в jQuery Mobile. С помощью конструкции `overflow: auto` в таблице CSS она создаст область прокрутки, меньшую, чем экран, так что наши панели инструментов всегда будут находиться на своих местах. В версии 1.0 эта функциональность отключена по умолчанию, но мы можем ее включить в коде JavaScript. Если мы это сделаем, браузеры, несовместимые с jQuery Mobile, будут отображать стандартные фиксированные панели. С выходом будущих версий платформы эта функциональная возможность станет морально устаревшей, поскольку появятся подлинно фиксированные панели инструментов, по умолчанию отображаемые браузерами, совместимыми с jQuery Mobile.

Далее в этой книге мы обсудим API-интерфейсы jQuery Mobile для JavaScript, а сейчас, чтобы идея была понятна, я приведу код, включающий отображение подлинно фиксированных панелей:

```
$.mobile.touchOverflowEnabled = true;
```

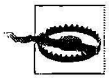
Добавление содержимого в заголовок

Стандартный верхний колонтитул jQuery Mobile содержит собственно заголовок (элемент `<hX>`) и, возможно, одну или две кнопки, расположенные слева и/или справа. Конечно, мы можем изменять внешний вид и содержимое верхнего колонтитула по своему желанию.

Добавление кнопок

Обычно (но не всегда) принципы юзабилити сенсорных устройств предписывают размещать элементы для позитивных действий справа, а элементы негативных действий — слева. Примерами позитивных действий являются кнопки **Done** (Готово),

Save (Сохранить), **Yes** (Да), **OK** и **Send** (Отправить). Негативные действия — **Cancel** (Отмена), **Back** (Назад), **No** (Нет), **Exit** (Закрыть) и **Log Out** (Выход).



Мы помним из *главы 2*, что нам не нужно самостоятельно реализовывать действие "Назад". Платформа jQuery Mobile автоматически обработает действие соответствующей браузерной кнопки, а мы можем написать атрибут `data-add-back-btn="true"` в коде страницы, чтобы расположить кнопку слева от заголовка. Если мы воспользуемся этой возможностью, то должны будем воздержаться от добавления других кнопок слева.

Кнопка в заголовке — это всего лишь гиперссылка, определяемая элементом `<a>` внутри заголовка. Если мы напишем только один элемент `<a>`, он появится у левого края заголовка. Если мы создадим две кнопки, первая будет расположена у левого края, а вторая — у правого.

Если мы хотим принудительно разместить кнопку справа или слева, то можем использовать CSS-класс `class="ui-btn-right"` или, соответственно, `class="ui-btn-left"`.

Следующий заголовок будет иметь только одну кнопку (рис. 3.2):

```
<div data-role="header">
  <a href="logout">Log out</a>
  <!-- Выйти -->
  <h1>Title</h1>
  <!-- Заголовок -->
</div>
```

Рис. 3.2. Заголовки могут включать в себя кнопки, расположенные слева или справа

Рис. 3.3. Чтобы интерфейс выглядел понятнее, принято использовать значки на кнопках заголовка

В следующем заголовке мы увидим две кнопки, причем правая содержит значок:

```
<div data-role="header">
  <a href="logout">Log out</a>
    <!-- Выйти -->
  <h1>Title</h1>
    <!-- Заголовок -->
  <a href="settings" data-icon="gear">Settings</a>
    <!-- Параметры -->
</div>
```

Как видно на рис. 3.3, мы можем использовать в кнопке любой из стандартных значков jQuery Mobile, добавив в код атрибут `data-icon`, а также любое допустимое значение из перечисленных в предыдущей главе и обсуждаемых еще раз в соответствующих разделах этой главы.

По умолчанию каждая кнопка панели инструментов наследует тему панели (а если эта тема не определена, то тему страницы). Если мы хотим, чтобы кнопки отличались друг от друга, мы можем для одной из них задать атрибут `datatheme` и сменить ее образец цвета (рис. 3.4). Например:

```
<div data-role="header">
  <a href="cancel" data-icon="delete">Cancel</a>
  <!-- Отмена -->
  <h1>New record</h1>
  <!-- Новая запись -->
  <a href="save" data-icon="check" data-theme="b">Save</a>
  <!-- Сохранить -->
</div>
```

Рис. 3.4. Мы можем изменить внешний вид кнопки, поменяв образец цвета ее темы

Если мы хотим, чтобы наша кнопка заголовка ссылалась на предыдущую страницу, мы должны написать атрибут `data-rel="back"` в элементе `<a>`, чтобы платформа jQuery Mobile могла корректно обработать анимацию возврата и журнал посещений.

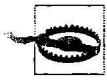
Добавление логотипа

Желание показать картинку или логотип вместо текста вполне обычно. Тема использования изображений в мобильных приложениях выходит за рамки этой книги, но если вы хотите вывести изображение в заголовке с помощью стандартных кнопок, расположенных слева и справа, то лучшим способом будет применение тега `<img>` внутри тега `<h1>`:

```
<div data-role="header">
  <a href="cancel" data-icon="delete">Cancel</a>
  <!-- Отмена -->
  <h1><img src="images/logo.png"></h1>
  <a href="save" data-icon="check" data-theme="b">Save</a>
  <!-- Сохранить -->
</div>
```

Заголовок автоматически увеличится в соответствии с высотой изображения (рис. 3.5). Не забудьте позаботиться о подходящем цвете фона. Чтобы избежать проблем с совместимостью, не используйте изображения размером более 125 пикселов.

Рис. 3.5. Использовать изображение в заголовке страницы не труднее, чем написать элемент `img`



Помните, что при работе с jQuery Mobile вы пишете приложения, предназначенные для экранов самого разного размера и с различным разрешением. Поэтому использование изображений требует некоторых действий по выяснению возможностей целевого устройства. Так вы обеспечите оптимальный пользовательский опыт в каждом конкретном случае.

Настройка внешнего вида заголовка

Если вас не устраивает стандартное отображение заголовка в jQuery Mobile, вы можете включить автоматическое поведение заголовка (т. е. обработку элементов <hX> и <a>), поместив все содержимое в контейнер блока (как правило, в элемент).

Настройка внешнего вида заголовка позволяет добавлять элементы формы, например, раскрывающиеся списки или флажки для фильтрации данных или доступа к меню. Эти элементы интерфейса обсуждаются в *главе 5*.

Рассмотрим пример (проиллюстрированный на рис. 3.6):

```
<div data-role="header">
  <div>
    <h1>A custom title</h1>
    <!-- Нестандартный заголовок -->
    <a href="#">A non-button link</a>
    <!-- Ссылка, не являющаяся кнопкой -->
  </div>
</div>
```

Для обработки нестандартного содержимого заголовка вам придется написать собственный код HTML и CSS. Высота заголовка увеличится автоматически в соответствии с содержимым.

Рис. 3.6. Мы можем отменить автоматическое поведение элементов <hX> и <a> в заголовках, используя собственные заголовки

Добавление содержимого в нижний колонтитул

Нижний колонтитул является гораздо более гибким элементом, чем заголовок. Как и в заголовке, любой элемент <a> будет отображен в виде кнопки, хотя в нижнем колонтитуле нет принудительного размещения кнопок слева или справа. Каждая кнопка встраивается в элемент и располагается после предыдущей. Мы можем разместить столько кнопок, сколько потребуется, как в обычной панели.

По умолчанию платформа не оставляет просвета между границей нижнего колонтитула и кнопками, поэтому для обеспечения привлекательного внешнего вида мы должны добавить в нижний колонтитул класс `uibar`.

Рассмотрим пример (проиллюстрированный на рис. 3.7):

```
<div data-role="footer" class="ui-bar">
  <a href="refresh">Refresh</a>
  <!-- Обновить -->
  <a href="filter">Filter</a>
  <!-- Фильтровать -->
  <a href="search">Search</a>
  <!-- Поиск -->
  <a href="add" data-theme="b">New Item</a>
  <!-- Новый пункт -->
</div>
```

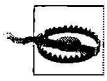
Рис. 3.7. Нижняя панель инструментов обладает большей гибкостью, чем заголовок, в том, что касается количества кнопок и их выравнивания

Далее мы обсудим наборы кнопок, позволяющие обращаться нескольким кнопкам как одним большим групповым элементом. Такой элемент может быть добавлен в нижний колонтитул.

Нижние колонтитулы могут также содержать элементы формы и панели навигации, и это наша следующая тема для обсуждения.

Панели навигации

Панель навигации — это набор из нескольких ссылок, который может быть помещен на панели инструментов, как правило, в нижний колонтитул, и представляющий собой группу взаимоисключающих опций. Идея применения панели навигации состоит в представлении основного способа навигации пользователю веб-приложения. На некоторых платформах, например iOS или Nokia, панель навигации называется *панелью вкладок*.



Чтобы избежать ошибочного обращения с элементами пользовательского интерфейса, не применяйте панели навигации для размещения кнопок действия, таких как **Save** (Сохранить), **Cancel** (Отмена), **Search** (Поиск). Эта панель — основной навигационный инструмент вашего приложения. Кнопки действия располагайте в виде обычных кнопок в заголовке или нижнем колонтитуле.

Панель навигации — это всего лишь контейнер (обычно элемент `<div>`), служащий оболочкой для неупорядоченного списка ссылок, соответствующих действиям. В качестве роли этого контейнера указывается значение `navbar`:

```
<div data-role="navbar">
  <ul>
    <li><a href="link1">Option 1</a>
    <!-- Вариант 1 -->
    <!-- ... -->
    <li><a href="linkn">Option n</a>
    <!-- Вариант n -->
  </ul>
</div>
```

Как видно на рис. 3.8, кнопки панели навигации внешне отличаются от стандартных кнопок, изображенных на рис. 3.7. Ширина кнопок рассчитывается в соответствии со следующим алгоритмом:

- одна кнопка на панели навигации занимает 100% площади;
- две кнопки — по 50%;

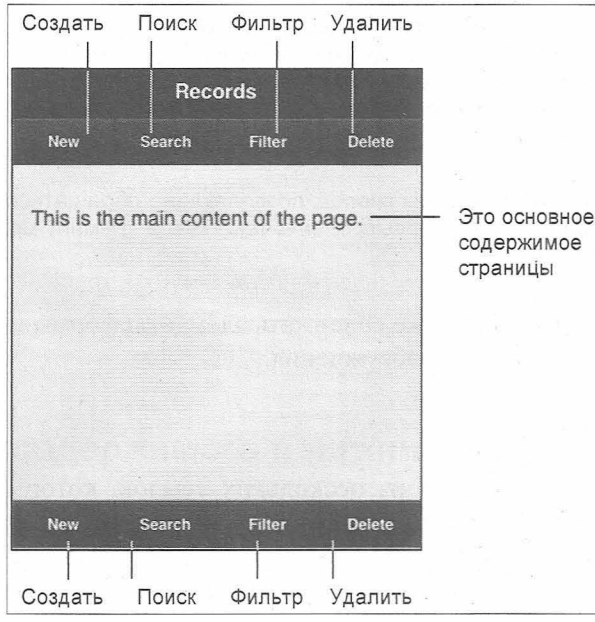


Рис. 3.8. Панели навигации в заголовке и в нижнем колонтитуле

- три, четыре или пять кнопок — 33%, 25% или 20% соответственно;
- шесть и более кнопок занимают несколько строчек по две кнопки в строке (по 50% площади строки), как показано на рис. 3.9.

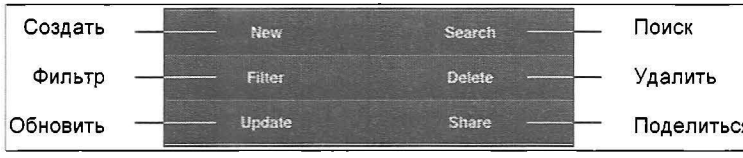


Рис. 3.9. Рекомендуется использовать на панели навигации не более шести элементов, чтобы они могли быть отображены в одной строчке. Максимальное количество кнопок — пять, потому что у большинства сенсорных смартфонов одна пятая ширины экрана является оптимальной шириной тактильных областей

Применение пиктограмм

Мы можем воспользоваться стандартным механизмом jQuery Mobile для работы со значками, чтобы разместить их на каждой панели навигации. Иными словами, мы можем написать атрибут `data-icon` со стандартным именем или даже настроить его на применение других пиктограмм.

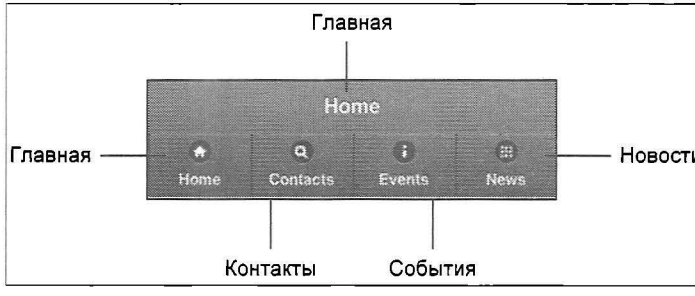


Рис. 3.10. Применение значков является хорошим стилем программирования, повышающим юзабилити и внешнюю привлекательность

По умолчанию значок размещается над текстом (рис. 3.10). Если хотя бы у одного элемента навигации есть значок, высота всей панели навигации соответственно увеличивается:

```
<div data-role="header" data-position="fixed">
  <h1>Home</h1>
  <!-- Главная -->
  <div data-role="navbar">
    <ul>
      <li><a href="#index" data-icon="home">Home</a>
      <!-- Главная -->
      <li><a href="#contacts" data-icon="search">Contacts</a>
      <!-- Контакты -->
      <li><a href="#events" data-icon="info">Events</a>
```

```
      <!-- События -->
      <li><a href="#news" data-icon="grid">News</a>
      <!-- Новости -->
    </ul>
  </div>
</div>
```

С помощью стандартных методов jQuery Mobile для создания собственных значков мы можем размещать на элементах любые пиктограммы навигации. Во Всемирной паутине имеется огромное количество бесплатных пиктограмм, а самым популярным ресурсом является **http://glyphish.com**.

Выделенный элемент

На каждой панели навигации имеется выделенный элемент, в котором используется класс `ui-btn-active`. Активный элемент внешне контрастирует среди элементов пользовательского интерфейса, отображаемых согласно текущей теме (в теме по умолчанию он имеет голубой цвет). На рис. 3.11 приводится пример стиля выделения для образца цвета.

Рис. 3.11. Состояние "выделен" отчетливо обозначается с помощью другого образца цвета

Если панель навигации является основным навигационным инструментом в веб-приложении, то в качестве первого элемента рекомендуется поставить кнопку перехода на главную страницу. Кроме того, рекомендуется выделить первый элемент `<a>` с помощью класса `class="ui-btn-active"`. Например:

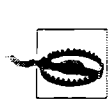
```
<div data-role="footer" data-position="fixed">
  <div data-role="navbar">
    <ul>
      <li><a href="#index" class="ui-btn-active">Home</a>
      <!-- Главная -->
      <li><a href="#contacts">Contacts</a>
      <!-- Контакты -->
      <li><a href="#events">Events</a>
      <!-- События -->
      <li><a href="#news">News</a>
      <!-- Новости -->
    </ul>
  </div>
</div>
```

Когда пользователь нажимает на элемент панели навигации, этот элемент автоматически выделяется. Это означает, что нам не нужно задумываться об обновлении класса `uibtn-active`. Кроме того, мы можем с помощью обработчика событий, напи-

санного на языке JavaScript, обрабатывать щелчки по элементам и управлять пользовательским интерфейсом и/или навигацией, а выделение элемента панели навигации будет происходить автоматически.

Постоянный нижний колонтитул

Проектируя панели навигации, мы должны отдавать себе отчет, что при смене документа jQuery Mobile появится новый нижний колонтитул вместо старого. Отсюда следует, что, например, состояние "выделен" может вызвать проблемы при отображении (оно пропадает на некоторое время или подвергается анимации при переходе на другую страницу). Для решения этой и многих других проблем можно воспользоваться постоянным нижним колонтитулом. Будучи однажды определенным, он постоянно находится в объектной модели документа, даже если произойдет смена страницы. Поэтому пользователь не сталкивается ни с какими визуальными проблемами при переходе с одной страницы на другую.

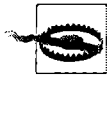


Платформа jQuery Mobile не предоставляет никакого официального способа определения постоянного заголовка. Один из возможных трюков состоит в определении постоянного нижнего колонтитула и написании таких CSS-таблиц, что он займет положение наверху экрана.

Для создания постоянной панели инструментов нам нужно определить идентификатор `data-id` для фиксированного нижнего колонтитула на каждой странице, на которой мы хотим его выводить. Тогда любая страница, содержащая нижний колонтитул с тем же значением `data-id`, будет сохранять предыдущий нижний колонтитул, не обновляя его.

Кроме того, нам придется использовать один и тот же код нижнего колонтитула и, возможно, обновлять выделенный элемент. Здесь возникает новая проблема, связанная с переходом вперед и назад с помощью панели навигации при постоянном нижнем колонтитуле: как активно поддерживать состояние "выделен"? Для ее решения при наличии панели навигации на постоянном нижнем колонтитуле мы должны применять к выделенному элементу два класса — `ui-btn-active` и `ui-statepersist`.

Рассмотрим полноценный документ с постоянным нижним колонтитулом, представляющим собой панель навигации с кнопками перехода на две страницы. Результат отображения документа показан на рис. 3.12.



Постоянный нижний колонтитул должен быть определен как фиксированный с помощью атрибута `data-position="fixed"`.

```
<!DOCTYPE html>
<html>
  <head>
    <!-- Здесь должен быть обычный заголовок jQuery Mobile -->
  </head>
```

```
  <body>
    <div data-role="page" id="home">
      <div data-role="header">
        <h1>Home</h1>
          <!-- Главная -->
        </div>
        <div data-role="content">
          <p>This is content for the home</p>
          <!-- Содержимое главной страницы -->
        </div>
        <div data-role="footer" data-id="main" position="fixed">
          <div data-role="navbar">
            <ul>
              <li><a data-icon="home"
                class="ui-btn-active ui-state-persist">Home</a></li>
              <!-- Главная -->
              <li><a href="#help" data-icon="alert">Help</a></li>
              <!-- Справка -->
            </ul>
          </div>
        </div>
      </div>
      <div data-role="page" id="help">
        <div data-role="header">
          <h1>Help</h1>
          <!-- Справка -->
        </div>
        <div data-role="content">
          <p>This is content for Help</p>
          <!-- Содержимое справочной страницы -->
        </div>
        <div data-role="footer" data-id="main" position="fixed">
        <div data-role="navbar">
          <ul>
            <li><a href="#home" data-icon="home">Home</a></li>
            <!-- Главная -->
            <li><a data-icon="alert"
              class="ui-btn-active ui-state-persist">Help</a></li>
            <!-- Справка -->
          </ul>
        </div>
      </div>
    </div>
  </body>
</html>
```

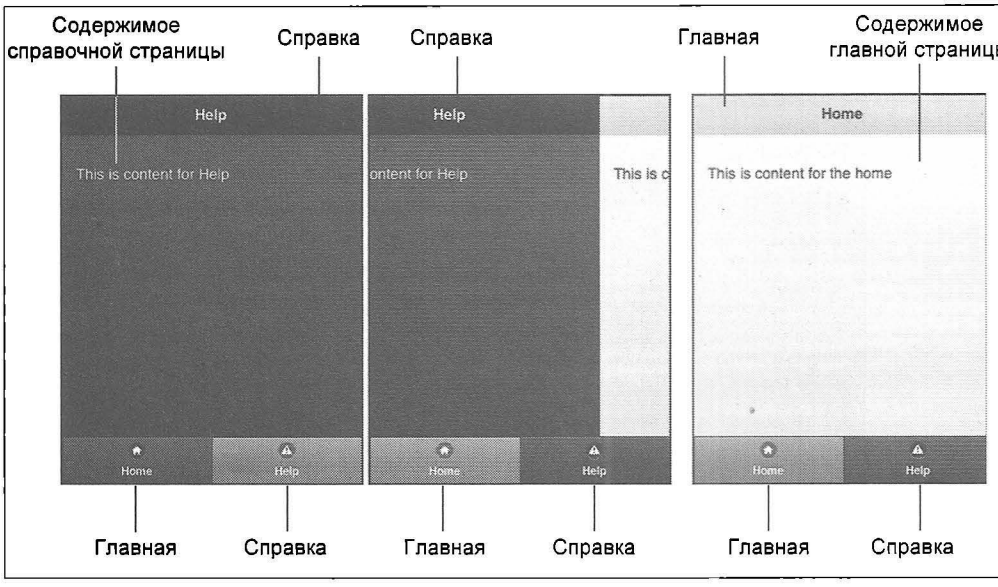


Рис. 3.12. Использование постоянного нижнего колонтитула

Форматирование содержимого

Сейчас настало время подробнее разобраться с областью содержимого нашего веб-приложения. Нам важно знать, что код будет работать внутри элемента `<div data-role="content">`.

Каждая тема jQuery Mobile содержит стили, обеспечивающие аккуратное выравнивание, поля, размеры и цвета для каждого стандартного элемента. Эти стили оптимизированы как для текущей темы, так и для мобильного устройства, и в их число входят стили элементов `<hX>`, ссылок, полужирного начертания, символов, выделенного текста, цитат, списков и таблиц.

Если мы планируем создание собственных CSS-стилей для содержимого, то должны хорошо разбираться в системе определения тем, чтобы избежать проблем с пользовательским интерфейсом при смене темы. Более подробно этот вопрос будет обсуждаться в последующих главах.

Кроме базовых HTML-элементов, существуют компоненты, предоставляемые платформой, которые определяются с помощью атрибутов `data-role`. Как мы уже видели, в заголовке и нижнем колонтитуле некоторые элементы (например, `<hX>` или `<a>`) автоматически отображаются предопределенным образом. Это не относится к элементам в области содержимого, но и тут есть свои исключения для элементов формы, о чем мы еще поговорим в *главе 5*.

На рис. 3.13 показано, как jQuery Mobile отображает некоторые из основных элементов в соответствии с темой, устанавливаемой по умолчанию.

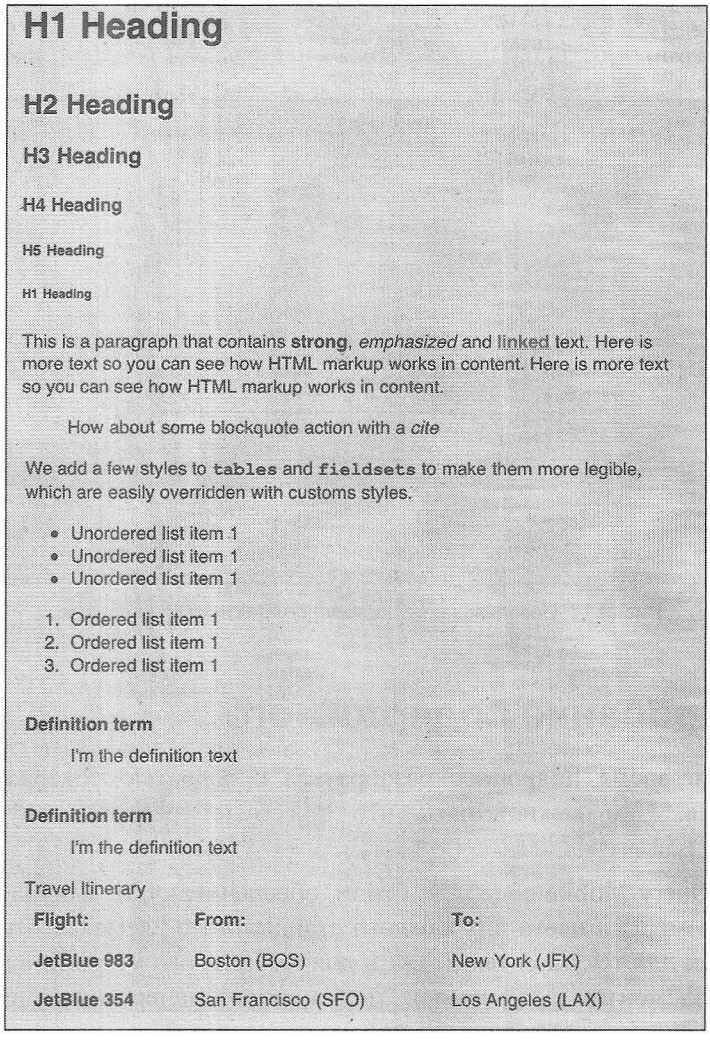


Рис. 3.13. Некоторые из основных HTML-элементов, отображаемых в соответствии с темой по умолчанию

Сворачиваемое содержимое

Не будем забывать, что мы пишем приложения для мобильных устройств, у которых место на экране сильно ограничено. В этой ситуации очень кстати оказывается сворачиваемое содержимое. Его можно скрыть и снова показать с помощью кода JavaScript, реагирующего на прикосновение к заголовку или кнопке.

Платформа jQuery Mobile предлагает автоматическую поддержку такого поведения пользовательского интерфейса, избавляя нас от необходимости писать код JavaScript. Для создания сворачиваемого содержимого нам достаточно определить контейнер с атрибутом `data-role="collapsible"`. В этом контейнере должен находиться элемент `<hX>`, который будет всегда присутствовать как заголовок, играю-

щий роль кнопки **Скрыть/Показать**. Сворачиваемое содержимое — это любой HTML-код внутри контейнера, кроме упомянутого заголовка.

По умолчанию jQuery Mobile показывает сворачиваемое содержимое при загрузке страницы. Мы можем изменить это умолчание с помощью атрибута `data-collapsed="true"` у контейнера.

Рассмотрим пример:

```
<div data-role="content">
  <div data-role="collapsible">
    <h2>History of Rome</h2>
    <!-- История Рима -->
    <p>There is archaeological evidence of human occupation
       of the Rome area from at least 14,000 years, but the
       dense layer of much younger debris obscures
       Palaeolithic and Neolithic sites.[11] Evidence of stone
       tools, pottery and stone weapons attest to at least
       10,000 years of human presence.
    </p>
    <!-- Существуют археологические свидетельства, что люди живут
         на территории современного Рима как минимум 14 тыс. лет,
         но гораздо более поздние плотные наслоения скрывают
         палеолитические и неолитические поселения. [11] Такие находки,
         как каменные орудия труда, посуда и каменное оружие, говорят
         о присутствии людей в течение как минимум 10 тыс. лет. -->
  </div>
  <div data-role="collapsible" data-collapsed="true">
    <h2>Government of Rome</h2>
    <!-- Городская администрация Рима -->
    <p>Rome constitutes one of Italy's 8,101 communes,
       and is the largest both in terms of land area and population.
       It is governed by a mayor, currently Gianni Alemanno,
       and a city council.
    </p>
    <!-- Рим является одной из 8101 коммун в Италии, самой крупной
         по занимаемой площади и населению. Управление осуществляется
         мэром, в настоящее время Джанни Алеманно (Gianni Alemanno),
         и муниципальным советом. -->
  </div>
</div>
```

Как видно на рис. 3.14, платформа jQuery Mobile добавляет значки в виде плюса и минуса в качестве кнопок для показа и, соответственно, сокрытия содержимого. На момент работы над этой книгой не было никакого способа изменить эти пиктограммы.

Как и в случае с любым другим сложным элементом управления, мы можем менять образец цвета этого элемента с помощью атрибута `datatheme`. Мы также можем оп-

ределить дополнительный атрибут `data-content-theme`, который будет влиять только на внешний вид содержимого, а не на кнопку **Скрыть/Показать** на сворачиваемой панели.

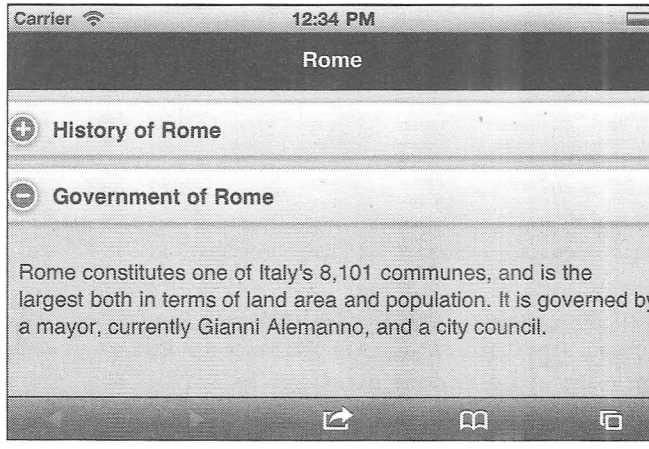
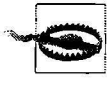


Рис. 3.14. Сворачиваемые панели — очень удобный способ вывода большого объема информации на одну страницу, при котором пользователь имеет возможность убирать и открывать подробности



Если вы не определите элемент `<hX>` внутри контейнера сворачиваемого содержимого, оно будет открыто без возможности свернуть его. Если вы определите несколько элементов `<hX>`, первый будет работать как заголовок, сворачивающий/разворачивающий содержимое, а остальные будут отображены как содержимое.

Вложенное сворачиваемое содержимое

Вы имеете возможность вкладывать сворачиваемые панели друг в друга. Платформа автоматически добавит поля на каждом новом уровне сворачиваемого содержимого. Рекомендуется не добавлять более двух уровней во избежание излишнего усложнения пользовательского интерфейса и объектной модели документа:

```
<div data-role="content">
  <div data-role="collapsible">
    <h2>Rome</h2>
        <!-- Рим -->
    <div data-role="collapsible">
    <h3>History</h3>
    <!-- История -->
    <p>There is archaeological evidence of human occupation
       of the Rome area from at least 14,000 years, but
       the dense layer of much younger debris obscures
       Palaeolithic and Neolithic sites.[11] Evidence of stone
       tools, pottery and stone weapons attest to at least
       10,000 years of human presence. </p>
```

```
    <!-- Существуют археологические свидетельства, что люди живут
         на территории современного Рима как минимум 14 тыс. лет,
         но гораздо более поздние плотные наслоения скрывают
         палеолитические и неолитические поселения. [11] Такие находки,
         как каменные орудия труда, посуда и каменное оружие, говорят
         о присутствии людей в течение как минимум 10 тыс. лет. -->
    </div>
    <div data-role="collapsible" data-collapsed="true">
      <h3>Government</h3>
      <!-- Администрация -->
      <p>Rome constitutes one of Italy's 8,101 communes, and
         is the largest both in terms of land area and population.
         It is governed by a mayor, currently Gianni Alemanno,
         and a city council. </p>
      <!-- Рим является одной из 8101 коммун в Италии, самой крупной
           по занимаемой площади и населению. Управление осуществляется
           мэром, в настоящее время Джанни Алеманно (Gianni Alemanno),
           и муниципальным советом. -->
    </div>
  </div>
  <div data-role="collapsible">
      <h2>Madrid</h2>
      <!-- Мадрид -->
    <div data-role="collapsible">
      <h3>History</h3>
      <!-- История -->
      <p>Although the site of modern-day Madrid has been occupied
         since pre-historic times,[23] in the Roman era this territory
         belonged to the diocese of Complutum
        (present-day Alcala de Henares).</p>
      <!-- Хотя территория современного Мадрида была заселена еще
           в доисторические времена, [23] в эпоху Римской империи,
           она входила в епархию Комплутума (современное название -
           Алькала де Энарес -->
    </div>
    <div data-role="collapsible" data-collapsed="true">
      <h3>Government</h3>
      <!-- Администрация -->
      <p>The City Council consists of 57 members, one of them
         being the Mayor, currently Alberto Ruiz-Gallardon Jimenez.
         The Mayor presides over the Council.</p>
      <!-- Муниципальный совет состоит из 57 членов, один из которых
           является мэром, в настоящее время Альберто Руис-Галлардо
           Хименес (Alberto Ruiz-Gallardyn Jimenez). Мэр
           председательствует в Совете. -->
    </div>
  </div>
</div>
```

Аккордеон

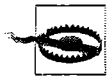
Существует еще одна распространенная модель поведения насыщенного веб-приложения, связанная со сворачиванием содержимого: *аккордеон*. Она позволяет группировать сворачиваемое содержимое так, чтобы была видна только одна панель. Иначе говоря, пока вы изучаете одну панель, остальные скрыты.

Платформа jQuery Mobile поддерживает такой тип компонентов в виде сворачиваемых наборов элементов. Набор представляет собой контейнер с атрибутом `data-role="collapsible-set"` и группу сворачиваемых панелей в качестве его потомков.

Если мы применим к сворачиваемому набору атрибут `data-theme` или `data-content-theme`, каждый потомок будет наследовать эти значения, если обратное не будет указано явно.

Шаблон сворачиваемого набора таков:

```
<div data-role="collapsible-set">
  <div data-role="collapsible">
    <!-- Сворачиваемый заголовок и содержимое -->
  </div>
  <!-- Любой объем сворачиваемого содержимого -->
  <div data-role="collapsible">
    <!-- Сворачиваемый заголовок и содержимое -->
  </div>
</div>
```



По умолчанию jQuery Mobile открывает последнюю сворачиваемую панель в сворачиваемом наборе. Если вы хотите, чтобы по умолчанию открывалась другая панель, поставьте атрибут `data-collapsed="false"` у соответствующего ей элемента и `data-collapsed="true"` у всех остальных элементов.

Рассмотрим пример:

```
<div data-role="page" id="home">
  <div data-role="header">
    <h1>@firt</h1>
  </div>
  <div data-role="content" data-theme="b">
    <!-- Здесь определяется весь сворачиваемый набор (аккордеон) -->
    <div data-role="collapsible-set">
      <div data-role="collapsible" data-collapsed="false">
        <h2>Books</h2>
        <!-- Книги -->
        <ul>
          <li>Programming the Mobile Web</li>
          <!-- Программирование для мобильной Всемирной паутины -->
          <li>jQuery Mobile: Up & Running</li>
          <!-- jQuery Mobile: разработка приложений
               для смартфонов и планшетов -->
```

```
          <li>Mobile HTML5</li>
          <!-- Мобильный HTML5 -->
        </ul>
      </div>
      <div data-role="collapsible" data-collapsed="true">
        <h2>Talks</h2>
        <!-- Доклады -->
        <ul>
          <li>Velocity Conference</li>
          <li>OSCON</li>
          <li>Mobile World Congress</li>
          <li>Google DevFest</li>
        </ul>
      </div>
    </div>
    <!-- Конец сворачиваемого набора (аккордеон) -->
  </div>
</div>
```

На рис. 3.15 представлен сворачиваемый набор, в котором каждая сворачиваемая зона имеет границы с закругленными углами.

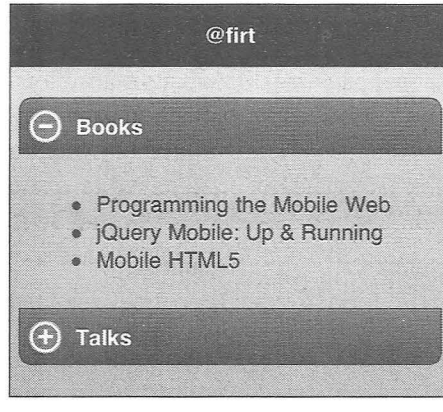
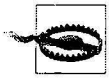


Рис. 3.15. Элемент управления "аккордеон" может быть создан из набора сворачиваемых панелей так, что в каждый момент будет видна только одна из них

Столбцы

Платформа jQuery Mobile предлагает ряд шаблонов для определения содержимого, выводимого в несколько столбцов. Эти шаблоны называются *сетками компоновки*. Сетки действуют как таблица, но без семантических сложностей использования элемента `<table>` (пожалуйста, не применяйте `<table>` ни для чего, кроме табличных данных).



Не забывайте, что ваше приложение будет работать на мобильном устройстве. Пользуйтесь столбцами осторожно, только для размещения небольших элементов, таких как кнопки, ссылки или короткие пункты списка. Если вы пишете приложение для планшетов, у вас будет больше места под столбцы.

Метод сеток компоновки основан на применении классов CSS, определяющих области сетки и столбцы. Сетка может иметь от двух до 5 столбцов. Каждая сетка невидима, занимает 100% ширины экрана, а в ее определении отсутствуют отступы и поля.

Чтобы создать сетку, достаточно использовать блочный контейнер, например `<div>`, с классом `ui-grid-a` для двух столбцов, `ui-grid-b` для трех, `ui-grid-c` для четырех и `ui-grid-d` для пяти столбцов. По умолчанию каждая сетка делит экран на столбцы одинаковой ширины.

Каждая ячейка сетки должна быть блочным контейнером с атрибутом `ui-block-<буква>`, где *<буква>* от `a` до `d` обозначает столбец сетки, с первого по пятый.

Рассмотрим пример компоновки из двух столбцов с использованием элемента `<section>`, появившегося в HTML5 (рис. 3.16):

```
<div data-role="content">
  <section class="ui-grid-a">
    <div class="ui-block-a">Column 1</div>
    <!-- Столбец 1 -->
    <div class="ui-block-b">Column 2</div>
    <!-- Столбец 1 -->
  </section>
</div>
```

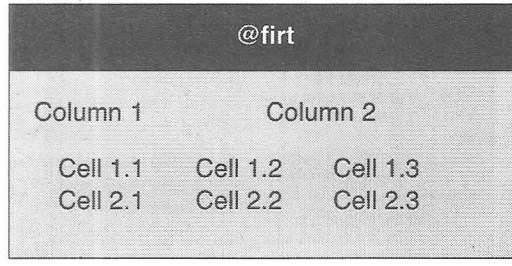


Рис. 3.16. Мы можем расположить элементы в несколько (максимум, пять) столбов

Сетки компоновки позволяют также реализовать табличное размещение. Это означает, что если мы добавим больше ячеек, чем имеется столбцов, то сможем эмулировать использование одной сетки в нескольких строках:

```
<div data-role="content">
  <section class="ui-grid-b">
    <!-- Row 1 -->
    <!-- Строка 1 -->
    <div class="ui-block-a">Cell 1.1</div>
    <!-- Ячейка 1.1 -->
```

```
    <div class="ui-block-b">Cell 1.2</div>
    <!-- Ячейка 1.2 -->
    <div class="ui-block-c">Cell 1.3</div>
    <!-- Ячейка 1.3 -->
    <!-- Row 2 -->
    <!-- Строка 1 -->
    <div class="ui-block-a">Cell 2.1</div>
    <!-- Ячейка 1.1 -->
    <div class="ui-block-b">Cell 2.2</div>
    <!-- Ячейка 1.2 -->
    <div class="ui-block-c">Cell 2.3</div>
    <!-- Ячейка 1.3 -->
  </section>
</div>
```

Кнопки

Мы уже видели, что для ссылок с одной страницы на другую или на внешнее содержимое можно применять любой элемент `<a>`. Однако типичный элемент `<a>` не очень удачно отображается в сенсорных устройствах. Элемент, как правило, является встроенным, а областью для щелчков служит текст. Это не вполне удобно пользователю сенсорного устройства. Именно поэтому платформа jQuery Mobile предоставляет нам кнопки.

Кнопка — это элемент пользовательского интерфейса, который выглядит и ведет себя как... кнопка. То есть он представляет собой большую область для щелчков, содержащую текст и, возможно, значок.

Кнопку можно создать разными способами:

- с помощью элемента `<button>`;
- с помощью элемента `<input>`, который обычно отображается в виде кнопки и имеет атрибуты `type="button"`, `type="submit"`, `type="reset"` и `type="image"`;
- с помощью любого элемента `<a>` с атрибутом `data-role="button"`.

Кнопка jQuery Mobile обычно отображается с отцентрованной надписью, скругленными углами и тенью, в зависимости от того, насколько браузер совместим с форматом CSS3.

При выводе группы кнопок хороший стиль создания пользовательского интерфейса предписывает, чтобы одна из них, — обозначающая позитивное действие и соответствующая самому вероятному выбору, — имела тему, отличную от темы остальных кнопок.

По умолчанию кнопка занимает всю ширину экрана, так что каждая кнопка выводится на отдельной строчке. Рассмотрим типичный пример, представленный на рис. 3.17:

```
<a href="#" data-role="button">Click me!</a>
<!-- Щелкни по мне! -->
<button data-theme="b">Click me too!</button>
<!-- И по мне тоже! -->
<input type="button" value="Don't forget about me!">
<!-- Обо мне не забудь! -->
```

Рис. 3.17. Кнопки — лучший инструмент взаимодействия между веб-приложением и пользователем сенсорного устройства

Встроенные кнопки

Мы можем создавать встроенные кнопки, которые не будут занимать всю ширину экрана. Для этого нам нужно применить к элементу атрибут `data-inline="true"`. Таким образом, мы получим три кнопки, расположенные в одной строке, с помощью следующего кода (см. рис. 3.17):

```
<a href="#" data-role="button" data-inline="true">Button 1</a>
<!-- Кнопка 1 -->
<a href="#" data-role="button" data-inline="true">Button 2</a>
<!-- Кнопка 2 -->
<a href="#" data-role="button" data-inline="true">Button 2</a>
<!-- Кнопка 2 -->
```

Если мы хотим, чтобы встроенные кнопки заняли всю ширину экрана, то можем воспользоваться сетками компоновки и определить до пяти столбцов, в каждом из которых будет кнопка.

Сгруппированные кнопки

Если у нас есть несколько кнопок, имеющих отношение друг к другу, мы можем сгруппировать их и получить другой пользовательский интерфейс, в котором каждая кнопка содержится в групповом контейнере. На платформе jQuery Mobile эта техника называется *группированием элементов управления*, а ее применение включает в себя использование нового компонента, называемого *группой элементов управления*, и нескольких кнопок.

Создавая сгруппированные кнопки, не объявляйте их встроенными.

Группа элементов управления — это просто контейнер, как правило, элемент с атрибутом data-role="controlgroup". Такая группа включает в себя несколько кнопок и выглядит примерно так, как показано на рис. 3.17:

```
<div data-role="controlgroup">
  <a href="#" data-role="button">Button 1</a>
  <!-- Кнопка 1 -->
  <a href="#" data-role="button">Button 2</a>
  <!-- Кнопка 2 -->
  <a href="#" data-role="button">Button 2</a>
  <!-- Кнопка 2 -->
</div>
```

На рис. 3.18 видно, что кнопки отображаются вертикально, друг под другом. Мы можем изменить такую компоновку на горизонтальную с помощью атрибута data-type="horizontal" в элементе-группе (рис. 3.19):

```
<div data-role="controlgroup" data-type="horizontal">
  <!-- Кнопка 1 -->
  <a href="#" data-role="button" data-inline="true">Button 1</a>
  <!-- Кнопка 2 -->
  <a href="#" data-role="button" data-inline="true">Button 2</a>
  <!-- Кнопка 2 -->
  <a href="#" data-role="button" data-inline="true">Button 2</a>
</div>
```

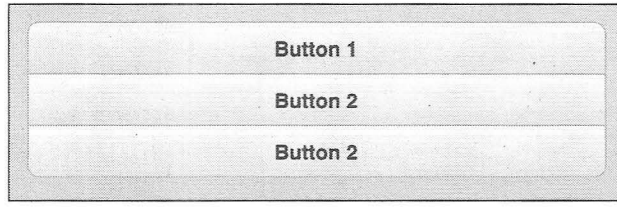


Рис. 3.18. Для улучшения внешнего вида набора встроенных кнопок рекомендуется собрать их в группу

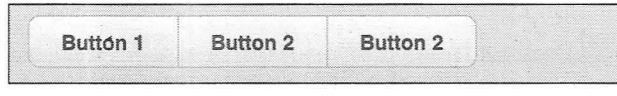


Рис. 3.19. При наличии не более пяти кнопок с короткими надписями можно использовать горизонтальное размещение

Горизонтальная группа кнопок может выглядеть как набор альтернативных вариантов (выбираемых кнопками). Однако сейчас мы имеем просто несколько кнопок, собранных вместе. Используя эту технику, мы не можем указать состояние "выделена". Позже мы обсудим группирование альтернативных кнопок.

Эффекты

По умолчанию каждая кнопка изображается со скругленными углами и тенью. Мы можем изменить ее внешний вид булевыми атрибутами `data-corners` и `data-shadow`. Далее в этой книге описывается, как изменить эти свойства кнопок глобально с помощью кода JavaScript.

Итак, можно отменить у кнопок оба стиля:

```
<a href="#" data-role="button" data-shadow="false"
            data-corners="false">Help</a>
```

Значки

Платформа jQuery Mobile имеет мощный механизм работы со значками, позволяющий применять темы к кнопкам и многим другим компонентам, на которых могут присутствовать пиктограммы. Мы вернемся к разговору о значках в последующих главах, причем опции, галерея и функциональность будут теми же, о которых мы поговорим сейчас.

В любой кнопке значок определяется атрибутом `data-icon`, в котором указано имя пиктограммы.

Значки, доступные в jQuery Mobile 1.0, перечислены в табл. 3.1.

***Таблица 3.1.** Значки, доступные в jQuery Mobile*

Описание значка	Значение
Стрелка влево	`arrow-l`
Стрелка вправо	`arrow-r`
Стрелка вверх	`arrow-u`
Стрелка вниз	`arrow-d`
Удалить (×)	`delete`
Плюс	`plus`
Минус	`minus`
Галочка	`check`
Шестеренка (настройки)	`gear`
Обновить	`refresh`
Вперед	`forward`
Назад	`back`
Сетка	`grid`
Звездочка	`star`
Внимание (пиктограмма предупреждения)	`alert`

Таблица 3.1 (окончание)

Описание значка	Значение
Информация (i)	`info`
Главная страница	`home`
Поиск	`search`

Помните, что файлы с изображениями значков поставляются вместе в платформой jQuery Mobile в загружаемом архиве или хранятся в сети CDN. Для пиктограмм, наносимых на кнопки, нужна папка с графическими файлами на тот случай, если мы решим хранить платформу у себя или будем создавать низкоуровневое приложение, содержащее платформу в себе. Для уменьшения изобразительной нагрузки на значки платформа jQuery Mobile применяет технологию CSS Sprites, которая автоматически выбирает отображение с низким или высоким разрешением.

На рис. 3.20 представлены все значки.

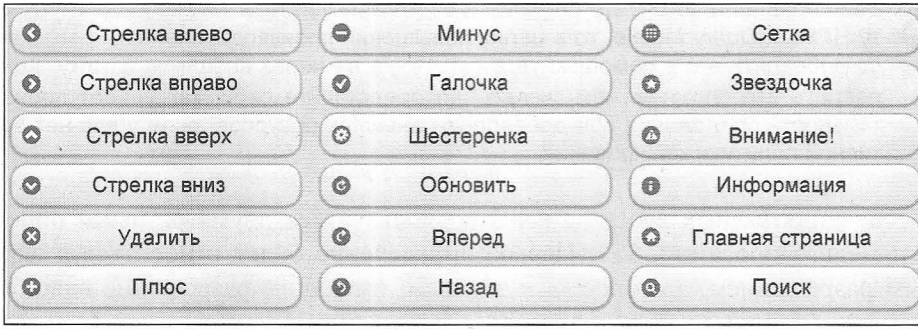


Рис. 3.20. Все значки платформы jQuery Mobile

Создание нестандартных значков

Если мы хотим использовать в приложении нестандартный значок, нам достаточно указать его уникальное имя в атрибуте `data-icon`. Рекомендуется придерживаться такой схемы наименования: *<имя-приложения>-<имя-значка>*, например, `myapp-tweet`.

Платформа jQuery Mobile автоматически добавит к значку фон в виде белого круга, чтобы он хорошо смотрелся на любом фоне.

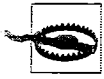
Кроме того, платформа jQuery ожидает, что мы создадим CSS-класс с названием `ui-icon-`*<имя>*, например `uiicon-myapp-tweet`, и укажем атрибут `background-image`. Чтобы избежать нежелательных последствий в пользовательском интерфейсе, мы должны нарисовать пиктограмму белым (или прозрачным) цветом и сохранить в виде изображения 18×18 пикселов в формате PNG-8 с альфа-прозрачностью. У значка не

должно быть границы; платформа сама добавит границу при отображении пиктограммы. Собственно рисунок должен быть меньше 18×18 пикселов, т. к. будет отображаться внутри кружка диаметром 18 пикселов. Поскольку значок изображается в круге, у него не должно быть углов.

Код класса должен быть примерно таким:

```
<style>
  .ui-icon-myapp-tweet { background-image: url(icons/tweet.png); }
</style>
```

Наш нестандартный значок будет плохо смотреться на устройствах с высоким разрешением экрана, таких как iPhone 4 и iPod Touch 4G с дисплеем Retina. По этой причине мы должны также обеспечить версию значка с высоким разрешением. Для этого потребуется определение альтернативного селектора CSS. Размеры пиктограммы должны быть ровно в два раза больше: 36×36. В обеих версиях можно использовать одно изображение, но для достижения наилучшего результата рекомендуется все-таки иметь две разные картинки.



Если предполагается, что стиль CSS будет применяться только к одному документу HTML5 jQuery Mobile, то в целях повышения производительности рекомендуется добавлять его с помощью тега `<style>`, а не через внешнюю ссылку. Конечно, сделать это сложнее, чем сказать, но представьте себе, как упадет производительность приложения при добавлении внешних ресурсов, если у вас нет эффективной политики кэширования.

Для обеспечения разных версий значка воспользуйтесь двумя разными определениями и медиазапросами CSS3. Первое определение будет относиться к экранам с любым разрешением, а второе — к экранам, имеющим разрешение вдвое выше стандартного (например, в устройствах iPhone 4 или более новых). Мы должны явно указать, что размер фона равен 18×18, потому что на этих устройствах суффикс `px` относится к виртуальным точкам, каждая из которых равна двум реальным пикселам. В результате получится примерно такой класс:

```
<style>
  .ui-icon-myapp-tweet {
    background-image: url(icons/tweet.png);
  }
  @media only screen and (-webkit-min-device-pixel-ratio: 2) {
    .ui-icon-myapp-tweet {
       background-image: url(icons-hd/tweet.png) !important;
       background-size: 18px 18px;
    }
  }
</style>
```

Сравним стандартную кнопку с нашей (рис. 3.21). В нашем приложении используются только что добавленные стили:

```
<a href="#" data-role="button" data-icon="gear">Help</a>
<!-- Справка -->
<a href="#" data-role="button" data-icon="myapp-tweet">Tweet</a>
<!-- Общение -->
```

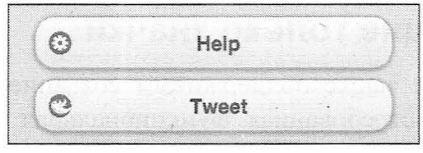


Рис. 3.21. Мы можем создавать нестандартные значки для кнопок и других компонентов пользовательского интерфейса

Расположение значков

По умолчанию каждый значок выводится слева от текста кнопки. Мы можем изменить его положение с помощью атрибута `data-iconpos`, принимающего значения `right`, `left` (по умолчанию), `bottom` и `top`. Если указано `bottom` или `top`, высота кнопки увеличится. Это очень удобно на панелях навигации или на горизонтальных группах кнопок.

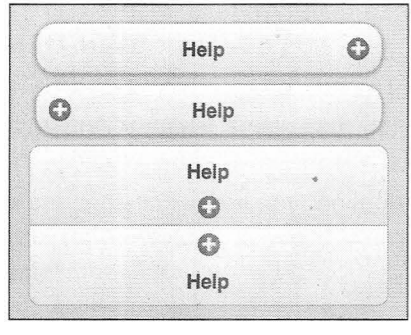


Рис. 3.22. Атрибут `data-iconpos` позволяет использовать любой из четырех вариантов размещения значка

Рассмотрим, как происходит размещение значков на одной и той же ссылке (рис. 3.22):

```
<a href="#" data-role="button" data-icon="help"
            data-iconpos="right">Help</a>
  <!-- Справка, значок справа -->
<a href="#" data-role="button" data-icon="help"
            data-iconpos="left">Help</a>
  <!-- Справка, значок слева -->
<div data-role="controlgroup">
  <a href="#" data-role="button" data-icon="help"
              data-iconpos="bottom">Help</a>
  <!-- Справка, значок снизу -->
```

```
  <a href="#" data-role="button" data-icon="help"
             data-iconpos="top">Help</a>
  <!-- Справка, значок сверху -->
</div>
```

Кнопки, содержащие только значки

Платформа jQuery Mobile также поддерживает создание кнопок, не содержащих текста. Это не самая востребованная функциональная возможность, поскольку кнопки получаются слишком маленькими для сенсорных интерфейсов. Чтобы создать кнопку, содержащую только значок, укажите для нее атрибут `data-iconpos="notext"`.

Тени на значках

Платформа jQuery Mobile предоставляет нам атрибут, позволяющий убрать эффект "тень" со значка. Для этого следует указать атрибут `data-iconshadow="false"` в коде соответствующей кнопки.

ГЛАВА 4

Списки

До сих пор мы создавали очень простые страницы, отображаемые платформой jQuery Mobile. Следующим важным шагом будет использование мощных элементов управления и представлений, реализуемых платформой. Почти каждый мобильный проект имеет в своем содержимом хотя бы один список, поэтому мы начнем обсуждение с этого элемента.

Трудно начать эту главу с определения. Вы уже знаете, что такое список, и я ничего не могу сказать нового о нем, по крайней мере в общих словах. В технологии jQuery Mobile список может быть упорядоченным (HTML-элемент `<ol>`) или неупорядоченным (HTML-элемент `<ul>`). Он располагается на странице и содержит хотя бы один пункт (HTML-элемент `<li>`), а его роль определена значением `listview` в HTML5-атрибуте `data-role="listview"`.

Если вы всего лишь хотите вывести маркированный или нумерованный список, то можете написать обычные HTML-элементы `<ul>` и `<ol>`; просто не нужно присваивать им никакой роли jQuery Mobile.

Списки — эффективное решение многих задач платформы, в чем мы вскоре убедимся.

Самым распространенным списком является неупорядоченный (`<ul>`), который просто существует на странице, не имеющей другого содержимого. Рассмотрим простой пример, проиллюстрированный на рис. 4.1:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Up and Running with jQuery Mobile</title>
    <!-- jQuery Mobile: разработка приложений
         для смартфонов и планшетов -->
    <link rel="stylesheet" href="jquery.mobile-1.0.min.css" />
    <script src="jquery-1.6.4.min.js"></script>
    <script type="text/javascript" src="jquery.mobile-1.0.min.js">
    </script>
    <meta name="viewport" content="width=device-width, initial-scale=1">
  </head>
  <body>
    <div data-role="page" id="main">
      <div data-role="header">
```

```
        <h1>HTML5 and APIs</h1>
        <!-- HTML5 и API-интерфейсы -->
      </div>
      <div data-role="content">
        <ul data-role="listview">
          <li>Offline Access
          <!-- Автономный доступ -->
          <li>Geolocation API
          <!-- API для геопозиционирования -->
          <li>Canvas
          <!-- Элемент canvas -->
          <li>Offline Storage
          <!-- Автономное хранение данных -->
          <li>New semantic tags
          <!-- Новые семантические теги -->
        </ul>
      </div>
    </div>
  </body>
</html>
```

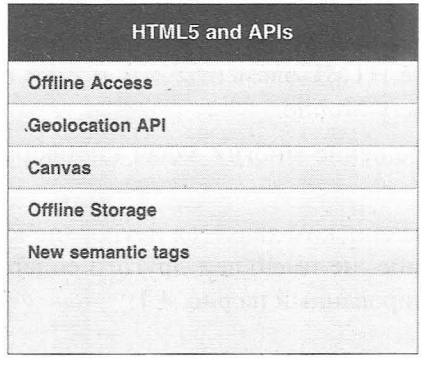


Рис. 4.1. Типичное представление списка состоит в аккуратном отображении строк в сенсорных устройствах

Платформа jQuery Mobile отображает списки, оптимизируя их под сенсорные экраны, о чем свидетельствует высота строк, принимаемая по умолчанию. Каждый пункт списка автоматически занимает всю ширину экрана, что типично для интерфейса сенсорных мобильных устройств. (Если список достаточно длинный, мы можем использовать фиксированные панели инструментов.)

Помните, что jQuery Mobile работает на стороне клиента. Документ со списком может быть без труда сгенерирован динамически с помощью любой серверной технологии, такой как PHP, Java, ASP.NET или Ruby.

Мы можем также создавать упорядоченные списки (элемент `<ol>`). Однако если мы не определим интерактивные строки со ссылками, внешне список не будет

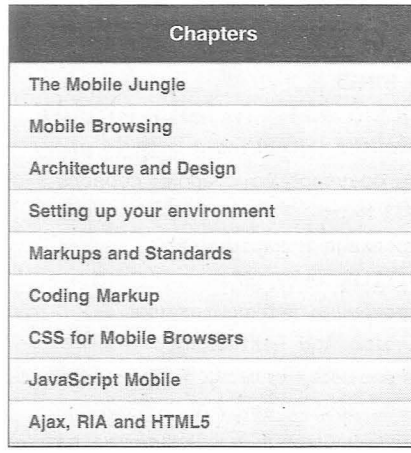


Рис. 4.2. Упорядоченный список отображается точно так же, как неупорядоченный, если мы не придали ему интерактивность

отличаться от неупорядоченного списка `<ul>`, в чем легко убедиться, глядя на рис. 4.2:

```
<!DOCTYPE html>
<html>
  <head>
    <!-- Здесь должен быть обычный заголовок jQuery Mobile -->
  </head>
  <body>
    <div data-role="page" id="main">
      <div data-role="header">
        <h1>Chapters</h1>
        <!-- Названия глав -->
      </div>
      <div data-role="content">
        <ol data-role="listview">
          <li>The Mobile Jungle
          <!-- Мобильные джунгли -->
          <li>Mobile Browsing
          <!-- Использование мобильных браузеров -->
          <li>Architecture and Design
          <!-- Архитектура и дизайн -->
          <li>Setting up your environment
          <!-- Настройка окружения -->
          <li>Markups and Standards
          <!-- Разметки и стандарты -->
          <li>Coding Markup
          <!-- Кодирование разметки -->
          <li>CSS for Mobile Browsers
          <!-- CSS-таблицы для мобильных браузеров -->
```

```
        <li>JavaScript Mobile
        <!-- Мобильный JavaScript -->
        <li>Ajax, RIA and HTML5
        <!-- Ajax, RIA и HTML5 -->
        <li>Server-Side Browser Detection
        <!-- Распознавание браузера со стороны сервера -->
        <li>Geolocation and Maps
        <!-- Геопозиционирование и карты -->
        <li>Widgets and Offline webapps
        <!-- Виджеты и автономные веб-приложения -->
        <li>Testing, Debugging and Performance
        <!-- Вопросы тестирования, отладка и производительность -->
        <li>Distribution and Social Web 2.0
        <!-- Распространение и Социальная Всемирная паутина 2.0 -->
      </ol>
    </div>
  </div>
 </body>
</html>
```

В этом примере продемонстрировано, как jQuery Mobile отображает пункты списка. Если текст не умещается в одну строку, размер пункта автоматически изменится.

В языке HTML5 не используется синтаксис XML, поэтому нет необходимости закрывать все теги, в частности, элементы `<li>`. Конечно, вы можете закрыть их, если вам так комфортнее (например, мне, как программисту, незакрытый тег `<li>` режет глаз).

Полностраничные и вставленные списки

По умолчанию списки отображаются в *полностраничном режиме*. Это означает, что список занимает всю страницу, как на рис. 4.1 и 4.2. Однако в некоторых проектах бывает необходимо сочетать списки с другим HTML-содержимым. Для этой цели платформа jQuery Mobile предлагает так называемые *вставленные списки*. Они определяются добавлением атрибута `data-inset="true"` в элемент `<ul>` или `<ol>`:

```
<ol data-role="listview" data-inset="true">
  <!-- строки с пунктами списка -->
</ol>
```

На рис. 4.3 показано, как предыдущий пример будет отображаться после добавления этого атрибута. Нетрудно заметить, что дизайн таблицы немного изменился: увеличились интервалы, поскольку на той же странице имеется и другое содержимое. Кроме того, у списка появились аккуратные закругленные углы и другие CSS3-эффекты.

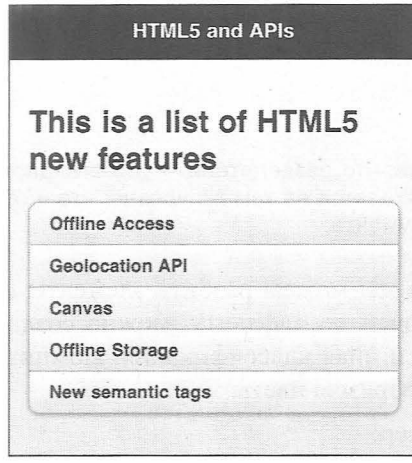


Рис. 4.3. Вставленный список имеет специфический пользовательский интерфейс и может находиться на странице вместе с другим содержимым, в том числе с другими списками

Цветовые образцы для списков определяются атрибутом `data-theme` элемента `<ul>` и каждого элемента `<li>`.

В случае со вставленными списками мы можем создать страницу, состоящую из нескольких списков и, возможно, другого HTML-содержимого между ними.

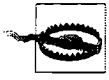
Визуальные разделители

Вы можете поставить *разделители*, разбивающие пункты списка на группы со своими названиями (рис. 4.4). В мобильных операционных системах, таких как iOS на iPhone и iPad, это вполне обычный прием. Чтобы добиться этого эффекта сред-

Рис. 4.4. Отображение разделителей в списке

ствами jQuery Mobile, можно просто использовать новое значение атрибута роли для элементов <li>, а именно:

```
data-role="list-divider"
```



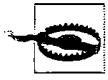
Обратите внимание, что разделители — это стандартные элементы со специальной ролью. Они находятся на том же уровне, что и обычные пункты списка в объектной модели документа.

С помощью этой техники мы можем разбить элементы списка на группы. Например, если список упорядочен по алфавиту, можно создать отдельную группу для каждой буквы. Возможна и иная классификация. Помните, что для выделения разделителей можно менять образцы цвета.

Рассмотрим простой пример:

```
<!DOCTYPE html>
<html>
  <head>
    <!-- Здесь должен быть обычный заголовок jQuery Mobile -->
  </head>
  <body>
    <div data-role="page" id="main">
      <div data-role="header">
        <h1>World Cup</h1>
        <!-- Чемпионат мира -->
      </div>
      <div data-role="content">
        <ul data-role="listview">
          <li data-role="list-divider">Group A
          <!-- Группа А -->
          <li>Argentina
          <!-- Аргентина -->
          <li>Nigeria
          <!-- Нигерия -->
          <li>England
          <!-- Англия -->
          <li>Japan
          <!-- Япония -->
          <li data-role="list-divider">Group B
          <!-- Группа В -->
          <li>United States
          <!-- США -->
          <li>Mexico
          <!-- Мексика -->
          <li>Korea
          <!-- Корея -->
          <li>Greece
          <!-- Греция -->
```

```
            <li data-role="list-divider">Group C
            <!-- Группа C -->
            <li>Germany
            <!-- Германия -->
            <li>Finland
            <!-- Финляндия -->
            <li>Chile
            <!-- Чили -->
            <li>South Africa
            <!-- ЮАР -->
          </ul>
        </div>
      </div>
    </body>
</html>
```



Если мы заполняем список на стороне клиента, например, в коде JavaScript, то должны после этого вызвать метод обновления страницы, чтобы платформа могла учесть внесенные изменения и корректно отобразить их. Мы обсудим этот вопрос позже, а пока достаточно знать, что если у вас есть только один список на странице с идентификатором `page1`, то вам потребуется такой код jQuery Mobile:

```
ul").listview("refresh")
```

На рис. 4.4 показано, как выглядят разделители списка.

Интерактивные строки

Списки становятся более мощным инструментом, когда мы реализуем в них взаимодействие с пользователем.

Если элемент списка содержит элемент `<a>`, соответствующая строка на экране станет интерактивной и будет реагировать на перемещение курсора и/или прикосновение пальцем. Замечательная особенность jQuery Mobile состоит в том, что целая строка автоматически становится интерактивной и реагирует на прикосновение, независимо от местоположения ссылки внутри нее.

Интерактивная строка отличается от обычной по высоте. Такой дизайн диктуется необходимостью реагировать на прикосновение, поскольку активная область должна иметь большой размер, позволяющий избежать ошибок. Текст интерактивной строки также имеет более крупный шрифт. На устройствах, работающих под управлением iOS, рекомендуемая высота для наиболее часто используемых элементов управления составляет 44 пиксела. На других устройствах высота элементов примерно такая же.

Платформа автоматически помещает аккуратную стрелку у правого конца строки, позволяющую пользователю догадаться, что строка реагирует на прикосновение (рис. 4.5). Очень скоро мы увидим, что эту пиктограмму можно заменить с помощью атрибута `data-icon`.

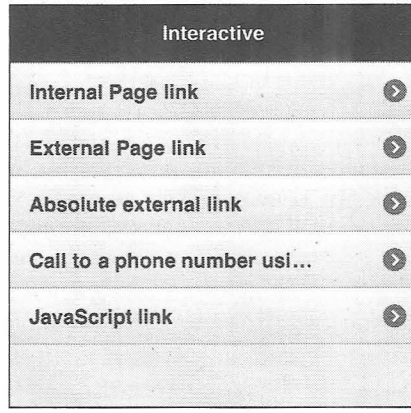


Рис. 4.5. Интерактивные строки получаются в результате помещения гиперссылок в пункты списка, после чего они начинают реагировать на щелчки и прикосновения

В списке можно комбинировать интерактивные строки с обычными. Однако большинство дизайнерских решений сводится к использованию либо полностью интерактивных списков, либо списков вида "только для чтения".

В следующем примере (см. рис. 4.5) определяются интерактивные строки:

```
<!DOCTYPE html>
<html>
<head>
  <!-- Здесь должен быть обычный заголовок jQuery Mobile -->
</head>
<body>
  <div data-role="page" id="main">
    <div data-role="header">
      <h1>Interactive</h1>
      <!-- Интерактивный список -->
    </div>
    <div data-role="content">
      <ul data-role="listview">
        <li><a href="#page2">Internal Page link</a>
        <!-- Ссылка на внутреннюю страницу -->
        <li><a href="other.html">External Page link</a>
        <!-- Ссылка на внешнюю страницу -->
        <li><a href="http://www.mobilexweb.com">Absolute external link</a>
        <!-- Абсолютная внешняя ссылка -->
        <li><a href="tel:+13051010200">Call to a phone number using a link</a>
        <!-- Позвонить с помощью ссылки -->
        <li><a href="javascript:alert('Hi!')">JavaScript link</a>
        <!-- Ссылка JavaScript -->
      </ul>
    </div>
  </div>
```

```
</body>
</html>
```

Как видно из рис. 4.5, при работе с интерактивными строками платформа старается поддерживать одинаковую высоту строк. Поэтому, если текст не умещается на одной строке, как в четвертом пункте списка, jQuery Mobile усекает текст, добавляя многоточие в конец (на устройствах, поддерживающих CSS3). Старайтесь минимизировать длину текста в строках, если у вас нет страницы с подробным текстом.

Элемент `<a>` должен находиться внутри тега `<li>`, и мы не обязаны вставлять в ссылку текст строки. Мы можем оставить ссылку пустой, а результат будет таким, как надо:

```
<li><a href="#page2"> Internal Page link </a>
<!-- Ссылка на внутреннюю страницу -->
```

На устройствах, в которых возможна навигация курсором или переносом фокуса (таких как BlackBerry или Android), пользователь может выполнять навигацию с помощью клавиатуры или трекбола. Активизация строки списка с помощью клавиши **OK** или кнопки трекбола приведет к переходу по ссылке, "спрятанной" в элементе списка. На рис. 4.6 показано, как этот механизм работает на устройствах Android, на которых браузеры, как правило, обозначают фокус оранжевой рамкой на выбранной строке.

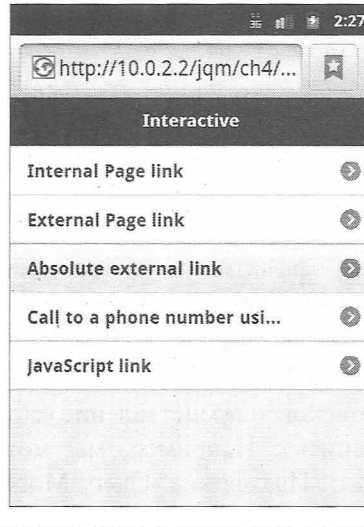


Рис. 4.6. На браузерах, показывающих текущее положение фокуса, переход по элементам интерактивного списка можно осуществлять с помощью клавиш или трекбола

Интерактивные списки являются более предпочтительным инструментом создания ссылок на странице, чем обычные теги `<a>`, поскольку они оптимизированы как под сенсорное управление, так и под навигацию на основе перемещения курсора.

Когда пользователь прикасается к интерактивной строке списка, генерирующей переход на другую страницу в пределах платформы jQuery Mobile, строка выделя-

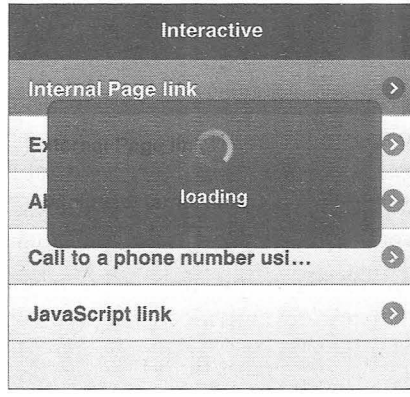


Рис. 4.7. При выделении интерактивной строки платформа меняет ее цветовую схему; например, в теме, выбираемой по умолчанию, фон принимает голубой цвет

ется на то время, пока выполняется переход, или новая страница загружается из сети. Выделенная строка имеет специальную цветовую схему (рис. 4.7).

Мы можем изменить пиктограмму на интерактивной строке с помощью атрибута `data-icon` элемента `<li>`, например:

```
<li data-icon="info"><a href="#moreinfo">More information</a></li>
<!-- Более подробная информация -->
```

Если мы хотим убрать значок из интерактивной строки, то можем указать специальное значение `false` атрибута `data-icon`:

```
<li data-icon="false"><a href="#page2">No icon interactive row</a></li>
<!-- Интерактивная строка без значка -->
```

Вложенные списки

Вложенные списки — очень эффективная функциональная возможность jQuery Mobile. Если у вас есть иерархическая структура или многостраничная схема навигации, например, "континенты → страны → города", вы можете воспользоваться вложенным списком.

Вложенный список — это списковое представление (список со специальной ролью) внутри элемента другого списка. Например, мы можем вставить элемент `<ul>` в элемент `<li>` или другой `<ul>`. Платформа jQuery Mobile неявно сгенерирует страницу для каждого вложенного списка так, словно мы создали эту страницу явным образом, и автоматически добавит ссылки между уровнями.

Вложенный список будет отображаться с темой, отличной от основной, чтобы было очевидно, что на экране список второго уровня. Конечно, как мы увидим позднее, у нас есть возможность определять политику тем самостоятельно с помощью атрибута `data-theme`.

Итак, мы имеем разные уровни навигации с автоматическим созданием только одной страницы. После загрузки платформа jQuery Mobile показывает пункты только

первого уровня, а каждый пункт, содержащий в себе список, будет отображаться интерактивной строкой. Если пользователь выберет ее, произойдет переход на новую страницу и вывод списка следующего уровня с кнопкой, позволяющей вернуться на предыдущий уровень.

Список может иметь неограниченное количество вложенных уровней. Однако если вы создадите слишком много уровней и пунктов, то будет разумно использовать разные страницы, чтобы уменьшить размер объектной модели документа и время начальной загрузки. Вложенные списки рекомендуются только для особых случаев. Не следует создавать весь сайт в виде одного многоуровневого списка.

Конечно, мы можем комбинировать вложенные списки с обычными интерактивными строками. Например, в списке первого уровня некоторые элементы могут быть ссылками на другие документы или страницы, а другие элементы могут содержать вложенные списки.

Важно не забывать добавлять текст в элементы `<li>` в дополнение к элементам `<ul>` и `<ol>`, потому что платформе нужно выводить заголовок интерактивного элемента в неявно сгенерированной странице.

Итак, типичный вложенный список будет выглядеть следующим образом:

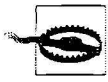
```
<li>Item title
<!-- Заголовок списка -->
  <ul data-role="listview">
  <!-- Элементы вложенного списка -->
</ul>
```

Напишем простой пример, изображенный на рис. 4.8:

```
<!DOCTYPE html>
<html>
  <head>
    <!-- Здесь должен быть обычный заголовок jQuery Mobile -->
  </head>
  <body>
    <div data-role="page" id="main">
      <div data-role="header">
        <h1>Training</h1>
        <!-- Обучение -->
      </div>
      <div data-role="content">
        <ul data-role="listview">
          <li><a href="order.html">Order Now!</a>
          <!-- Закажите прямо сейчас! -->
          <li>Cities available
          <!-- Города, где имеются курсы -->
          <ul data-role="listview">
            <li>Boston
            <!-- Бостон -->
```

```
        <li>New York
        <!-- Нью-Йорк -->
        <li>Miami
        <!-- Майами -->
        <li>San Francisco
        <!-- Сан-Франциско -->
        <li>San Jose
        <!-- Сан-Хосе -->
      </ul>
      <li>Topics
      <!-- Темы -->
      <ul data-role="listview">
        <li>Intro to Mobile Web
        <!-- Введение в мобильную Всемирную паутину -->
        <ul data-role="listview">
          <li>WML and other oldies
          <!-- WML и другие стандарты -->
          <li>XHTML MP
          <li>HTML 4.01
          <li>HTML5
        </ul>
        <li>Mobile Browsers
        <!-- Мобильные браузеры -->
        <ul data-role="listview">
          <li>Safari for iOS
          <!-- Safari для iOS -->
          <li>Android Browser
          <li>Firefox for Mobile
          <!-- Firefox для Mobile -->
          <li>Opera
        </ul>
        <li>Tablet Browsers
        <!-- Браузеры для планшетов -->
        <li>Using jQuery
        <!-- Работа с jQuery -->
      </ul>
      <li>Promotions
      <!-- Специальные предложения -->
      <ul data-role="listview">
        <li>10% off before May
        <!-- Скидка 10% до мая -->
        <li><a href="promo3x2.html">3x2</a>
        <li>10% off to subscribers
        <!-- Скидка 10% подписчикам -->
      </ul>
    </ul>
  </div>
```

```
    </div>
  </body>
</html>
```



Во вложенных таблицах платформа jQuery Mobile автоматически использует текст строки в качестве заголовка сгенерированной страницы. Поэтому вы должны писать как можно более простой и короткий текст, чтобы он уместился в области, отведенной для заголовка.

На рис. 4.8 показаны все навигационные элементы и страницы, которые мы получаем без каких-либо усилий с нашей стороны, когда определяем вложенные списки.

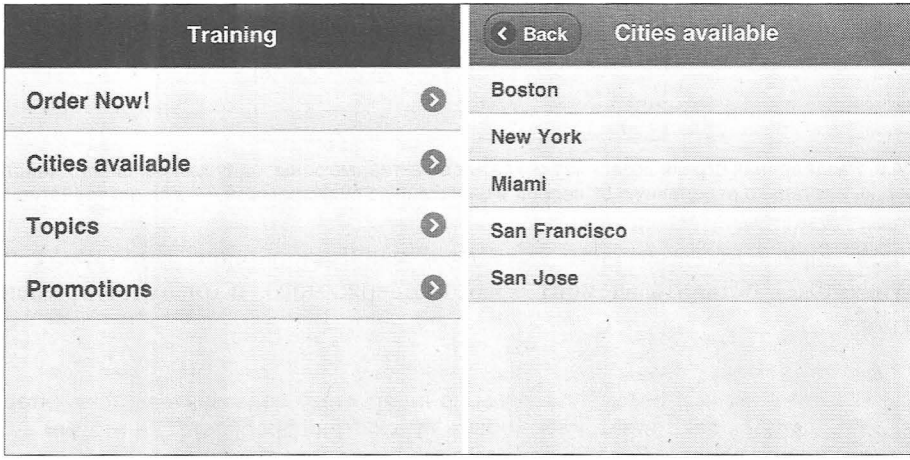


Рис. 4.8. Вложенные списки предлагают способ навигации, аналогичный использованию нескольких страниц, причем нам не нужно создавать эти страницы

С помощью некоторых трюков и манипуляций с хешем URL на платформе jQuery Mobile мы можем создавать ссылки на автоматически сгенерированные страницы. Однако если нам понадобится это сделать, то будет проще явно создать страницу и не связываться с вложенными списками.

Списки с разделенными кнопками

Иногда бывает необходимо реализовать возможность совершения двух действий на одном элементе списка. Самый распространенный сценарий включает в себя возможность увидеть подробности в сочетании с возможностью вносить изменения.

Например, при выводе списка контактов мы можем предложить пользователю два действия — просмотр деталей и редактирование данных. Для этой цели в jQuery Mobile предусмотрены *разделенные строки*. Если элемент списка `<li>` содержит две гиперссылки (два элемента `<a>`), он будет автоматически отображен в виде разделенной строки.

Как видно на рис. 4.9, строка разделяется на две части — левую и правую. Первая ссылка в объектной модели документа будет применяться для первого действия, и она активизируется в левой части строки. Вторая ссылка соответствует альтернативному действию и активизируется в правой части.

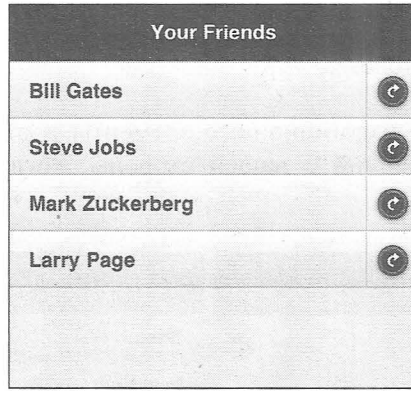


Рис. 4.9. Разделенная строка позволяет создать две активные зоны; одну для основного действия, а другую, визуально отделенную от первой и снабженную пиктограммой, — для альтернативного

Во второй ссылке необязательно указывать какой-либо текст. Впрочем, в первой тоже. Мы может оставить элемент `<a>` без содержимого, и он будет применен ко всей строке.

В соответствии с рекомендациями по построению пользовательского интерфейса iOS, первым действием должен быть просмотр подробностей, а вторым — редактирование данных. Однако, это всего лишь рекомендация.

По умолчанию платформа jQuery Mobile помещает на правой кнопке (активизирующей второе действие) значок стрелки в круглом ободке, отличающейся от стандартной, применяемой в интерактивных строках.

Если мы захотим определить другую тему, для правой пиктограммы, то сможем сделать это с помощью атрибута `datasplit-theme` в теге `<ul>`.

Обсудим пример, изображенный на рис. 4.10.

```
<!DOCTYPE html>
<html>
  <head>
    <!-- Здесь должен быть обычный заголовок jQuery Mobile -->
  </head>
  <body>
    <div data-role="page" id="main">
      <div data-role="header">
        <h1>Your Friends</h1>
        <!-- Ваши контакты -->
      </div>
```

```
      <div data-role="content">
        <ul data-role="listview">
          <li><a href="details/bill">Bill Gates</a>
              <!-- Билл Гейтс -->
              <a href="edit/bill"></a>
          <li><a href="details/steve">Steve Jobs</a>
              <!-- Стив Джобс -->
              <a href="edit/steve"></a>
          <li><a href="details/mark">Mark Zuckerberg</a>
              <!-- Марк Цукерберг -->
              <a href="edit/mark"></a>
          <li><a href="details/larry">Larry Page</a>
              <!-- Ларри Пейдж -->
              <a href="edit/larry"></a>
        </ul>
      </div>
    </div>
  </body>
</html>
```

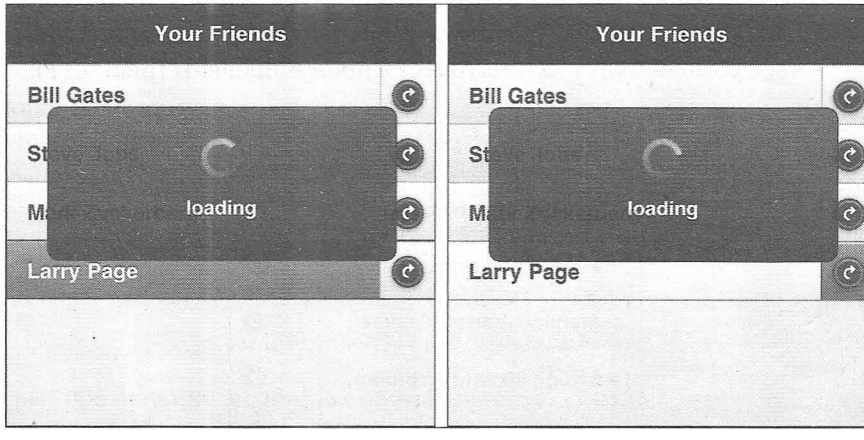


Рис. 4.10. Здесь видно, что каждая зона выделяется индивидуально

Мы можем изменить значок второго действия с помощью модификатора `data-split-icon`, примененного ко всему списку (элементу `<ul>` или `<ol>`), а не только к одному пункту (элементу `<li>`).

В представлении списка используется тот же набор значков, что и для кнопок (в чем мы убедимся в следующих главах), но для отличия от обычных кнопок эти пиктограммы снабжаются круглым ободком.

Значки, используемые в jQuery Mobile 1.0, перечислены в табл. 3.1.

Мы можем использовать собственный набор значков, взяв одно изображение и применив технологию CSS Sprites, аналогично тому, как это сделано в платформе jQuery Mobile.

Как указать степень важности строк

Если мы хотим показать, что некоторые или все строки важнее, чем обычные, то можем вставить заголовок строки в любой тег <hX> (например, в тег <h2> или <h3>). В результате высота строки увеличится.

Если же мы хотим показать, что важность строки невелика (например, она активизирует не самое популярное действие), мы должны будем поместить заголовок строки в тег <p>, чтобы уменьшить ее высоту.

В следующих главах мы научимся более точно настраивать представления списков и свойства строк.

Упорядоченные интерактивные списки

Ранее в этой главе мы говорили о возможности использования элемента <ol> вместо элемента <ul> при определении списков. Мы также убедились, что применение элемента <ol> для списков, доступных только для чтения, не имеет особого эффекта при отображении на экране. Ситуация в корне меняется, когда списки содержат интерактивные строки.

Вначале следует запомнить одно важное требование: чтобы список вел себя корректно, все его строки должны быть интерактивными. Тогда при использовании элемента <ol> все ссылки будут автоматически пронумерованы (рис. 4.11).

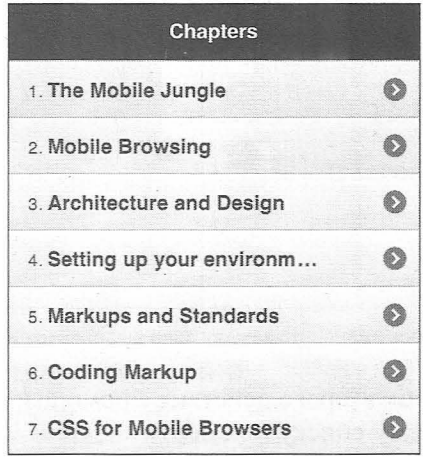


Рис. 4.11. Если мы определим роль `listview` для элемента <ol> с интерактивными строками, то получим пронумерованные строки

Применение изображений

В каждом элементе списка мы можем определить одно изображение для графической идентификации пункта. Существуют два вида изображений, которые мы можем добавлять в строки: значки и миниатюры.

Значки строк

Значок строки — это изображение, выводимое слева от текста. Не следует путать значок строки с кнопкой у правого края интерактивной строки или со значком разделенной строки.

Значок представляет собой изображение размером 16×16 в элементе `<li>`, имеющее класс `ui-li-icon`. Например:

```
<li>
<img src="/icons/email.png" class="ui-li-icon">
Send by e-mail
<!-- Отправить по электронной почте -->
```

Значки обычно используются в списках действий, например в списках, перечисляющих операции с некоторой записью (удалить, редактировать, отправить по электронной почте, поделиться в социальной сети и т. д.).

Миниатюры

Миниатюра — это изображение размером 80×80 пикселов, также размещаемое слева от текста. Миниатюры следует предпочитать значкам, когда мы выводим фотографии, картинки и иную графическую информацию, связанную с пунктами списка.

Миниатюры обычно используются в списках, показывающих записи из базы данных, такие как "друзья", "книги", "диски", "города" и т. д.

Миниатюра определяется как изображение внутри элемента и не имеет специального класса:

```
<li>
<img src="/thumb/washington.png">
George Washington
<!-- Джордж Вашингтон -->
```

Рассмотрим применение значков и миниатюр в двух вставленных списках, изображенных на рис. 4.12.

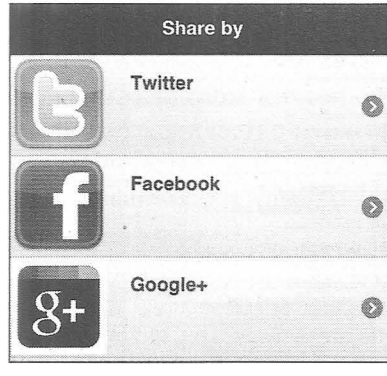


Рис. 4.12. "Лучше один раз увидеть", и поэтому мы можем добавлять значки и миниатюры в каждую строку списка

Дополнительное содержимое

До сих пор каждый создаваемый нами список имел только один столбец для текстового содержимого. Мы можем добавить значок или миниатюру, но только к одному текстовому столбцу. Кроме того, мы можем определить в каждом ряду столбец второго уровня, содержащий дополнительную информацию, которую тоже необходимо вывести.

Для этого годится любой HTML-элемент с классом `ui-li-aside`, например, элемент `<span>` или `<div>`.

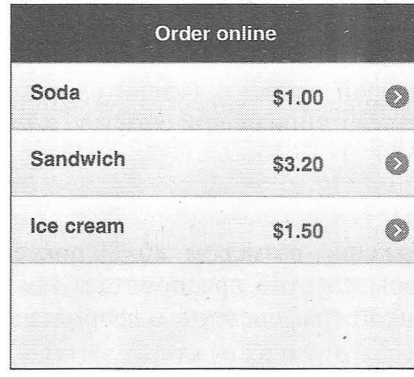


Рис. 4.13. Определив дополнительное содержимое, мы можем выводить больше информации в строке, не создавая новую страницу

Создадим пример, в котором цена товара будет выводиться в качестве дополнительной информации (рис. 4.13):

```
<!DOCTYPE html>
<html>
  <head>
    <!-- Здесь должен быть обычный заголовок jQuery Mobile -->
  </head>
  <body>
    <div data-role="header">
      <h1>Order online</h1>
      <!-- Закажите через Интернет -->
    </div>
    <div data-role="content">
      <ul data-role="listview">
        <li><a href="buy.html">Soda</a>
        <!-- Газированная вода -->
        <span class="ui-li-aside">$1.00</span>
        <li><a href="buy.html">Sandwich</a>
        <!-- Сэндвич -->
        <span class="ui-li-aside">$3.20</span>
        <li><a href="buy.html">Ice cream</a>
        <!-- Мороженое -->
```

```
        <span class="ui-li-aside">$1.50</span>
      </ul>
    </div>
  </body>
</html>
```

Название и описание

Если мы хотим показать название и описание товара в одном пункте списка, то можем воспользоваться каким-либо заголовочным элементом <hX> для названия и элементом <p> для описательного текста. Как видно на рис. 4.14, описание не будет отображаться в виде отдельного столбца.

Конечно, в одной строке мы можем комбинировать название и описание с дополнительным содержимым, а также со значками и миниатюрами.

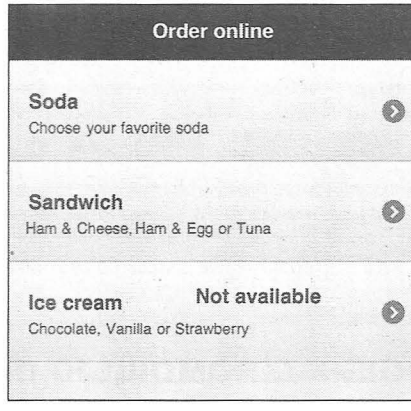
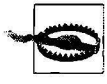


Рис. 4.14. Если мы хотим помимо основного текста вывести длинную строку с описательной информацией, оптимальным решением будет "название и описание", а не "дополнительное содержимое"

Применение счетчиков

Счетчик представляется кружком с числом внутри и выводится в правой части строки. Как правило, он показывает количество пунктов в интерактивных строках. Он также годится для показа количества непрочитанных элементов, невыполненных работ и другой цифровой информации, представляя ее в простом и удобном виде.



Нет никаких ограничений на числовые значения, выводимые счетчиком. Однако использование в нем текста не рекомендуется, потому что место под счетчик оптимизировано для вывода чисел. Если вам нужно показать текст, пользуйтесь техникой вывода дополнительного содержимого или описания.

Для счетчика можно использовать любой элемент, например , с классом ui-li-count, помещенный в элемент списка.

Например:

```
<li><a href="inbox.html">Inbox</a>
<!-- Входящие -->
<span class="ui-li-count">86</span>
```

На рис. 4.15 показано, как счетчики отображаются в интерактивном списке. Цвет счетчика можно определить в атрибуте data-count-theme элемента <ul>, что позволяет сменить белый цвет фона, принимаемый по умолчанию.

Рис. 4.15. Интерактивные списки со счетчиками

Фильтрация данных с помощью поиска

Самое интересное я приберег на конец главы. Отложите книгу ненадолго. Если вы читаете ее в электронном виде, закройте приложение для чтения или браузер. Выберите какой-нибудь пример списка из тех, что вы создали, читая эту главу. Найдите в нем элемент <ul> или <ol> и укажите для него атрибут data-filter="true". Протестируйте пример и возвратитесь к чтению.

Волшебство jQuery Mobile сработало. После добавления этого простого атрибута в верхней части списка (полностраничного или вставленного) автоматически появилось окно поиска, и оно работает!

Эта функция фильтрует элементы нашего списка с учетом того, что вводит пользователь. Поле для ввода искомого текста выглядит очень аккуратно (рис. 4.16): значок поиска у левого края, текст с подсказкой, закругленные углы и кнопки очистки поля у правого края.

Повторюсь: для того чтобы это работало, достаточно написать код, приведенный далее. Срочно посетите сайт этой книги и скопируйте исходный код, ведь набирать его вручную невыносимо трудно:

```
<ul data-role="listview" data-filter="true">
  <!-- пункты списка -->
</ul>
```

Мы можем менять текст подсказки в поле поиска с помощью атрибута `data-filter-placeholder`, например:

```
<ul data-role="listview" data-filter="true"
    data-filter-placeholder="Search contacts...">
  <!-- пункты списка -->
</ul>
```

Чтобы изменить образец цвета поля поиска, следует написать атрибут `data-filtertheme`.

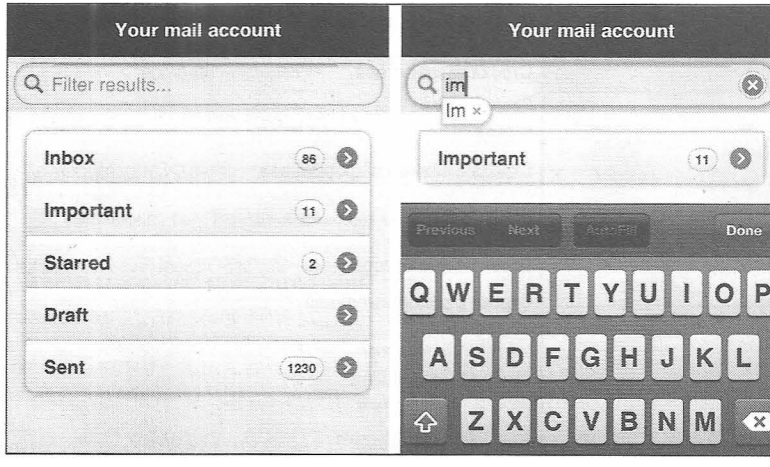


Рис. 4.16. Фильтры поиска — волшебная функция, добавляющая систему фильтрации и не требующая от вас никакого кода

Шпаргалка по представлениям списка

В этой главе мы обсудили много функциональных особенностей списков. На рис. 4.17 представлена шпаргалка, подсказывающая, как будут отображаться ваши списки, созданные с помощью разметки HTML.

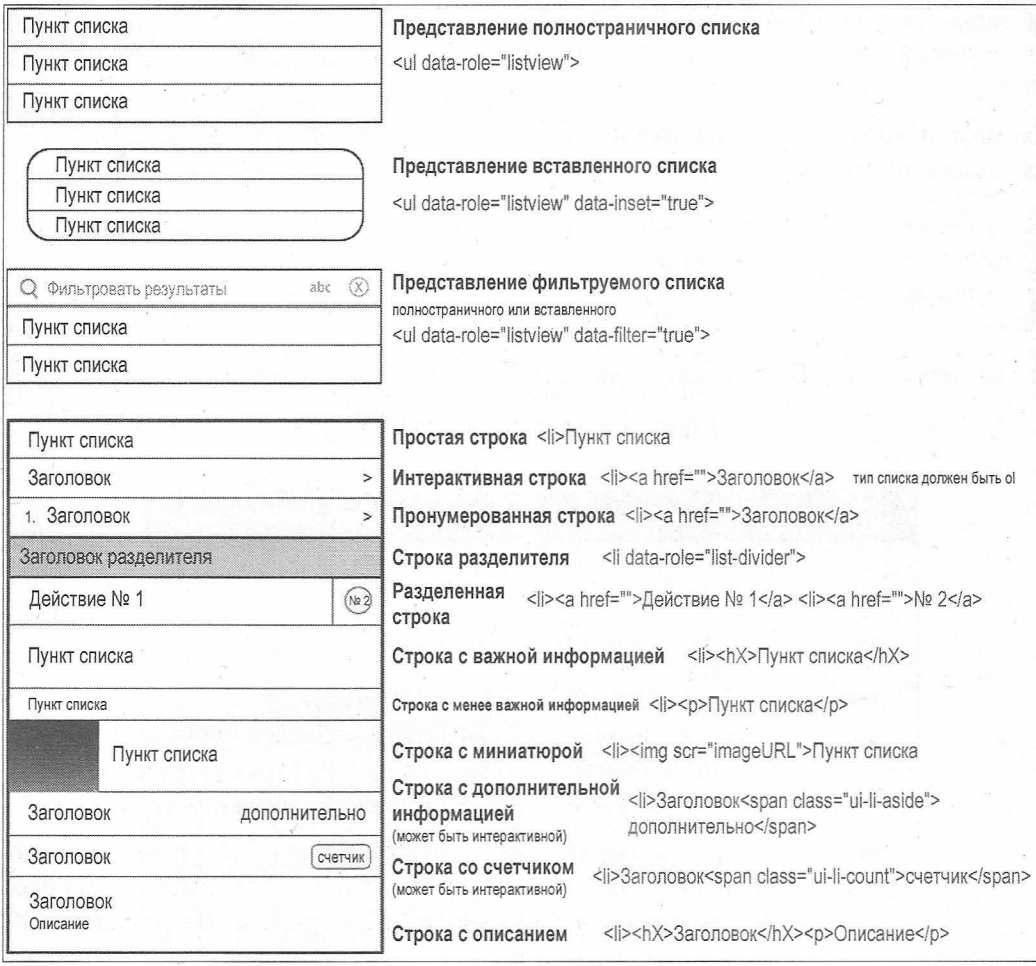


Рис. 4.17. Все варианты представления списка на одной схеме

ГЛАВА 5

Компоненты формы

Платформа jQuery Mobile поддерживает стандартные веб-формы с автоматической обработкой данных по технологии AJAX на совместимых устройствах и с отображением элементов формы, оптимизированным под сенсорные устройства. Это первая хорошая новость относительно элементов формы.

Когда я говорю о стандартной веб-форме, то имею в виду группу элементов формы, таких как `<input>`, `<textarea>` и `<select>`, помещенных в элемент `<form>` с атрибутом `action`, определенным как URL-адрес, на который будут отправлены данные из формы.

Действие формы

Любое действие формы будет выполнено так, как мы ожидаем. То есть данные будут отправлены, когда пользователь нажмет кнопку **Отправить** или клавишу <Enter>. Неужели я произнес "клавиша <Enter>", говоря о мобильном устройстве?

Мы пишем приложения для самых разных устройств с различными способами ввода. У некоторых имеются физические клавиатуры (не исключено, что с клавишей <Enter>); у других — только виртуальные клавиатуры на экране. На устройствах с виртуальной клавиатурой обычно есть физическая кнопка, действующая как кнопка **Отправить**, когда пользователь вводит данные формы.

Когда пользователь отправляет форму, jQuery Mobile выполняет AJAX-переход на страницу отправки. Это аналогично переходу по ссылке на внешнюю страницу, если форма не отправляется в другой домен. Передача данных может осуществляться методом `get` или `post`, который определен атрибутом `action` элемента `<form>`.

Типичная форма jQuery Mobile выглядит аналогично типичной веб-форме:

```
<form action="send.php" action="get">
</form>
```

Форме jQuery Mobile необходимо присутствие документа jQuery Mobile на странице результатов, и в этом наблюдается аналогия с внешней ссылкой. Дело в том, что по умолчанию платформа jQuery Mobile пытается отправить форму по технологии AJAX, выполняемой на платформе.

Страница результатов будет добавлена в журнал браузера, если данные передаются методом `get` с применением хеша адреса.

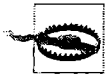
К элементам <form> можно применять атрибуты data-transition и data-direction, о которых мы говорили в *главе 2*. Таким образом, мы можем добиться, чтобы действие по отправке формы предоставлялось в виде всплывающего окна перехода:

```
<form action="send.php" action="get" data-transition="pop">
</form>
```

Форма без использования AJAX

Если мы хотим принудительно отправить HTTP-запрос без применения AJAX, то можем добавить к элементу <form> атрибут data-ajax="false". Это особенно удобно, когда обработка происходит на другом домене или когда выполняется выгрузка файла на сервер (подробности см. далее в этой главе).

Другим способом принудительной отправки без использования AJAX является указание атрибута target="_blank"для элемента <form>.



У нас есть возможность определить булево значение autofocus для элемента формы, принимающего ввод пользователя, чтобы этот элемент получил фокус после загрузки страницы. Однако в документе jQuery Mobile этот атрибут сработает только на первой странице и не будет действовать на страницах, загружаемых в результате переходов по ссылкам jQuery Mobile.

Элементы формы

Платформа jQuery Mobile позволяет размещать на одной форме как стандартные элементы веб-форм, так и новые, более богатые функциональными возможностями. Платформа имеет функцию, называемую автоинициализацией, действие которой заключается в замене каждого элемента формы элементом, дружественным по отношению к сенсорным способам взаимодействия.

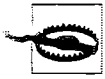
Кроме того, платформа переводит новые элементы ввода HTML5 на более прогрессивный уровень, поддерживая их даже на браузерах, не имеющих официальной поддержки.

Следующие элементы будут отображаться как "продвинутые" элементы управления:

- кнопки, определяемые элементами <button> и <input>;
- текстовые поля, определяемые элементами <input> и <textarea>;
- флажки и переключатели, определяемые элементами <input>;
- меню, определяемые элементами <select> и <option>;
- ползунковые регуляторы, определяемые новым элементом <input type="range">;
- ползунковые переключатели, определяемые элементами <select> и <option> с новой ролью.

Если мы не хотим, чтобы платформа jQuery Mobile отображала элемент формы в виде элемента с более богатой функциональностью, то можем запретить это, указав атрибут `data-role="none"` для каждого элемента формы.

Элемент формы занимает одну строчку и не делит ее ни с каким другим элементом. Это оптимальное решение для мобильных форм. Мы можем искусственно выводить элементы в несколько столбцов, но делать это не рекомендуется.



При использовании технологии AJAX все страницы, в том числе элементы веб-формы, имеют одну общую объектную модель документа. Поэтому мы должны давать элементам формы идентификаторы, уникальные на уровне сайта. Если вы создаете две формы, не используйте одинаковые идентификаторы для схожих элементов.

Метки

Метка (надпись) — очень важная составляющая любого элемента формы. Мы должны следить за тем, чтобы элемент `<label>` указывал на соответствующий элемент формы в своем атрибуте `for`. Рассмотрим пример:

```
<label for="company">Company Name:</label>
<!-- Название фирмы -->
<input type="text" id="company">
```

Самое важное, что мы должны знать о метках, — это то, что после щелчка по метке соответствующий элемент формы получает фокус, и пользователь может работать с ним. Иными словами, активная область элемента включает в себя сам элемент и его метку, что неплохо для сенсорных устройств.

Если по какой-либо причине вам потребуется элемент формы без метки, вы все равно не должны забывать о доступности интерфейса для всех пользователей. Поэтому вы всегда должны создавать метку, а если хотите скрыть ее, примените класс `ui-hiddenaccessible` к ней или к любому другому HTML-элементу.

Контейнеры полей

Контейнер поля — это необязательный элемент-оболочка для метки и соответствующего компонента, который обеспечивает более качественный пользовательский опыт на широких экранах, например планшетных. Контейнером может быть любой блочный элемент с атрибутом `data-role="fieldcontainer"`.

Этот контейнер аккуратно выравнивает метки и элементы формы и рисует тонкую рамку в качестве разделителя полей. Другой важной особенностью контейнера является автоматическое определение компоновки в зависимости от ширины экрана. Говоря конкретно, если устройство имеет узкий экран, метка размещается над элементом формы; в противном случае она выводится слева от него, образуя отдельный столбец (рис. 5.1).

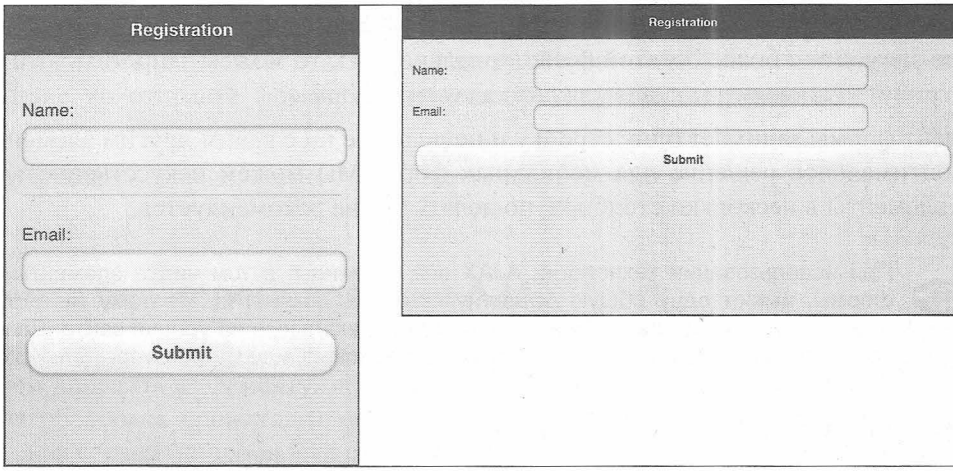


Рис. 5.1. Метка имеет разное расположение на экране смартфона и планшета в соответствии с общеизвестными принципами дизайна мобильного интерфейса

Мы можем создать контейнер поля для каждой пары "метка — элемент", как показано в следующем фрагменте кода:

```
<div data-role="fieldcontainer">
  <label for="company">Company Name:</label>
  <!-- Название фирмы -->
  <input type="text" id="company" name="company">
</div>
<div data-role="fieldcontainer">
  <label for="email">Email:</label>
  <!-- Электронная почта -->
  <input type="email" id="email" name="email">
</div>
```

Текстовые поля

Платформа jQuery Mobile поддерживает базовые элементы HTML5 для ввода текста и отображает их в соответствии с текущей темой и образцом цвета. Доступны следующие текстовые поля:

- `<input type="text">`;
- `<input type="password">`;
- `<input type="email">`;
- `<input type="tel">`;
- `<input type="url">`;
- `<input type="search">`;
- `<input type="number">`;
- `<textarea>`.

Написав любой из этих элементов в коде, мы автоматически получим элемент управления jQuery Mobile. Помните, что мы не должны явно указывать атрибут `data-role`.

Каждый элемент `<input>` со значением типа `button`, `submit`, `clear` и `image` будет автоматически отображен в виде кнопки jQuery Mobile.

Если вам знакомы только типы `text` и `password`, не расстраивайтесь. Типы `email`, `tel`, `url`, `search` и `number` — это новые типы в спецификации HTML5, весьма важные в мобильном мире. Как вам известно, у большинства мобильных устройств нет физической клавиатуры. Когда мы определяем один из этих типов, ему ставится в соответствие специфическая оптимизированная виртуальная клавиатура (рис. 5.2).

Рис. 5.2. С помощью новых типов ввода текста, появившихся в HTML5, мы получаем оптимизированные виртуальные клавиатуры на некоторых мобильных устройствах

Не стоит беспокоиться по поводу представления вашей формы на старых устройствах, не поддерживающих новые типы ввода, определенные в HTML5. Браузер, не понимающий атрибут `type`, отобразит базовое текстовое поле.

Тип `search` имеет два отличия от обычного типа элемента ввода: особый интерфейс на платформе jQuery Mobile и особую клавишу "ввода" на виртуальной клавиатуре некоторых устройств (рис. 5.3).

```
<div data-role="fieldcontainer">
  <label for="search">Search:</label>
  <!-- Поиск -->
  <input type="search" id="search" name="search">
</div>
```

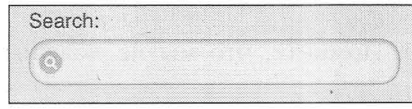


Рис. 5.3. Поле поиска имеет специальный внешний вид в документе jQuery Mobile

Автоматически увеличивающаяся текстовая область

Когда мы выполняем ввод многострочного текста, то получаем дополнительную услугу: автоматическое увеличение поля. Платформа jQuery Mobile начинает с показа двухстрочного поля. Когда мы введем текст, длина которого потребует создания третьей строки, платформа автоматически расширит текстовую область, чтобы поместилась новая строка. Это позволяет перемещать курсор вверх-вниз в поле `<textarea>`, избавляя пользователя от прокрутки, весьма неудобной на большинстве сенсорных мобильных устройств.

```
<div data-role="fieldcontainer">
  <label for="comments">Your comments:</label>
  <!-- Ваш комметарий -->
  <textarea id="comments" name="comments></textarea>
</div>
```

На рис. 5.4 показано, как увеличивается область текстового ввода.

Вопросы безопасности

Необходимость в поле для ввода пароля (`<input type="password" />`) в мобильных устройствах вызывает жаркие споры. Изначально это поле (с классическими кружочками и звездочками вместо вводимых букв) было создано для защиты пароля или иных важных данных от взгляда человека, стоящего позади пользователя и смотрящего на экран. В мобильной экосистеме ситуация принципиально иная. При малом размере экрана и мелком шрифте постороннему человеку трудно увидеть, что вводит пользователь на своем мобильном телефоне. Более того, ввод символов с нестандартной клавиатуры затруднителен, а если на экране отображаются только звездочки, пользователь может быть не уверен, правильно ли он ввел текст (даже если символ показывается на секунду, а потом заменяется на звездочку, что происходит в некоторых устройствах). Если вы по-прежнему хотите применять стандартный элемент для ввода пароля, я рекомендую вам разрешить использование только цифровых паролей. Якоб Нильсен (Jakob Nielsen), гуру в области веб-юзабилити, придерживается того же мнения (**http://useit.com**). В своей колонке Alertbox он писал еще в 2009 году: "Юзабилити страдает, когда пользователи вводят пароли, имея в качестве обратной связи только ряд звездочек или кружочков. Как правило, маскировка паролей ничуть не повышает безопасность, но требует дополнительных усилий из-за ошибок при аутентификации". В системе Facebook mobile реализована такая функциональность: если вы допустили ошибку при вводе пароля, при следующей попытке вы будете видеть вводимые символы, причем на экране появится надпись, что пароль по-прежнему в безопасности, хотя и виден. Другое возможное решение заключается в том, чтобы использовать для ввода пароля обычное текстовое поле, сопровождаемое флажком с меткой "Скрыть пароль". Когда флажок установлен, для ввода пароля будет использоваться поле типа `password`.

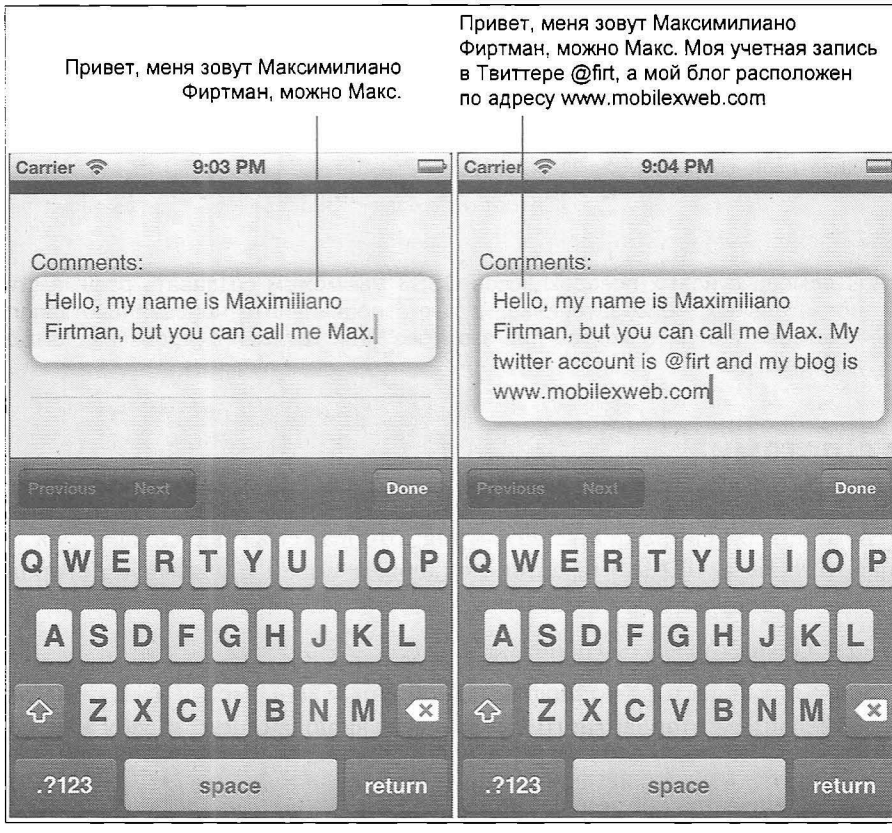


Рис. 5.4. Платформа jQuery Mobile автоматически расширяет текстовую область, когда тексту требуются дополнительные строчки

Новые атрибуты HTML5

Вы можете без опасений использовать атрибуты HTML5 в текстовых полях, поскольку они будут корректно обработаны на совместимых браузерах и проигнорированы на браузерах старых версий. Среди новых атрибутов элементов формы следует упомянуть `required`, `pattern`, `placeholder`, а также `min` и `max` (имеющие смысл только для элементов `<input type="number">`).

Таким образом, обязательные поля могут быть помечены булевым атрибутом `required`. Кроме того, мы можем определить атрибут `placeholder` — текст, выводимый в поле серым шрифтом в качестве подсказки, пока пользователь не начал вводить данные:

```
<div data-role="fieldcontainer">
  <label for="name">Your Name:</label>
  <!-- Имя: -->
  <input type="text" id="name" required placeholder="Enter your name">
  <!-- Введите свое имя -->
</div>
```

```
<div data-role="fieldcontainer">
  <label for="age">Your Age:</label>
  <!-- Возраст -->
  <input type="number" id="age" required placeholder="Enter your age" min="10"
Max="110">
  <!-- Введите свой возраст -->
</div>
```

С помощью новых псевдоклассов CSS3 мы можем создавать разные стили для обязательных, необязательных, а также правильно и неправильно заполненных текстовых полей, обходясь при этом без кода JavaScript, проверяющего допустимость ввода.

Поля с датами

Ввод дат в HTML-формах всегда представлял сложную задачу, и мы обычно прибегали к помощи библиотек JavaScript для отображения календарей на HTML-страницах. Теперь в HTML5 имеется поддержка полей с датами со стороны элемента `<input>` со следующими типами:

- `date` — для селектора даты (день, месяц, год);
- `datetime` — для селектора полноформатной даты (день, месяц, год, часы, минуты) с использованием стандартного синтаксиса, включающего в себя часовой пояс GMT;
- `time` — для селектора времени (часы, минуты);
- `datetime-local` — для селектора полноформатной даты без указания часового пояса;
- `month` — для селектора месяца (обычно отображаемого в виде раскрывающегося списка);
- `week` — для селектора недели (обычно отображаемого в виде раскрывающегося списка).

Типы для ввода дат являются новыми и реализованы не на всех браузерах в мобильных устройствах и настольных компьютерах. На момент работы над этой книгой большинство этих типов корректно работало на браузерах Safari для iOS, начиная с версии 5.0, BlackBerry Browser для PlayBook и смартфонов, начиная с версии 5.0, Opera Mobile и Firefox для Android. Более свежую информацию о совместимости вы можете найти на сайте **http://mobilehtml5.org**.

Даже если вы явно не указываете атрибут `data-role`, вы должны помнить, что элементы ввода отображаются на экране как элементы jQuery Mobile. Поэтому глобальные атрибуты, например `data-theme` для определения образца цвета, будут работать по-прежнему.

Как мы помним, если браузер не поддерживает элемент `<input>`, он автоматически будет воспринимать обычный текстовый ввод:

```
<div data-role="fieldcontainer">
  <label for="birth">Your Birthdate:</label>
  <!-- Ваша дата рождения -->
  <input type="date" id="birth" name="birth">
  <label for="favmonth">Your favorite month:</label>
  <!-- Ваш любимый месяц -->
  <input type="month" id="favmonth" name="favmonth">
</div>
```

Рис. 5.5. В системе iOS, начиная с версии 5, имеется поддержка элементов с датой, так что пользователь получает удобный селектор дат, когда мы применяем эти элементы HTML5

Ползунковый регулятор

Ползунковый регулятор — идеальный элемент для ввода числовых значений, ограниченный некоторым диапазоном. Он отображается числовым текстовым полем и горизонтальным регулятором справа от него (рис. 5.6). Чтобы использовать ползунковый регулятор в приложении, мы должны определить элемент `<input type="range">` из стандарта HTML5. Он принимает атрибуты `min`, `max` и `step`:

```
<div data-role="fieldcontainer">
  <label for="qty">Quantity:</label>
  <!-- Количество -->
  <input type="range" id="qty" name="qty" min="1" max="100" value="5">
</div>
```

Ползунковый регулятор поддерживает определение образца цвета атрибутом `data-theme` для числового поля. Кроме того, мы можем определить атрибут `data-track-theme`, относящийся только к шкале регулятора (см. рис. 5.6):

```
<div data-role="fieldcontainer">
  <label for="qty">Quantity:</label>
  <!-- Количество -->
  <input type="range" id="qty" name="qty" min="1" max="100"
         value="5" data-theme="e" data-track-theme="b">
</div>
```

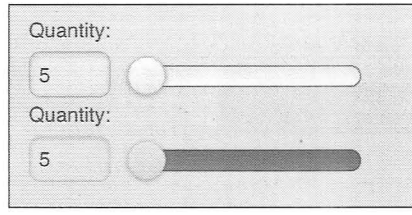


Рис. 5.6. Элемент для ввода чисел из диапазона отображается платформой jQuery Mobile в виде маленького числового поля и синхронизированного с ним горизонтального регулятора в соответствии со стандартом HTML5

Ползунковый регулятор поддерживает события клавиатуры на совместимых устройствах. Следовательно, когда этот элемент в фокусе, пользователь может увеличивать или уменьшать значение в поле с помощью клавиш со стрелками, колесика или джойстика.

Двухпозиционный переключатель

Двухпозиционный переключатель (тумблер) — это селектор булевых значений ("истина" или "ложь", "включено" или "выключено"), аналогичный флажку по функциональности, но радикально отличающийся от него по интерфейсу. По виду он похож на физический переключатель, который может быть переведен в другое состояние постукиванием или перетаскиванием.

Это первый элемент формы, для которого вы должны явно указать значение `slider` в атрибуте роли `data-role`. Для него требуется элемент `<select>` с двумя элементами `<option>` в качестве потомков: первый для значения "выключено"/"ложь", а второй — для "включено"/"истина":

```
<label for="updated">Receive updates</label>
  <!-- Следить за обновлениями -->
<select id="updated" name="updated" data-role="slider">
  <option value="no">No</option>
  <!-- Нет -->
  <option value="yes">Yes</option>
  <!-- Да -->
</select>
```

Если не определен контейнер поля, двухпозиционный переключатель отображается во всю ширину страницы. Если же для него имеется контейнер, он принимает более привычные очертания (рис. 5.7).

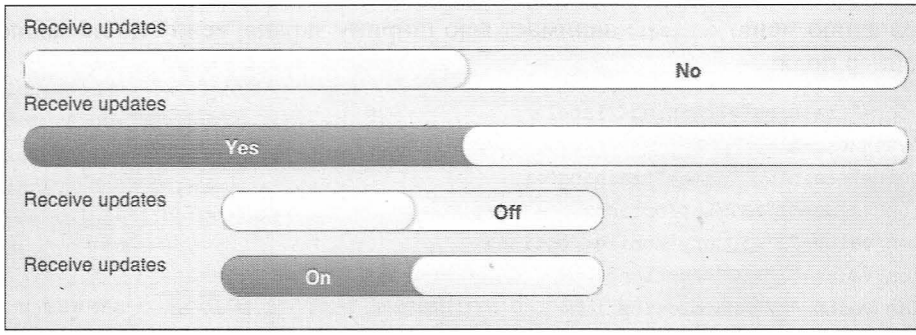
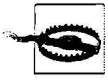


Рис. 5.7. Двухпозиционный переключатель, помещенный в контейнер, не занимает всю ширину страницы



Селектор цвета `<input type="color">`, определенный в HTML5, не работает на большинстве мобильных браузеров и отображается как обычный элемент ввода текста.

Меню

Меню, реализованные элементами `<select>`, являются распространенным элементом формы, позволяющим выбрать один или несколько вариантов в раскрывающемся списке. Каждый мобильный браузер поддерживает такие меню (с возможностью выбора одного или нескольких пунктов). Платформа jQuery Mobile заменяет привычный интерфейс элемента `<select>` кнопочным элементом, вызывающим на экран меню, реализованное в собственных кодах устройства (рис. 5.8).

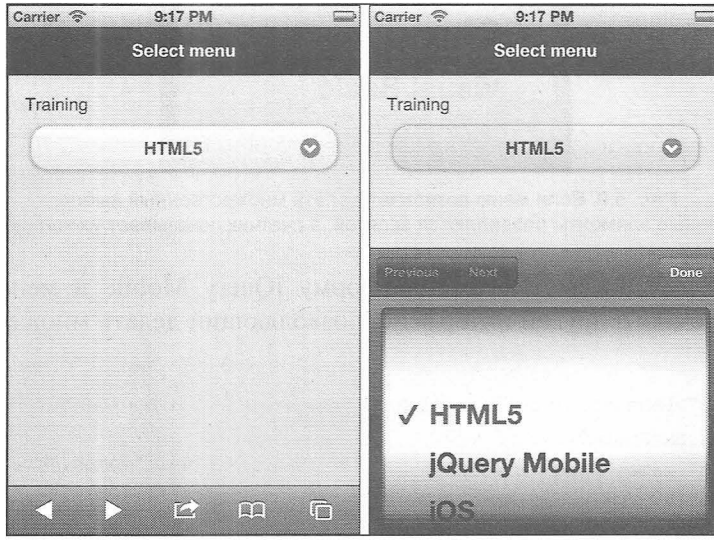


Рис. 5.8. Меню автоматически представляется в виде кнопочного элемента со значком, приглашающим пользователя открыть меню

По умолчанию меню <select> занимает всю ширину экрана, если оно не заключено в контейнер поля:

```
<label for="training">Training</label>
<!-- Обучение -->
<select id="training" name="training">
  <option value="1">HTML5</option>
  <option value="2">jQuery Mobile</option>
  <option value="3">iOS</option>
  <option value="4">Android</option>
  <option value="5">BlackBerry</option>
  <option value="6">Qt for Meego</option>
</select>
```

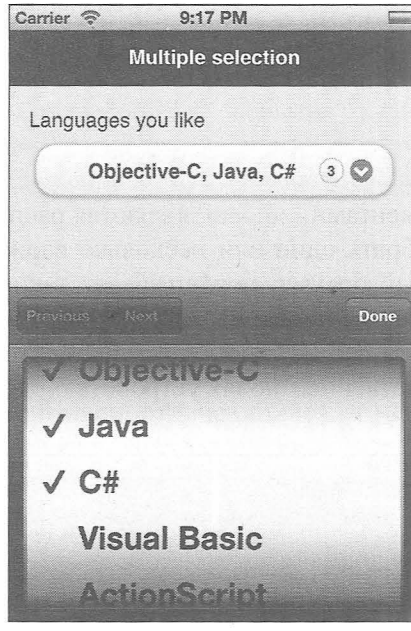


Рис. 5.9. Если меню позволяет делать множественный выбор, выбранные элементы разделяются запятой, а счетчик показывает их количество

Булев атрибут multiple заставляет платформу jQuery Mobile и меню устройства представлять немного другой интерфейс, позволяющий делать множественный выбор (рис. 5.9):

```
<label for="lang">Languages you like</label>
<!-- Какие языки вы предпочитаете -->
<select id="lang" name="lang" multiple>
  <option value="1">C/C++</option>
  <option value="2">Objective-C</option>
  <option value="3">Java</option>
  <option value="4">C#</option>
```

```
  <option value="5">Visual Basic</option>
  <option value="6">ActionScript</option>
  <option value="7">Delphi</option>
  <option value="8">Phyton</option>
  <option value="9">JavaScript</option>
  <option value="10">Ruby</option>
  <option value="11">PHP</option>
</select>
```

Как и в случае с любым другим элементом формы, мы можем изменить образец цвета, принимаемый по умолчанию, с помощью атрибута `data-theme`, при этом элемент `<optgroup>` позволяет группировать элементы, предоставляющие варианты выбора.

Если мы с помощью кода JavaScript изменим параметры элемента формы, например атрибут `value` у элемента-меню или атрибут `checked` у переключателя или флажка, то jQuery Mobile не обновит элемент, пока мы не обновим виджеты с помощью API jQuery Mobile. Эта тема обсуждается в следующей главе.

Группирование элементов меню

Мы можем группировать элементы меню вертикально или горизонтально с помощью элемента `controlgroup`. Горизонтальная компоновка удобна для коротких элементов, например, день/месяц/год. Чтобы сгруппировать элементы меню, необходимо вложить их в элемент `<div>` с атрибутом `data-role="controlgroup"`, как показано в следующих примерах. Результат изображен на рис. 5.10.

```
<div data-role="controlgroup">
  <legend>Color and Size</legend>
  <!-- Цвет и размер -->
  <select id="color" name="color">
    <option value="1">Blue</option>
    <!-- Синий -->
    <option value="2">White</option>
    <!-- Белый -->
    <option value="3">Red</option>
    <!-- Красный -->
    <option value="4">Black</option>
    <!-- Черный -->
    <option value="5">Pink</option>
    <!-- Розовый -->
  </select>
  <select id="size" name="size">
    <option value="1">X-Small</option>
    <!-- Очень маленький -->
    <option value="2">Small</option>
    <!-- Маленький -->
```

```
    <option value="3">Medium</option>
    <!-- Средний -->
    <option value="4">Large</option>
    <!-- Большой -->
    <option value="5">X-Large</option>
    <!-- Очень большой -->
  </select>
</div>
<div data-role="controlgroup" data-type="horizontal">
  <legend>Week day and time</legend>
  <!-- День недели и время -->
  <select id="weekday" name="weekday" multiple>
    <option value="1">Mon</option>
    <!-- Понедельник -->
    <option value="2">Tue</option>
    <!-- Вторник -->
    <option value="3">Wed</option>
    <!-- Среда -->
    <option value="4">Thu</option>
    <!-- Четверг -->
    <option value="5">Fri</option>
    <!-- Пятница -->
  </select>
  <select id="time" name="time">
    <option value="1">Morning</option>
    <!-- Утром -->
    <option value="2">Midday</option>
    <!-- В середине дня -->
    <option value="3">Afternoon</option>
    <!-- Во второй половине дня -->
  </select>
</div>
```

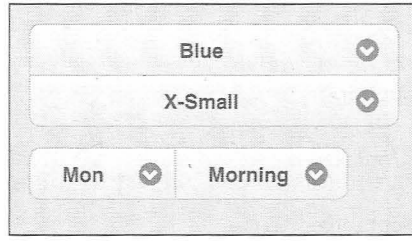


Рис. 5.10. Мы можем группировать элементы меню с помощью контейнера, имеющего роль `controlgroup`

Когда используется элемент с ролью `controlgroup`, отдельные метки убраны из пользовательского интерфейса. Чтобы определить метку для целой группы, можно применить элемент `<legend>`, как показано в следующем фрагменте кода, которому соответствует рис. 5.11:

```
<div data-role="controlgroup" data-type="horizontal">
  <legend>Delivery options</legend>
  <!-- Условия доставки -->
  <label for="weekday">Week day</label>
  <!-- День недели -->
  <select id="weekday" name="weekday" multiple>
    <option value="1">Mon</option>
    <!-- Понедельник -->
    <option value="2">Tue</option>
    <!-- Вторник -->
    <option value="3">Wed</option>
    <!-- Среда -->
    <option value="4">Thu</option>
    <!-- Четверг -->
    <option value="5">Fri</option>
    <!-- Пятница -->
  </select>
  <label for="time">Time</label>
  <!-- Время -->
  <select id="time" name="time">
    <option value="1">Morning</option>
    <!-- Утром -->
    <option value="2">Midday</option>
    <!-- В середине дня -->
    <option value="3">Afternoon</option>
    <!-- Во второй половине дня -->
  </select>
</div>
```

Рис. 5.11. Короткие элементы меню можно компоновать горизонтально

Нестандартные меню

Как следует из рис. 5.10, каждый элемент `<select>` отображается по умолчанию в виде элемента управления jQuery Mobile, но если его открыть, его внешний вид зависит от представления элемента `<select>` мобильным браузером. На случай, если мы захотим переопределить это поведение, платформа jQuery Mobile предусматривает альтернативный пользовательский интерфейс для меню.

Для активизации этой функциональной возможности следует поставить атрибут `data-native-menu="false"` в элемент `<select>`. На рис. 5.12 видно, что закрытый элемент выглядит как прежде, но если его открыть, на экране появляется совсем иной интерфейс:

```
<label for="training">Training</label>
<!-- Обучение -->
<select id="training" name="training" data-native-menu="false">
  <option value="1">HTML5</option>
  <option value="2">jQuery Mobile</option>
  <option value="3">iOS</option>
  <option value="4">Android</option>
  <option value="5">BlackBerry</option>
  <option value="6">Qt for Meego</option>
</select>
```

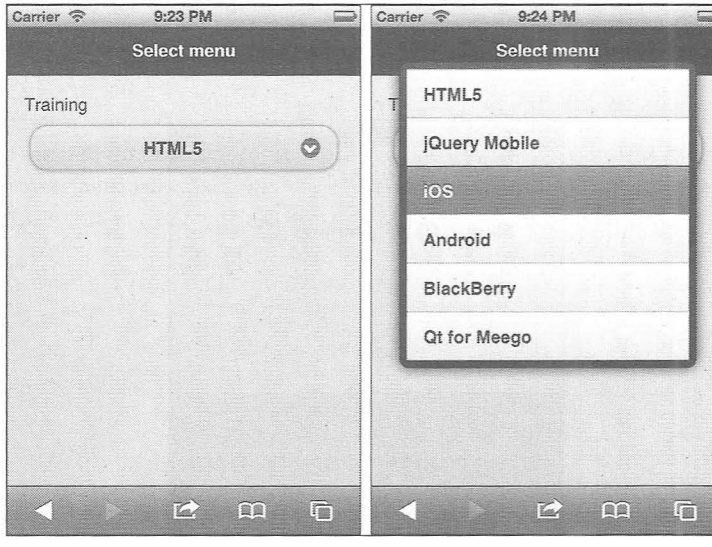


Рис. 5.12. При использовании нестандартного меню оригинальное меню заменяется интерактивным списком в режиме, напоминающем работу с диалоговой страницей

Внешний вид элемента управления будет другим и в том случае, когда список пунктов меню такой длинный, что не умещается на экране. Тогда появляется нечто вроде диалогового окна (рис. 5.13).

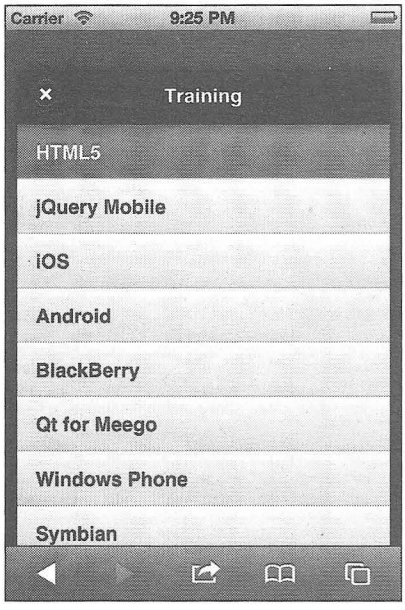


Рис. 5.13. Если пунктов меню очень много, при его открытии создается диалоговая страница

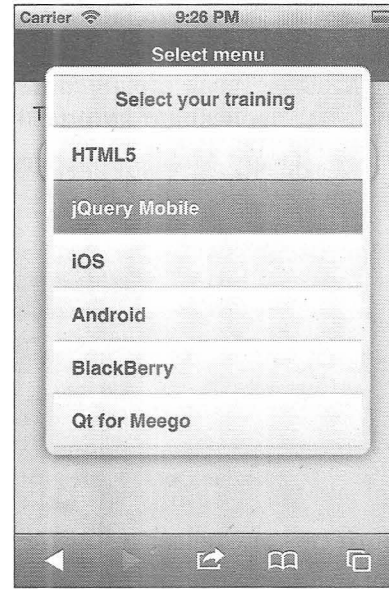


Рис. 5.14. Атрибут `placeholder` позволяет определить заголовок диалоговой страницы в случае нестандартного меню, представляемого в виде списка

При отображении нестандартного меню элементы `<option>` выводятся в виде разделителей списка, как это происходит в списковом представлении.

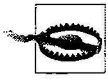
Нестандартные меню, будучи открытыми, перекрывают форму. Атрибут `data-overlay-theme` позволяет определить цветовой образец этого перекрытия. Если в меню присутствует элемент `<option>` с явно указанным пустым значением (`value=""`) или с атрибутом `data-placeholder="true"`, он будет представлен как заголовок перекрывающей области, а не как один из пунктов меню. Рассмотрим пример кода и его представление на рис. 5.14:

```
<label for="training">Training</label>
<!-- Обучение -->
<select id="training" name="training"
        data-native-menu="false" data-overlay-theme="e">
  <option value="0" data-placeholder="true">Select your training</option>
  <!-- Выберите курс -->
  <option value="1">HTML5</option>
  <option value="2">jQuery Mobile</option>
  <option value="3">iOS</option>
  <option value="4">Android</option>
  <option value="5">BlackBerry</option>
  <option value="6">Qt for Meego</option>
</select>
```

На рис. 5.15 показано, что нестандартное меню прекрасно работает с меню множественного выбора, если определен булев атрибут `multiple`. Когда атрибут `multiple` определен, диалоговая страница перекрытия имеет закрывающую кнопку; в противном случае происходит автоматическое закрытие меню после выбора элемента.

Кроме того, jQuery Mobile корректно распознает и обрабатывает атрибут `disabled` у элементов `<option>`.

Рис. 5.15. Нестандартные меню с множественным выбором имеют интерфейс списка, позволяющего выбирать несколько пунктов



Чтобы нестандартное меню работало корректно, на элементы `<option>` накладывается одно ограничение. Они должны явно содержать атрибут `value` с номером варианта. Если вы определите пустой атрибут `value`, элемент будет использован в качестве заголовка всплывающего окна.

Переключатели

Мы все знаем, что такое переключатели. Замечательной особенностью переключателей в jQuery Mobile является тот факт, что они отображаются без каких-либо усилий с нашей стороны. Вначале перечислю требования, которым должны удовлетворять переключатели для корректной работы в jQuery Mobile:

- каждый переключатель должен определяться элементом `<input type="radio">`;
- все переключатели в одной строке должны иметь одно и то же значение атрибута `name`;
- у каждого переключателя должен быть уникальный атрибут `id` и соответствующий уникальный элемент `<label>`.

Существенное отличие меню от переключателей состоит в том, что у элемента <select> метка относится ко всему меню, а каждый переключатель имеет собственную метку. На рис. 5.16 видно, что метка является текстом переключателя, а не внешней надписью как у элемента ввода.

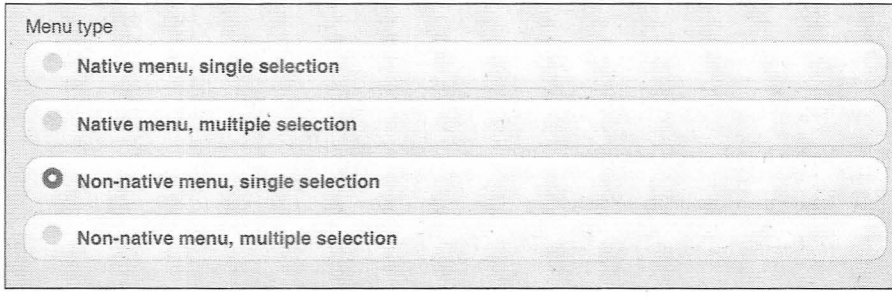


Рис. 5.16. Переключатели отображаются с текстом на кнопке

Если мы хотим определить надпись для всей группы переключателей, то должны воспользоваться элементом <legend> языка HTML.

Рассмотрим пример:

```
<legend>Menu type</legend>
<label for="menuNative1">Native menu, single selection</label>
<!-- Стандартное меню, одиночный выбор -->
<input type="radio" id="menuNative1" name="menuType" value="1">
<label for="menuNative2">Native menu, multiple selection</label>
<!-- Стандартное меню, множественный выбор -->
<input type="radio" id="menuNative2" name="menuType" value="2">
<label for="menuNonNative1">Non-native menu, single selection</label>
<!-- Нестандартное меню, одиночный выбор -->
<input type="radio" id="menuNonNative1" name="menuType" value="3">
<label for="menuNonNative2">Non-native menu, multiple selection</label>
<!-- Нестандартное меню, множественный выбор -->
<input type="radio" id="menuNonNative2" name="menuType" value="4">
```

Если мы встроили все переключатели в контейнер controlgroup, внешний вид пользовательского интерфейса станет привлекательнее (рис. 5.17):

```
<div data-role="controlgroup">
  <!-- Здесь должны быть метки и переключатели -->
</div>
```

Если мы изменим тип элемента controlgroup на horizontal, то получим совершенно другой внешний вид элемента. Переключатели потеряют характерную круглую кнопку-селектор, а элемент в целом превратится в группу кнопок (рис. 5.18):

```
<legend>Delivery method</legend>
<!-- Способ доставки -->
<div data-role="controlgroup" data-type="horizontal">
  <label for="deliveryUPS">UPS</label>
```

```
  <input type="radio" id="deliveryUPS" name="delivery" value="ups">
  <label for="deliveryDHL">DHL</label>
  <input type="radio" id="deliveryDHL" name="delivery" value="dhl">
  <label for="deliveryFedex">FedEx</label>
  <input type="radio" id="deliveryFedex" name="delivery" value="fedex">
</div>
```

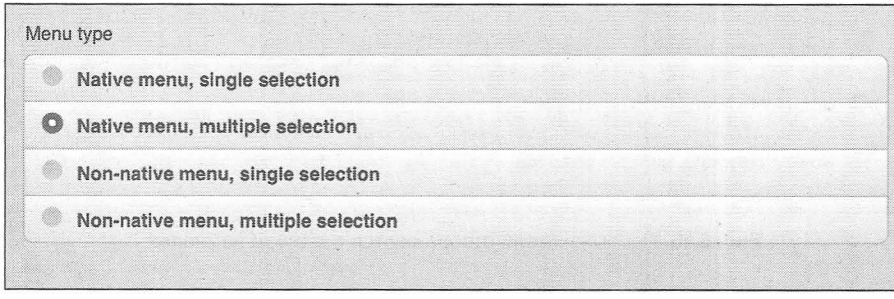


Рис. 5.17. Группирование переключателей в элемент `controlgroup` — лучший способ организации ясного пользовательского интерфейса

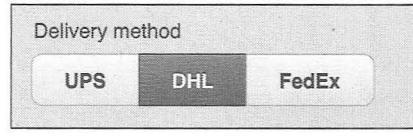


Рис. 5.18. В горизонтальном контейнере `controlgroup` переключатели превращаются в обычные кнопки, причем выбранная кнопка отличается от остальных цветом

Даже если платформа jQuery Mobile отображает непривычный интерфейс группы переключателей, форма работает стандартным образом. Это означает, что свойство `value` всегда соответствует значению выбранного элемента.

Флажки

Флажки действуют аналогично переключателям (рис. 5.19), но позволяют делать множественный выбор. Мы можем определить одиночный флажок, как в следующем примере:

```
<label for="accept">I accept terms and conditions</label>
<!-- Я принимаю условия -->
<input type="checkbox" id="accept" name="accept" value="yes">
```

Кроме того, мы можем поместить несколько флажков в контейнер `controlgroup`, в результате чего получим элемент интерфейса с множественным выбором (рис. 5.20):

```
<legend>Delivery options</legend>
<!-- Варианты оформления -->
```

```
<div data-role="controlgroup">
  <label for="optionGift">Pack it as a Gift</label>
  <!-- Упаковать как подарок -->
  <input type="checkbox" id="optionGift" name="optionGift" value="yes">
  <label for="optionBag">Send it with a bag</label>
  <!-- Отправить в пакете -->
  <input type="checkbox" id="optionBag" name="optionBag" value="yes">
  <label for="optionRemove">Remove the box</label>
  <!-- Вынуть из коробки -->
  <input type="checkbox" id="optionRemove"
        name="optionRemove" value="yes">
</div>
```

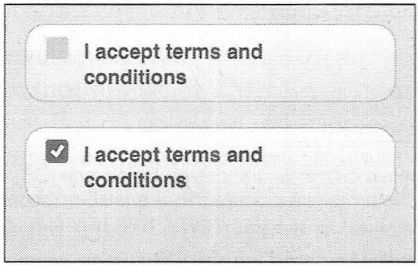


Рис. 5.19. Флажок имеет метку, а когда он выбран, в квадратике появляется галочка

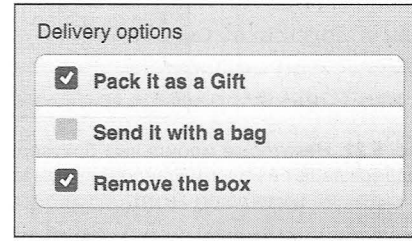


Рис. 5.20. Группирование флажков с элементом <legend> создает аккуратный список с возможностью выбора нескольких элементов

При помощи горизонтальной версии элемента controlgroup можно создать кнопочный элемент с множественным выбором (рис. 5.21). Нажатая кнопка соответствует выбранному флажку.

```
<div data-role="controlgroup" data-type="horizontal">
  <label for="bold">B</label>
  <input type="checkbox" id="bold" name="bold" value="yes">
  <label for="italic">I</label>
  <input type="checkbox" id="italic" name="italic" value="yes">
  <label for="underline">U</label>
  <input type="checkbox" id="underline" name="underline" value="yes">
</div>
```

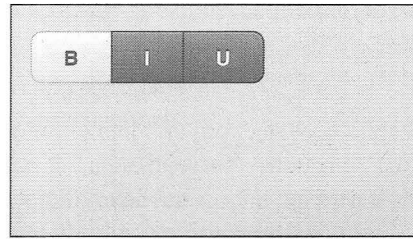


Рис. 5.21. Если метки элементов невелики, и группа содержит не более пяти флажков, мы можем воспользоваться горизонтальным контейнером и создать кнопочный элемент с множественным выбором

Выгрузка файлов

Выгрузка файлов на удаленный компьютер представляет собой реальную проблему в мобильном мире, обусловленную отсутствием поддержки со стороны некоторых операционных систем для смартфонов и планшетов, таких как iOS (iPhone и iPad), Android до версии 2.2 и webOS (рис. 5.22). На этих платформах не работает элемент `<inputtype="file">` по разным причинам, включая отсутствие открытой пользователю файловой системы. Платформа jQuery Mobile не гарантирует никакой специальный внешний вид интерфейса выгрузки файлов, и мы должны быть очень осторожны при реализации его на мобильных браузерах.

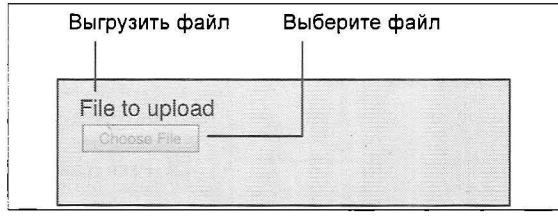


Рис. 5.22. Некоторые мобильные браузеры и операционные системы, например Safari для iOS, не поддерживают выгрузку файлов, и пользователь видит неактивный стандартный элемент формы

На некоторых современных устройствах, таких как Android версии 3.0 и выше, доступен API-интерфейс HTML Media Capture, позволяющий запросить фото- или видеоизображение, а также аудиозапись с помощью элемента, выполняющего выгрузку файлов. Вопросы совместимости с этим API-интерфейсом стандарта HTML5 обсуждаются на сайте **http://mobilehtml5.org**.

ГЛАВА 6

Платформа и JavaScript

Платформа jQuery Mobile предоставляет API-интерфейс для JavaScript, позволяющий разработчику взаимодействовать с ней и управлять содержимым из кода JavaScript. Вы должны понимать, что в основе платформы лежит HTML5, и лучшим способом создания содержимого является использование ненавязчивого HTML5.

Создание страниц и наполнение их содержимым с помощью JavaScript, а не языка разметки приведет к несовместимости приложения с некоторыми браузерами категории B и со старыми платформами, не работающими с jQuery Mobile. Если вы нацеливаетесь на последние модели смартфонов и планшетов и готовы тестировать свой код на различных реальных устройствах, то можете смело создавать содержимое, используя JavaScript и AJAX вместо непосредственной разметки.

Интерфейс API для JavaScript позволяет создавать динамическое содержимое, которое будет работать на платформе jQuery Mobile. Кроме того, он предоставляет возможность обрабатывать новые события и определять глобальные конфигурации.

Для понимания материала этой главы от вас потребуется знание основ JavaScript и ядра платформы jQuery.

События документов

Общепринятая практика включает в себя использование события `load` на вебстранице для конфигурации умолчаний и инициализации кода. Если вы работаете на платформе jQuery, то, вероятно, с удовольствием используете и событие `ready`, генерируемое элементом `document`.

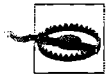
Когда мы имеем дело с документом jQuery Mobile, мы должны понимать смысл нового события `mobileinit` и уметь его обрабатывать. Это событие генерируется, когда платформа jQuery Mobile загружена и готова к инициализации кода. Событие следует обрабатывать в элементе `document` с помощью метода `bind`, предлагаемого платформой jQuery:

```
$(document).bind('mobileinit', function() {
  // Здесь должен быть код инициализации
});
```

Мобильная инициализация запускается после загрузки платформы jQuery Mobile в память, но непосредственно перед отображением элементов интерфейса. Поэтому мы можем использовать этот обработчик события для изменения некоторых глобальных настроек интерфейса.

Порядок выполнения событий документа jQuery Mobile, как правило, такой:

- mobileinit;
- ready;
- load.



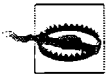
Если вы хотите выполнить какой-либо код после того, как страница загружена или выведена на экран, вы не должны использовать события load, ready или mobileinit. Каждый элемент страницы jQuery Mobile имеет набор событий, к которым мы можем привязаться.

Первое, что нужно обсудить, говоря о событии mobileinit, — это размещение его обработчика. Мы должны привязаться к этому событию в особом месте внутри элемента <header>, а именно между подключением ядра jQuery и подключением ядра jQuery Mobile. Дело в том, что нам требуется готовность объекта $ ядра jQuery, а привязаться к нему мы должны до выполнения jQuery Mobile.

Поэтому типичный шаблон документа jQuery Mobile с некоторым сценарием выглядит так:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>My first jQuery Mobile code</title>
    <!-- Мой первый код jQuery Mobile -->
    <link rel="stylesheet" href="http://code.jquery.com/mobile/1.0/jquery.mobile-
1.0.min.css" />
    <script src="http://code.jquery.com/jquery-1.6.4.min.js"></script>
    <!-- МОЙ КОД ИНИЦИАЛИЗАЦИИ -->
    <script src="customcode.js"></script>
    <script src="http://code.jquery.com/mobile/1.0/jquery.mobile-
    1.0.min.js"></script>
    <meta name="viewport" content="width=device-width, initial-scale=1">
  </head>
</html>
```

Помните, что размещение большого количества тегов <script> на одной мобильной странице является неудачным решением, поскольку снижает производительность приложения. В некоторых ситуациях лучше встроить весь код инициализации в HTML-документ в качестве сценария, чем хранить его во внешнем файле.



Если вы работаете с внешними документами, то должны отдавать себе отчет, что теги сценариев во внешних документах, загруженных по технологии AJAX, не будут выполняться, если пользователь посетил ваш сайт не с главной страницы. Поэтому все должно быть объявлено в одном внешнем файле JavaScript, и соответствующие ссылки должны присутствовать в каждом документе.

Например, мы можем привязаться к событию mobileinit следующим образом:

```
<script src="http://code.jquery.com/jquery-1.6.4.min.js"></script>
<script>
  $(document).bind("mobileinit", function() {
  // Здесь должен быть наш код инициализации
  });
</script>
<script src="http://code.jquery.com/mobile/1.0/jquery.mobile-1.0.min.js"></script>
```

Конфигурация

Платформа jQuery Mobile прикрепляет новый объект mobile к главному объекту $ платформы jQuery, также доступному по имени jQuery. Следовательно, большая часть работы с API-интерфейсом будет происходить при посредстве объекта $.mobile или jQuery.mobile. При создании веб-приложения jQuery Mobile мы сможем пользоваться разнообразными глобальными атрибутами и методами. Объект становится доступным только после генерирования события mobileinit.

Платформа использует архитектуру виджетов платформы jQuery Mobile UI, предназначенной для настольных компьютеров. Виджет — это элемент интерфейса, управляемый платформой. На платформе jQuery Mobile 1.0 доступные виджеты обычно указываются в атрибуте data-role, но существуют также элементы формы без роли. Таким образом, элементы page, button и listview — это виджеты платформы.

У каждого виджета имеется конструктор объектов и конфигурация, устанавливаемая по умолчанию, которую мы можем изменить в обработчике события mobileinit, что повлияет на каждый экземпляр виджета на странице.

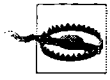
Список виджетов, доступных на платформе jQuery Mobile 1.0, таков:

- page;
- dialog;
- collapsible;
- fieldcontain;
- navbar;
- listview;
- checkboxradio;
- button;
- slider;
- textinput;
- selectmenu;
- controlgroup.

Некоторые элементы jQuery Mobile, имеющие богатую функциональность, сгруппированы в один виджет. Например, все элементы текстового ввода — даже элемент textarea — обращаются к одному виджету textinput. Флажки и переключатели также сгруппированы в виджет checkboxradio.

У каждого виджета есть свой конструктор объектов, который определяет поведение объекта на странице. Мы можем обратиться к этому прототипу с помощью конструкции $.mobile.<*имя_виджета*>.prototype. Обычно каждый конструктор виджета име-

ет объект option, в котором мы можем определить атрибуты виджета, работающие по умолчанию. Например, объект $.mobile.page.prototype.options позволяет определить атрибуты, которые будут применяться к каждому экземпляру страницы (data-role="page").



Не забудьте установить глобальные умолчания или умолчания виджета в обработчике события mobileinit. Если изменить умолчания после или до этого события, не исключено, что новые атрибуты не будут применены к документу.

Глобальная конфигурация

Большое количество значений, которые могут быть явно определены с помощью атрибута data-* у каждого элемента, можно определить глобально, так чтобы эти значения применялись к каждому элементу интерфейса, если в разметке не определено новое значение какого-либо атрибута.

Большинство значений глобальной конфигурации может быть изменено с помощью объекта $.mobile в обработчике события mobileinit.

Пользовательский интерфейс

По умолчанию платформа jQuery Mobile присваивает классы конкретным элементам страницы и меняет эти классы на другие, когда элементы активны. Эти классы используются таблицей стилей темы для придания элементам разного внешнего вида. Мы можем изменить имя класса текущей активной страницы (в многостраничном документе) или имя класса активной кнопки, обратившись к строковым свойствам activePageClass и activeBtnClass. По умолчанию приняты значения ui-page-active и ui-btn-active. Состояние активной кнопки используется во многих других виджетах, в которых применяется механизм кнопок, например в панелях навигации, переключателях и флажках.

Существуют два глобальных атрибута (остающихся неизменными в большинстве ситуаций), которые позволяют изменить поведение прокрутки. По умолчанию после загрузки документа jQuery Mobile он немного прокручивается вниз ровно настолько, чтобы скрылась адресная строка. Мы можем изменить это умолчание, установив значение defaultHomeScroll. Когда мы открываем страницу, а затем возвращаемся на нее, платформа прокручивает окно просмотра до того положения, в котором была страница изначально (т. е. позиция запоминается). Однако если первая страница была открыта очень близко к началу (но не у нулевого пиксела), то платформа оставит ее в том же положении и не будет прокручивать. По умолчанию минимальное значение, вызывающее прокрутку при возврате, равно 250 пикселам по вертикали, и мы можем изменить его с помощью атрибута minScrollBack.

И наконец, следует упомянуть два глобальных атрибута, которые мы будем менять относительно часто. Это переходы, выполняемые по умолчанию во время загрузки страницы и диалогового окна. Изначально применяются переходы slide и pop, но мы можем поменять их с помощью атрибутов defaultPageTransition и defaultDialogTransition.

В следующем примере изменяются некоторые умолчания пользовательского интерфейса:

```
$(document).bind("mobileinit", function() {
  // Меняем значения по умолчанию
  $.mobile.defaultPageTransition = "fade";
  $.mobile.minScrollBack = 150;
  $.mobile.activeBtnClass = "active-button";
});
```

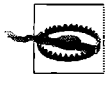
Функциональность ядра и AJAX

Некоторыми функциональными возможностями ядра платформы можно манипулировать через глобальные атрибуты. Если мы обращаемся к другой платформе, которая может конфликтовать с jQuery Mobile, мы можем определить пространство имен с помощью глобального атрибута `ns`. По умолчанию пространство имен не определено. Предположим, мы определили пространство имен, например, так:

```
$(document).bind("mobileinit", function() {
  // Меняем значения по умолчанию
  $.mobile.ns = "firt";
});
```

Тогда все атрибуты `data-*` превращаются в атрибуты `data-`*<пространство_имен>*`-*`, в нашем примере — в атрибуты `data-firt-*`. Это относится ко всем нестандартным атрибутам jQuery Mobile, таким как `data-role` (который преобразуется в `data-`*<пространство_имен>*`-role`). В нашем новом пространстве имен типичный шаблон страницы будет выглядеть так:

```
<div data-firt-role="page">
  <div data-firt-role="header" data-firt-theme="a">
  </div>
  <div data-firt-role="content">
  </div>
</div>
```

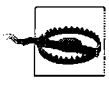


Если вы определяете пространство имен, то должны вручную отредактировать CSS-файл (как структурный, так и файл темы), чтобы стала понятной смена пространства имен. Например, вы должны заменить условные селекторы в соответствии с новым пространством имен: `[data-`*<пространство_имен>*`-role=page]`.

Одной из важных функциональных возможностей ядра jQuery Mobile является использование технологии AJAX для загрузки внешних страниц. Мы можем отключить работу AJAX с помощью булева атрибута `ajaxEnabled`. Если мы это сделаем оператором `$.mobile.ajaxEnabled=false`, любая внешняя страница будет загружаться в браузер в результате HTTP-запроса.

По умолчанию объект `XMLHttpRequest`, лежащий в основе работы AJAX, не позволяет выполнять кроссплатформенные запросы.

Это означает, что если наша страница находится в домене **domain1.com**, мы не можем загружать страницы из домена **domain2.com** с помощью AJAX. Платформа автоматически отправит HTTP-запрос в этом случае. Бывают особые ситуации, когда мы можем делать кроссдоменные запросы и заставить платформу поддерживать их, если определим атрибут `allowCrossDomainPages`.



Некоторые новые мобильные браузеры поддерживают стандарт CORS (Cross Origin Resource Sharing, обмен ресурсами между доменами), рабочий проект которого предложен консорциумом W3C и доступен по адресу **http://w3.org/TR/cors**. Этот стандарт позволяет браузеру поддерживать кроссдоменные запросы, если ответ с другого сервера включает в себя некоторые специальные HTTP-заголовки. Совместимость конкретных браузеров со стандартом CORS можно выяснить на сайте **http://mobilehtml5.org**.

Если вы создаете автономное или гибридное приложение (например, под PhoneGap или RhoMobile), вы загружаете страницы, в основном, по протоколу `file://` (локальные файлы). Эти платформы позволяют выполнять AJAX-запросы к любому домену в Интернете, так что если вы захотите, чтобы ваше приложение загружало внешние страницы из Всемирной паутины, вы должны обеспечить эту возможность оператором `$.mobile.allowCrossDomainPages=true`.

Некоторые виджеты, например представления вложенных списков, генерируют страницы динамически. Платформе нужно имя для каждой новой страницы (для хеша, URL-адресации и иных целей). По умолчанию jQuery Mobile использует атрибут `ui-page` в качестве имени параметра, если мы не изменим это поведение с помощью свойства `subPageUrlKey` из объекта `$.mobile`. Когда мы работаем с диалоговыми страницами, в нашем распоряжении имеется свойство `dialogHashKey`, которое по умолчанию связано с атрибутом `ui-state=dialog`. В большинстве ситуаций нам не придется изменять эти свойства.

Если вы хотите, чтобы платформа не меняла поведение каждой ссылки для поддержки различных действий jQuery Mobile, вы можете отключить это поведение оператором `$.mobile.linkBindingEnabled=false`. По умолчанию платформа jQuery Mobile инициализирует первую страницу документа, когда готова объектная модель документа. Это действие тоже можно отключить с помощью `$.mobile.autoInitializePage=false`.

Кроме того, можно отключить автоматическое чтение хеша, обеспечивающее переход по страницам назад и вперед в ответ на нажатие соответствующих кнопок на устройстве или в браузере. Отключение производится оператором `$.mobile.hashListeningEnabled=false`.

Локализуемые строки

В коде платформы имеется несколько жестко закодированных строковых значений, которые могут быть локализованы в соответствии с другими языковыми настройками. Некоторые из них вообще не видны в обычном документе jQuery Mobile, поскольку хранят семантическую информацию или служат для обеспечения доступ-

ности содержимого всем пользователям (так что программы чтения текста могут озвучить их).

Строки, которые можно изменить, содержат сообщение, появляющееся при загрузке внешней страницы с помощью AJAX, сообщение об ошибке, когда загрузка внешней страницы невозможна, и ряд других сообщений, выдаваемых конкретными виджетами.

Приведу список сообщений с указанием их значений по умолчанию:

```
// Глобальные строки
$.mobile.loadingMessage = "loading";
// загружается
$.mobile.pageLoadErrorMessage = "Error Loading Page";
// Ошибка загрузки страницы
// Строки с сообщениями виджетов
$.mobile.page.prototype.options.backBtnText = "Back";
// Назад
$.mobile.dialog.prototype.options.closeBtnText = "Close"
// Закрыть
$.mobile.collapsible.prototype.options.expandCueText =
    "click to expand contents";
// Щелкните, чтобы развернуть содержимое
$.mobile.collapsible.prototype.options. collapseCueText =
    "click to collapse contents";
// Щелкните, чтобы свернуть содержимое
$.mobile.listview.prototype.options.filterPlaceholder = "Filter items...";
// Фильтровать пункты
$.mobile.selectmenu.prototype.options.closeText = "Close";
// Закрыть
```

Таким образом, чтобы создать испаноязычную версию сообщений jQuery Mobile, мы должны будем написать такой код:

```
$(document).bind('mobileinit', function() {
  // Глобальные строки
  $.mobile.loadingMessage = "cargando";
  $.mobile.pageLoadErrorMessage = "Error Cargando Pagina";
  // Строки с сообщениями виджетов
  $.mobile.page.prototype.options.backBtnText = "Atras";
  $.mobile.dialog.prototype.options.closeBtnText = "Cerrar"
  $.mobile.collapsible.prototype.options.expandCueText =
     " click para expandir contenido";
  $.mobile.collapsible.prototype.options. collapseCueText =
     " click para cerrar contenido";
  $.mobile.listview.prototype.options.filterPlaceholder =
     "Filtrar items...";
  $.mobile.selectmenu.prototype.options.closeText = "Cerrar";
});
```

Сенсорное переполнение

Как уже было сказано в предыдущих главах, создание фиксированных панелей инструментов (с атрибутом `data-position="fixed"`) не обеспечивает появления подлинно фиксированных панелей в jQuery Mobile 1.0. Платформа просто эмулирует фиксированную панель, когда не прокручивается страница (рис. 6.1).

Рис. 6.1. Когда мы создаем фиксированные панели с применением сенсорного переполнения, дизайн страницы меняется: область содержимого имеет собственную область прокрутки

Под управлением операционной системы iOS 5.0 браузер Safari поддерживает элементы `position:fixed` и прокрутку блочных областей одним пальцем. При этом используются элемент `overflow:scroll` и новое префиксное расширение `overflowscrolling:touch`. По умолчанию это поведение отключено, но если вы хотите создать подлинно фиксированные панели инструментов в iOS 5 (или на других платформах в будущем), вам будет достаточно установить свойство `touchOverflowEnabled` в значение `true`. На платформах, не поддерживающих эту функциональность, будут отображаться обычные фиксированные панели инструментов в стиле jQuery Mobile:

```
$(document).bind('mobileinit', function() {
  $.mobile.touchOverflowEnabled=true;
});
```

Прокрутка внутри блочных элементов с использованием `overflow:scroll` также поддерживается в Android 3.0 и BlackBerry Browser для PlayBook. Платформа jQuery Mobile будет поддерживать эти браузеры, начиная с версии 1.1.

Если мы установили это свойство в значение `true`, мы также сможем включить масштабирование с помощью свойства `touchOverflowZoomEnabled`. Однако этой опцией следует пользоваться осторожно, потому что ее включение может понизить удобство приложения. Эти два свойства морально устареют после выхода следующих версий платформы, потому что в дальнейшем панели инструментов будут фиксированными по умолчанию.

Конфигурация страницы

Для каждой страницы создается элемент с атрибутом `data-role="page"`, имеющий параметры по умолчанию, которые могут быть переопределены с помощью атрибутов `data-*`. Если мы хотим изменить умолчания, то можем изменить параметры прототипа у любого экземпляра страницы.

Например, если нам нужно, чтобы на каждой странице, имеющей предшествующую страницу в журнале, появлялась кнопка **Назад**, мы можем установить свойство `addBackBtn` в значение `true`. Текст и тему этой кнопки можно изменить с помощью свойств `backBtnText` и `backBtnTheme` соответственно.

Кроме того, мы можем изменить темы, устанавливаемые по умолчанию у других элементов, если поменяем значения свойств `headerTheme`, `footerTheme` и `contentTheme`.

Например:

```
$(document).bind('mobileinit', function() {
  $.mobile.page.prototype.options.addBackBtn = true;
  $.mobile.page.prototype.options.backBtnTheme = "e";
  $.mobile.page.prototype.options.headerTheme = "b";
  $.mobile.page.prototype.options.footerTheme = "d";
});
```

Если вы создаете полноэкранное веб-приложение, гибрид или приложение PhoneGap, вы должны явным образом обеспечить присутствие кнопки **Назад** в заголовке. Для этого вы должны перехватить событие `mobileinit` и написать в его обработчике оператор `$.mobile.page.prototype.options.addBackBtn=true`.

Загрузка страницы

Каждый раз, когда внешняя страница загружается с помощью AJAX, применяются некоторые атрибуты со значениями по умолчанию. Эти атрибуты определены в объекте `$.mobile.loadPage.defaults object`, а в табл. 6.1 перечислены их возможные значения.

Таблица 6.1. *Свойства объекта* `$.mobile.loadPage.defaults`

Свойство	Допустимое значение	Значение по умолчанию	Описание
`type`	`"get"/"post"`	`"get"`	Определяет тип AJAX-запроса
`data`	Объект/строка		При запросе типа `"post"` здесь можно указать post-объект для отправки
`reloadPage`	`true/false`	`false`	Определяет, следует ли загружать новую страницу независимо от наличия ее кэшированной версии в объектной модели документа

Таблица 6.1 (окончание)

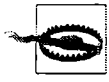
Свойство	Допустимое значение	Значение по умолчанию	Описание
role	Строка	Определяется атрибутом data-role	Определяет роль целевой страницы
showLoadMsg	true/false	true	Определяет, должно ли выводиться сообщение "загружается", если время запроса превышает указанный тайм-аут
loadMsgDelay	Миллисекунды	50	Продолжительность тайм-аута в миллисекундах, по истечении которого следует выводить сообщение "загружается"
theme	a–z	c	Образец цвета, применяемый по умолчанию к каждой странице
domCache	true/false	false	Определяет, следует ли кэшировать страницы в объектной модели документа

Если мы хотим изменить атрибут загрузки только для одной страницы, то можем воспользоваться утилитой `$.mobile.changePage`.

Конфигурация виджетов

Каждый виджет платформы jQuery Mobile имеет свои атрибуты конфигурации, устанавливаемые по умолчанию. Вспомним, что мы можем менять умолчания, изменяя объект `options` прототипа виджета: `$.mobile.<widget_name>.prototype.options`. Например:

```
$(document).bind('mobileinit', function() {
  // Включить фильтрацию для всех списков
  $.mobile.listview.prototype.filter = true;
  // Включить нестандартные меню для всех меню select
  $.mobile.selectmenu.prototype.nativeMenu=false;
});
```



У большинства виджетов есть атрибут `theme`, имеющий значение по умолчанию. Поэтому, если мы хотим, чтобы, например, все меню имели по умолчанию тему `e`, мы можем написать оператор `$.mobile.selectmenu.prototype.theme="e"`.

В табл. 6.2 перечислены наиболее популярные атрибуты каждого виджета, которые мы можем изменить.

Таблица 6.2. *Свойства по умолчанию, которые можно изменить у виджетов*

Виджет	Свойство	Значения (по умолчанию)	Описание
Любой виджет	theme	От a до z	Цветовой образец, применяемый ко всем экземплярам виджета
listview	filter	true/(false)	Включает фильтрацию у каждого спискового представления
listview	filterPlaceHolder	Строка	Текст-подсказка в поле поиска у фильтра
listview	filterTheme	От a до z	Цветовой образец для поля поиска фильтра
navbar	iconpos	(top)/bottom/ left/right	Положение значка на панели навигации
slider	trackTheme	От a до z	Цветовой образец дорожки ползункового регулятора
selectmenu	icon	Значение, определяющее значок (arrow_d)	Значок кнопки открытия меню
selectmenu	iconpos	top/bottom/ left/(right)	Положение значка
selectmenu	corners	(true)/false	Наличие закругленных углов у кнопки открытия
selectmenu	shadow	(true)/false	Наличие тени у кнопки открытия
selectmenu	iconshadow	(true)/false	Наличие тени у значка на кнопке открытия
selectmenu	menuPageTheme	От a до z (b)	Цветовой образец для нестандартного меню
selectmenu	overlayTheme	От a до z (a)	Цветовой образец для нестандартного перекрытия
selectmenu	closeText	Строка	Текст на кнопке закрытия в нестандартном меню
selectmenu	nativeMenu	(true)/false	Определяет, используется ли стандартное или нестандартное меню
dialog	closeBtnText	Строка	Текст на кнопке закрытия
dialog	overlayTheme	От a до z (a)	Цветовой образец диалоговой перекрывающей страницы
collapsible	expandCueText	Строка	Текст для кнопки разворачивания
collapsible	collapseCueText	Строка	Текст для кнопки сворачивания
collapsible	collapsed	(true)/false	Определяет состояние сворачиваемого элемента по умолчанию: свернут или развернут

Таблица 6.2 (окончание)

Виджет	Свойство	Значения (по умолчанию)	Описание
collapsible	heading	Значения, разделяемые запятыми	Список тегов, используемых в качестве заголовка сворачиваемого элемента
collapsible	contentTheme	От a до z	Цветовой образец для содержимого
collapsible	iconTheme	От a до z	Цветовой образец для значка заголовка
button	icon	Значение, определяющее значок	Значок по умолчанию для кнопки
button	iconpos	Значение, определяющее положение значка	Положение значка на кнопке
button	inline	true/(false)	Определяет, должна ли кнопка быть встроенной
button	corners	(true)/false	Наличие закругленных углов у кнопки
button	shadow	(true)/false	Наличие тени у кнопки
button	iconshadow	(true)/false	Наличие тени у значка на кнопке

Утилиты

Платформа jQuery Mobile предлагает разработчику множество утилит, позволяющих управлять приложением из кода JavaScript. Утилиты предоставляются методами и атрибутами, доступными только для чтения, и в результате мы имеем возможность обеспечивать более качественный пользовательский опыт с помощью языка JavaScript.

Утилиты *data-**

Общепринятая практика работы с jQuery Mobile предполагает интенсивные манипуляции с нестандартными атрибутами `data-*`. Например, если мы хотим создать набор кнопок на странице, то можем написать код для jQuery:

```
var buttons = $("a[data-role=button"]);
```

Платформа jQuery Mobile предлагает новый фильтр по имени `jqmData`, который учитывает пространство имен, если оно используется в коде. Поэтому проще и безопаснее заменить предыдущий фрагмент кода таким:

```
var buttons = $("a:jqmData(role='button')");
```

Следует также использовать методы jqmData и jqmRemoveData вместо традиционных функций data() и removeData() платформы jQuery, когда мы работаем с объектом-коллекцией:

```
$("a").jqmRemoveData("transition");
$("#button1").jqmData("theme", "a");
```

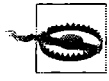
Утилиты страниц

На случай, если нам понадобится обратиться к текущей странице, платформа jQuery Mobile предлагает атрибут $.mobile.activePage, который автоматически связывается с элементом data-role="page", отображаемым на экране в данный момент. Это свойство связано с объектом из объектной модели документа jQuery Mobile (как правило, с элементом):

```
var currentPageId = $.mobile.activePage.id;
```

Мы можем обратиться к контейнеру текущей страницы (обычно это элемент <body>) с помощью атрибута $.mobile.pageContainer.

Самой полезной утилитой платформы является метод $.mobile.changePage. Он позволяет выполнить переход к другой странице так, словно пользователь щелкнул по ссылке. Мы можем вызывать этот метод в коде JavaScript для показа как внутренних, так и внешних страниц.



Некоторые утилиты становятся доступны только после генерирования события mobileinit. Поэтому мы не можем вызывать их в коде обработчика этого события.

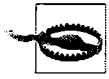
Первым (обязательным) параметром может быть строка (содержащая URL-адрес внешней страницы) или объект jQuery, содержащий внутреннюю страницу.

Чтобы загрузить внешнюю страницу external.html, достаточно написать:

```
$.mobile.changePage("external.html");
```

Чтобы перейти на внутреннюю страницу в том же документе, можно написать:

```
$.mobile.changePage($("#pageId"));
```



Мы не можем открывать внутренние страницы, просто загружая их по имени "#pageId", мы должны отправлять объект модели DOM с помощью платформы jQuery, $("#pageId").

Опции перехода между страницами

Вторым (необязательным) аргументом метода changePage является объект, как правило, определяемый с помощью синтаксиса JSON, с необязательными атрибутами перехода и/или загрузки по технологии AJAX.

Все возможные опции перечислены в табл. 6.3.

Таблица 6.3. *Свойства, доступные методу* `changePage`

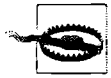
Свойство	Значения	Значение по умолчанию	Описание
`transition`	Название перехода	`slide`	Применяемый переход
`reverse`	`true/false`	`false`	Определяет, должен ли переход воспроизводиться в обратном направлении (обычно это делается при возврате на предыдущую страницу)
`type`	`"get"/"post"`	`"get"`	HTTP-метод для загрузки внешней страницы
`data`	Объект/строка		Данные, отправляемые методом `post`, если он указан
`allowSamePageTransition`	`true/false`	`false`	Разрешает переход на страницу, являющуюся активной (переход "на себя")
`changeHash`	`true/false`	`true`	Определяет, должна ли новая страница заноситься в журнал
`data-url`	Строка		URL-адрес, добавляемый в строку с местоположением страницы
`pageContainer`	Объект модели DOM jQuery		Контейнер для новой страницы
`reloadPage`	`true/false`	`false`	Выполняет принудительную перезагрузку страницы, даже если она кэширована в объектной модели документа
`showLoadMsg`	`true/false`	`true`	Определяет, должно ли появляться сообщение "страница загружается" по истечении некоторого количества миллисекунд
`role`	`page/dialog`	Определяется атрибутом `data-role`	Роль, применяемая к новой странице

Если мы хотим форсировать переход "сдвиг" в обратном направлении, то можем написать такой код:

```
$.mobile.changePage($("#page2"), {
  transition: "slide",
  reverse: true
});
```

Следующий фрагмент кода загружает внешнюю страницу, отправляя данные методом `post`:

```
<script>
  function viewProduct(idProduct) {
    $.mobile.changePage("productdetail.php", {
      method: "post",
      data: {
        action: 'getProduct',
        id: idProduct
      },
      transition: "fade"
    });
  }
</script>
<!-- ... -->
<a href="javascript:viewProduct(5200)" data-role="button">Product details</a>
<!-- Описание товара -->
```



Нельзя забывать, что даже при загрузке страницы методом `post` целевым объектом должен быть документ jQuery Mobile, содержащий заголовки и атрибут `datarole="page"`.

Существует метод `$.mobile.loadPage`, вызываемый методом `changePage` при загрузке внешней страницы. Этот метод заносит указанную страницу в объектную модель документа, но не выполняет переход на нее. Для этой цели служит метод `changePage`. Мы можем вызывать метод `loadPage`, когда хотим заранее загрузить содержимое, занести его в объектную модель документа, а впоследствии вызвать метод `changePage` для перехода на объект из модели DOM jQuery.

Утилиты платформы

Платформа предоставляет нам собственные утилиты, способствующие разработке веб-приложений. В табл. 6.4 приводятся наиболее полезные из утилит платформы.

Таблица 6.4. *Утилиты, доступные для запросов к работающей платформе из объекта* `$.mobile`

Метод/свойство	Описание
`orientationChangeEnabled`	Доступно ли низкоуровневое событие `orientationchange`
`gradeA()`	Возвращает `true`, если браузер принадлежит к категории A в таблице совместимости jQuery Mobile
`urlHistory`	Коллекция страниц, посещенных в пределах jQuery Mobile без перезагрузки страниц. Каждый элемент имеет свойства `pageUrl`, `title` и `transition`
`getDocumentUrl()`	Возвращает URL-адрес оригинального документа (документа, загруженного в первый раз)
`getDocumentBase()`	Возвращает базу оригинального документа

Таблица 6.4 (окончание)

Метод/свойство	Описание
keyCode	Константы, определяющие коды для обработки событий от клавиш, в том числе: ALT, BACKSPACE, COMMAND, COMMAND_LEFT, COMMAND_RIGHT, DELETE, DOWN, UP, RIGHT, LEFT, END, ENTER, ESCAPE, HOME, INSERT, MENU, PAGE_DOWN, PAGE_UP, PERIOD, SHIFT, SPACE, TAB, WINDOWS
getScreenHeight()	Высота экрана данного устройства

Утилиты пути

Существуют утилиты для работы с путями, используемые платформой и доступные как открытые методы в объекте `$.mobile.path`. Они перечислены в табл. 6.5.

Таблица 6.5. Утилиты пути, доступные в объекте `$.mobile.path`

Метод	Описание
parseUrl(url)	Возвращает объект со свойствами, соответствующими частям URL-адреса (протокол, имя хоста, порт, путь, каталог, имя файла, хеш и др.)
makePathAbsolute(relativePath, absolutePath)	Принимает относительный путь и возвращает абсолютный
makeUrlAbsolute(relativeUrl, absoluteUrl)	Принимает относительный URL-адрес и возвращает абсолютный
isSameDomain(Url1, Url2)	Возвращает true, если оба URL-адреса находятся в одном домене
isRelativeUrl(Url)	Возвращает true, если URL-адрес относительный
isAbsolute(Url)	Возвращает true, если URL-адрес абсолютный

Утилиты пользовательского интерфейса

Последняя группа утилит относится к пользовательскому интерфейсу. Метод `$.mobile.getInheritedTheme(element, defaultSwatch)` позволяет узнать, какой цветовой образец должен быть применен к документу на основании его собственного определения образца цвета или цепочки наследования.

Метод `$.mobile.silentScroll(y)` выполняет прокрутку до любой позиции на странице без анимации и генерирования событий, а методы `$.mobile.showPageLoadingMsg()` и `$.mobile.hidePageLoadingMsg()` позволяют показать или скрыть сообщение "страница загружается" по вашему усмотрению.

```
// Этот код показывает сообщение о загрузке страницы
// и убирает его через 2 секунды
$.mobile.showPageLoadingMsg();
setTimeout(function() {
  $.mobile.hidePageLoadingMsg();
}, 2000);
```

Наконец, методы `$.mobile.fixedToolbars.show()` и `$.mobile.fixedToolbars.hide()` показывают и скрывают фиксированные панели инструментов (как мы видели в предыдущих главах). Панели могут быть полноэкранными или просто фиксированными. Вы не можете скрыть подлинно фиксированные панели в iOS 5. По умолчанию они показываются и скрываются с использованием перехода "постепенное появление/исчезновение". Это поведение можно изменить, передав параметр `true` методу `$.mobile.fixedToolbars.show(true)`, который выведет панели инструментов немедленно и без анимации.

Нестандартные переходы

Ранее мы обсуждали все переходы, доступные в jQuery Mobile 1.0. Можем ли мы определить собственные переходы? Да, можем, и для этого существуют два способа:

- с использованием анимаций CSS3;
- с помощью кода JavaScript.

Когда мы определяем атрибут `data-transition` (или вызываем переход из кода JavaScript), платформа jQuery Mobile вначале проверяет, является ли имя перехода стандартным для платформы. Если не является, она ищет переход в коллекции `$.mobile.transitionHandlers`. Если переход не является стандартным и не определен в коллекции, выполняется переход, установленный по умолчанию.

Переход, выполняемый по умолчанию, может быть заменен, и он должен быть определен в виде обработчика перехода. По умолчанию такой "резервный" переход отображается на нестандартный, который должен быть определен в CSS-анимации *(см. следующую главу)*.

Мы также можем отобразить переход, выполняемый по умолчанию, на метод `$.mobile.noneTransitionHandler`, который изначально показывает новую страницу и скрывает предыдущую без какой-либо анимации.

Например, можно добавить переход `explode`:

```
$.mobile.transitionHandlers.explode = explodeTransitionHandler;
```

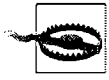
Кроме того, мы можем заменить переход, выполняемый по умолчанию, и использовать любое имя в том же обработчике:

```
$.mobile.defaultTransitionHandler = explodeTransitionHandler;
```

Обработчик перехода — это функция на языке JavaScript, принимающая четыре аргумента:

- имя перехода;
- `reverse` — булево значение, показывающее, должен ли переход выполняться в обратном направлении (если `true`, то должен);
- `toPage` — объект модели DOM jQuery, соответствующий целевой странице;
- `fromPage` — объект модели DOM jQuery, соответствующий исходной странице.

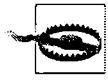
Для создания перехода мы можем написать код JavaScript по своему усмотрению. Просто убедитесь, что в конце вы удалили класс `$.mobile.activePageClass` из исходной страницы и применили его к целевой странице.



Будьте осторожны с переходами, написанными на JavaScript. На некоторых платформах возможны проблемы с совместимостью и снижение производительности. Прежде чем использовать переход, проведите его всестороннее тестирование.

Динамическое содержимое

Если наше содержимое имеет динамический характер, например, берется из базы данных сайта, мы, скорее всего, не захотим создавать весь документ динамически, а предпочтем воспользоваться технологиями JavaScript и AJAX для изменения, показа и сокрытия информации в веб-приложении.



Использование элементов, основанных на коде JavaScript вместо семантической разметки, может привести к проблемам на браузерах, не совместимых с jQuery Mobile. Если же вы пишете приложения только для смартфонов и планшетов, такие проблемы вам не грозят.

Создание страниц

Можем ли мы создавать страницы, как говорится, "на лету"? Мы знаем, что страница — это всего лишь элемент `<div>` с соответствующим значением атрибута `data-role`, так что на первый взгляд кажется, что мы сможем заставить его работать. Попробуем разобраться, что произойдет. Мы хотим создать базовую страницу, которая будет динамически создавать четыре других с помощью кода JavaScript:

```
<div data-role="page">
  <div data-role="header">
    <h1>Dynamic page</h1>
    <!-- Динамическая страница -->
  </div>
  <div data-role="content">
    <a id="button1" href="javascript:addPages()"
       data-role="button">Add Pages</a>
    <!-- Добавить страницы -->
```

```
    <ul id="list1">
    </ul>
  </div>
</div>
```

Теперь в сценарии напишем функцию, создающую динамические страницы и кнопки для перехода на них:

```
function addPages() {
  for (var i=1; i<5; i++) {
    var page = $("<div>").jqmData("role", "page").attr("id", "page" + i);
    // заголовок
    $("<div>").attr("data-role", "header").append($("<h1>")
      .html("Page " + i)).appendTo(page);
    // содержимое
    $("<div>").attr("data-role", "content").append($("<p>")
      .html("Contents for page " + i)).appendTo(page);
    $("body").append(page);
    $("<li>").append($("<a>").attr("href", "#page"+i)
      .html("Go to page " + i)).appendTo("#list1");
  }
  $("#button1").hide();
};
```

Если рассмотреть результаты этого кода (рис. 6.2), легко заметить, что при динамическом создании страниц после загрузки основной все работает, но заголовки страниц имеют неправильные CSS-стили.

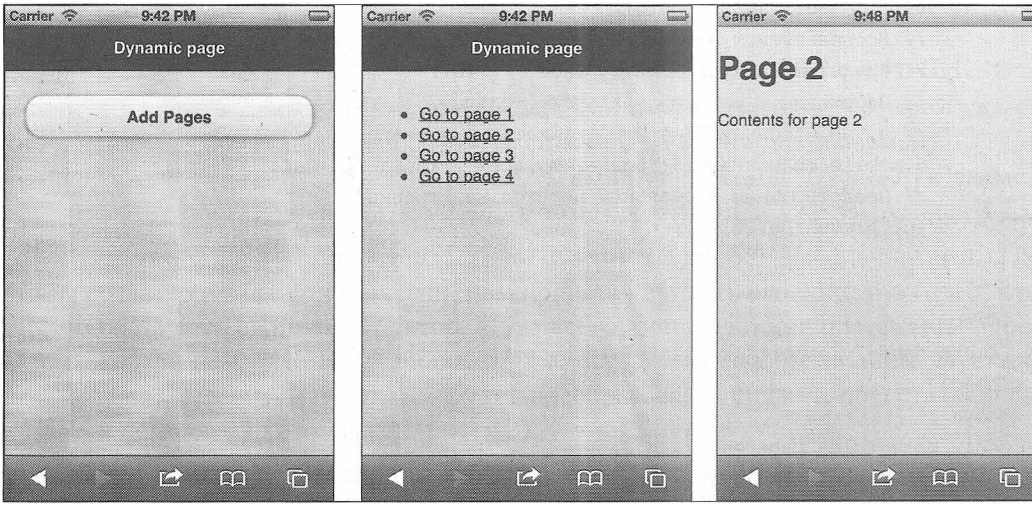
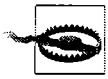


Рис. 6.2. Страницы, созданные динамически, ведут себя так, словно присутствовали с самого начала



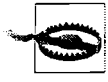
Динамически создаваемые страницы имеют один недостаток: когда пользователь выполняет перезагрузку, находясь на одной из новых страниц, у него ничего не получится, если мы не перехватим событие `mobileinit` и не проверим, пытается ли он загрузить одну из динамических страниц (читая хеш-значение или событие страницы). К моменту повторной загрузки эта страница уже не существует, и ее нужно создавать по требованию.

Чтобы улучшить функциональность страницы, созданной динамически, мы можем вызвать метод `page()` элемента модели DOM jQuery, например, `$("#page1").page()`.

Оптимальным способом создания динамических страниц является указание ссылок на них (например, `#page1`) и обработка события `pagebeforechange` с изменением поведения платформы. Это событие подробно обсуждается далее, но следующего кода достаточно, чтобы понять, в чем его суть:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta charset="UTF-8" />
    <title>jQuery Mobile</title>
    <script src="jquery.js"></script>
    <link rel="stylesheet" type="text/css" href="jquery.mobile-1.0.css">
    <script src="jquery.mobile-1.0.js"></script>
    <script>
      $(document).bind('pagebeforechange', function(event, data) {
        // Получаем целевую страницу в объекте data.toPage
        // и нормализуем ее
        var url = $.mobile.path.parseUrl(data.toPage).hash;
        if (url!=undefined && url.length>5 &&
                 url.substring(0, 5)=="#page") {
          // Динамически вставляем новую страницу
          var id = url.substring(5);
          // Воспользуемся шаблоном страницы, уже присутствующим в DOM
          $("#pageTemplate h1").html("Page " + id);
          // Переходим к шаблону реальной страницы,
          // но не используем ее идентификатор в журнале
          $.mobile.changePage($("#pageTemplate"), {dataUrl: data.toPage});
          // Предотвращаем нормальный переход на страницу
        event.preventDefault();
        }
      });
    </script>
    <meta name="viewport" content="width=device-width,user-scalable=no">
  </head>
  <body>
    <div data-role="page">
      <div data-role="header">
        <h1>Dynamic pages</h1>
        <!-- Динамические страницы -->
      </div>
```

```
      <div data-role="content">
        <a id="button1" href="#page1" data-role="button">Page 1</a>
        <!-- Страница 1 -->
        <a id="button1" href="#page2" data-role="button">Page 2</a>
        <!-- Страница 2 -->
        <a id="button1" href="#page3" data-role="button">Page 3</a>
        <!-- Страница 3 -->
        <a id="button1" href="#page4" data-role="button">Page 4</a>
        <!-- Страница 4 -->
      </div>
    </div>
    <div data-role="page" id="pageTemplate">
      <div data-role="header">
        <h1>Header</h1>
        <!-- Заголовок -->
      </div>
      <div data-role="content">Content</div>
      <div data-role="footer">
        <h4>Footer</h4>
        <!-- Нижний колонтитул -->
      </div>
    </div>
  </body>
</html>
```



Если мы попытаемся вставить другие виджеты с помощью JavaScript в то время, когда отображается страница, их отображение может произойти некорректно, пока мы не сгенерируем событие `create`.

Создание виджетов

Взглянув на рис. 6.2, мы поймем, что фактически имеем список из четырех элементов, и что было бы лучше преобразовать список с именем `list1` в элемент `listview`. Я знаю, что вы думаете: "Нужно добавить атрибут `datarole="listview"` в элемент `<ul>`". Но этот прием не сработает. Дело в том, что страница уже загружена, а наш список не был распознан как виджет `listview` с самого начала.

Чтобы создать виджет динамически, нам придется вызвать его конструктор. У каждого виджета есть свой конструктор, и это всего лишь функция платформы jQuery с тем же именем, что и у виджета. То есть, если мы выполним функцию `$("#list1").listview()`, элемент `<ul>` будет преобразован в виджет, представляющий список, и немедленно отображен (рис. 6.3).

Если мы хотим преобразовать список элементов `<a>` в набор кнопок, нам достаточно вызвать функцию:

```
$("a").button();
```

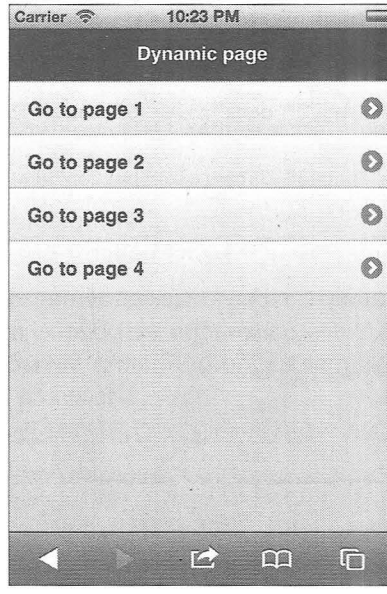


Рис. 6.3. С помощью конструкторов мы можем создать любой виджет динамически; в данном случае — представление списка

Если же мы создаем его динамически, нужно написать:

```
var button = $("<a>").attr("href", "somewhere.html").button();
```

Обновление виджетов

Итак, мы знаем, как динамически создавать виджеты jQuery Mobile. А что произойдет, если мы изменим содержимое виджета после того, как он создан и отображен на экране? Например, нам нужно будет добавить элементы в список или поменять состояние флажка.

В этих случаях необходимо обновить виджет, для чего следует вызвать его конструктор со строкой `refresh` в качестве аргумента:

```
$("#list1").listview('refresh');
$("#checkbox").val('true').checkboxradio('refresh');
```

Вернемся к последнему примеру. Если изначально определить элемент `list1` в виде пустого виджета `listview`, то после добавления элементов придется обновить его:

```
<ul id="list1">
</ul>
```

После добавления элементов вызовем функцию:

```
$("#list1").listview('refresh');
```

Создание сеток

Существует специальный виджет для создания CSS-сеток, о котором шла речь в одной из предыдущих глав. Нам нужно взять HTML-элемент, имеющий потомков, и преобразовать его в таблицу из *n* столбцов, по числу потомков.

Для этого достаточно вызвать функцию `grid()` платформы jQuery, например:

```
$("#element").grid();
```

В зависимости от количества потомков, к элементу будет применен соответствующий класс `ui-grid-<буква>`, а к потомкам — класс `ui-block-<буква>`.

Изменение содержимого страницы

Если мы изменим большую группу HTML-элементов, содержащую много виджетов, например, создающих различные элементы типа `collapsible` на основании информации из JSON-файла, полученного в ответ на AJAX-запрос, то нам придется обновить весь контейнер. То же самое произойдет, если мы, например, добавим пару элементов `<input>` в текущую форму и захотим преобразовать их в виджеты, словно они присутствовали на странице с самого начала.

Чтобы обновить контейнер и позволить каждому виджету проверить, не нужно ли создавать новые экземпляры элементов, мы можем сгенерировать для страницы событие `create`.

Например:

```
$("#content").html(newHTMLcontentWithWidgets);
$("#page1").trigger("create");
```

Конструктор любого виджета, как правило, обрабатывает событие `create` на странице, и он распознает необходимость в создании новых элементов.

Обработка событий

В jQuery Mobile имеются новые события, доступные при посредстве обычных методов платформы jQuery, таких как `bind` или `live`.

События страницы

Мы привыкли к нормальным событиям HTML-страницы, таким как `load` и `DOMready`, которые генерируются браузером для каждой страницы, загруженной в текущем сеансе. На платформе jQuery Mobile существуют самые разные элементы, к которым можно отнести события. Как мы знаем, документ jQuery Mobile обычно имеет несколько страниц (внутренних или загруженных извне), поэтому мы должны мыслить в терминах загрузки страниц jQuery Mobile.

Каждая страница (элемент с атрибутом `data-role="page"`) имеет набор событий, которые мы можем обработать глобально (одновременно для всех страниц) или одиночным образом, для каждой страницы в отдельности.

Чтобы обработать страничные события глобально, мы можем вызвать метод `$(document).bind` или, говоря более конкретно, метод `$(":jqmData(role='page')").bind`. Мы также можем вызвать метод `live` вместо `bind` для обеспечения связывания со страницами, которые будут добавлены в объектную модель документа в будущем.

Каждая страница имеет события, соответствующие ее созданию, загрузке и отображению.

События создания страницы

У каждой страницы есть события, связанные с ее созданием и инициализацией. Доступны следующие события:

- `pagebeforecreate` — после добавления страницы в объектную модель документа, но до создания ее виджетов;
- `pagecreate` — после создания страницы, но до отображения виджетов;
- `pageinit` — после того как страница полностью загружена (это наиболее часто используемое событие страницы);
- `pageremove` — после удаления страницы из объектной модели документа (как правило, если страница была загружена с помощью AJAX, а сейчас неактивна).

Например, мы можем связать обработчик события `pageinit` со страницей при помощи метода `live` платформы jQuery:

```
$("#page2").live("pageinit", function(event) {
});
```

Помните, что если вызывается метод `bind`, элемент должен присутствовать в объектной модели документа на момент связывания. Если он недоступен, можно воспользоваться методом `live`. Платформа jQuery 1.7 поддерживает новую функцию "включения", которая недоступна при работе с jQuery Mobile 1.0 и jQuery 1.6.4.

События загрузки страницы

Не всякая страница загружается по умолчанию вместе с первым документом jQuery Mobile. Для страниц, загружаемых с помощью AJAX, существуют специальные обработчики событий, обычно связанные с объектом `$(document)`, поскольку в объектной модели документа еще нет страниц, с которыми можно было бы связать обработчик.

Доступны следующие события загрузки страницы:

- `pagebeforeload` — до выполнения любого AJAX-запроса;
- `pageload` — генерируется после загрузки страницы и занесения ее в объектную модель документа;
- `pageloadfailed` — генерируется, если не удалось загрузить страницу.

Каждый из обработчиков этих событий принимает два аргумента: объект-событие и объект-данные.

Первый параметр принимает значения, обычные для обработчиков событий, а в коде можно вызывать методы, например `preventDefault()`, предотвращающий поведение по умолчанию. Опираясь на эту идею, мы можем отменить показ сообщения об ошибке, выводимого платформой по умолчанию, и реализовать нужное нам поведение интерфейса:

```
$(document).bind("pageloadfailed", function(event, data) {
  data.preventDefault();
  // Нестандартная обработка ошибки
});
```

Второй аргумент представляет собой объект, содержащий различные атрибуты, в том числе:

- `url` — абсолютный или относительный URL-адрес в том виде, в каком он был запрошен в методе `$.mobile.loadPage`;
- `absUrl` — абсолютный URL-адрес;
- `dataUrl` — URL-адрес, используемый в качестве идентификатора страницы;
- `options` — все параметры, переданные методу `$.mobile.loadPage`, чтобы можно было узнать, например, был ли запрос сделан методом `get` или `post`;
- `xhr` — объект `XMLHttpRequest` для программирования на более низком уровне;
- `textStatus` — сообщение об ошибке;
- `errorThrown` — объект-исключение, свидетельствующий об ошибке и допустимый только для события `pageloadfailed`;
- `deferred` — аргумент, допустимый только для событий `pagebeforeload` и `pageloadfailed` и только при вызове метода `event.preventDefault()`. В этих случаях мы должны вызвать метод `resolve()` или `reject()` данного объекта, чтобы платформа знала, как обрабатывать ситуацию.

Если вы хотите реагировать на инициализацию страницы, вам не следует обрабатывать события `load`, `ready` и даже `mobileinit`. Вам нужно событие `pageinit`, доступное для каждой страницы. Если вы связываетесь с ним внутри обработчика `mobileinit`, вызывайте метод `live` вместо `bind`.

События показа страницы

Страница может быть проинициализирована один раз, но много раз показана, поскольку пользователь может переходить туда-сюда с одной страницы на другую. Поэтому существует возможность обработки событий, связанных с показом и сокрытием страниц.

Эти события делятся на события смены страницы и события переходов.

Доступны следующие события смены страниц:

- `pagebeforechange` — генерируется до того, как произойдет смена страницы и начнется выполнение перехода;
- `pagechange` — генерируется после того, как произойдет смена страницы;
- `pagechangefailed` — генерируется, если смена страницы невозможна.

Каждый обработчик события принимает два аргумента:

- `toPage` — строка с URL-адресом целевой страницы или объект из модели DOM с целевой страницей, если загружается внутренняя страница;
- `options` — те же самые параметры, которые были переданы методу `$.mobile.changePage`.

Доступны следующие события переходов:

- `pagebeforeshow` — генерируется непосредственно перед выводом страницы на экран с помощью перехода (страница все еще скрыта);
- `pageshow` — генерируется после совершения перехода на страницу, которая в этот момент видна на экране;
- `pagebeforehide` — генерируется непосредственно перед сокрытием страницы (страница все еще видна);
- `pagehide` — выполняется после совершения перехода со страницы, которая в момент уже скрыта.

Любой обработчик события, связанного с переходом, принимает один аргумент, объект из модели DOM jQuery, соответствующий странице. Если событие относится к показу страницы — это объект, соответствующий предыдущей странице; если к сокрытию, то объект соответствует следующей странице.

События виджетов

Каждый виджет, показывающий или скрывающий содержимое динамически, например `collapsible`, генерирует событие `updatelayout`, поскольку меняется компоновка страницы. Вы можете обрабатывать это событие, чтобы обновить еще какие-нибудь параметры пользовательского интерфейса.

Событие смены ориентации

Мобильное устройство все время перемещается, и оно может иметь как минимум два варианта ориентации: книжную и альбомную. При желании мы можем изменить внешний вид приложения или поведение приложения при смене ориентации. Платформа jQuery Mobile предоставляет нам событие `orientationchange`, которое можно привязать к документу.

В настоящий момент это событие связано с событием `resize` на тех платформах, которые не поддерживают низкоуровневое событие `orientationchange`. На некоторых платформах, генерирующих `orientationchange`, рамка окна не меняется, так что вам

не удастся получить корректные значения ширины и высоты. Если вы хотите принудительно сгенерировать это событие, когда обновляются значения, вы можете выполнить оператор `$.mobile.orientationChangeEnabled=false`.

Обработчик события принимает в качестве первого аргумента строку со значением `portrait` или `landscape`. Это значение корректно на любой платформе, поскольку не связано с определением ширины и высоты:

```
$(document).bind("orientationchange", function(orientation) {
  if (orientation=="landscape") {
    // Альбомная ориентация
    } else {
    // Книжная ориентация
  }
});
```

События жестов

Платформа jQuery Mobile предлагает нам события, вызываемые некоторыми жестами пользователя, которые можно связать с любым элементом из объектной модели документа. В версии jQuery Mobile 1.0 доступны следующие события жестов:

- `tap` — генерируется после быстрого прикосновения к экрану;
- `taphold` — генерируется, когда пользователь прикасается к экрану и давит на него в течение одной секунды. Это событие удобно применять для показа контекстных меню;
- `swipeleft` — генерируется, когда пользователь проводит пальцем по экрану справа налево;
- `swiperight` — генерируется, когда пользователь проводит пальцем по экрану слева направо.

В следующем примере мы связываем событие `swiperight` со страницей, на которую нужно вернуться:

```
$(document).bind("mobileinit", function() {
  $("#page2").live("swiperight", goBackToPage1);
});
function goBackToPage1() {
  $.mobile.changePage("#page1", { reverse: true });
  $("#page2").unbind("swiperight", goBackToPage1);
}
```

События виртуальных щелчков

Виртуальный щелчок? Звучит странно, не так ли? Поговорим немного на эту тему. Большинство браузеров в мобильных сенсорных устройствах имеет задержку на 300—500 миллисекунд, когда генерируются события щелчков, такие как `click` или

mouseover), но эта задержка отсутствует, когда генерируются сенсорные события (touchstart или touchmove). Еще одна проблема заключается в том, что не всякий браузер в сенсорном устройстве поддерживает сенсорные события.

События виртуальных щелчков — это оболочки, которыми можно пользоваться вместо событий, связанных со щелчками или прикосновениями, и эти оболочки сами выбирают правильное событие в зависимости от платформы, на которой работает приложение. Такая оболочка нормализует информацию о позиции и может быть использована только для одиночного прикосновения (не многократного).

События виртуальных щелчков обрабатываются точно так же, как события щелчков, но их имена содержат префикс v. Платформа предлагает нам события vclick, vmouseover, vmousedown, vmousemove, vmouseup и vmousecancel.

ГЛАВА 7

Создание тем

Платформа jQuery Mobile позволяет нам настраивать интерфейс приложения с помощью тем и CSS-таблиц. Необходимо помнить, что jQuery Mobile генерирует HTML-код и CSS-таблицы, так что любой элемент интерфейса может быть переопределен с помощью CSS-таблиц.

На рис. 7.1 показано несколько сайтов с разными интерфейсами. Если вы хотите просмотреть больше примеров, посетите сайт **http://jqmgallery.com**.

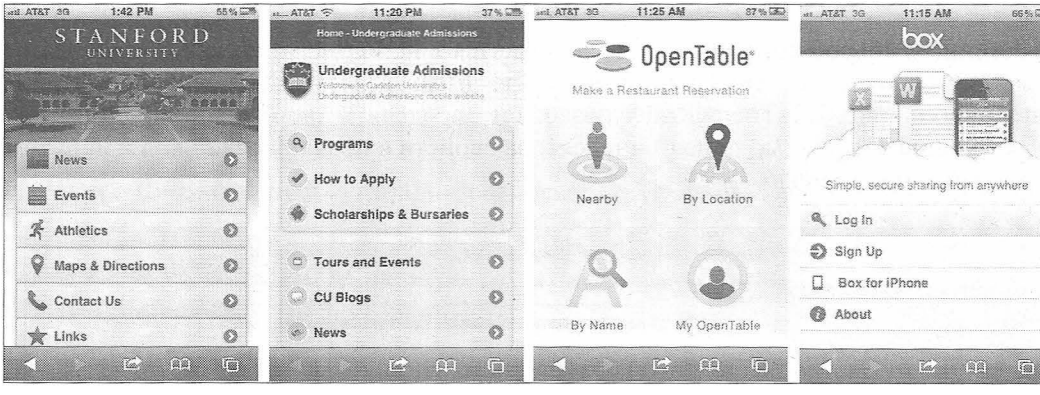


Рис. 7.1. На сайте приводятся сотни мобильных веб-приложений, созданных с помощью jQuery Mobile, и интерфейс некоторых из них значительно отличается от стандартного

Тема представляет собой набор образцов цвета, определяющих:

- цвета текста;
- цвета фона и градиенты;
- шрифт.

Мы можем определить до 26 образцов цвета, от `a` до `z`, хотя в типичном приложении применяется около пяти различных цветов.

Тема также содержит глобальное определение, применяемое к каждому образцу цвета, включающее в себя:

- эффекты, применяемые при отображении цвета и прямоугольников, такие как тени и скругленные углы;
- внешний вид кнопок и других элементов в активном состоянии.

Цель глобального определения состоит в поддержании единого стиля, независимо от того, какой цвет был применен. Например, в теме, принятой по умолчанию, выделенная кнопка всегда имеет синий цвет, независимо от цветового образца, который ей назначен.

Тема хранится в CSS-файле, который подключается к нашему HTML-файлу вместе со структурным CSS-файлом, предоставляемым платформой. Не модифицируйте структурный CSS-файл, поскольку это может привести к несовместимости приложения со следующими версиями платформы. Если вы хотите переопределить поведение, объявленное в структурном CSS-файле, рекомендуется описать новое поведение в отдельном CSS-файле, который будет загружаться после структурного (тем самым переопределяя описания стилей).

Тема не должна включать в себя определение размеров и расположение элементов. Эти параметры определены в структурном файле, и нам не следует изменять их, если мы не очень хорошо представляем себе последствия.

Приложение ThemeRoller

Простейший способ создания темы заключается в использовании бесплатного приложения ThemeRoller, доступного на сайте **http://jquerymobile.com/themeroller**. Как видно на рис. 7.2, ThemeRoller позволяет определять цвет каждого элемента на веб-странице с помощью панели инспектора свойств в правой части окна или путем перетаскивания мышью.

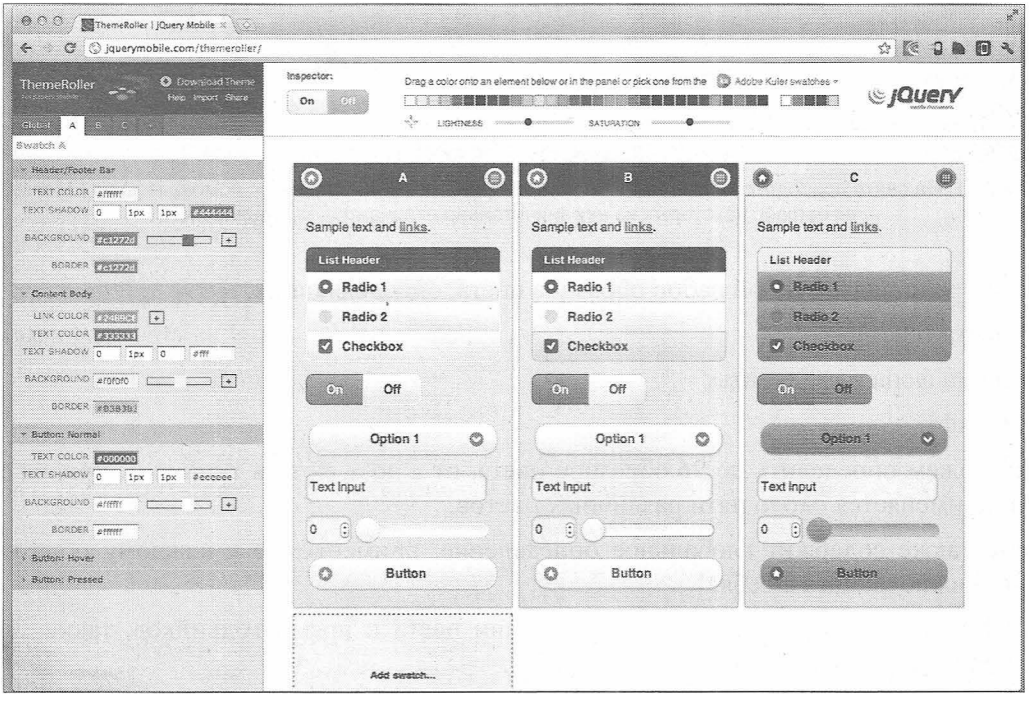


Рис. 7.2. Окно ThemeRoller в браузере настольного компьютера

Интерфейс утилиты разбит на три части:

- селектор образцов цвета слева;
- панель с палитрой вверху;
- панель предварительного просмотра справа.

Если вы перетащите мышью цвет палитры и отпустите его на панели предварительного просмотра в области фона или текста, этот цвет будет автоматически применен к стилю.

Чтобы получить большее разнообразие цветов на палитре, вы можете передвигать регуляторы яркости и насыщенности.

Глобальные настройки

На вкладке **Global** (Глобальные), изображенной на рис. 7.3, мы можем настроить следующие параметры:

- **Font Family** (Семейство шрифтов);
- **Active State** (Активное состояние) — внешний вид кнопок и других элементов в активном состоянии;
- **Corner Radii** (Радиусы углов) — радиусы закругления углов у кнопок и групп элементов;
- **Icon** (Значок) — свойства значков;
- **Box Shadow** (Тень прямоугольника).

Настройки образцов цвета

Мы можем перейти на любую вкладку с буквой, каждая из которых представляет образец цвета (рис. 7.4). Здесь вы можете настроить:

- цвет текста (**TEXT COLOR**), внешний вид тени (**TEXT SHADOW**) и цвет фона (**BACKGROUND**) для заголовка и нижнего колонтитула (группа **Header/Footer Bar**);
- цвета и рамки содержимого (группа **Content Body**);
- цвета и рамки для кнопок в нормальном состоянии, в ситуации, когда на кнопке находится курсор, и в нажатом состоянии (группы **Button: Normal**, **Button: Hover** и **Button: Pressed** соответственно).

Вкладка + позволяет нам добавлять в тему новые образцы цвета.

При определении цвета фона можно воспользоваться небольшим регулятором. Передвигая его, можно отслеживать значения в маленьком поле предварительного просмотра слева в реальном времени. Если вы щелкнете по значку с плюсом справа от регулятора, то получите более глубокую конфигурацию градиента.

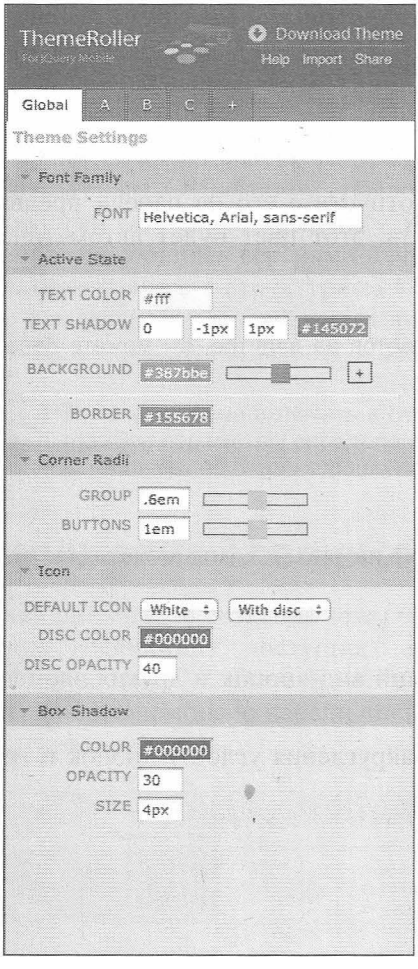


Рис. 7.3. Вкладка **Global** с параметрами

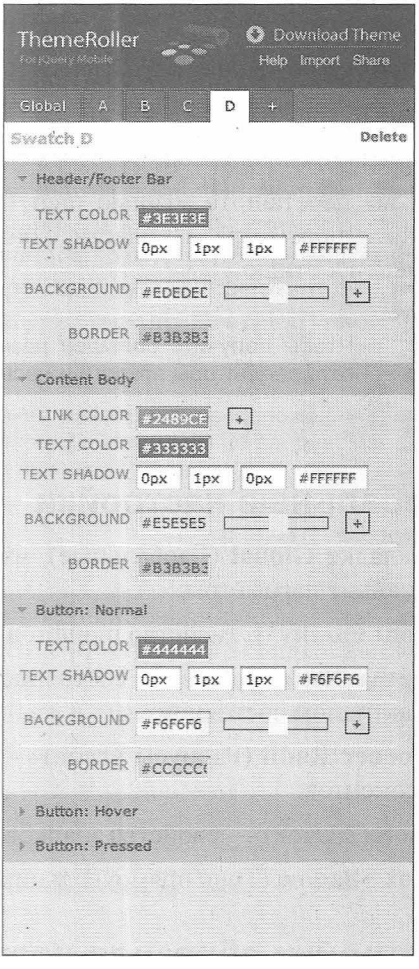


Рис. 7.4. Под каждой буквой скрывается объявление цветов, эффектов и фона для панелей инструментов

Инспектор свойств

Если вы включите инспектор свойств с помощью кнопки в верхней части окна, то щелчок по любому элементу на панели предварительного просмотра приведет к открытию панели со свойствами этого элемента.

Виджет Adobe Kuler

Вообще говоря, Adobe Kuler — это сообщество разработчиков, существующее при поддержке компании Adobe. В рамках этой системы люди делятся своими цветовыми палитрами, а мы можем рассматривать их на сайте **http://kuler.adobe.com** или в любом приложении из комплекта Creative Suite.

Приложение ThemeRoller на платформе jQuery Mobile включает в себя виджет Kuler (рис. 7.5), позволяющий просматривать тысячи цветовых палитр в сети и перетаскивать их на панель предварительного просмотра.

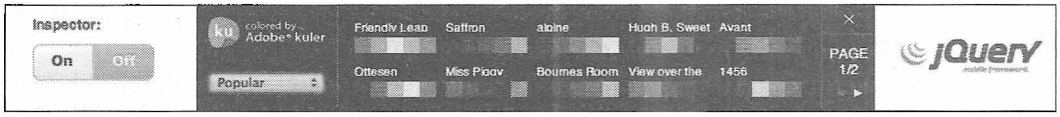


Рис. 7.5. Виджет Adobe Kuler открывается в верхней части окна ThemeRoller и позволяет искать во Всемирной паутине лучшие цветовые палитры

Если у вас есть готовое изображение, например логотип вашего сайта, и вы хотите получить его цветовую палитру, создайте палитру на основе этого изображения на сайте **http://kuler.adobe.com**, сделайте ее публично доступной и найдите ее с помощью ThemeRoller.

Экспорт темы

Закончив разработку темы, вы можете экспортировать ее с помощью кнопки **Download Theme** (Загрузить тему) в верхней части окна ThemeRoller. Вы должны будете указать имя темы (рис. 7.6) и в результате получите фрагмент кода, который можно скопировать в приложение, где будет использована эта тема. Если вы щелкнете по кнопке **Download Zip** (Загрузить ZIP-архив), то получите ZIP-файл, содержащий CSS-файл с вашей темой, уже минимизированный и готовый к отладке.

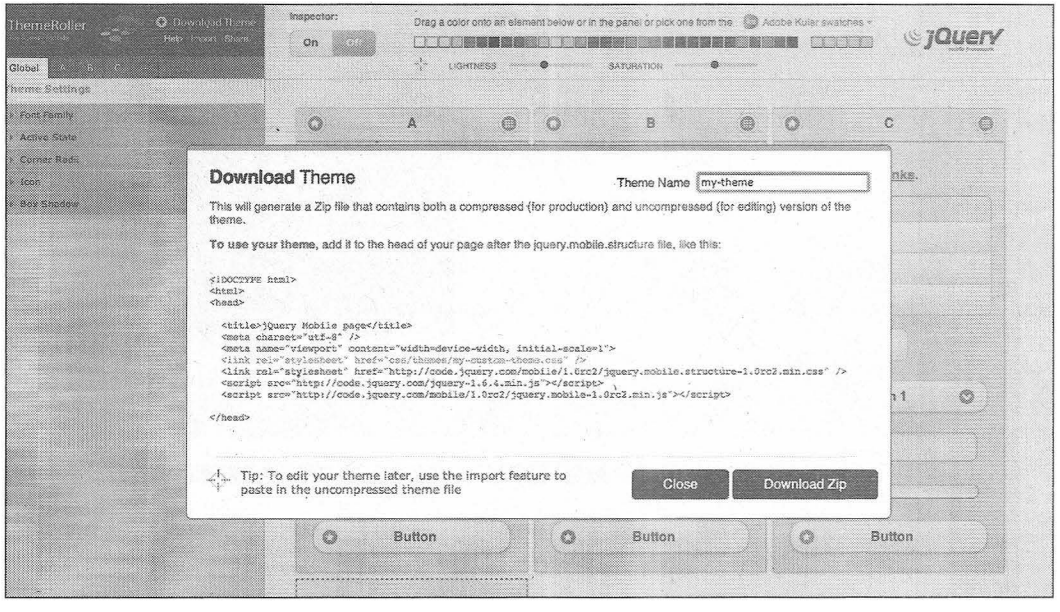


Рис. 7.6. Экспорт темы сводится к указанию имени и загрузке ZIP-файла

Вы можете импортировать тему в приложение ThemeRoller и без труда изменить ее в интерфейсе приложения. Воспользуйтесь функцией **Import** (Импортировать) и скопируйте в окно ваш CSS-файл с темой.

Редактор тем Fireworks

Если вы работаете с приложением Adobe Fireworks, в вашем распоряжении имеется редактор тем jQuery Mobile. Для версии CS5.1 можно загрузить бесплатный добавляемый модуль Fireworks CSS3 Mobile Pack с сайта **http://labs.adobe.com**.

Когда вы установите пакет, откройте Fireworks и выберите **Commands | jQuery Mobile | Create New Theme** (Команды | jQuery Mobile | Создать новую тему). Откроется окно, изображенное на рис. 7.7.

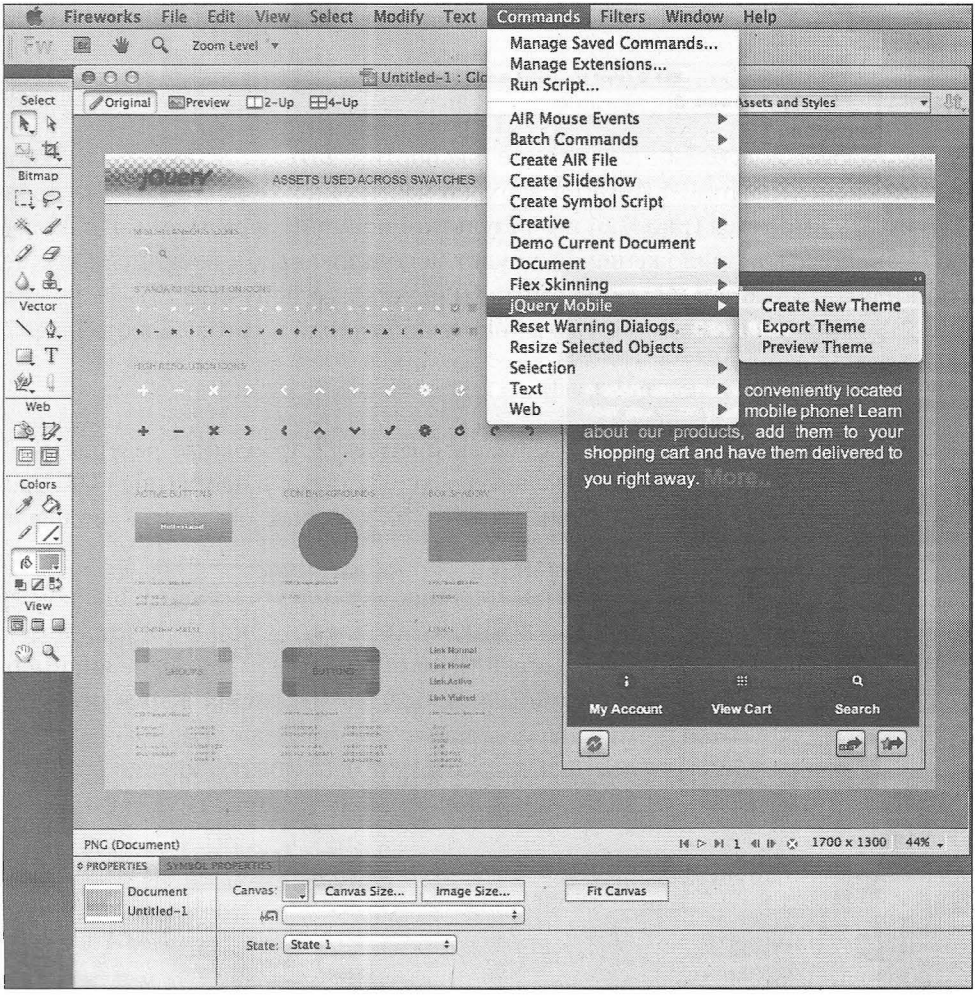


Рис. 7.7. Приложение Adobe Fireworks позволяет создавать темы jQuery Mobile

Если у вас нет Adobe Fireworks, вы можете загрузить 30-дневную пробную версию с сайта **http://adobe.com/go/fireworks**.

Появившееся окно действует как шаблон: мы можем редактировать его содержимое, сохраняя имена экземпляров. Окно содержит шесть страниц, которые можно увидеть, открыв панель **PAGES** (Страницы) (выберите в меню **Windows | Pages** (Окна | Страницы)), изображенную на рис. 7.8.

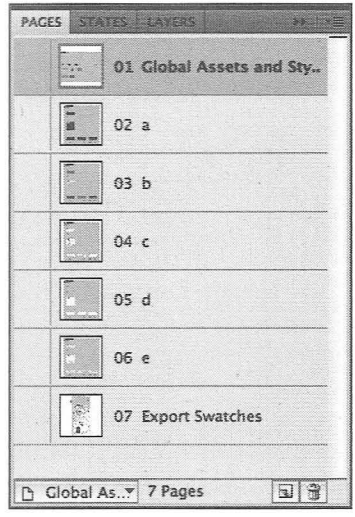


Рис. 7.8. Открыв панель **PAGES**, вы обнаружите общий конструктор стилей и по одной странице на каждый образец цвета

Первая страница называется **Global Assets and Styles** (Глобальные ресурсы и стили) и содержит глобальные стили и значки для всех образцов цвета (рис. 7.9). Далее идут страницы, обозначенные буквами с `a` по `e`, на которых мы можем определить и редактировать образцы цвета (рис. 7.10).

Мы можем добавить образцы цвета, создав копию страницы (сделайте щелчок правой кнопкой мыши и выберите команду **Duplicate Page** (Копировать страницу)) и указав в панели **PAGE** (Страница) имя страницы из одной буквы, например, `f`. Кроме того, мы можем удалить образцы цвета, не нужные в теме, но рекомендуется оставить образцы хотя бы от `a` до `c`.

Пользуясь интерфейсом Fireworks, мы можем выбрать любой элемент на экране и отредактировать его свойства, в том числе:

- цвет текста;
- цвет фона (ровный цвет или линейный градиент);
- стиль и размер шрифта;
- фильтр тени;
- прозрачность (точнее, степень непрозрачности).

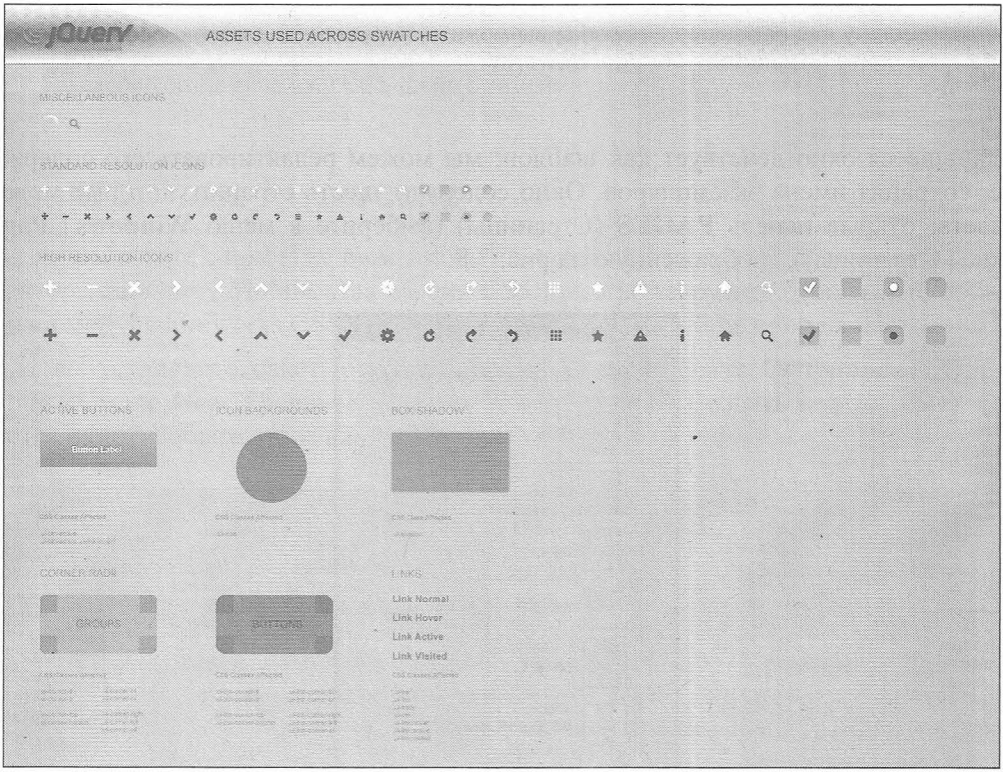


Рис. 7.9. Страница глобальной конфигурации позволяет нам менять значки и глобальные настройки

На странице с глобальными ресурсами мы можем определить:

- значки для экранов с низким и высоким разрешением. Мы можем менять эти пиктограммы при условии, что оставим то же имя экземпляра, например, `ui-icon-plus` для знака "плюс";
- стиль активной кнопки;
- фон значка (степень его непрозрачности);
- тень от прямоугольника (цвет и форму тени);
- радиус закругления углов у кнопок и групп элементов управления (можно выделить любой узел и индивидуально изменить радиус его закругления в панели **Properties** (Свойства);
- все стили ссылок (нормальное состояние, указатель наведен на ссылку, активное состояние, посещенная ссылка).

На любой странице с образцом цвета мы можем определить:

- области заголовка и нижнего колонтитула;
- область содержимого;
- все стили кнопок (нормальное состояние, указатель наведен на кнопку, нажатая кнопка).

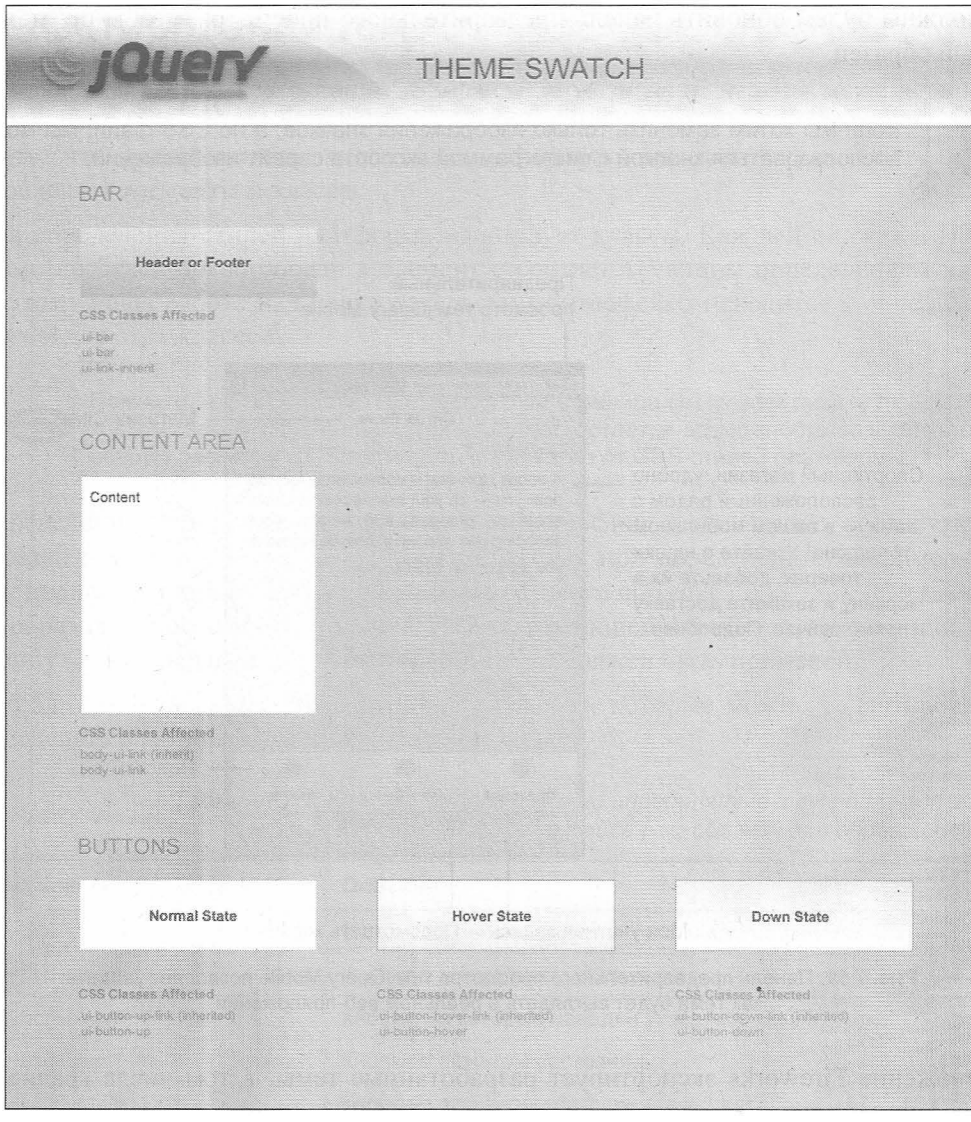


Рис. 7.10. На странице с однобуквенным именем от a до z можно определить любой образец цвета

После того как мы закончим вносить изменения, можем просмотреть файл с темой (рис. 7.11). Выберите **Commands | jQuery Mobile | Preview Theme** (Команды | jQuery Mobile | Предварительный просмотр темы). Если тема нас устраивает, ее можно экспортировать, выбрав **Commands | jQuery Mobile | Export Theme** (Команды | jQuery Mobile | Экспортировать тему).

Мы можем оставить панель предварительного просмотра тем jQuery Mobile постоянно открытой, для чего следует выбрать **Windows | Extensions | jQuery Mobile Theme Preview** (Окна | Расширения | Предварительный просмотр тем jQuery Mobile). Если мы сменим образец цвета (перейдем на другую страницу с образцом),

то должны будем обновить панель предварительного просмотра, чтобы применить новый образец.

Если мы хотим заменить только изображения значков, а не CSS-файл, мы можем воспользоваться кнопкой с пиктограммой экспорта спрайт-изображений.

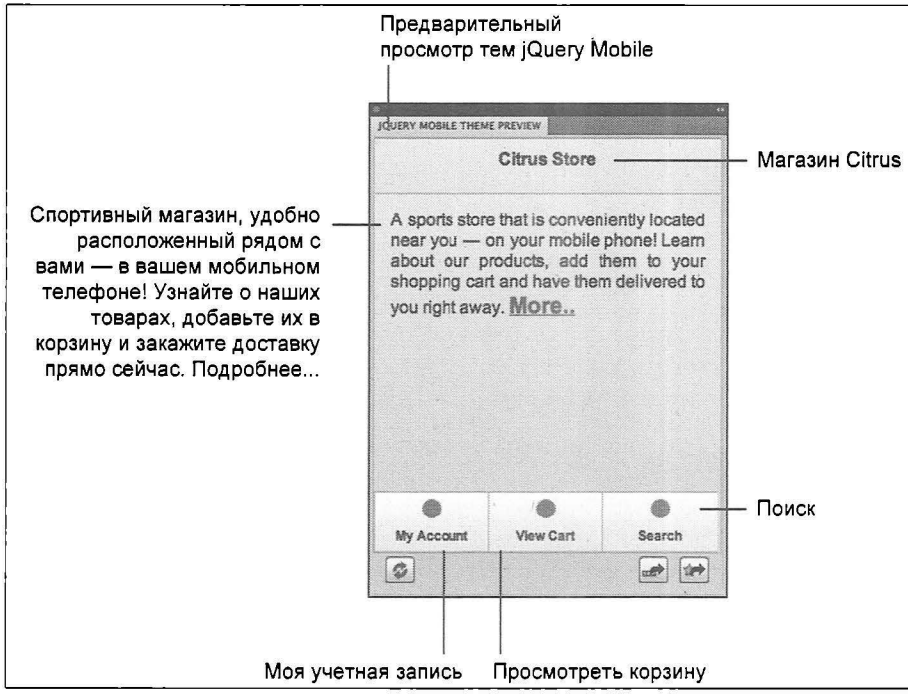
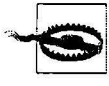


Рис. 7.11. Панель предварительного просмотра тем jQuery Mobile позволяет увидеть, как тема будет выглядеть в реальном веб-приложении

Приложение Fireworks экспортирует разработанные темы, в том числе градиенты (с префиксами для всех видов браузеров), в файл CSS3 и подключает большое изображение со всеми значками к платформе для использования его в качестве CSS-спрайта. Чтобы экспорт был выполнен, мы должны указать имя темы и папку. В результате мы получим папку с изображениями и CSS-файл, которые можно подключать к проекту.

Хороший стиль предписывает сохранять файл в формате Fireworks PNG, чтобы в будущем можно было использовать его для внесения небольших изменений без повторения всего процесса с самого начала.

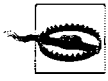


Если мы работаем с собственным CSS-файлом темы, то должны использовать в веб-приложении структурный CSS-файл, а не файл, содержащий тему по умолчанию.

Редактирование тем

Каждая тема — это всего лишь CSS-файл. Это означает, что мы можем написать такой файл "с нуля" и модифицировать его в любом текстовом редакторе. Чтобы знать, что редактировать, необходимо понимать, как платформа jQuery Mobile определяет элементы и классы.

Для определения стилей платформа использует классы. Каждый виджет в HTML-разметке будет преобразован в элемент с соответствующим определением класса. Поэтому наша работа над пользовательским интерфейсом сводится к определению стилей для этих классов.



Помните, что в области содержимого на странице мы можем писать любой HTML-код по своему усмотрению, так что у нас остается возможность создавать собственную разметку с CSS-стилями независимо от CSS-стилей платформы.

Определение любого класса включает в себя префикс `ui` и суффикс, обозначающий образец цвета: `ui-`*<имя>*`-`*<образец_цвета>*. Поэтому при определении, например, кнопок полное имя класса будет `uibtn-a` для цветового образца `a` и `ui-btn-c` для цветового образца `c`. После этого в коде HTML мы напишем атрибут `data-theme` или иной атрибут для определения, какой образец цвета должен быть применен.

В табл. 7.1 приведены имена типичных классов, которые можно редактировать и определять в CSS-файле темы.

Таблица 7.1. *Классы, которые можно определять в теме для настройки пользовательского интерфейса (<x> обозначает образец цвета)*

Имя класса	Описание
`ui-bar-<x>`	Заголовки, нижние колонтитулы и другие панели
`ui-btn-up-<x>`	Кнопки в нормальном состоянии
`ui-btn-hover-<x>`	Кнопки, на которые наведен указатель мыши
`ui-btn-down-<x>`	Кнопки в нажатом состоянии
`ui-btn-active`	Кнопки в активном состоянии (все цвета)
`ui-body-<x>`	Тело страницы
`ui-link-<x>`	Ссылки
`ui-icon-<x>`	Пиктограмма для кнопок других виджетов
`ui-corner-all`	Относится ко всем элементам управления, имеющим закругленные углы
`ui-corner-<tl/tr/bl/br>`	Относится к левому верхнему/правому верхнему/левому нижнему/правому нижнему закругленному углу
`ui-corner-<top/bottom>`	Относится к верхним/нижним закругленным углам
`ui-corner-<left/right>`	Относится к левым/правым закругленным углам
`ui-shadow`	Относится к любому элементу, у которого может быть тень

Таблица 7.1 (окончание)

Имя класса	Описание
ui-disabled	Относится к любому элементу, отключенному с помощью HTML-кода

Если мы захотим, например, обеспечить разный интерфейс для заголовка и нижнего колонтитула (имеющих один класс `ui-bar`), нам придется изменить используемые ими образцы цвета в коде разметки.

Нестандартные переходы

Мы можем создавать собственные переходы в коде на языке JavaScript (как было показано в предыдущей главе) или с помощью таблиц CSS3. Если мы собираемся использовать CSS3-анимации, то должны понимать, как они работают.

Когда мы определяем переход `data-transition`, неизвестный системе, она ищет обработчик на языке JavaScript с тем же именем. Если обработчик не найден, система пытается применить CSS3-анимацию.

Имя перехода используется в качестве имени класса, применяемого как к текущей, так и к следующей странице. Класс `in` применяется к следующей странице, а класс `out` — к текущей.

Это означает, что если мы определим переход по имени `card`, мы должны будем определить селектор для классов `.card.in` и `.card.out`. В качестве необязательного дополнения мы можем создать обратный переход, который будет работать при возврате на исходную страницу. В этом случае добавляются класс `reverse` и определения для классов `.card.in.reverse` и `.card.out.reverse`.

Нам не нужно создавать функции, управляющие временем анимации, поскольку они уже присутствуют в глобальном структурном CSS-файле.

С помощью CSS3-анимаций мы можем создать собственный переход. Переход `card` будет аналогичен переходу `slide`, но у нас страницы будут накладываться одна на другую, как карты в колоде. Переходя к следующей карте-странице, мы снимаем верхнюю (текущую) и открываем ту, что под ней (без анимации).

```
.card.out {
  -webkit-transform: translateY(-100%);
  -webkit-animation-name: cardout;
  z-index: 1; /* Эта страница сверху */
}
.card.in {
  -webkit-transform: translateY(0);
```

```
  z-index: 0; /* Эта страница снизу */
}
@-webkit-keyframes cardout {
  from {
    -webkit-transform: translateY(0%);
  }
  to {
    -webkit-transform: translateY(-100%);
  }
}
```

Это объявление можно поместить в CSS-файл темы или в другой CSS-файл. Чтобы использовать переход, мы определили его с помощью атрибута `data-transition` или в коде JavaScript:

```
<a href="#page2" data-role="button" data-transition="card">Page 2</a>
<!-- Страница 2 -->
```

Если мы хотим, чтобы эта анимация работала в браузерах Firefox для Android, Opera или Internet Explorer 10, то должны создать альтернативные варианты кода с соответствующими префиксами `-moz`, `-o` и `-ms`.

ГЛАВА 8

Установка и автономная работа

Применяя HTML5 и некоторые другие расширения, мы можем заставить свое приложение jQuery Mobile работать совершенно автономно, словно оно было установлено на мобильном устройстве как низкоуровневое приложение.

Мы также можем упаковать веб-приложение jQuery Mobile как низкоуровневое для целей его распространения, однако это тема одной из следующих глав. А в этой главе мы создадим автономный вариант веб-приложения, не предназначенный для распространения.

При таком решении пользователь может обратиться к веб-приложению из мобильного браузера и установить его на своем устройстве. Это означает, что при последующем обращении к тому же приложению по тому же URL-адресу (или посредством значка приложения) оно будет загружено из локальной памяти, а не с нашего сервера.

Определение пакета

Первое, что мы должны сделать, — определить пакет. Для этого мы воспользуемся API-интерфейсом из HTML5, называемым Application Cache (кэш приложения). Он также известен под именем Offline API (API для автономной работы) и доступен в проекте консорциума W3C.

В настоящее время этот API-интерфейс работает не на каждом мобильном браузере, но совместим с большинством смартфонов и планшетов. Самую свежую информацию о его совместимости вы найдете на сайте **http://mobilehtml5.org**.

Для начала разберемся, чего мы хотим. Нужно ли нам полностью автономное приложение? Хотим ли мы, чтобы некоторые страницы или данные обновлялись с сервера при каждой загрузке? Или мы предпочтем иметь локальный кэш, обновляемый при обращении к приложению через Интернет?

Второй шаг заключается в определении пакета. Пакет — это список файлов, которые должны быть загружены браузером, когда пользователь заходит на наш сайт. Этот список должен включать в себя все файлы с кодом JavaScript, CSS-файлы, изображения и ресурсы, к которым приложение будет обращаться в автономном режиме. А поскольку мы создаем приложение jQuery Mobile, мы должны добавить в пакет все файлы jQuery Mobile, в том числе структурный CSS-файл, темы, изображения и JavaScript-файлы.

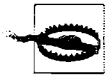
Манифест HTML

Список-пакет содержится в текстовом файле, который называется *манифестом кэша*. В первой строке этого файла должны быть слова `CACHE MANIFEST`, за которыми идет список URL-адресов (относительных или абсолютных) всех ресурсов, подлежащих загрузке в устройство.

Наличие главного HTML-файла подразумевается, так что нам не нужно добавлять его в манифест.

Не имеет значения, находятся ли указанные файлы на одном сервере. Отсюда следует, что мы всегда можем получить файлы платформы jQuery Mobile из сети CDN.

В файле манифеста могут присутствовать строки с комментариями, начинающиеся с символа `#`.



Если при установке пакета не удастся загрузить хотя бы один файл, весь пакет становится недействительным. Таким образом, если мы указываем ресурсы, расположенные на сторонних серверах, установка нашего приложения попадает в зависимость от работы этих серверов.

Например, манифест типичного приложения jQuery Mobile, состоящего из одного документа (без внешних страниц), будет выглядеть так:

```
CACHE MANIFEST:
# ядро jQuery Mobile
http://code.jquery.com/jquery-1.6.1.min.js
# файлы jQuery Mobile без нестандартной темы
http://code.jquery.com/mobile/1.0/jquery.mobile-1.0.min.css
http://code.jquery.com/mobile/1.0/jquery.mobile-1.0.min.js
http://code.jquery.com/mobile/1.0/images/ajax-loader.png
http://code.jquery.com/mobile/1.0/images/icons-18-black.png
http://code.jquery.com/mobile/1.0/images/icons-18-white.png
http://code.jquery.com/mobile/1.0/images/icons-36-black.png
http://code.jquery.com/mobile/1.0/images/icons-36-white.png
# Мои файлы, адреса указаны относительно HTML-документа
images/logo.png
data/countries.json
```

Такой файл обычно хранится под именем offline.appcache, и для корректной работы он должен иметь MIME-тип `text/cache-manifest`. Если вы не знаете, как устанавливать этот MIME-тип, посоветуйтесь с администратором вашего сервера.

Если ваш сервер поддерживает PHP, вы можете просто изменить расширение файла на PHP и воспользоваться следующим шаблоном:

```
<?php header('Content-Type: text/cache-manifest');
?>CACHE MANIFEST:
```

Тогда вы можете обойтись без какой-либо специальной конфигурации.

Следующий шаг состоит в указании URL-адреса файла манифеста в HTML-коде. Это делается с помощью атрибута `manifest`, появившегося в стандарте HTML5. Этот атрибут должен быть указан в элементе `<html>`.

```
<html manifest="offline.appcache">
  <!-- Наше веб-приложение -->
</html>
```

Процедура загрузки

Когда браузер, совместимый с jQuery Mobile, обнаружит объявление манифеста, он загрузит файл манифеста в фоновом режиме. Если это окажется корректный файл с соответствующим MIME-типом, начнется загрузка веб-приложения.

Этот фоновый процесс абсолютно независим от обычной загрузки страницы. Каждый файл манифеста загружается отдельно и сохраняется в специальном месте.

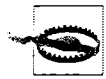
Если не удается загрузить хотя бы один ресурс манифеста (файл не существует, сервер не работает и т. д.), весь пакет считается недействительным, и в локальной памяти ничего не сохраняется.

Если пакет установлен удачно, то при следующем обращении пользователя по тому же URL-адресу браузер загрузит локальную версию и не станет искать приложение на сервере. Если пользователь подключен к Интернету, браузер автоматически перейдет в режим автономной работы. Это означает, что по умолчанию мы не сможем обратиться ни к какому ресурсу в Интернете, не объявленному заранее в файле манифеста.

Если же установить пакет не удалось, то в следующий раз страница будет загружена с сервера. Причины неудачи в установке пакета могут быть следующими:

- файл манифеста некорректен, не существует или имеет неправильный MIME-тип;
- хотя бы один из ресурсов манифеста недоступен;
- пользователь закрыл браузер или покинул веб-страницу до того, как все ресурсы были загружены.

Инструменты отладки позволяют нам увидеть содержимое пакета. Для симулятора iOS Simulator можно загрузить бесплатную утилиту iWebInspector (с сайта **http://iwebinspector.com**). Информация о кэше приложения будет доступна на вкладке **Resources** (Ресурсы), как показано на рис. 8.1.



Чтобы иметь доступ к ресурсам, загруженным в кэш приложения, мы не должны изменять код. Мы просто обращаемся к ресурсу так, словно он находится в Интернете, указывая его относительный или абсолютный URL-адрес, указанный в манифесте. Браузер "догадается" загрузить локальную версию этого файла.

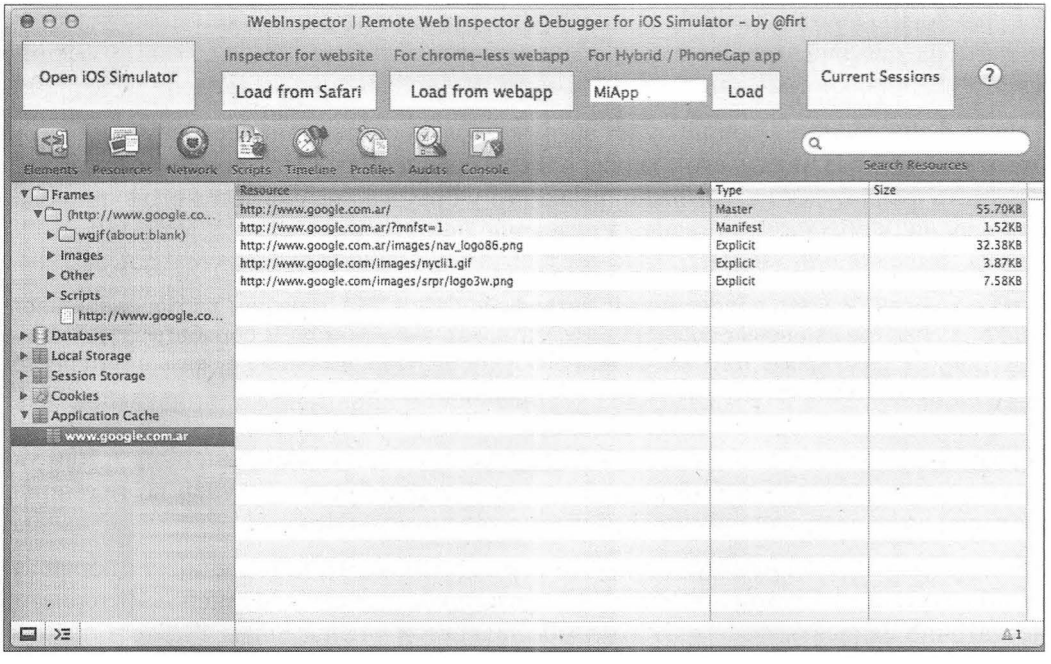


Рис. 8.1. На вкладке **Resources** приложения iWebInspector видно содержимое пакета кэша приложения

Обращение к ресурсам в Интернете

Если мы попытаемся обратиться к какому-либо ресурсу, изначально не определенному в файле манифеста, у нас ничего не получится, потому что приложение работает автономно и изолированно. В случае, когда мы уверены, что приложению понадобится информация из Всемирной паутины, мы можем определить ресурс в файле манифеста в специальном разделе `NETWORK:`. По умолчанию все ресурсы определены в неявно создаваемом разделе `CACHE:`. Раздел — это всего лишь строчка, заканчивающаяся двоеточием. Итак, если мы хотим, чтобы файл countries.json всегда загружался с сервера, мы можем изменить манифест следующим образом:

```
CACHE MANIFEST:
# ядро jQuery Mobile
http://code.jquery.com/jquery-1.6.1.min.js
# файлы jQuery Mobile без нестандартной темы
http://code.jquery.com/mobile/1.0/jquery.mobile-1.0.min.css
http://code.jquery.com/mobile/1.0/jquery.mobile-1.0.min.js
http://code.jquery.com/mobile/1.0/images/ajax-loader.png
http://code.jquery.com/mobile/1.0/images/icons-18-black.png
http://code.jquery.com/mobile/1.0/images/icons-18-white.png
http://code.jquery.com/mobile/1.0/images/icons-36-black.png
http://code.jquery.com/mobile/1.0/images/icons-36-white.png
# Мои файлы приложения, адреса указаны относительно HTML-документа
images/logo.png
```

```
# Ресурсы, загружаемые из Всемирной паутины
NETWORK:
data/countries.json
```

Теперь файл countries.json не будет загружаться с остальными ресурсами, и браузер будет каждый раз читать его с сервера. Если соединения с Интернетом отсутствует, получить файл будет невозможно, разве что у браузера не окажется его кэшированной версии (в обычном веб-кэше, а не в кэше приложения).

Если браузер поддерживает интерфейс Offline API, в нашем распоряжении появляются два новых события документа, которые мы можем обрабатывать: `online` и `offline`. Кроме того, мы можем опросить браузер, имеется ли связь с Интернетом, с помощью булева свойства `window.onLine`.

В разделе `NETWORK:` можно использовать символы подстановки, например `*`, или указывать папки, и тогда любой ресурс из такой папки будет доступен во Всемирной паутине, даже если приложение работает в автономном режиме.

Таким образом, если нам нужно автономное приложение, которое может обращаться к ресурсам во Всемирной паутине, когда пользователь соединен с Интернетом, достаточно написать:

```
NETWORK:
*
```

И тогда в предыдущем примере только файлы, указанные до конструкции `NETWORK:`, будут загружены из автономного пакета, а все остальные ресурсы будут загружаться из Всемирной паутины.

В файле манифеста допускается также наличие раздела `FALLBACK:`, обсуждение которого выходит за рамки этой книги. В этом разделе мы можем определить альтернативные URL-адреса для сетевых ресурсов на случай отсутствия интернет-соединения.

Обновление ресурсов

Как было сказано ранее, если пакет установлен, то все ресурсы, в том числе основной HTML-документ, будут всегда загружаться из локального хранилища, а не из Всемирной паутины. Поэтому разумно задать вопрос: как обновлять тот или иной ресурс? Что делать, когда нам понадобится обновить CSS-файл темы, изменить изображение или добавить новую ссылку в HTML-документ?

Я должен признаться, что чуть-чуть солгал. Нет, я кое-что опустил. Конечно, это была не ложь; я просто не все сказал. Когда пользователь открывает наше приложение во второй раз (и все последующие разы), браузер загружает приложение из локального хранилища и в фоновом режиме пытается получить в сервера обновленную версию файла манифеста.

Если интернет-соединение отсутствует, ничего не произойдет, и пользователь получит локальную версию приложения. Если соединение имеется, браузер сравнива-

ет новый файл манифеста, только что загруженный с сервера, с локальной версией манифеста, полученной вместе с загруженным приложением. Браузер производит побайтовое сравнение двух файлов и, если хотя бы один байт изменялся, старый манифест считается недействительным, а с сервера заново загружаются все ресурсы в соответствии с новым файлом манифеста.

Прочитайте предыдущий абзац снова. Перечитали? Давайте разберемся, что происходит. Если мы изменим CSS-файл, а его имя останется прежним, манифест не изменится, и загруженные файлы обновлены не будут. Чтобы приложение обновилось, нужно изменить манифест.

Изменение манифеста ради обновления приложения может заключаться в добавлении пробела, изменении имени ресурса (например, указании другой версии) или даже добавлении комментария со случайным содержимым, а еще лучше, с датой модификации:

```
CACHE MANIFEST
# Приложение обновлено 2012-01-01
```

Если мы изменим хотя бы один байт, например, укажем другую дату, старый манифест станет недействительным, и платформа заново загрузит все файлы. Да, все файлы. Этот API-интерфейс не позволяет обновить только один ресурс.

Используя комбинацию API-интерфейсов Application Cache и Web Storage, мы можем создать механизм загрузки ресурсов, таких как CSS-файлы, JavaScript-код или изображения, и сохранения их в локальной памяти без повторной загрузки всего приложения.

С обновлением приложения через обновление манифеста связана одна неприятная проблема. Если имеется обновление, платформа заново загружает все ресурсы, указанные в манифесте. Однако это происходит в фоновом режиме, пока старые файлы отображаются на экране. То есть, при наличии обновления пользователь получает предыдущую версию, пока не перезагрузит приложение. В следующий раз установленный пакет уже будет другим, и он загрузит новые ресурсы.

Все это означает, что если мы изменим манифест, пользователю для получения новой версии придется загрузить страницу дважды. Проблему можно разрешить, если задействовать обработку событий.

Объект JavaScript

Существует глобальный объект JavaScript, помогающий нам узнать статус кэша приложения. Объект `applicationCache` имеет свойство `status`, принимающее одно из значений, перечисленных в табл. 8.1.

Чтобы наш код был совместим со всеми браузерами, мы должны всегда начинать с проверки, доступен ли объект:

```
if (window.applicationCache!=undefined) {
  // API доступен
}
```

Таблица 8.1. Допустимые значения свойства `applicationCache.status`

Значение	Константа	Описание
0	UNCACHED	Первая загрузка страницы, или файл манифеста недоступен
1	IDLE	Кэш не используется
2	CHECKING	Локальный файл манифеста сравнивается с файлом манифеста на сервере
3	DOWNLOADING	Ресурсы загружаются (впервые или в результате обновления)
4	UPDATEREADY	Обновление загружено, но будет доступно при следующей загрузке

Для опросов статуса можно применять константы, например:

```
if (window.applicationCache!=undefined) {
  // API доступен
  if (applicationCache.status==applicationCache.UPDATEREADY) {
    // Имеется обновление, ожидающее перезагрузки
  }
}
```

Объект `applicationCache` имеет методы `update()` и `swapCache()`. Первый принудительно выполняет проверку обновлений, а второй переключает приложение со старого кэша ресурсов на новый (если новый уже загружен). Однако уже загруженные HTML-документ и ресурсы не будут доступны, пока мы не сделаем полную перезагрузку методом `history.reload()`.

События

Объект `applicationCache` имеет события, обработка которых позволит нам справиться с любой ситуацией. Например, если пользователь посещает наш сайт впервые, мы можем показывать сообщение "Приложение загружается", пока выполняется загрузка ресурсов, чтобы пользователь подождал, а вероятность завершения загрузки повысилась.

События объекта `applicationCache` перечислены в табл. 8.2.

Таблица 8.2. События, которые мы можем обрабатывать с помощью объекта `applicationCache`

Имя события	Генерируется, когда
checking	Браузер проверяет манифест
downloading	Браузер приступил к загрузке ресурсов манифеста
progress	Очередной ресурс загружен (генерируется для каждого ресурса, так что мы можем создать индикатор хода загрузки)
cached	Процесс первой загрузки завершился удачно

Таблица 8.2 (окончание)

Имя события	Генерируется, когда
`noupdate`	Произведено сравнение локального манифеста с серверным, и обновление отсутствует
`updateready`	Было обнаружено обновление, новые ресурсы загружены корректно, и приложение ждет перезагрузки
`error`	При загрузке ресурса произошла ошибка
`obsolete`	При проверке обновлений манифест стал недействительным, приложение было удалено из памяти и не будет работать в автономном режиме в следующий раз

В типичной ситуации мы поступаем следующим образом:

- перехватываем событие `downloading`, чтобы показать пользователю сообщение и, возможно, анимацию загрузки;
- перехватываем событие `progress`, чтобы показать индикатор загрузки (пример представлен на рис. 8.2);
- перехватываем событие `cached`, чтобы убрать сообщение о загрузке и сообщить пользователю об установке приложения;

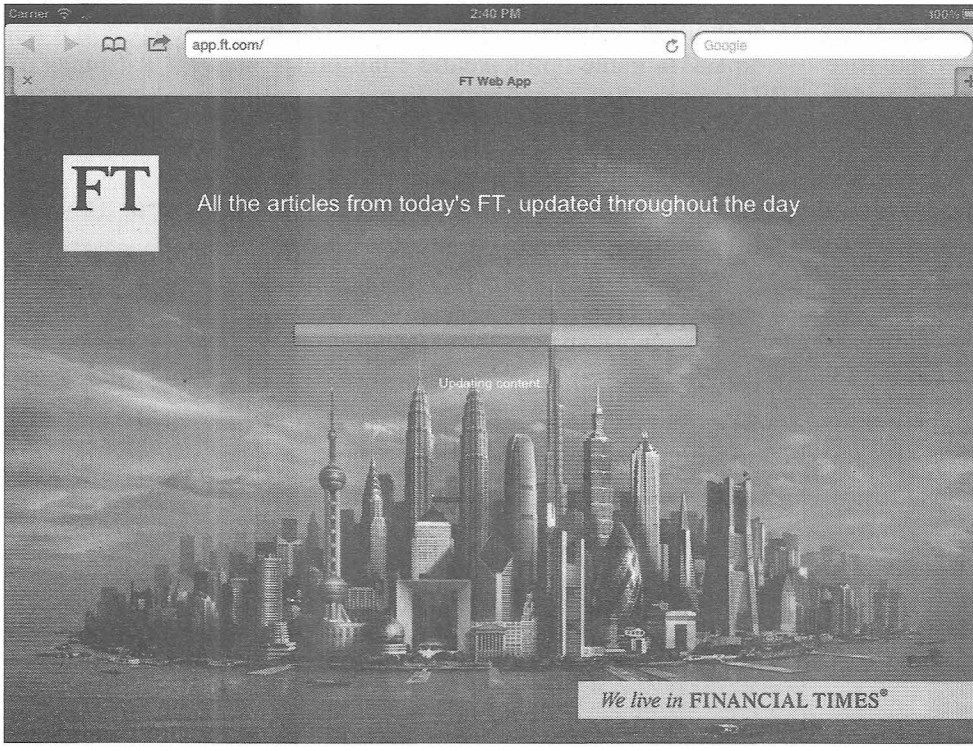


Рис. 8.2. Выполняется загрузка приложения Financial Times для iPad (**app.ft.com**)

- перехватываем событие `error`, чтобы убрать сообщение о загрузке и сообщить пользователю об ошибке;
- перехватываем событие `updateready`, чтобы сообщить пользователю о загрузке обновления и предложить ему перезагрузить приложение и работать с обновленной версией.

Мы можем привязаться к этим событиям с помощью объекта `addEventListener`, например:

```
if (window.applicationCache!=undefined) {
  // API доступен
  applicationCache.addEventListener('updateready', function() {
    // Имеется обновление, ожидающее перезагрузки
    if(confirm("There is an update ready. Do you want to load it now?")){
    // "Имеется обновление. Загрузить сейчас?"
    history.reload();
    }
  });
}
```

Установка значка

Когда приложение загружается из кэша приложения, мы можем предложить пользователю добавить значок на главный экран устройства или в меню приложений. Это позволит ему открывать наше приложение с помощью ярлыка, избавляя от повторного набора URL-адреса. Даже сообразительному пользователю идея открытия автономного приложения (которое можно запустить, находясь в самолете) путем ввода URL-адреса покажется странной.

Пользователь может добавить значок даже при отсутствии манифеста. В этом случае он будет ярлычком для интернет-версии приложения.

Можно предложить пользователю добавить приложение в закладки. Однако типичный владелец мобильного устройства чаще использует значки, чем закладки.

Ни одна платформа не позволяет нам устанавливать значок автоматически, так что мы должны предложить пользователю сделать это, предоставив ему инструкцию.

Предложение по установке

Предложение пользователю установить наше приложение может выглядеть поразному. Например, YouTube, Google Maps и Facebook на iOS выводят всплывающее окно (элемент `<div>`), которое выглядит так, как показано на рис. 8.3[1].

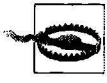
[1] Всплывающее окно на экране устройства содержит текст: "Установите это приложение на своем телефоне: нажмите на стрелку и выберите **Добавить на главный экран**". — *Прим. перев.*

Рис. 8.3. Многие мобильные сайты выводят всплывающие окна с предложением добавить значок на главный экран

Нам нужно будет сохранить cookie-файл или значение `localStorage` стандарта HTML5, чтобы знать, что мы уже предлагали пользователю установить приложение, и не выводить его повторно.

Имя значка

На некоторых платформах имя значка будет показано на главном экране (рис. 8.4), а на других — в меню приложения (рис. 8.5)[1]. По умолчанию имя под знаком определяется в HTML-элементе `<title>`. Отсюда следует, что если мы хотим выводить предложение пользователю, у нас должен быть очень короткий заголовок (одно-два слова), умещающийся в отведенную область.



Если при открытом документе jQuery Mobile пользователь захочет добавить веб-приложение на главный экран или в меню приложений, находясь не на первой странице (а на какой-то другой, внутренней или внешней), то ярлык будет указывать не на главную страницу, а на текущую; а в качестве имени значка будет взято значение атрибута `data-title`.

На рис. 8.6 показано, как пользователь может добавлять наш сайт на главный экран на разных платформах.

---

[1] Текст сообщения на рис. 8.4: "На главный экран будет установлен значок, позволяющий быстро перейти на этот сайт", а на рис. 8.5: "На ваш главный экран будет добавлен ярлык для этого сайта". — *Прим. перев.*

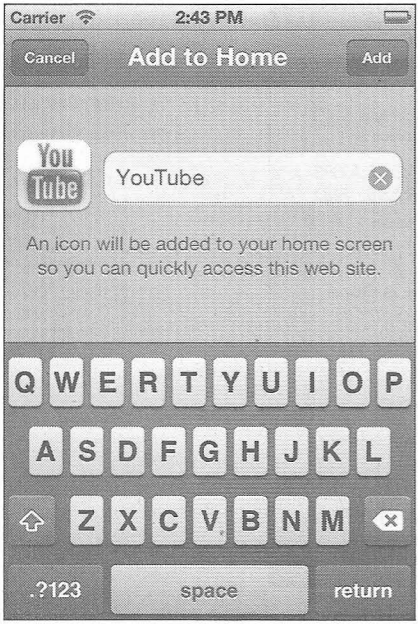


Рис. 8.4. На платформе iOS пользователь может добавить значок на главный экран

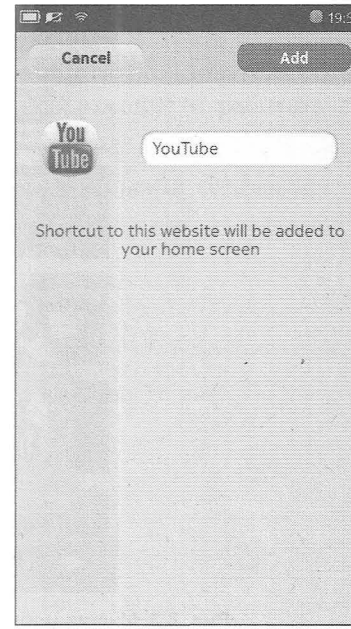


Рис. 8.5. Платформа Nokia N9 тоже позволяет добавить значок на главный экран, как и Android и iPhone

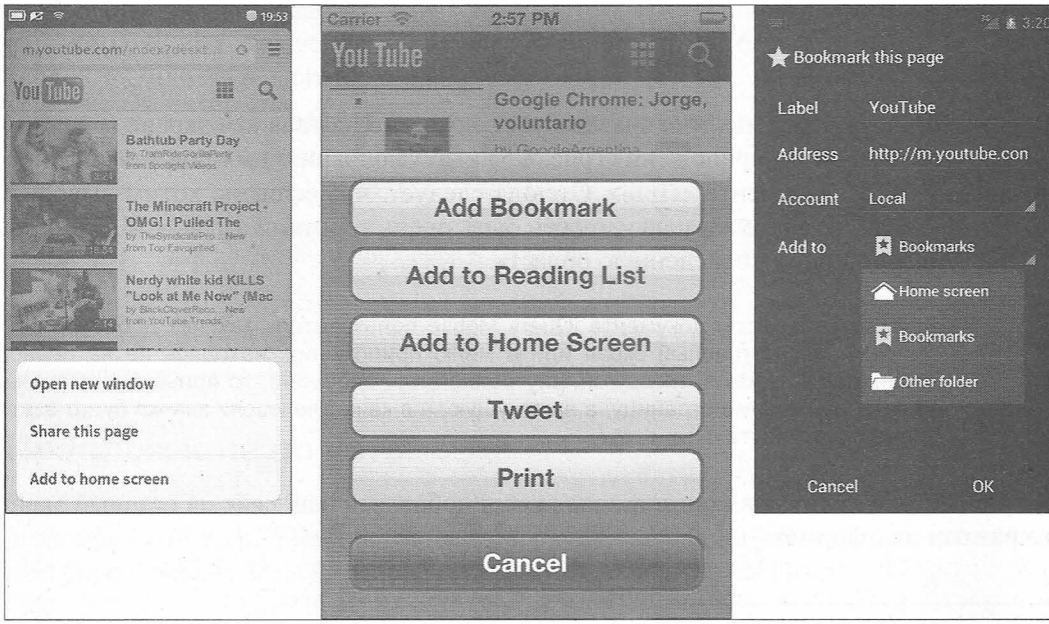


Рис. 8.6. Меню разных платформ, позволяющие добавить значок на главный экран или в меню приложения

Определение значка

На обычной веб-странице единственное определение значка делается с помощью классического элемента `<link>` с указанием файла `favicon`. Как правило, это определение включает в себя атрибут `rel="icon"` и `rel="shortcut icon"` для обеспечения совместимости со всеми браузерами:

```
<link rel="icon" type="image/png" href="favicon.png" />
<link rel="shortcut icon" type="image/png" href="favicon.png" />
```

Проблема заключается в том, что изначально использовался формат ICO (растровое изображение 16×16 пикселов), а теперь поддерживаются и другие форматы, в частности PNG, но они обычно имеют небольшой размер (максимум 32×32 пиксела) и оказываются малы по сравнению со значками на главных экранах смартфонов и планшетов. Поэтому такие значки практически не применяются или, в крайнем случае, располагаются в углу "настоящей" пиктограммы.

Если платформа не поддерживает значки главного экрана, мы всегда имеем возможность упаковать приложение jQuery Mobile в гибридное (например, с помощью PhoneGap) и распространять его как низкоуровневое приложение на нашем сайте или через интернет-магазины. Эта тема обсуждается в последующих главах.

Для определения значка нет других стандартов. Однако разработчики Apple для iOS создали тег `<meta>` для собственных значков, который теперь можно применить на других платформах, например, в браузерах Android Browser и MeeGo 1.2 (браузер для Nokia N9). Мы должны обеспечить столько значков, сколько сможем, а каждая платформа будет выбирать ту, которую поддерживает. Если платформа не поддерживает ни одну пиктограмму, она воспользуется стандартной или возьмет снимок левого верхнего угла сайта, где обычно расположен логотип.

Стандарт, фактически установленный компанией Apple, включает в себя также тег `<link>` с атрибутом `rel="apple-touch-icon"` или `rel="apple-touch-icon-precomposed"` и, возможно, с необязательным атрибутом `sizes`.

По умолчанию, принятому в Apple, каждый значок, созданный с атрибутом `rel="apple-touch-icon"`, имеет закругленные углы и отбрасывает тень, кроме того, к нему применяется 3D-эффект "сияние". Если мы хотим отказаться от тени и сияния (имеющих смысл для прозрачных пиктограмм), мы можем указать атрибут `rel="apple-touch-icon-precomposed"`.

Браузер Android Browser поддерживает только атрибут `rel="apple-touch-icon-precomposed"` и не применяет к значку никакого эффекта.

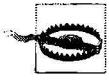
Мы можем создать значки разных размеров для разных платформ. Если мы предоставим только одну пиктограмму, она будет масштабироваться на каждой платформе, и результаты не всегда окажутся удовлетворительными. Лучше всего написать несколько элементов `<link>` с разными значениями атрибута `sizes`, например `sizes="57x57"`. Всем другим форматам следует предпочесть PNG.

Допустимые размеры значков перечислены в табл. 8.3.

Таблица 8.3. Допустимые размеры значков на главном экране

Платформа	Размер	Значение атрибута rel	Атрибут sizes
iPhone 3, 3GS iPod Touch 1-3	57×57	apple-touch-icon и apple-touch-iconprecomposed	Поддерживается, начиная с версии 4.2
iPhone 4, 4S, iPod Touch 4	114×114	apple-touch-icon и apple-touch-iconprecomposed	Поддерживается, начиная с версии 4.2
iPad 1, 2	72×72	apple-touch-icon и apple-touch-iconprecomposed	Поддерживается, начиная с версии 4.2
Android Browser	Любой	apple-touch-iconprecomposed	Не поддерживается
Nokia N9 Browser	80×80	apple-touch-icon	Поддерживается

Браузер Android Browser не поддерживает атрибут `sizes` и просто проигнорирует его. Если мы определим несколько значков, Android выберет последний, атрибут которого имеет суффикс `precomposed`. Если нам нужно, чтобы платформа iOS игнорировала тег, предназначенный для Android, поможет следующий трюк. Укажите недопустимое значение атрибута `sizes`, например `sizes="android-only"`.



Устройства на платформе iOS версии 4.2 или ниже не воспринимают атрибут `sizes`, поэтому, если мы объявим значки разного размера, взят будет последний. Отсюда следует рекомендация ставить значок меньшего размера ниже.

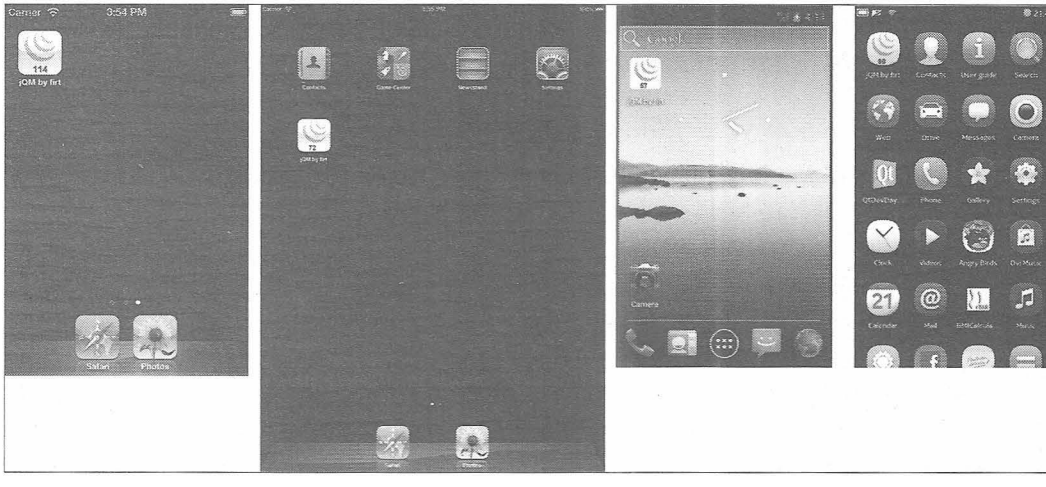


Рис. 8.7. Один и тот же значок разного размера установлен на главных экранах или экранах приложений разных платформ

Итак, для определения всех возможностей значков нужно написать следующий код внутри элемента `<head>` и сохранить значок во всех указанных размерах:

```
<link rel="icon" href="icons/icon32.png">
<link rel="shortcut icon" href="icons/icon32.png">
<link rel="apple-touch-icon" href="icons/icon57.png" sizes="57x57">
```

```
<link rel="apple-touch-icon" href="icons/icon114.png" sizes="114x114">
<link rel="apple-touch-icon" href="icons/icon72.png" sizes="72x72">
<link rel="apple-touch-icon" sizes="80x80" href="icons/icon80.png">
<link rel="apple-touch-icon-precomposed"
      sizes="android-only" href="icons/icon57.png">
```

На рис. 8.7 показано, как наш значок выглядит на главных экранах устройств с разными платформами.

Полноэкранный режим

На устройствах, работающих только под управлением iOS, в частности, на iPhone и iPad, мы можем пойти дальше в разработке нашего приложения. Мы можем создать полноэкранное приложение без пользовательской оболочки, т. е. без каких бы то ни было элементов интерфейса Safari (адресной строки или панели инструментов). Это будет работать, только если пользователь откроет приложение из главного экрана, и если мы определим на HTML-странице следующий тег `<meta>`:

```
<meta name="apple-mobile-web-app-capable" content="yes">
```

Тогда, после открытия страниц, она заполнит весь экран, и мы должны будем явным образом обеспечить в интерфейсе кнопки возврата (рис. 8.8).

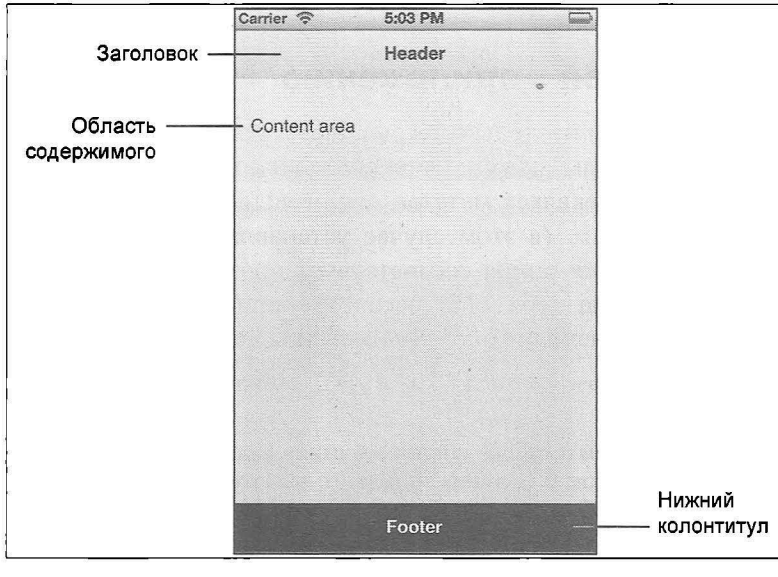


Рис. 8.8. Одно из наших приложений jQuery Mobile, открытое в полноэкранном режиме на iPhone или iPad

Распознавание полноэкранного режима

На платформе iOS мы можем принудительно устанавливать полноэкранный режим приложения с помощью свойства `navigator.standalone`. Когда это свойство доступно,

оно принимает значение `false` (пользователь открыл Safari) или `true` (приложение было открыто прикосновением к пиктограмме на главном экране).

Вооружившись этой идеей, мы можем форсировать установку и работу в полноэкранном режиме. После установки пользователь не видит разницы между низкоуровневым приложением, загруженным с сайта AppStore, и нашим веб-приложением без пользовательской оболочки.

Например, мы можем создать специальную страницу jQuery Mobile, предлагающую пользователю установить наше приложение или запускать его из меню главного экрана:

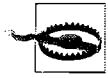
```
if (navigator.standalone!=undefined) {
  // Это iOS
  if (!navigator.standalone) {
    // Это Safari
    $.mobile.changePage($("#install"), {transition: "none"});
  }
}
```

Если этот фрагмент кода поместить в обработчик события `mobileinit`, пользователь увидит страницу установки, когда откроет приложение в браузере Safari, и не сможет ничего сделать на странице. Если же он откроет приложение из меню главного экрана, мы предоставим ему первую страницу jQuery Mobile.

Применение стилей к приложению

В компании Apple созданы несколько тегов `<meta>`, позволяющих менять стили веб-приложения. Во-первых, мы можем изменить цвет строки состояния (у верхнего края экрана), которым управляет метатег `apple-mobile-webapp-status-bar-style`, принимающий значения `default` (в этом случае устанавливается серый цвет), `black` и `black-translucent`. Последняя опция соответствует прозрачной черной области, которая частично примет на себя цвет заголовка приложения jQuery Mobile. На рис. 8.9 показано, что происходит после применения каждого из этих значений:

```
<meta name="apple-mobile-web-app-status-bar-style" content="black">
```



Если мы применяем к строке состояния стиль `black-translucent`, в нашем распоряжении окажется окно браузера в полную высоту. Строка состояния будет "плавать" над нашей веб-страницей. Если страница не имеет фиксированных панелей, и мы не оставим свободными 20 пикселов сверху, то качество дизайна может пострадать. Еще одно изменение, которое мы можем внести в наше полноэкранное приложение, заключается в создании изображения запуска. Это изображение, которое выводится операционной системой, пока приложение открывается (рис. 8.10).

Изображение запуска может быть определено элементом `<link>` с атрибутом `rel="apple-touchstartup-image"`, а его размер зависит от платформы, на которой работает приложение.

Допустимы следующие размеры:

- все iPhone/iPod: 320×460;
- все iPad с книжной ориентацией: 748×1004;
- все iPad с альбомной ориентацией: 748×1024 (повернуто на 90°).

К сожалению, применение медийных запросов с целью добавления изображений с высоким разрешением в iPhone 4, 4S и iPod Touch четвертого поколения невозможно. Единственный обходной путь состоит в написании кода на языке JavaScript для динамического создания элемента `<link>`. В таком случае изображение должно иметь размер 640×920.

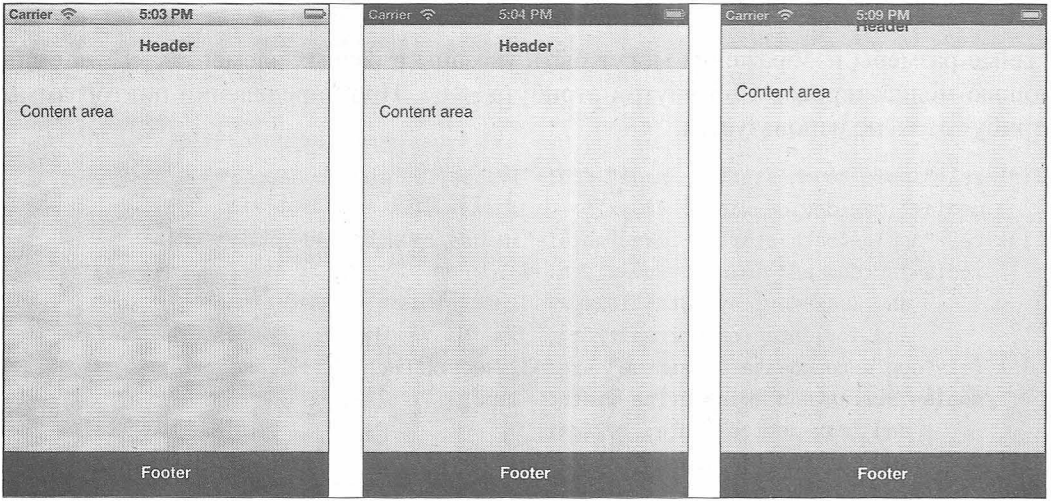


Рис. 8.9. Различные стили строки состояния, примененные на одной странице

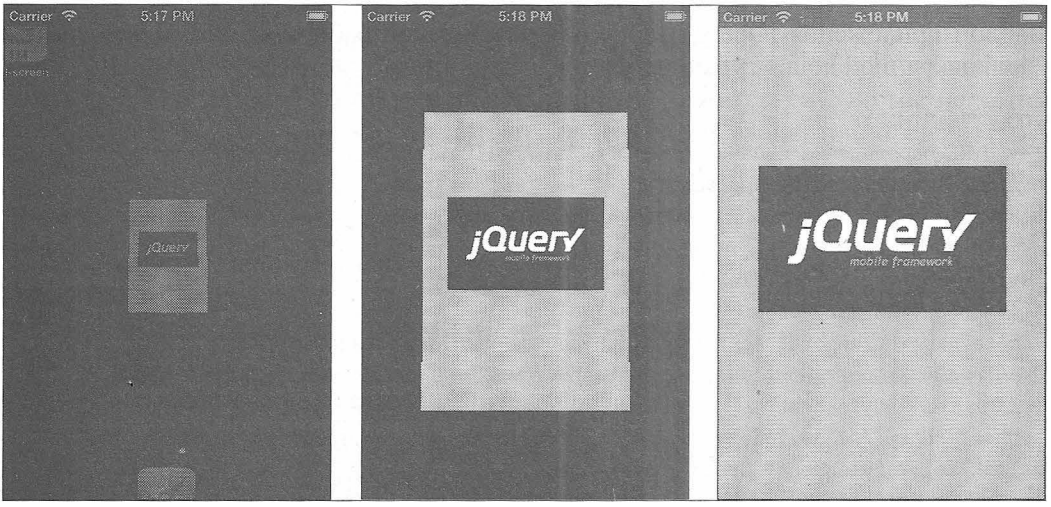


Рис. 8.10. Изображение, выводимое операционной системой в процессе открытия приложения

Если мы определяем только одно изображение, его размер должен составлять 320×460 (20 пикселов используются строкой состояния в верхней части экрана). Такое изображение подойдет для всех устройств iPhone и iPod Touch (в том числе и имеющих экраны с высоким разрешением). На устройствах iPad такой подход не сработает, и будет использовано обычное изображение запуска (снимок экрана, сделанный при последнем сеансе работы с веб-приложением):

```
<link rel="apple-touch-startup-image" href="images/launch.png">
```

Если у вас есть устройство, работающее под управлением iOS, вы можете сделать снимок экрана, одновременно нажав на кнопки **Главный** и **Питание**. Полученный снимок можно использовать в качестве изображения запуска, совпадающего с тем, что было при первой загрузке веб-приложения.

Разные размеры изображения могут быть указаны в разных элементах `<link>` с помощью медиазапросов CSS внутри атрибута `media`. При определении пиктограммы атрибут `sizes` не используется:

```
<link rel="apple-touch-startup-image" href="images/launch-iphone.png"
      media="(max-device-width: 480px)">
<link rel="apple-touch-startup-image" href="images/launch-iPad-p.png"
      media="screen and (min-device-width: 481px)
            and (max-device-width: 1024px)
            and (orientation:portrait)">
<link rel="apple-touch-startup-image" href="images/launch-iPad-l.png"
      media="screen and (min-device-width: 481px)
            and (max-device-width: 1024px)
            and (orientation:landscape)">
```

Подведем итоги

Шаблон приложения jQuery Mobile для автономной работы со всеми значками и в полноэкранной конфигурации будет выглядеть так:

```
<!DOCTYPE HTML>
<!-- HTML-определение с манифестом Offline Application Cache -->
<html manifest="offline.appcache">
<head>
  <meta charset="UTF-8">
  <title>short title</title>
  <!-- короткий заголовок -->
  <meta name="viewport" content="width=device-width,user-scalable=no">
  <!-- Файлы jQuery Mobile со стандартной темой -->
  <link rel="stylesheet"
    href="http://code.jquery.com/mobile/1.0/jquery.mobile.structure-1.0.min.css" />
  <link rel="stylesheet" href="custom_theme.css">
  <script src="http://code.jquery.com/jquery-1.6.4.min.js"></script>
  <script src="http://code.jquery.com/jquery-1.6.4.min.js"></script>
```

```
  <script src="http://code.jquery.com/mobile/1.0/jquery.mobile-1.0.min.js">
  </script>
  <!-- Значки -->
  <link rel="icon" href="icons/icon32.png">
  <link rel="shortcut icon" href="icons/icon32.png">
  <link rel="apple-touch-icon" href="icons/icon57.png" sizes="57x57">
  <link rel="apple-touch-icon" href="icons/icon114.png" sizes="114x114">
  <link rel="apple-touch-icon" href="icons/icon72.png" sizes="72x72">
  <link rel="apple-touch-icon" sizes="80x80" href="icons/icon80.png">
  <link rel="apple-touch-icon-precomposed" sizes="android-only"
        href="icons/icon57.png">
  <!-- Если мы создаем приложение без пользовательской оболочки -->
  <meta name="apple-mobile-web-app-capable" content="yes">
  <meta name="apple-mobile-web-app-status-bar-style" content="black">
  <link rel="apple-touch-startup-image" href="images/launch-iphone.png"
        media="(max-device-width: 480px)">
  <link rel="apple-touch-startup-image" href="images/launch-iPad-p.png"
        media="screen and (min-device-width: 481px)
               and (max-device-width: 1024px)
               and (orientation:portrait)">
  <link rel="apple-touch-startup-image" href="images/launch-iPad-l.png"
        media="screen and (min-device-width: 481px)
               and (max-device-width: 1024px)
               and (orientation:landscape)">
</head>
<body>
  <!-- Страницы jQuery Mobile -->
</body>
```

Хранение данных в автономном режиме

Чтобы хранить информацию локально, когда приложение работает в автономном режиме, — или даже когда оно имеет подключение к Интернету, но мы не хотим использовать AJAX, — мы можем выбрать один из трех вариантов, предлагаемых нам технологией HTML5:

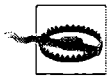
- Web Storage API;
- Web SQL Database API;
- IndexedDB API.

Совместимость устройства с каждым API-интерфейсом, предлагаемым в HTML5, можно выяснить на сайте **http://mobilehtml5.org**.

В этой книге описывается работа с API-интерфейсом Web Storage, поскольку он самый простой и имеет максимальную совместимость с браузером jQuery Mobile

категории A. Простота этого API-интерфейса заключается в том, что он поддерживает две коллекции пар "ключ/значение": `localStorage` и `sessionStorage`.

Первая, `localStorage`, представляет собой коллекцию строк, постоянно хранящихся в памяти устройства. Коллекция `sessionStorage` полностью аналогична, но удаляется из системы после закрытия браузера.



Обычно мы можем без проблем хранить в локальной памяти одного устройства до 5 Мбайт данных. Однако большинство браузеров хранит строки в кодировке Unicode, в которой один символ занимает 2 байта. Поэтому безопаснее считать, что мы можем хранить 2,5 Мбайт текста.

Этот API-интерфейс доступен в виде глобального атрибута `window` по имени `localStorage` и предлагает нам несколько методов, в том числе `getItem` и `setItem` для загрузки и сохранения данных коллекции.

Стандартный API-интерфейс позволяет сохранять только строки, но отсюда следует, что мы можем хранить:

- массивы и объекты, преобразованные в строки JSON;
- простые значения;
- значения, разделенные запятыми;
- код JavaScript (который можно выполнить впоследствии);
- таблицы стилей CSS (которые можно вставить в код впоследствии);
- код HTML;
- изображения в кодировке base64 (данные URI).

Современные мобильные браузеры поддерживают API-интерфейс к JSON, позволяющий преобразовать объекты в строки JSON с помощью метода `JSON.stringify(object)`, а также преобразовывать строки JSON в объекты с помощью метода `JSON.parse(string)`. Если API-интерфейс недоступен, вы можете воспользоваться бесплатной библиотекой JSON2, разработанной Дугласом Крокфордом (Douglas Crockford) и доступной по адресу **http://github.com/douglascrockford/JSON-js**.

Чтобы сохранить значение, мы должны написать:

```
localStorage.setItem("name", "value");
```

Чтобы получить значение, пишем:

```
var value = localStorage.getItem("name");
```

ГЛАВА 9

Законченное веб-приложение

В этой главе мы создадим полноценное приложение, опираясь на знания, полученные из предыдущих глав. Это будет мобильное приложение для конференции. Перечислю основные цели приложения:

- оно будет официальным источником информации для участников конференции;
- оно будет предоставлять обновляемый список заседаний по секциям;
- оно будет предоставлять общую информацию о конференции.

Возможно, что вы, подобно мне, участвовали в большом количестве технических конференций. Кроме того, вам наверняка известно, что сетевые соединения, как правило, работают неудовлетворительно. Поэтому наше приложение будет работать в автономном режиме (на устройствах, которые это позволяют). В последующих главах мы научимся упаковывать его с целью распространения.

Структура приложения

Для обеспечения задуманной функциональности структура приложения будет достаточно простой. Там должны быть следующие разделы:

- главная страница;
- заседания;
- место проведения;
- обратная связь;
- общая информация о конференции.

Раздел "Заседания" будет содержать список всех секций и всех заседаний по секциям и аудиториям. Список заседаний будет присылаться с сервера использованием JSON-подобного объекта, создаваемого на "серверном" языке, таком как PHP или Java. Здесь мы не будем обсуждать ни генерирование этого объекта на стороне сервера, ни генерирование базы данных с помощью системы CMS (Content Management System, система управления содержимым).

Для устройств iPhone, iPod и iPad мы создадим приложение без пользовательской оболочки, поэтому мы постараемся уговорить пользователя установить наше приложение в его системе. Если он не захочет устанавливать веб-приложение, мы предоставим ему версию, работающую при подключении к Интернету.

Первый список файлов будет включать в себя следующие HTML-документы:

- index.html (с наиболее часто используемыми страницами);
- feedback.html;
- feedback.php.

В папке images мы будем хранить изображения, в том числе:

- logo.png;
- sponsors.png;
- background.png;
- launch-ios.png (значок запуска приложения в iOS);
- icon114.png (значок для iPhone с высоким разрешением экрана);
- icon72.png (значок для iPad);
- icon57.png (значок для Android и iPhone с низким разрешением экрана).

И, конечно, нам понадобятся все файлы платформы jQuery Mobile, включая файлы с кодом JavaScript, таблицами CSS и изображениями.

Единственным постоянно обновляемым ресурсом будет файл sessions.json, содержащий JSON-подобный объект и поставляемый с сервера. Для повышения производительности и обеспечения автономной работы мы будем кэшировать этот JSON-файл с применением технологии Local Storage API стандарта HTML5.

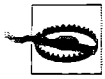
Манифест автономной работы

Все приложение будет содержаться в манифесте кэша приложения для совместимых устройств. Поэтому сначала мы создадим файл с манифестом кэша на языке HTML5.

Наш файл с манифестом выглядит следующим образом:

```
CACHE MANIFEST
CACHE:
# файлы jQuery Mobile
jquery.js
jquery.mobile.js
jquery.mobile.css
images/ajax-loader.png
images/icons-18-black.png
images/icons-18-white.png
images/icons-36-black.png
images/icons-36-white.png
# HTML-документы
# index.html всегда кэшируется неявно
feedback.html
# Наши сценарии CSS-таблицы
index.js
index.css
```

```
# Наши изображения
logo.png
sponsors.png
background.png
# Нам не нужно кэшировать пиктограммы iOS и значок запуска,
# т. к. платформа кэширует их сама
NETWORK:
# Мы будем выходить в сеть только для получения JSON-файла и
# результатов сценария, обрабатывающего форму обратной связи.
# Замените mobilexweb.com на свой домен
http://mobilexweb.com/jqmbook/sessions.json
http://mobilexweb.com/jqmbook/feedback.php
```



После установки нашего приложения ему не будет доступен весь Интернет, потому что мы не указали конструкцию * в разделе NETWORK: манифеста автономной работы.

Страницы

Начнем с главного файла index.html. Это будет типичный документ jQuery Mobile со всеми элементами, о которых мы говорили в этой книге (включая метатеги, значки и метатег, определяющий окно просмотра). Он будет содержать все страницы, кроме формы, поскольку велика вероятность, что пользователь будет обращаться к ним постоянно. Впоследствии мы сможем перенести внутренние страницы во внешние HTML-файлы, если понадобится. Страница #sessions будет содержать главный список. Кроме того, у нас будет страница #details, служащая шаблоном для подробной информации о заседаниях.

На странице #main не будет заголовка, вместо которого мы будем выводить логотип конференции. Все остальные страницы будут с заголовками.

В HTML-документе нет никакой информации о заседаниях, потому что она будет извлекаться кодом JavaScript из JSON-подобного объекта, получаемого с сервера с помощью технологии AJAX. По этой причине страницы #sessions и #details оставлены пустыми.

Файл index.html имеет следующий код разметки:

```
<!DOCTYPE HTML>
<html manifest="manifest.appcache">
<head>
  <meta charset="UTF-8">
  <title>jQM Conference</title>
  <!-- Конференция по jQM -->
  <meta name="viewport" content="width=device-width,user-scalable=no">
  <link rel="stylesheet" href="jquery.mobile-1.0.css" />
  <link rel="stylesheet" href="index.css" />
```

```
  <script src="jquery.js"></script>
  <script src="index.js"></script>
  <script src="jquery.mobile-1.0.js"></script>
  <meta name="apple-mobile-web-app-capable" content="yes">
  <meta name="apple-mobile-web-app-status-bar-style" content="black">
  <link rel="apple-touch-icon" sizes="80x80" href="icons/icon80.png">
  <link rel="apple-touch-icon" href="images/icon57.png" sizes="57x57">
  <link rel="apple-touch-icon" href="images/icon114.png" sizes="114x114">
  <link rel="apple-touch-icon" href="images/icon72.png" sizes="72x72">
  <link rel="apple-touch-icon-precomposed" sizes="android-only"
        href="images/icon57.png">
</head>
<body>
  <!-- **** ГЛАВНАЯ СТРАНИЦА **** -->
  <div data-role="page" id="home">
    <div data-role="content">
      <p><img src="images/logo.png" alt="jQM Conference"
              width="300" height="62" align="middle">
      </p>
      <h3>November 5th</h3>
      <!-- 5 ноября -->
      <div class="ui-grid-a">
        <div class="ui-block-a">
          <a href="http://www.twitter.com/fakeaccount"
             data-role="button" data-theme="b"
             target="_blank">Twitter</a>
        </div>
        <div class="ui-block-b">
          <a href="http://www.facebook.com/fakeaccount"
             data-role="button" data-theme="b"
             target="_blank">Facebook</a>
        </div>
      </div>
    </div>
    <div data-role="footer" data-position="fixed" data-id="toolbar">
      <div data-role="navbar">
        <ul>
          <li><a class="ui-btn-active" href="#home" data-icon="home"
          data-transition="fade">Home</a></li>
                          <!-- Главная -->
          <li><a href="#sessions" data-icon="grid"
                 data-transition="fade">Sessions</a>
                                  <!-- Заседания -->
          </li>
          <li><a href="#where" data-icon="info"
                 data-transition="fade">Where</a></li>
                                  <!-- Где -->
```

```
        <li><a href="#about" data-icon="star"
               data-transition="fade">About</a></li>
                                <!-- О конференции -->
        <li><a href="feedback.html" data-icon="plus"
               data-rel="dialog">Feedback</a>
                                <!-- Обратная связь -->
        </li>
      </ul>
    </div>
  </div>
</div>
<!-- **** СТРАНИЦА "ЗАСЕДАНИЯ" **** -->
<div data-role="page" id="sessions">
  <div data-role="header">
    <h1>Sessions</h1>
    <!-- Заседания -->
    <a href='javascript:refresh();' data-icon="refresh" id="refresh"
       data-theme="c" class="ui-btn-left" data-iconpos="notext"></a>
  </div>
  <div data-role="content">
    <p id="console"></p>
    <ul data-role="list-view" data-inset="true" id="slots">
    </ul>
  </div>
  <div data-role="footer" data-position="fixed" data-id="toolbar">
    <div data-role="navbar">
      <ul>
        <li><a href="#home" data-icon="home"
               data-transition="fade">Home</a></li>
                                <!-- Главная -->
        <li><a class="ui-btn-active" href="#sessions" data-icon="grid"
               data-transition="fade">Sessions</a></li>
                                <!-- Заседания -->
        <li><a href="#where" data-icon="info"
               data-transition="fade">Where</a></li>
                                <!-- Где -->
        <li><a href="#about" data-icon="star"
               data-transition="fade">About</a></li>
                                <!-- О конференции -->
        <li><a href="feedback.html" data-icon="plus"
               data-rel="dialog">Feedback</a>
                                <!-- Обратная связь -->
        </li>
      </ul>
    </div>
  </div>
</div>
```

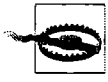
```
<!-- **** СТРАНИЦА С ПОДРОБНОСТЯМИ О ЗАСЕДАНИЯХ **** -->
<div data-role="page" id="details" data-add-back-btn="true">
  <div data-role="header">
    <h1>Session detail</h1>
    <!-- Подробнее -->
  </div>
  <div data-role="content">
    <div id="sessionInfo"> </div>
  </div>
  <div data-role="footer" data-position="fixed" data-id="toolbar">
    <div data-role="navbar">
      <ul>
        <li><a href="#home" data-icon="home"
               data-transition="fade">Home</a></li>
                                <!-- Главная -->
        <li><a href="#sessions" data-icon="grid"
               data-transition="fade">Sessions</a></li>
                                <!-- Заседания -->
        <li><a href="#where" data-icon="info"
               data-transition="fade">Where</a></li>
                                <!-- Где -->
        <li><a href="#about" data-icon="star"
               data-transition="fade">About</a></li>
                                <!-- О конференции -->
        <li><a href="feedback.html" data-icon="plus"
               data-rel="dialog">Feedback</a></li>
                                <!-- Обратная связь -->
      </ul>
    </div>
  </div>
</div>
<!-- **** СТРАНИЦА "ГДЕ" **** -->
<div data-role="page" id="where">
  <div data-role="header">
    <h1>Where</h1>
    <!-- Место проведения -->
  </div>
  <div data-role="content">Hasta la vista 1234, ACME City </div>
  <!-- (вымышленный адрес) -->
  <div data-role="footer" data-position="fixed" data-id="toolbar">
    <div data-role="navbar">
      <ul>
        <li><a href="#home" data-icon="home"
               data-transition="fade">Home</a></li>
                                <!-- Главная -->
        <li><a href="#sessions" data-icon="grid"
               data-transition="fade">Sessions</a>
                                <!-- Заседания -->
        </li>
```

```
        <li><a class="ui-btn-active" href="#where" data-icon="info"
        data-transition="fade">Where</a></li>
                        <!-- Где -->
        <li><a href="#about" data-icon="star"
               data-transition="fade">About</a></li>
                               <!-- О конференции -->
        <li><a href="feedback.html" data-icon="plus"
               data-rel="dialog">Feedback</a>
                               <!-- Обратная связь -->
        </li>
      </ul>
    </div>
  </div>
</div>
<!-- **** СТРАНИЦА "О КОНФЕРЕНЦИИ" **** -->
<div data-role="page" id="about">
  <div data-role="header">
    <h1>About</h1>
    <!-- О конференции -->
  </div>
  <div data-role="content">
    <div data-role="collapsible">
      <h3>Organization</h3>
      <!-- Организатор -->
      <p>This congress is organized by ACME</p>
      <!-- Этот конгресс организован компанией ACME -->
    </div>
    <div data-role="collapsible">
      <h3>Dates</h3>
      <!-- Время проведения -->
      <p>November 5th, 2015 9am to 6pm</p>
      <!-- 5 ноября 2015 г. с 9:00 до 18:00 -->
    </div>
    <div data-role="collapsible">
      <h3>History</h3>
      <!-- История -->
      <p>First edition of this congress was.....</p>
      <!-- Этот конгресс впервые был проведен ... -->
    </div>
    <div data-role="collapsible">
      <h3>Sponsors</h3>
      <!-- Спонсоры -->
      <img src="images/sponsors.png" width="300" height="400">
    </div>
  </div>
  <div data-role="footer" data-position="fixed" data-id="toolbar">
    <div data-role="navbar">
```

```
        <ul>
          <li><a href="#home" data-icon="home"
                 data-transition="fade">Home</a></li>
                                 <!-- Главная -->
          <li><a href="#sessions" data-icon="grid"
                 data-transition="fade">Sessions</a>
                                 <!-- Заседания -->
          </li>
          <li><a href="#where" data-icon="info"
                 data-transition="fade">Where</a></li>
                                 <!-- Где -->
          <li><a class="ui-btn-active" href="#about" data-icon="star"
                 data-transition="fade">About</a></li>
                                 <!-- О конференции -->
          <li><a href="feedback.html" data-icon="help"
                 data-transition="fade">Feedback
                                 <!-- Обратная связь -->
          </a></li>
        </ul>
      </div>
    </div>
  </div>
  <!-- **** ДИАЛОГОВАЯ СТРАНИЦА ДЛЯ УСТАНОВКИ В iOS **** -->
  <div data-role="page" id="ios">
    <div data-role="header">
      <h1>installation</h1>
      <!-- Установка -->
    </div>
    <div data-role="content">
      <p id="consoleInstall">Complete the installation</p>
      <!-- Закончить установку -->
      <div id="install">
        <p>To finish the installation, you must add this webapp
           to your Home Screen. To do that, touch the arrow
           and select "Add to Home Screen"</p>
        <!-- Чтобы завершить установку, вы должны добавить
             это приложение на главный экран. Для этого прикоснитесь
             к стрелке и выберите "Добавить на главный экран" -->
      </div>
      <a href="javascript:openWithoutInstallation()"
         class="openWithoutInstall">Open without installation</a>
         <!-- Открыть без установки -->
    </div>
  </div>
</body>
</html>
```

Приложение имеет фиксированный нижний колонтитул, так что панель навигации не анимируется при переходе с одной страницы на другую.

На странице `sessions` присутствуют пустой элемент `<p>` с идентификатором `console` и компонент `listview` с идентификатором `slots`. Мы заполним эти элементы из кода JavaScript после того, как страница будет выведена на экран.

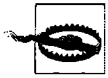


Помните, что если мы используем элемент `navbar`, то его следует вставлять на каждую страницу (внутреннюю или внешнюю), в заголовок или нижний колонтитул. Если мы хотим избежать дублирования кода, можно реализовать вставку элемента на языке JavaScript. Единственное различие между панелями навигации на разных страницах может заключаться в выделении разных элементов.

Форма будет представлять собой диалоговую страницу во внешнем файле feedback.html. Вот код этого файла:

```
<!DOCTYPE HTML>
<html>
<head>
  <meta charset="UTF-8">
  <title>jQM Conference</title>
  <!-- Конференция по jQM -->
  <meta name="viewport" content="width=device-width,user-scalable=no">
  <link rel="stylesheet" href="jquery.mobile-1.0.css" />
  <script src="jquery.js"></script>
  <script src="script.js"></script>
  <script src="jquery.mobile-1.0.js"></script>
</head>
<body>
  <div data-role="dialog">
    <div data-role="header">
      <h1>Feedback</h1>
      <!-- Обратная связь -->
    </div>
    <div data-role="content">
      <form action="feedback.php" method="post" data-transition="none">
        <div data-role="fieldcontain">
          <label for="name">Name:</label>
          <!-- Имя -->
          <input type="text" name="name" id="name" value="" required />
        </div>
        <div data-role="fieldcontain">
          <label for="email">Email:</label>
          <!-- Адрес электронной почты -->
          <input type="email" name="email" id="email"
                 value="" required />
        </div>
```

```
        <div data-role="fieldcontain">
          <label for="comments">Comments:</label>
          <!-- Комментарий -->
          <textarea cols="40" rows="8" name="comments"
                    id="comments"></textarea>
        </div>
        <div data-role="fieldcontain">
          <label for="contacted">Can we contact you?</label>
          <!-- Можно ли связаться с Вами? -->
          <select name="contacted" id="contacted" data-role="slider">
            <option value="no">No</option>
            <!-- Нет -->
            <option value="yes">Yes</option>
            <!-- Да -->
          </select>
        </div>
        <input type="submit" value="Send" data-theme="a">
      </form>
    </div>
  </div>
</body>
</html>
```



Здесь мы не обсуждаем вопросы проверки допустимости данных на стороне клиента и сервера. Не забывайте проверять корректность данных формы перед их обработкой.

Файл feedback.php должен производить проверку допустимости информации, сохранять ее в базе данных или отправлять по электронной почте, а также выводить в качестве результата страницу jQuery Mobile:

```
<!DOCTYPE HTML>
<html>
<head>
  <meta charset="UTF-8">
  <title>jQM Conference</title>
  <!-- Конференция по jQM -->
  <meta name="viewport" content="width=device-width,user-scalable=no">
  <link rel="stylesheet" href="jquery.mobile-1.0.css" />
  <script src="jquery.js"></script>
  <script src="script.js"></script>
  <script src="jquery.mobile-1.0.js"></script>
</head>
<body>
  <div data-role="dialog">
    <div data-role="header">
      <h1>Feedback</h1>
```

```
      <!-- Обратная связь -->
    </div>
    <div data-role="content">
      <?php
      // Здесь происходит проверка допустимости и обработки данных
      ?>
      Thanks for your feedback.
      <!-- Спасибо за комментарий -->
      <a data-role="button" data-inverse="true"
         href="index.html">Close</a>
      <!-- Закрыть -->
    </div>
  </div>
</body>
</html>
```

Таблица стилей

В файле index.css определены нестандартные стили:

```
/* Стили для заголовка */
.ui-header {
  background-color: #69C;
  background-image: url('images/background.png');
  background-position: top right;
  background-repeat: no-repeat
}
/* Стили для процедуры установки автономного приложения */
#console, #consoleInstall {
  background-color: #FF6;
  border-radius: 10px;
  -webkit-border-radius: 10px;
  padding: 5px;
  margin: 0;
  text-align: center;
  border: 1px solid #CCC;
  font-size: small;
}
/* Область содержимого главной страницы */
#home [data-role=content] {
  background-color: white;
  text-align: center;
}
/* Кнопки */
.openWithoutInstall {
  margin-top: 50px;
  display: block;
}
```

```
/* Текст на панели навигации */
[data-role=navbar] .ui-btn-text {
  font-size: smaller;
}
```

Данные

Файл JSON с информацией о заседаниях будет иметь примерно следующую структуру:

```
{
  "slots":
  [
    // Время отдельных событий, таких как открытие, перерывы и т. д.
    { "id": 1, "time": "09:00", "message": "Opening" },
    // Время заседаний
    { "id": 2, "time": "09:20"}
  ],
  "sessions":
  [
    { "id": 1, "title": "A great session", "timeId": 2,
      "description": "...", "speaker": "...", "room": "..." }
  ]
}
```

Сценарий

Часть функциональности программы может быть реализована только в коде на языке JavaScript:

- автономное распознавание iOS;
- загрузка сведений о заседаниях;
- динамическое генерирование списка и подробностей о заседаниях.

Файл index.js содержит следующий код:

```
var data;
$(document).bind("mobileinit", function() {
  if (navigator.platform=="iPhone" || navigator.platform=="iPad" ||
    navigator.platform=="iPod" || navigator.platform=="iPad" ||
    navigator.platform=="iPhone Simulator" ) {
    // Устройство работает под управлением iOS, и мы проверим,
    // находится ли приложение в автономном режиме
    if (!navigator.standalone) {
      showIOSInvitation();
    }
  }
```

```
  /* Мы перехватываем загрузку страницы sessions и
     проверяем динамические данные о заседании */
  $("#sessions").live("pageshow", function() {
    if (window.localStorage!=undefined) {
      // Интерфейс HTML5 Local Storage доступен
      if (window.localStorage.getItem("data")!=undefined &&
      window.localStorage.getItem("data")!=null) {
        // Вначале загружаем информацию о заседаниях, хранящуюся
        // локально, и одновременно проверяем наличие обновлений
        showSessions(window.localStorage.getItem("data"));
        $("#console").html("Checking data update");
        // Проверка наличия обновлений
      } else {
        // There is no local storage cache
        // Кэш локального хранения отсутствует
        $("#console").html("Downloading session data...");
        // Загрузка информации о заседании...
      }
    } else {
      // Интерфейс HTML5 Local Storage не поддерживается,
      // каждый раз загружаем JSON-файл
      $("#console").html("Downloading session data...");
      // Загрузка информации о заседании...
    }
    loadSessionsAjax();
  });
});
function showIOSInvitation() {
  setTimeout(function() {
    // Скрываем информацию о сохранении,
    // пока не загрузится все приложение
    $("#install").hide();
    // Открываем инструкцию по добавлению приложения
    // на главный экран в iOS
    $.mobile.changePage($("#ios"), {transition: "slideup",
      changeHash: false});
  }, 100);
  // Перехватываем события HTML5 Application Cache
  // для выдачи нужной информации
  if (window.applicationCache!=undefined) {
    window.applicationCache.addEventListener('checking', function() {
      $("#consoleInstall").html("Checking version");
      // Проверка версии
  });
  window.applicationCache.addEventListener('downloading', function() {
    $("#consoleInstall").html("Downloading application.
      Please wait...");
```

```
      // Приложение загружается. Пожалуйста, подождите...
  });
  window.applicationCache.addEventListener('cached', function() {
    $("#consoleInstall").html("Application downloaded");
    // Приложение загружено
    $("#install").show();
  });
  window.applicationCache.addEventListener('updateready', function() {
    $("#consoleInstall").html("Application downloaded");
    // Приложение загружено
    $("#install").show();
  });
  window.applicationCache.addEventListener('noupdate', function() {
    $("#consoleInstall").html("Application downloaded");
    // Приложение загружено
    $("#install").show();
  });
  window.applicationCache.addEventListener('error', function(e) {
    $("#consoleInstall").html("There was an error downloading this
    app");
    // Во время загрузки этого приложения произошла ошибка
  });
  window.applicationCache.addEventListener('obsolete', function(e) {
    $("#consoleInstall").html("There was an error downloading this
    app");
    // Во время загрузки этого приложения произошла ошибка
  });
  }
}
function loadSessionsAjax() {
  // Получаем JSON-объект в текстовом виде для упрощения его хранения
  // в локальной памяти
  $.ajax(/*"http://www.mobilexweb.com/congress/*/"sessions.json", {
  complete: function(xhr) {
    if (window.localStorage!=undefined) {
      if (window.localStorage.getItem("data")!=undefined &&
        window.localStorage.getItem("data")!=null) {
          if (xhr.responseText==window.localStorage.getItem("data")) {
            // Загруженная информация о заседании совпадает
            // с выведенной на экран
            $("#console").html("Schedule is updated");
            // Расписание обновлено
          } else {
            // Имеется новая информация о заседаниях
            if (confirm("There is an update in the schedule
              available, do you want to load it now?")) {
```

```
            // Имеются изменения в расписании; загрузить сейчас?
              $("#console").html("Schedule is updated");
              // Расписание обновлено
              showSessions(xhr.responseText);
            } else {
              $("#console").html("Schedule will be updated later");
              // Расписание будет обновлено позже
            }
          }
        } else {
          // Локальная память доступна, но кэш не обнаружен
          $("#console").html("Schedule is updated");
          // Расписание обновлено
          showSessions(xhr.responseText);
          }
        } else {
          // Локальная память отсутствует;
          // показываем информацию без сохранения
          $("#console").html("Schedule is updated");
          // Расписание обновлено
          showSessions(xhr.responseText);
        }
      },
      dataType: 'text',
      error: function() {
        $("#console").html("Schedule update could not be downloaded");
        // Невозможно загрузить новое расписание
      }
  });
}
var isFirstLoad = true;
function showSessions(string) {
  if (window.JSON!=undefined) {
    data = JSON.parse(string);
  } else {
    data = eval("(" + string + ")");
  }
  if (window.localStorage!=undefined) {
    // Сохранить новые данные в виде строки
    window.localStorage.setItem("data", string);
  }
  $("#slots").html("");
  var html = "";
  for (var i=0; i<data.slots.length; i++) {
    if (data.slots[i].message!=null) {
      // Это специальная информация, и мы создаем разделитель
      html += "<li data-role='list-divider' data-groupingtheme='e'>" +
        data.slots[i].time + ": " + data.slots[i].message + "</li>";
```

```
    } else {
      // Это обычная информация о заседаниях
      html += "<li><a href='javascript:showDetails(" + i +
              ")'>Sessions of " + data.slots[i].time + "</a></li>";
    }
  }
  $("#slots").html(html);
  if (isFirstLoad) {
    $("#slots").listview();
    isFirstLoad = false;
  } else {
    $("#slots").listview('refresh');
  }
}
function showDetails(index) {
  $("#details h1").html("Sessions of " + data.slots[index].time);
  var html = "";
  for (var i=0; i<data.sessions.length; i++) {
    if (data.sessions[i].timeId==data.slots[index].id) {
      // Создаем по одному сворачиваемому компоненту на заседание
      html += "<div data-role='collapsible'>";
      html += "<h3>" + data.sessions[i].title + "</h3>";
      html +=" <h3>" + data.sessions[i].room + "</h3>";
      html += "<h4>Speaker/s: " + data.sessions[i].speaker;
      html += "</h4>";
      html += "<p>" + data.sessions[i].description + "</p>";
      html += "</div>";
    }
  }
  // Информация для страницы с подробностями
  $("#sessionInfo").html(html);
  $("#sessionInfo div").collapsible();
  // Переходим на страницу с подробностями
  $.mobile.changePage($("#details"));
}
function refresh() {
  $("#console").html("Verifying...");
  // Выполняется проверка...
    loadSessionsAjax();
}
function openWithoutInstallation() {
  // Удаляем псевдодиалоговую страницу по установке iOS
  $.mobile.changePage($("#home"), {transition:"slideup", reverse:true});
}
```

Для платформы Android и других платформ мы можем создать гибридное приложение или виджет и предоставить пользователю аналогичную страницу установки, ссылающуюся на пакет с низкоуровневым приложением на нашем сервере или в хранилище.

Это приложение можно совершенствовать и далее. Например, можно создать URL-ссылку для каждого заседания (скажем, `#details!22`), чтобы можно было отслеживать URL-адрес и обеспечивать динамическую загрузку подробностей путем глубокого связывания. На рис. 9.1 изображены страницы нашего веб-приложения, работающего и как браузерное, и как автономное приложение на устройствах с операционной системой iOS.

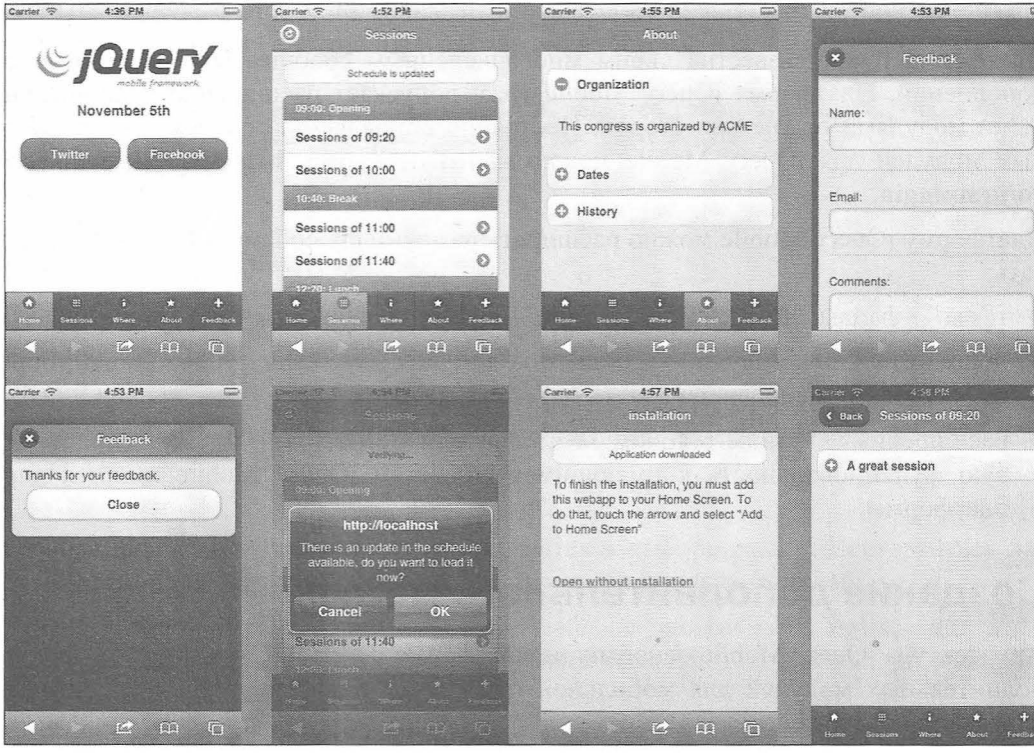


Рис. 9.1. Наше веб-приложение

ГЛАВА 10

Расширение возможностей платформы

Платформа jQuery известна своим многочисленным сообществом разработчиков дополнений. Платформа jQuery Mobile тоже допускает расширение дополнительными модулями от третьих сторон. Постоянно обновляемый список дополнительных модулей для jQuery Mobile можно найти по адресу **http://mobilexweb.com/go/jqmplugin**.

Платформу jQuery Mobile можно расширять разными способами. Мы можем создавать:

- темы — файлы с CSS-таблицей и/или несколько изображений;
- дополнительные модули — файлы с кодом JavaScript или CSS-таблицей, определяющие новые виджеты (`data-role`) платформы;
- расширения — файлы с кодом JavaScript или CSS-таблицей, добавляющие новую функциональность к имеющимся виджетам jQuery Mobile и/или к ядру платформы.

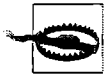
Создание дополнительного модуля

Архитектура jQuery Mobile основана на платформе jQuery UI, так что создание дополнительных модулей для мобильной платформы аналогично их созданию для платформы интерфейса.

Вначале мы должны усвоить некоторые правила, связанные с созданием совместимых дополнительных модулей.

- Необходимо учитывать вопросы совместимости jQuery Mobile с другими системами. Дополнительный модуль, совместимый только с одной-двумя платформами, не заинтересует сообщество.
- Установка модуля должна сводиться для пользователя к добавлению файла со сценарием и файла с CSS-таблицей.
- Мы должны максимально использовать атрибуты `data-*`, чтобы естественно расширять архитектуру платформы.
- Дополнительный модуль должен корректно работать с нестандартным пространством имен и набором тем.
- Дополнительный модуль также должен выполнять автоинициализацию виджетов, как делается в главной платформе.

- Для определения содержимого модуля необходимо использовать семантический код HTML5, где только возможно.
- Если возможно, обеспечивайте доступность пользователям с ограниченными возможностями (например, применяйте технологию ARIA).
- Чтобы избежать несовместимости модуля с будущими версиями платформы, не используйте имена общего характера в качестве собственных нестандартных значений атрибута `data-role`. Например, имена `tableview` или `datepicker`[1] могут быть впоследствии использованы разработчиками платформы при реализации соответствующих функциональных возможностей.



Обсуждение HTML5, CSS3 и ARIA выходит за рамки этой книги. Чтобы создать хороший виджет с широкой совместимостью, вы должны уверенно владеть этими технологиями.

Базовый шаблон

Виджет должен быть инициализирован одним из двух способов:

- путем автоинициализации, подразумевающей автоматическое создание значений на основании указанных ролей или семантического кода HTML5;
- путем явной инициализации с использованием имени виджета и синтаксиса jQuery, например:

```
$("#myelement").widgetname()
```

Первый шаг в создании виджета заключается в выборе имени. Постарайтесь дать ему имя, которое не вступит в конфликт с именами других модулей.

Виджет должен использовать один файл с кодом JavaScript, названный jquery.mobile.*<имя_виджета>*.js и (необязательно) файл jquery.mobile.*<имя_виджета>*.css.

Шаблон виджета выглядит следующим образом:

```
(function($){
  // Мы заключаем весь код в эту функцию-оболочку для уверенности,
  // что $ будет подсоединяться к глобальному объекту jQuery.
  // Определение виджета
  $.widget("mobile.ourWidgetName", $.mobile.widget, {
    options: {
      // Здесь можно определить опции виджета, выбираемые по умолчанию
    },
    // Закрытые методы
    _create: function() {
      // Функция-конструктор
    },
```

[1] "Представление таблицы" и "выбор даты". — *Прим. перев.*

```
    // Открытые методы
    enable: function() {
      // Сделать виджет активным
    },
    disable: function() {
      // Сделать виджет неактивным
    },
    refresh: function() {
      // Обновить виджет
    }
  });
  // Конец определения виджета.
  // Код автоинициализации
  $(document).bind("pagecreate", function(event) {
    // Находим значение атрибута data-role и вызываем конструктор виджета
    $(event.target).find(":jqmData(role='ourWidgetName')").ourWidgetName();
  });
}(jQuery));
```

Создание собственного дополнительного модуля

В качестве примера создадим виджет для динамического показа изображений, который будет использовать бесплатный сервис Sencha IO Src. Этот облачный сервис получает изображение из источника и масштабирует его на сервере в соответствии с параметрами мобильного устройства.

Официальную документацию по сервису Sencha IO Src можно найти на сайте **http://mobilexweb.com/go/senchaio**.

Назовем наш виджет `dynamicimage`. Мы можем автоматически применить его к элементу `<img>` или воспользоваться новым значением `dynamic-image` атрибута `data-role`. Выберем второй вариант.

Использование

Идея использования виджета заключается в том, чтобы определить элемент `<img>` с атрибутом `data-src`, который будет загружать изображение с Sencha IO Src. Мы решили отказаться от стандартного атрибута `src`, потому что он приводит к двойной загрузке: вместе с отмасштабированным файлом загружается и оригинальный.

У нас будут два специальных атрибута. Атрибут `data-width` будет определять, какой процент ширины экрана займет изображение, а атрибут `data-margin` будет управлять размером полей, вычисляемым на основании размера экрана данного устройства.

Мы не станем применять темы, поскольку на платформе jQuery Mobile отсутствует визуализация изображений.

Виджет

Каждый виджет определяется как объект со свойствами и функциями. Функция, имя которой начинается с подчеркивания, будет закрытой. В теле функции ключевое слово `this` означает объект-виджет, т. е. конструкция `this.element` будет относиться к HTML-элементу:

```
$.widget("mobile.dynamicimage", $.mobile.widget, {
  options: {
    width: "100%",
    margin: 0
  }
});
```

Конструктор всегда носит имя `_create`. Он вызывается один раз при первой инициализации виджета. Для обращения к коллекции опций можно воспользоваться конструкцией `this.options`.

Функция-конструктор нашего виджета выглядит следующим образом:

```
$.widget("mobile.dynamicimage", $.mobile.widget, {
  options: {
    width: 100,
    margin: 0
  },
  _create: function() {
    // Вызываем закрытую функцию
    this._loadURL();
  }
});
```

Любая открытая функция (имя которой не начинается с подчеркивания) может быть вызвана с использованием синтаксиса виджета, и обычно мы должны поддерживать как минимум функции `refresh`, `enable` и `disable`, широко используемые в платформе jQuery Mobile. Например, если мы изменим URL-адрес изображения в коде JavaScript, то можем принудительно обновить виджет с помощью конструкции `$("#ourImage").dynamicimage("refresh")`, которая приведет к вызову функции `refresh`.

Определим функции `refresh`, `enable` и `disable`:

```
$.widget("mobile.dynamicimage", $.mobile.widget, {
  options: {
    width: 100,
    margin: 0
  },
  _create: function() {
    // Вызываем закрытую функцию
    this._loadURL();
  },
```

```
  // Открытые методы
  enable: function() {
    // Сделать виджет активным
    $(this.element).attr('disabled', '');
  },
  disable: function() {
    // Сделать виджет неактивным
    $(this.element).removeAttr('disabled');
  },
  refresh: function() {
    // Обновить виджет
    this._loadURL();
  }
});
```

И, наконец, напишем метод ядра, закрытую функцию `_loadURL`:

```
_loadURL: function() {
  // Конструкция this.element обозначает элемент +img+
  var url;  // создаем URL-адрес сервиса
  url = "http://src.sencha.io/";
  var parameters = "";
  if (!isNaN(this.options.width)) {
    parameters += "x" + this.options.width;
  }
  if ((!isNaN(this.options.margin)) && this.options.margin>0) {
    parameters += "-" + this.options.margin;
  }
  if (parameters.length>0) {
    url += parameters + "/";
  }
  // Сервису Sencha IO требуется абсолютный URL-адрес
  var originalUrl = $(this.element).jqmData("src");
  if (originalUrl.length>1) {
    var newUrl = "";
    if ($.mobile.path.isAbsoluteUrl(originalUrl)) {
    // URL-адрес изображения уже абсолютный
    newUrl = originalUrl;
  } else {
    // URL-адрес изображения относительный; создадим абсолютный
    var baseUrl = $.mobile.path.parseUrl(location.href);
    var baseUrlWithoutScript = baseUrl.protocol + "//" +
      baseUrl.host + baseUrl.directory;
    newUrl = $.mobile.path.makeUrlAbsolute(originalUrl,
      baseUrlWithoutScript);
  }
  url += newUrl;
  $(this.element).attr("src", url);
}
```

Автоинициализация

Написав этот код виджета, мы можем добавить код автоинициализации, в котором будет выполняться поиск элемента `data-role="dynamic-image"` с последующим созданием виджета:

```
$(document).bind("pagecreate", function(event) {
  // Ищем элемент с указанной ролью и вызываем конструктор виджета
  $(event.target).find("img:jqmData(role='dynamic-image')").dynamicimage();
});
```

Конструкция `jqmData` используется вместо `attr` для совместимости с нестандартным пространством имен. Каждый атрибут `data-*` нашего элемента будет автоматически отображаться на `this.options` внутри нашего объекта-виджета.

Применение нашего модуля

Вначале мы должны указать наш JavaScript-файл после JavaScript-файлов платформы jQuery Mobile:

```
<script src="jquery.mobile-dynamicimage-1.0.js"></script>
```

После этого будет достаточно создавать элементы `<img>` с правильными параметрами:

```
<!-- Изображение займет всю ширину экрана -->
<img data-src="images/photo.png" data-role="dynamic-image">
<!-- Изображение займет 40% ширины экрана -->
<img data-src="images/photo.png" data-role="dynamic-image"
     data-width="40">
<!-- Изображение займет всю ширину экрана с полем в 20 пикселов -->
<img data-src="images/photo.png" data-role="dynamic-image"
     data-margin="50">
```

Наш модуль изображен на рис. 10.1.

Полный код модуля

Далее приводится полный исходный код модуля jquery.mobile-dynamicimage-1.0.js:

```
(function($){
  // Определение виджета
  $.widget( "mobile.dynamicimage", $.mobile.widget, {
    options: {
    margin: 0, width: 100
  },
  _create: function() {
    this._loadURL();
  },
  // Закрытые методы
  _loadURL: function() {
```

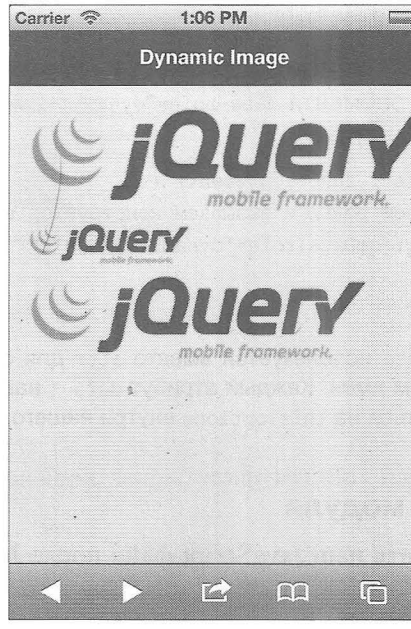


Рис. 10.1. Дополнительный модуль корректно работает на странице jQuery Mobile, показывая одно и то же изображение, по-разному отмасштабированное на сервисе Sencha IO Src

```
// Конструкция this.element обозначает элемент +img+
var url; // создаем URL-адрес сервиса
url = "http://src.sencha.io/";
var parameters = "";
if (!isNaN(this.options.width)) {
  parameters += "x" + this.options.width;
}
if ((!isNaN(this.options.margin)) && this.options.margin>0) {
  parameters += "-" + this.options.margin;
}
if (parameters.length>0) {
  url += parameters + "/";
}
// Сервису Sencha IO требуется абсолютный URL-адрес
var originalUrl = $(this.element).jqmData("src");
if (originalUrl.length>1) {
  var newUrl = "";
  if ($.mobile.path.isAbsoluteUrl(originalUrl)) {
    // URL-адрес изображения уже абсолютный
    newUrl = originalUrl;
  } else {
    // URL-адрес изображения относительный; создадим абсолютный
    var baseUrl = $.mobile.path.parseUrl(location.href);
    var baseUrlWithoutScript = baseUrl.protocol + "//"
      + baseUrl.host + baseUrl.directory;
```

```
        newUrl = $.mobile.path.makeUrlAbsolute(originalUrl,
          baseUrlWithoutScript);
      }
      url += newUrl;
      $(this.element).attr("src", url);
    }
  },
  // Открытые методы
  enable: function() {
    // Сделать виджет активным
    $(this.element).attr('disabled', '');
  },
  disable: function() {
    // Сделать виджет неактивным
    $(this.element).removeAttr('disabled');
  },
  refresh: function() {
    // Обновить виджет
    this._loadURL();
  }
  });   // Конец определения виджета.
  // Код автоинициализации
  $(document).bind("pagecreate", function(event) {
    // Находим значение атрибута data-role и вызываем конструктор виджета
    $(event.target)
      .find("img:jqmData(role='dynamic-image')").dynamicimage();
  });
}(jQuery));
```

Замечательные модули

Существует ряд модулей, достойных упоминания. Далее приводятся краткие описания самых полезных дополнительных модулей, которыми сейчас можно пользоваться в проектах jQuery Mobile.

Модуль Pagination

Добавляемый модуль Pagination (рис. 10.2) находится по адресу **http://filamentgroup.com/lab/jquery_mobile_pagination_plugin**. Он позволяет платформе jQuery Mobile разбивать на страницы содержимое, например, альбомы фотографий. На каждой странице присутствуют кнопки со стрелками влево и вправо, чтобы пользователь мог перелистывать страницы в обоих направлениях.

Пользователь может переходить со страницы на страницу следующими способами:

- прикасаясь к экранным кнопкам со стрелками влево и вправо;
- нажимая на клавиши со стрелками на устройствах с клавиатурой;

- сдвигая экранное изображение пальцем влево и вправо (модуль обрабатывает события перетаскивания).

Загрузив модуль, мы обнаружим два файла:

- jquery.mobile.pagination.css;
- jquery.mobile.pagination.js.

После включения их в элемент `<head>` мы должны будем создать элемент `<ul>` с атрибутом `data-role="pagination"`. Виджет разбивки на страницы должен находиться на каждой странице jQuery Mobile.

Рис. 10.2. Модуль Pagination в действии: показ нескольких изображений в виде отдельных страниц

Каждый виджет разбивки на страницы имеет два элемента `<li>`, которые содержат ссылки `<a>` на предыдущую и следующую страницы. Например:

```
<ul data-role="pagination">
  <li class="ui-pagination-prev"><a href="1.html">Prev</a></li>
  <!-- Предыдущая -->
  <li class="ui-pagination-next"><a href="3.html">Next</a></li>
  <!-- Следующая -->
</ul>
```

Модуль Bartender

Этот дополнительный модуль доступен на странице **http://www.stokkers.mobi/valuables/bartender.html**. Он выводит внизу экрана таблицу навигации в стиле собственных приложений платформы iOS (рис. 10.3).

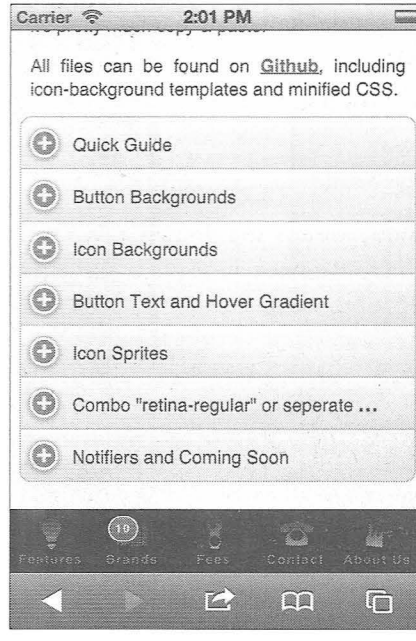


Рис. 10.3. Модуль Bartender с панелью навигации в стиле iOS

Модуль представляет собой CSS-файл и несколько изображений. Таким образом, вам не нужно обращаться ни к каким элементам `data-*`, а достаточно использовать определения классов.

Типичный код будет реализован в фиксированном нижнем колонтитуле в обычном виджете `navbar`. Шаблон выглядит примерно так:

```
<div data-role="navbar" data-grid="d">
  <ul class="apple-navbar-ui comboSprite">
    <!-- элементы -->
  </ul>
</div>
```

Модуль также поддерживает вкладки со счетчиками, как в iOS, для чего служит тег `<span class="uili-count">XXXX</span>` внутри элемента `<li>`.

Модуль DateBox

Модуль DateBox предоставляет селектор для выбора даты (рис. 10.4), используя синтаксис jQuery Mobile. Он определяется атрибутом `data-role="datebox"`, который может быть применен к любому элементу типа `<input type="date">`. Модуль имеет широкие возможности настройки предпочтений, причем стандартный код выглядит так:

```
<input type="date" data-role="datebox">
```

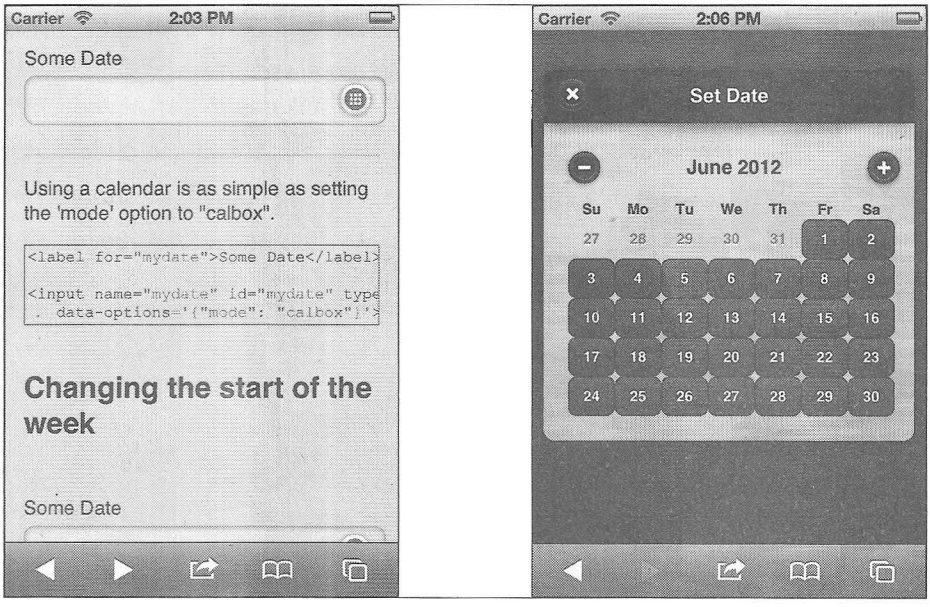


Рис. 10.4. Модуль DateBox позволяет создавать привлекательные селекторы дат даже на устройствах, не поддерживающих типы ввода данных стандарты HTML5

Официальный сайт с загружаемыми файлами, примерами и документацией находится по адресу **http://dev.jtsage.com/jQM-DateBox**. Модуль всегда имеет вид всплывающего окна (рис. 10.5) и работает в разных режимах:

- полнофункциональный календарный селектор;
- режим Android: календарь с кнопками "+" и "–" для изменения дня, месяца и года;
- режим Slider: три горизонтальных ползунковых регулятора для выбора года, месяца и дня;
- режим Flip: три вертикальных ползунковых регулятора для выбора месяца, дня и года;
- показ времени;
- показ продолжительности интервала.

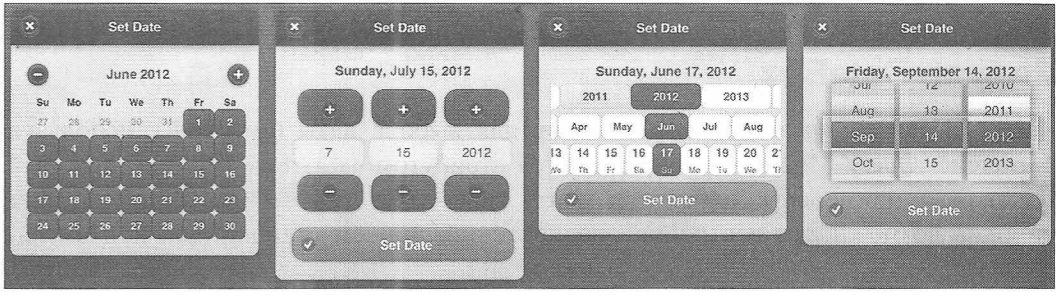


Рис. 10.5. Различные режимы работы модуля DateBox

Модуль Simple Dialog

Этот бесплатный дополнительный модуль позволяет заменить стандартные окна `window.alert`, `window.confirm` и `window.prompt` на окно в стиле jQuery Mobile, требующее ответа пользователя (рис. 10.6).

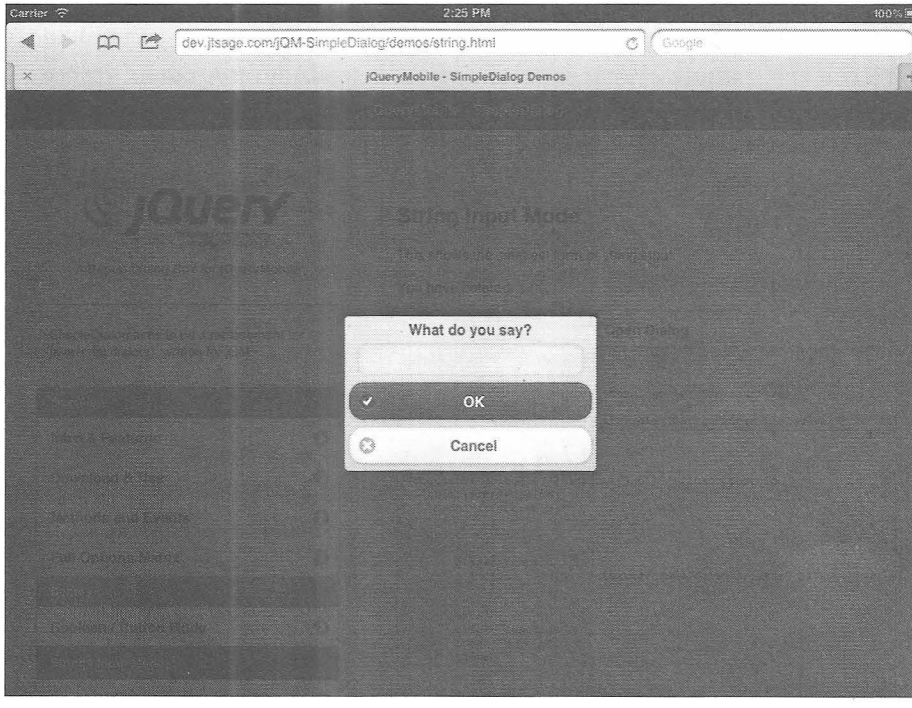


Рис. 10.6. Модуль Simple Dialog позволяет заменить диалоговыми окнами в стиле jQuery Mobile стандартные всплывающие окна, реализуемые в коде JavaScript

Документацию и сам модуль можно загрузить с сайта **http://dev.jtsage.com/jQM-SimpleDialog**.

Чтобы работать с дополнительным модулем Simple Dialog, мы должны загрузить и добавить в элемент `<head>` файл со сценарием модуля или получить модуль из сети CDN:

```
<script src="http://dev.jtsage.com/cdn/simpledialog/latest/
jquery.mobile.simpledialog.min.js"></script>
```

После этого, если мы хотим вывести предупреждение, то должны написать такой код:

```
$("#button").click(function() {
  $(this).simpledialog({
    mode: 'bool',
      // Для обычного предупреждения или простого окна подтверждения
    prompt: "We could not open the file",
```

```
      // Не удалось открыть файл
    useModal: true,
    buttons: [
      'Ok': {
        theme: "c", icon: "check"
      }
    ]
  });
});
```

Если же нам нужно диалоговое окно, код должен выглядеть так:

```
$("#button").click(function() {
  $(this).simpledialog({
    mode: 'bool',
    prompt: "Do you want to delete this file?",
    // Вы хотите удалить этот файл?
    useModal: true,
    buttons: [
      'Yes': {
        theme: "c", icon: "delete",
        click: function() { // Удалить }
      },
      'No': {
        theme: "a", icon: "cancel"
      },
    ]
  });
});
```

И наконец, приведу код диалогового окна с текстовой подсказкой:

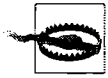
```
$("#button").click(function() {
  $(this).simpledialog({
    mode: 'string',
    prompt: "What is your name?",   // Назовите свое имя
    useModal: true,
    buttons: [
      'No': {
        theme: "c", icon: "delete",
        click: function() {
          alert("Your name is " + $("#button").jqmData("string");
          // Вас зовут
        }
      }
    ]
  });
});
```

Модуль Action Sheet

Есть еще один добавляемый модуль, авторов которого вдохновил внешний вид элементов управления iOS. Модуль Action Sheet представляет собой модальное всплывающее меню.

Вы найдете модуль (файлы CSS и JavaScript) на сайте **https://github.com/hiroprotagonist/jquery.mobile.actionsheet**.

Чтобы создать меню Action Sheet, нам нужно определить кнопку для его открытия. Это должен быть элемент `<a>` с атрибутом `data-role="actionsheet"`. В ответ на щелчок он откроет следующий элемент того же уровня или элемент с идентификатором (`id`), определенным в атрибуте `data-sheet`.



Не забудьте добавить в элемент `<head>` ссылки на файлы jquery.mobile.actionsheet.js и jquery.mobile.actionsheet.css.

Этот элемент должен содержать кнопки jQuery Mobile. Если пользователь щелкнет за пределами элемента, меню закроется. Кроме того, мы можем создать кнопку с атрибутом `data-rel="close"`, которая будет закрывать элемент. Например:

```
<a data-role="actionsheet" data-sheet="share" data-icon="plus">Share</a>
<div id="share">
  <a href="facebook.html" data-role="button">Share in Facebook</a>
  <!-- Поделиться в Facebook -->
  <a href="twitter.html" data-role="button">Share in Twitter</a>
  <!-- Поделиться в Twitter -->
  <a href="google.html" data-role="button">Share in Google+</a>
  <!-- Поделиться в Google+ -->
  <a data-rel="close" data-role="button">Cancel</a>
  <!-- Отмена -->
</div>
```

Дополнительные модули для планшетов

При разработке веб-приложения для планшетов иногда бывает полезно разбить страницу на два столбца, избежав тем самым стандартного для jQuery Mobile вывода меню шириной во весь экран. Для этого имеются два дополнительных модуля, рассчитанных на планшетные приложения: SplitView и MultiView.

Модуль SplitView

Модуль SplitView находится на сайте **http://asyraf9.github.com/jquery-mobile** и позволяет нам определить внутри документа две области, называемые панелями.

Каждая панель является страницей jQuery Mobile со своим заголовком, содержимым и нижним колонтитулом. Модуль SplitView позволяет двум панелям находиться на экране одновременно при альбомной ориентации (рис. 10.7). Это означает,

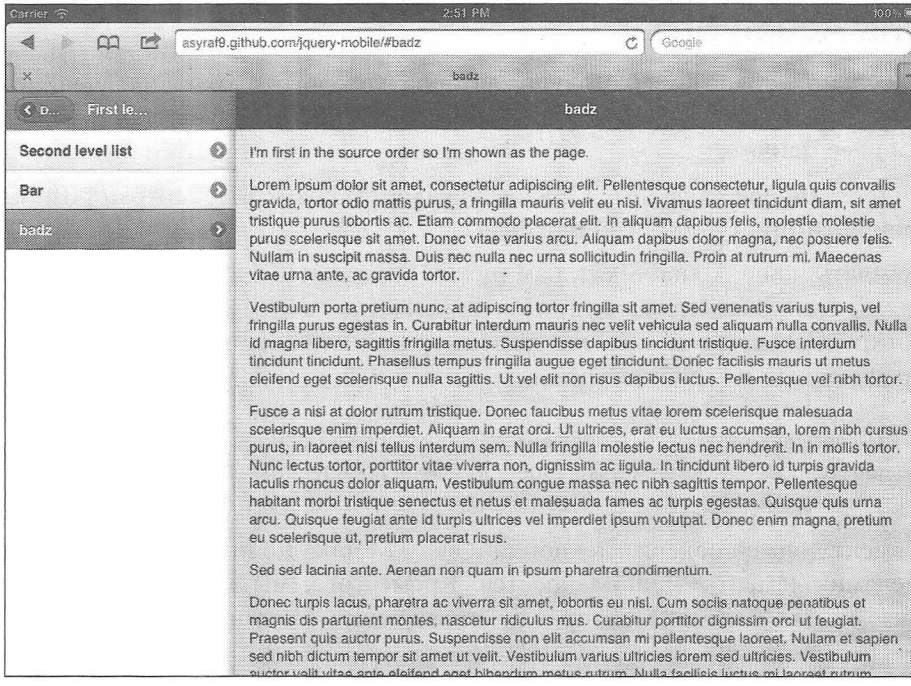


Рис. 10.7. Модуль SplitView позволяет создавать планшетные приложения с одновременным выводом на экран нескольких страниц, как правило, меню и области содержимого

что мы будем иметь две страницы, как правило, меню слева и область содержимого справа.

При книжной ориентации панель меню скрыта за кнопкой в левом верхнем углу страницы и может открывать как всплывающее меню (рис. 10.8).

Каждая панель имеет атрибут `data-id`, определяющий ее имя. Это полезно для ссылок:

```
<body>
  <div data-role="panel" data-id="menu">
    <div data-role="page">
    </div>
  </div>
  <div data-role="panel" data-id="main">
    <div data-role="page">
    </div>
  </div>
</body>
```

После определения идентификаторов мы можем в каждом элементе `<a>` указать панель, в которую должна загружаться страница. Анимация перехода работает внутри каждой панели так, словно это фреймы.

Например:

```
<a href="demo.html" data-panel="main">Demos</a>
```

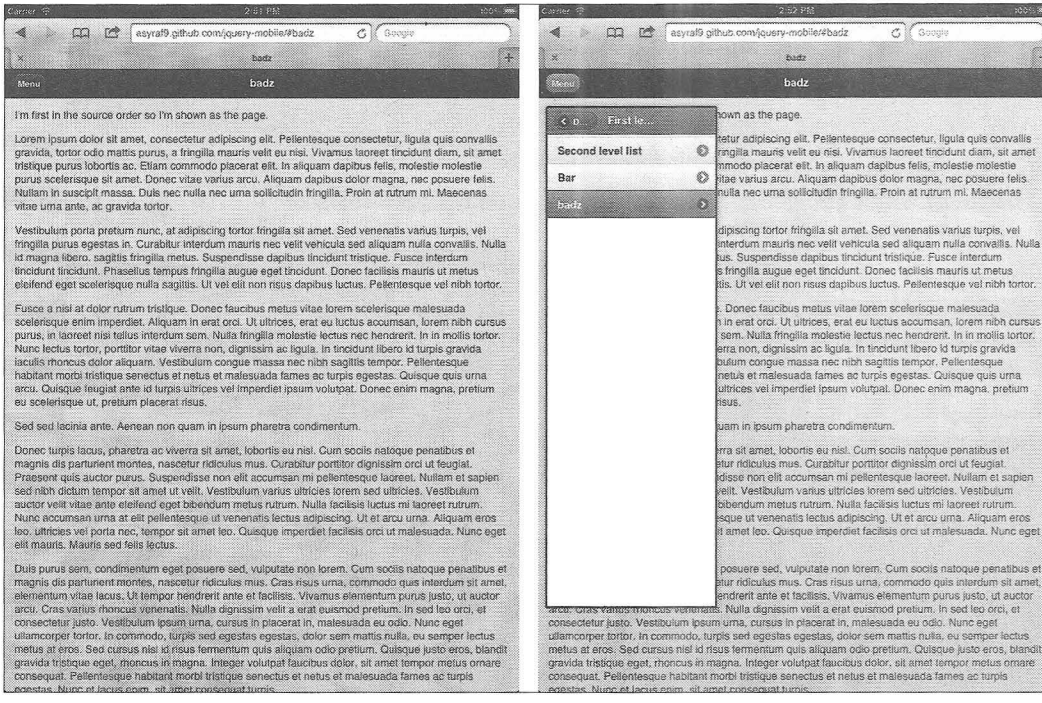


Рис. 10.8. После придания устройству книжной ориентации модули SplitView и MultiView преобразуют два столбца в один раздел, имеющий всплывающее меню, открываемое кнопкой в левом верхнем углу

Модуль MultiView

Модуль MultiView имеет то же предназначение, что и SplitView, но реализован иначе (рис. 10.9). Он тоже реализует роль `data-role="panel"`, но соответствующие элементы будут потомками элемента `page`. Модуль доступен в виде работающей демо-версии на сайте **http://www.stokkers.mobi/valuables/multiview/page1.html**, и он меняет обычное поведение платформы jQuery Mobile. Элемент с ролью `datarole="page"` состоит не из областей с содержимым, а максимум из четырех областей-панелей (атрибут `data-role="panel"`):

- панель меню;
- главная панель;
- полноэкранная панель (необязательная, для альбомной ориентации);
- всплывающая панель (необязательная, для альбомной ориентации).

Тип панели определяется атрибутом `data-id`, например, `data-id="menu"`.

Каждая панель содержит свои страницы, так что мы имеем вложенность страниц. Структура отношений объектов такова: страница → панель → страница → заголовок/содержимое/нижний колонтитул. Каждая панель может иметь столько страниц, сколько потребуется, но хотя бы одна из них должна иметь атрибут `data-show="first"`.

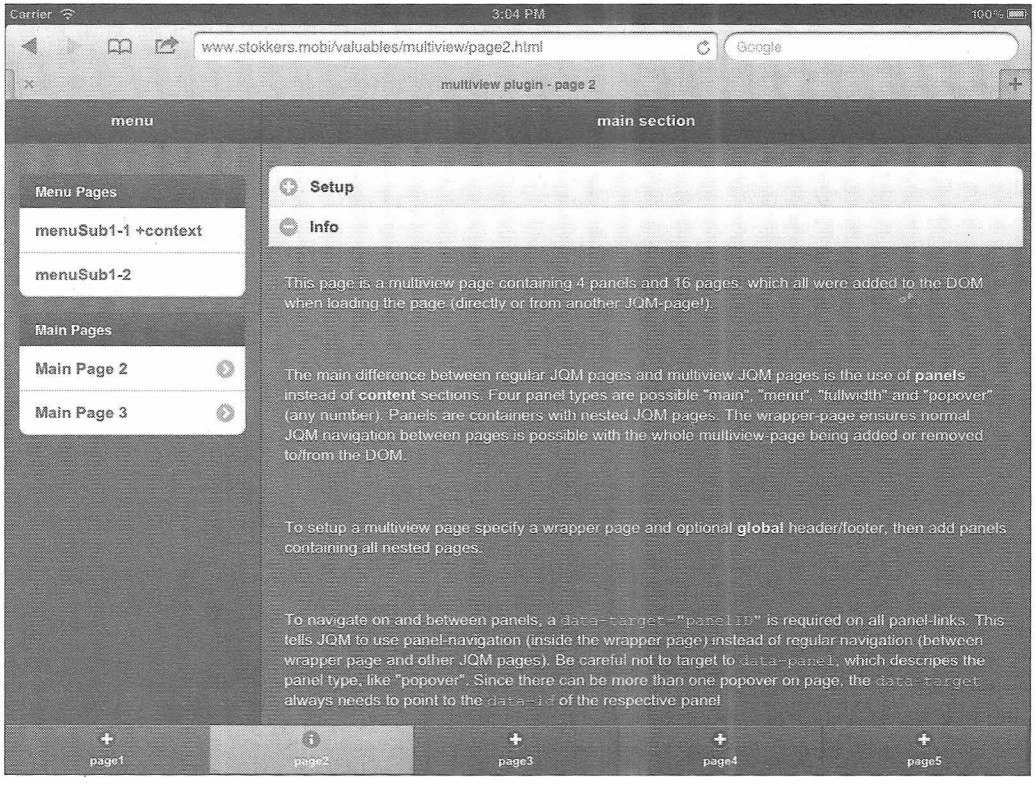


Рис. 10.9. Модуль MultiView позволяет создавать приложения в "планшетном" стиле с фиксированными нижними колонтитулами, общими для всех панелей

Атрибут `data-target` позволяет нам определить, где следует открывать страницу. Возможные варианты:

- загрузка всей страницы (с удалением всех панелей);
- загрузка панели (панели меню или главной);
- мультизагрузка (одна страница в меню и одна на главную панель).

Объем документации модуля довольно мал, но демонстрационная анимация позволяет решить все вопросы.

Совместимые модули

Существует несколько модулей jQuery (не мобильных), которыми можно пользоваться и на платформе jQuery Mobile. Однако в них не соблюдаются правила платформы, и поэтому в вашем распоряжении не будет ни атрибутов `data-role`, ни функциональности, обеспеченной чистым кодом JavaScript.

В следующем списке перечислены некоторые из этих дополнительных модулей.

- Photoswipe (**http://photoswipe.com**) позволяет создавать фотогалереи для iOS, Android и BlackBerry 6+;

- Diapo (**http://www.pixedelic.com/plugins/diapo**) представляет собой слайд-шоу с отличной CSS-анимацией;
- jQuery UI Maps (**http://code.google.com/p/jquery-ui-map**) обеспечивает интеграцию Google Maps в ваше мобильное приложение;
- MobiScroll (**http://www.mobiscroll.com**) предоставляет селекторы даты и времени в виде кнопочных или ползунковых виджетов.

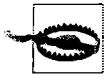
ГЛАВА 11

Упаковка приложения для продажи

Существует мнение, что будущее за магазинами приложений. Откровенно говоря, никто не знает, как будут распространяться приложения в будущем — через магазины, через браузеры или с помощью какого-либо гибридного механизма. Самая замечательная особенность содержимого, созданного на основе HTML5, в частности приложений jQuery Mobile, состоит в том, что мы можем упаковать свое веб-приложение как низкоуровневое и распространять его через магазины приложений.

Итак, если вы хотите распространять приложение через магазин, платформа jQuery Mobile позволит вам сделать это. В зависимости от целевой платформы, существует несколько подходов к реализации этой идеи.

Первое, что следует усвоить, — это необходимость создания отдельного пакета для каждой платформы. Тем или иным способом мы должны скопировать свои файлы (файлы HTML, JavaScript, CSS и jQuery Mobile) в разные проекты и создать разные пакеты.

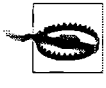


В состав консорциума W3C входит группа, работающая над будущим стандартом упаковки веб-приложений для распространения. Она называется Native Web apps, и ее страница находится по адресу **http://www.w3.org/community/native-web-apps/**.

Когда мы упаковываем веб-приложение как низкоуровневое, то обычно имеем возможность обращаться к некоторым новым API-интерфейсам, не описанным в HTML5, таким как Camera, Contacts или Accelerometer.

Чтобы упаковать веб-приложение для распространения через магазин, мы можем:

- создать низкоуровневый проект для каждой платформы, добавить файлы веб-приложения в виде локальных ресурсов и связать компонент Web View с нашим HTML-содержимым. Иногда результат называют гибридным приложением;
- воспользоваться официальной платформой веб-приложения, что обычно подразумевает упаковку файлов в ZIP-архив;
- воспользоваться средствами компиляции в коды устройства и откомпилировать наше приложение под разные платформы.



Компиляция приложения в низкоуровневый пакет обычно требует знания языка низкого уровня и SDK-комплекта целевой платформы.

Распространение через магазин

В качестве первого шага упаковки приложения мы должны решить, на какие платформы мы нацеливаемся и через какие магазины будем распространять приложение.

В табл. 11.1 приводится список доступных магазинов. В каждом вы должны зарегистрироваться как издатель.

Таблица 11.1. *Магазины, позволяющие распространять приложения*

Магазин	Владелец	Платформы	Форматы	Стоимость публикации	URL-адрес
AppStore	Apple	iOS (iPhone, iPod, iPad)	ipa	99 долл. в год	**http://developer.apple.com/programs/ios**
Android Market	Google	Android	apk	20 долл. однократно	**http://market.android.com/publish**
AppWorld	RIM BlackBerry	Smartphones/ PlayBook	cod/bar	Бесплатно	**http://appworld.blackberry.com/isvportal**
Nokia Store	Nokia	Symbian/N9	wgz/deb/ bar	1 евро	**http://info.publish.nokia.com/**
Amazon AppStore	Amazon	Android/ Kindle Fire	apk	99 долл. в год	**http://developer.amazon.com/**
Marketplace	Microsoft	Windows Phone		99 долл. в год	**http://create.msdn.com/**

Для каждой платформы мы должны собрать метаданные и сопоставить их с документацией магазина:

- размер значков в высоком разрешении (обычно 512×512);
- текст описания;
- категория;
- снимки экрана для каждой платформы;
- распространение — список совместимых устройств;
- распространение — страна и язык;
- маркетинговые баннеры.

Распространение своими силами

Некоторые платформы позволяют нам распространять приложение через наш собственный сервер при условии корректности конфигурации MIME-типов. Устройства, работающие под управлением Symbian и BlackBerry, без проблем принимают и устанавливают любой выполняемый файл с веб-приложением.

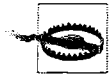
Устройства на платформах Android и Nokia N9 установят приложения с нашего сервера, если пользователь разрешил в настройках установку сторонних приложений.

Другие платформы, например iOS и Windows Phone, никогда не позволят установить приложение, полученное не из официального магазина.

Существуют способы конфигурирования устройств, используемых для разработки, таким образом, что установка наших приложений будет возможна, однако это разрешается делать только для целей тестирования. Обсуждение таких способов выходит за рамки этой книги.

Подготовка пакета

При создании приложений jQuery Mobile, работающих как низкоуровневые, мы не должны упускать из виду некоторые важные моменты. Во-первых, у приложения не будет панелей браузера. В частности, это означает, что в интерфейсе не будет кнопки **Назад**. Наше приложение должно явным образом содержать кнопку возврата в заголовке, о чем уже говорилось в этой книге.



При компилировании приложения в коды низкого уровня никогда не используйте сеть CDN в качестве источника файлов jQuery Mobile.

Вторая проблема будет связана с загрузкой внешних страниц. При создании приложения мы можем загружать локальные страницы, представляющие HTML-файлы, которые будут распространяться вместе с приложением.

Если мы хотим загружать документы jQuery Mobile, находящиеся по удаленному URL-адресу, то можем написать оператор `$.support.cors=true` в обработчике события `mobileinit`. Так мы разрешим платформе обращаться к удаленным серверам по технологии AJAX. Кроме того, следует объявить `$.mobile.allowCrossDomainPages=true`.

Разработчики платформы jQuery Mobile также рекомендуют отключить состояние `pushState`, чтобы избежать проблем с обработкой URL-адресов.

Таким образом, главный HTML-файл должен включать в себя следующий код:

```
$(document).bind("mobileinit", function() {
  $.support.cors = true;
  $.mobile.allowCrossDomainPages = true;
  $.mobile.pushState = false;
});
```

Если мы разрабатываем приложение под iOS 5.0+, то сможем воспользоваться такой функциональной возможностью, как подлинно фиксированные панели инструментов. Тем самым мы повысим производительность приложения и удобство фиксированных заголовков и нижних колонтитулов.

При упаковке низкоуровневых приложений мы должны подготовить некоторые дополнительные файлы, например пиктограмму-ярлык для главного экрана или меню приложений, а также изображение, которое будет выводиться при запуске приложения. На некоторых платформах приходится явно указывать список серверов для выхода в сеть.

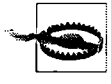
Упаковка с помощью PhoneGap

Платформа PhoneGap имеет открытый исходный код и предназначена для компиляции кода HTML в низкоуровневые приложения. Она полностью совместима с платформой jQuery Mobile. Сопровождается платформа PhoneGap (иногда называемая Apache CallBack) компанией Adobe и некоторыми другими известными фирмами. Полное описание PhoneGap выходит за рамки этой книги. Ограничусь базовыми сведениями, которые позволят вам приступить к работе с этой платформой.

С помощью PhoneGap (**http://phonegap.com**) мы можем откомпилировать приложение jQuery Mobile в коды следующих платформ:

- iOS;
- Android;
- webOS;
- Symbian;
- BlackBerry;
- Windows Phone.

Когда мы загрузим PhoneGap, то получим ZIP-файл с набором образцов пакетов, которыми можно пользоваться вместе с SDK каждой платформы, а также JavaScript-файл, который следует включать во все HTML-файлы приложения. Вплоть до версии PhoneGap 1.2 для каждой платформы требуется отдельный файл JavaScript. Этот файл нормализует особенности поведения приложения на разных платформах и обеспечит совместимость с некоторыми API.



Если вы хотите откомпилировать приложение под iOS, вам потребуется Mac или сервис в облаке. Вы можете легко арендовать устройство Mac на сайте **http://macincloud.com**.

Разработчикам веб-приложений под iOS платформа PhoneGap предоставляет механизм установки, который добавляет тип проекта PhoneGap в среду Xcode, официальную интегрированную среду разработки низкоуровневых приложений под iOS. В табл. 11.2 перечислены все комплекты SDK, которые вам придется установить для компиляции.

Когда мы создаем проект для нескольких платформ, то должны скопировать наши файлы (HTML, JavaScript, CSS, изображения и др.) в соответствующий каталог в каждом шаблоне проекта PhoneGap. Обычно этот каталог называется www. Вы

найдете его в файле index.html, который должны будете заменить файлами вашего приложения jQuery Mobile.

Таблица 11.2. *Инструментальные средства SDK и IDE, необходимые для компиляции веб-приложений в коды низкого уровня*

SDK	Платформа мобильного устройства	Платформа настольного компьютера	URL-адрес для загрузки
Xcode	iOS	Mac OS	Доступен на сайте магазина приложений Mac
ADT для Eclipse	Android	Win/Mac/Linux	**http://developer.android.com**
WebWorks SDK	BlackBerry	Win/Mac	**http:// blackberry.com/developers**
Visual Studio для WP	Windows Phone	Win	**http://microsoft.com/visualstudio**
Nokia Web Tools для Symbian	Symbian	Win/Mac	**http://developer.nokia.com**

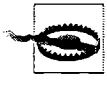
Сервис PhoneGap Build

Чтобы мы не запутались, создавая пакеты для каждой платформы с использованием разных комплектов SDK, компания Adobe предлагает сервис (с бесплатными и платными планами) для компиляции приложений в облаке. Сервис называется PhoneGap Build и доступен на сайте **http://build.phonegap.com**.

С помощью этого сервиса мы можем выгрузить на сервер архивный ZIP-файл с полным приложением jQuery Mobile (включая CSS, JavaScript и изображения) и файлом config.xml, удовлетворяющим стандарту W3C по написанию виджетов. После этого PhoneGap Build откомпилирует приложение в пакеты для:

- iOS;
- Android;
- webOS;
- BlackBerry (смартфоны);
- Symbian.

На момент работы над этой книгой сервис PhoneGap Build не поддерживал Windows Phone и планшеты BlackBerry, но эта поддержка должна появиться в ближайшем будущем.



Для iOS и BlackBerry мы должны предоставить сервису PhoneGap Build наши цифровые подписи, которые следует получить в рамках издательских программ этих платформ.

Предметный указатель

A

Action Sheet 239
Adobe Kuler 180
AJAX 60, 153
Amazon AppStore 245
Android Market 245
Application Cache 190
AppStore 245
AppWorld 245

B

Bartender 234

C

CDN 40
CORS 154
CSS-таблица 177

D

DateBox 235
Diapo 243
Dreamweaver 44

F

FaceTime 71
Fireworks 182
Fireworks CSS3 Mobile Pack 182

H

HTML5 29
◊ документ 42

I

iWebInspector 192

J

JavaScript 149, 195
jQTouch 63
jQuery Mobile, технология 15
jQuery UI Maps 243
JSON2 208

K

Kuler 181

M

Marketplace 245
MobiScroll 243
MultiView 241

N

Nokia Store 245

O

Offline API 190

P

Pagination 233
PhoneGap 28, 247
PhoneGap Build 248
Photoswipe 242

S

Sencha IO Src 228
Simple Dialog 237
Skype 71
SMS 73
SplitView 239

T

ThemeRoller 178

U

URI 72

X

Xcode 247

А

Автоинициализация 231
Аккордеон 94
Анимация перехода 63
Архитектура 46
Атрибуты данных, нестандартные 47

Б

Безопасность 132
Браузеры 25
◊ категории 28

В

Веб-приложение:
◊ способы реализации 17
◊ тестирование 33
Видеозвонок 71
Виджет 151
◊ инициализация 227
◊ конфигурация 158
◊ обновление 170
◊ создание 169, 227
◊ шаблон 227
Возврат на предыдущую страницу 53
Выбор множественный 146
Выгрузка файлов 148

Г

Гиперссылка 52
Группирование 98
◊ элементов меню 139

Д

Дата 134

Ж

Жест 175

З

Заголовок 51, 76
◊ нестандартный 82
Загрузка:
◊ предварительная 61
◊ страницы 157
Звонок 70
Значок строки 121

И

Изображение в заголовке 81
Инициализация кода 149
Инспектор свойств 180
Интеграция с телефоном 70

К

Канеда, Дэвид 63
Кнопка 79, 97
◊ "Назад" 53
Колонтитул нижний 76, 82
◊ постоянный 87
Контейнер поля 129
Конференция 209

Конфигурация 151
◊ виджетов 158
◊ глобальная 152
◊ страницы 157
Крокфорд, Дуглас 208

Л

Лаборатория удаленная 37
Логотип 81
Локализация 154

М

Магазины 245
Манифест 191, 210
◊ загрузка 192
Меню 137
◊ группирование элементов 139
◊ нестандартное 142
Метка 129
Метод:
◊ bind() 149
◊ changePage() 161
◊ getInheritedTheme() 164
◊ hidePageLoadingMsg() 164
◊ live() 172
◊ loadPage() 163
◊ page() 168
◊ showPageLoadingMsg() 164
◊ silentScroll() 164
Миниатюра 121
Модуль дополнительный, создание 226

Н

Навигация 52, 83
Настройки:
◊ глобальные 179
◊ образцов цвета 179
Нильсен, Якоб 132

О

Образец цвета 48
◊ настройки 179
Окно просмотра 43
Ориентация 174

П

Пакет 190
Панель инструментов 76
◊ навигации 83
◊ полноэкранная 77
◊ размещение 77
◊ фиксированная 79
Пароль 132
Переключатель 144
◊ двухпозиционный 136
Переход 161, 165, 174
◊ между страницами 62
◊ нестандартный 188
Пиктограмма 85, 100, 198
◊ допустимые размеры 201
◊ определение 201
Планшет 24, 239
Прогрессивное улучшение 32
Производительность 44
Прокрутка 152
Просмотр предварительный 46
Псевдобраузер 25
Путь 164

Р

Работа автономная 190, 210
Разделитель 109
Распространение 244
Регулятор ползунковый 135
Редактирование тем 187
Редактор тем 182
Режим полноэкранный 203
Ресиг, Джон 21
Ресурс:
◊ загрузка 193
◊ обновление 194
Роль 47

С

Сетка 171
◊ компоновки 95
Симулятор 35
Системы операционные 24
Смартфон 23
Событие:
◊ mobileinit 151
◊ orientationchange 174
◊ виджетов 174
◊ виртуальных щелчков 175
◊ документа 149
◊ жестов 175
◊ загрузки страницы 172

Событие *(прод.)*:
◊ объекта applicationCache 196
◊ переходов 174
◊ показа страницы 173
◊ страницы 171
Совместимость 26
Содержимое 51
◊ динамическое 166
◊ сворачиваемое 90
▫ вложенное 92
Соудерс, Стив 40
Список 105
◊ вложенный 114
◊ дополнительная информация 122
◊ интерактивный 111
◊ с разделенными кнопками 117
Ссылка:
◊ абсолютная внешняя 61
◊ внешняя 60
Старк, Джонатан 63
Степень важности строки 120
Стиль 204, 219
Столбцы 95
Страница 46, 49, 211
◊ диалоговая 64
◊ динамическая 166
◊ загрузка 157, 172
◊ конфигурация 157
◊ обновление 171
◊ показ 173
◊ события 171
◊ создание 166, 172
Строка:
◊ интерактивная 111
◊ разделенная 117
Счетчик 123

Т

Текст 130
Телефон 70
Тема 48, 177
◊ редактирование 187
◊ создание 178
◊ экспорт 181
Тестирование 33

У

Упаковка 244
Устройства мобильные:
◊ категории 21
◊ промежуточного и имиджевого уровня 22
Утилиты 160
◊ платформы 163
◊ пользовательского интерфейса 164
◊ пути 164

Ф

Фильтрация данных 124
Флажок 146
Форма 127
◊ без использования AJAX 128
◊ элементы 128
Форматирование 89

Х

Хранение файлов 38

Щ

Щелчок виртуальный 175

Э

Экспорт темы 181
Элемент выделенный 86
Эмулятор 33

jQuery Mobile: разработка приложений для смартфонов и планшетов

Вы хотите создать мобильное приложение, которое будет работать на iPad и Kindle Fire, а также на смартфонах iPhone и Android? В этой книге рассказано, как это сделать. Выполнив ряд практических упражнений, вы научитесь использовать многочисленные компоненты фреймворка jQuery Mobile для построения гибких мультиплатформенных приложений. От вас не потребуются навыки программирования и предыдущий опыт работы с библиотекой jQuery. В книге рассмотрено:

- Использование компонентов пользовательского интерфейса
- Оформление и настройка пользовательского интерфейса с помощью JavaScript, AJAX и ядра библиотеки jQuery
- Настройка внешнего вида пользовательского интерфейса с помощью тем и CSS3
- Создание динамического содержимого с помощью JavaScript, Ajax и библиотеки jQuery
- Создание приложений с возможностью автономной работы off-line
- Распространение приложений

Максимилиано Фиртман (Maximiliano Firtman), специалист в области разработки мобильных приложений и приложений с использованием HTML5. Основатель тренинговой компании ITMaster Professional, в которой преподает мобильные технологии. Имеет звание Adobe Community Professional и Nokia Developer Champion. Автор книги «Programming the Mobile Web», вышедшей в издательстве O'Reilly.

«Эта книга описывает фреймворк jQuery Mobile очень эффективным способом, который позволяет читателю научиться работать с ним как можно быстрее.»

Майк Хостетлер
(Mike Hostetler), генеральный директор компании appendTo

БХВ-ПЕТЕРБУРГ
191036, Санкт-Петербург,
Гончарная ул., 20
Тел.: (812) 717-10-50,
339-54-17, 339-54-28
E-mail: mail@bhv.ru
Internet: www.bhv.ru

O'REILLY®