Если вы – разработчик, ищущий серьезное руководство по созданию мобильных приложений на Objective-C и iPhone SDK, эта книга – для вас!

«Программирование для iPhone» предназначено для профессиональных программистов, желающих создавать приложения для iPhone 3G и iPod Touch с помощью Apple SDK. Освоение материала не требует предварительного изучения языка Objective-C – его основы даются в начале книги.
В книге подробно описываются основы iPhone SDK. Значительная часть материала посвящена разработке пользовательских интерфейсов, механизму баз данных SQLite и библиотеке обработки XML libxml2. Большое внимание уделено возможностям iPhone как GPS-навигатора.

Автор книги – опытный программист в области создания приложений под мобильные устройства. Кроме того, он обладает весьма впечатляющим набором титулов – научный сотрудник Bell Labs, доктор наук, известный ученый и активный участник ряда интернет-сообществ.

В книге подробно раскрываются следующие вопросы:

- язык программирования Objective-C;
- Cocoa Touch;
- разработка мобильных пользовательских интерфейсов;
- основы анимации и Quartz 2D;
- шаблоны типа «модель-представление-контроллер» (Model-view-controller, MVC);
- табличное представление данных;
- управление файлами;
- синтаксический анализ XML-документов с использованием SAX и DOM;
- работа с Google Maps API;
- использование веб-сервисов RESTful;
- разработка продвинутых приложений, основанных на анализе местоположения;
- создание баз данных с помощью механизма SQLite;
- разработка мультимедиа-приложений;
- использование камеры и видео.

ISBN 978-5-699-40764-4

9 785699 407644 >

эксмо

Махер Али

ПРОГРАММИРОВАНИЕ для iPhone

эксмо

Махер Али

ПРОГРАММИРОВАНИЕ для iPhone

Разработка мобильных приложений с помощью iPhone SDK

эксмо

iPhone SDK Programming

Maher Ali

ПРОГРАММИРОВАНИЕ для iPhone

Разработка мобильных приложений с помощью iPhone SDK

Махер Али

ЭКСМО
Москва
2010

УДК 004.42
ББК 32.973.26
М 36

Махер А.
М 36 Программирование для iPhone / Махер Али ; [пер. с англ.]. – М. : Эксмо, 2010. – 368 с.

ISBN 978-0-470-74282-2 (англ.)
ISBN 978-5-699-40764-4

Книга предназначена для профессиональных программистов, желающих создавать приложения для iPhone 3G и iPod Touch с помощью Apple SDK. Освоение материала не требует предварительного изучения языка Objective-C – его основы даются в начале книги. В книге подробно описываются основы iPhone SDK. Значительная часть материала посвящена разработке пользовательских интерфейсов, механизму баз данных SQLite и библиотеке обработки XML libxml2. Большое внимание уделено возможностям iPhone как GPS-навигатора.

УДК 004.42
ББК 32.973.26

ISBN 978-0-470-74282-2 (англ.)
ISBN 978-5-699-40764-4

Содержание

Введение **11**

Для кого предназначена книга **12**

Что вам необходимо **13**

Структура книги **14**

Глава 1. Objective-C и Cocoa **17**

1.1. Классы 18

1.1.1. Объявление класса 18

1.1.2. Как использовать другие объявления 19

1.1.3. Определение класса 20

1.1.4. Определение и вызов методов 20

1.1.5. Важные типы 21

1.1.6. Важные классы Cocoa 22

1.2. Управление памятью 23

1.3. Протоколы 26

1.4. Свойства 28

1.5. Категории 34

1.6. Позиционирование 35

1.7. Исключения и ошибки 36

1.7.1. Исключения 36

1.7.2. Ошибки 40

1.8. Кодирование «ключ-значение» 42

1.9. Многопотоковость 50

1.10. Резюме 54

Задачи 54

Глава 2. Коллекции **58**
2.1. Массивы 58
2.1.1. Неизменяемое копирование 62
2.1.2. Изменяемое копирование 64
2.1.3. Расширенное копирование 65
2.1.4. Сортировка массива 69
2.2. Множества 74
2.2.1. Неизменяемые множества 74
2.2.2. Изменяемые множества 75
2.2.3. Дополнительные важные методы 77
2.3. Словари 78
2.4. Резюме 80

Глава 3. Анатомия iPhone-приложения **82**
3.1. Приложение HelloWorld 82
3.2. Создание приложения HelloWorld 85

Глава 4. Представление **90**
4.1. Геометрия представления 90
4.1.1. Определения полезных геометрических типов данных 90
4.1.2. Класс UIScreen 91
4.1.3. Атрибуты frame и center 93
4.1.4. Атрибут bounds 94
4.2. Иерархия представлений 96
4.3. Multitouch-интерфейс 96
4.3.1. Класс UITouch 97
4.3.2. Класс UIEvent 98
4.3.3. Класс UIResponder 98
4.3.4. Обработка скольжений 104
4.3.5. Более совершенное распознавание жестов 108
4.4. Анимация 113
4.4.1. Использование поддержки анимации в классе UIView 113
4.4.2. Анимация перехода 117
4.5. Рисование 120

Глава 5. Элементы управления **122**
5.1. Основа всех элементов управления 122
5.1.1. Атрибуты UIControl 123
5.1.2. Механизм «цель-действие» 123
5.2. UITextField 126
5.2.1. Взаимодействие с клавиатурой 128
5.2.2. Делегат 131
5.2.3. Создание и работа с UITextField 132
5.3. Слайдеры 133
5.4. Переключатели 134
5.5. Кнопки 135
5.6. Сегментированные элементы управления 136
5.7. Страничные элементы управления 140
5.8. Элементы выбора даты 141
5.9. Резюме 142

Глава 6. Контроллеры представлений **143**
6.1. Простейший контроллер представления 143
6.1.1. Контроллер представления 143
6.1.2. Представление 146
6.1.3. Делегат приложения 146
6.1.4. Резюме 148
6.2. Радиоинтерфейсы 149
6.2.1. Детальный пример 150
6.2.2. Некоторые комментарии к контроллерам панелей закладок 155
6.3. Контроллеры навигации 160
6.3.1. Пример поведения класса навигации 160
6.3.2. Настройка 166
6.4. Модальные контроллеры представления 170
6.5. Резюме 176

Глава 7. Специализированные представления **177**
7.1. Представления подбора значений 177
7.1.1. Делегат 177

7.1.2. Пример ... 179
7.2. Представления индикаторов деятельности ... 183
7.3. Текстовое представление ... 186
7.3.1. Делегат ... 186
7.3.2. Пример ... 187
7.4. Представление предупреждения ... 190
7.5. Списки действий ... 192
7.6. Веб-представления ... 193
7.6.1. Простое приложение с веб-представлением ... 194
7.6.2. Просмотр локальных файлов ... 197
7.6.3. Выполнение JavaScript ... 202
7.6.4. Делегат веб-представления ... 208

Глава 8. Табличное представление ... 214
8.1. Обзор ... 214
8.2. Простейшее приложение с табличным представлением ... 215
8.3. Табличное представление, содержащее изображения и текст ... 219
8.4. Табличное представление с верхним и нижним колонтитулами секции ... 220
8.5. Табличное представление с возможностью удалять строки ... 222
8.6. Табличное представление с возможностью вставки строк ... 229
8.7. Упорядочивание табличных строк ... 235
8.8. Вывод иерархической информации ... 240
8.9. Сгруппированные табличные представления ... 249
8.10. Индексированные табличные представления ... 252
8.11. Резюме ... 257

Глава 9. Управление файлами ... 258
9.1. Домашняя директория ... 258
9.2. Поиск в директории ... 259
9.3. Создание и удаление директории ... 260
9.4. Создание файлов ... 262

9.5. Считывание и изменение атрибутов ... 265
9.6. Работа с ресурсами и низкоуровневый доступ к файлам ... 269
9.7. Резюме ... 272

Глава 10. Работа с базами данных ... 273
10.1. Основные операции с базой данных ... 273
10.2. Обработка результирующих строк ... 277
10.3. Подготавливаемые выражения ... 279
10.3.1. Подготовка ... 279
10.3.2. Выполнение ... 280
10.3.3. Финализация ... 280
10.4. Пользовательские функции ... 282
10.5. Хранение BLOB-значений ... 286
10.6. Получение BLOB-значений ... 289
10.7. Резюме ... 291

Глава 11. Обработка XML ... 292
11.1. XML и RSS ... 292
11.1.1. XML ... 292
11.1.2. RSS ... 294
11.2. Объектная модель документа ... 296
11.3. Простой интерфейс для XML ... 302
11.4. Приложение для чтения RSS ... 310
11.5. Резюме ... 313
Задачи ... 315

Глава 12. Информация о местоположении ... 317
12.1. Фреймворк Core Location ... 317
12.2. Простейшее приложение, предоставляющее информацию о местоположении ... 321
12.3. Google Maps API ... 326
12.4. Отслеживающее приложение с картами местности ... 331
12.5. Работа с почтовыми индексами ... 336
12.6. Резюме ... 339

Глава 13. Работа с устройствами **340**
13.1. Работа с акселерометром 340
13.2. Аудио 345
13.3. Видео 347
13.4. Информация об устройстве 347
13.5. Производство и просмотр снимков 348
13.5.1. Общий подход 349
13.5.2. Подробный пример 349
13.6. Резюме 357

Приложение А. Сохранение и восстановление состояния программы **358**

Приложение Б. Запуск внешних программ **361**

Ссылки и библиография **363**

Алфавитный указатель **364**

Введение

Добро пожаловать в программирование в iPhone SDK — введение в разработку мобильных приложений для iPhone и iPod Touch. В этой книге рассматриваются следующие темы:

- язык программирования Object-C;
- коллекции;
- Cocoa Touch;
- разработка многофункциональных мобильных пользовательских интерфейсов;
- основы анимации и Quartz 2D;
- шаблоны типа «модель-представление-контроллер» (Model-view-controller, MVC);
- табличное представление данных;
- управление файлами;
- синтаксический анализ XML-документов с использованием SAX и DOM;
- работа с Google Maps API;
- использование веб-сервисов RESTful;
- разработка многофункциональных программ, основанных на анализе местоположения;
- создание приложений баз данных, используя механизм SQLite;
- разработка мультимедиа-приложений.

Для кого предназначена книга

Основная читательская аудитория данной книги — это разработчики приложений, владеющие основами языка C и концепциями объектно-ориентированного программирования, такими как инкапсуляция и полиморфизм. Чтобы читать книгу, не нужно быть экспертом-программистом на C — достаточно базовых знаний о структурах, указателях и функциях. Рассматриваемые в книге общие темы, такие как базы данных и обработка XML, раскрыты из расчета, что у читателя минимальные знания по этой теме.

Что вам необходимо

Чтобы овладеть программированием в iPhone SDK, вам потребуется следующее:

- компьютер на платформе Intel с установленной операционной системой Mac OS X Leopard;
- iPhone SDK (доступен по адресу http://developer.apple.com/iphone);
- членство в программе iPhone Developer Program (необязательно); чтобы использовать ваше устройство для разработки, вы должны оплатить взнос за членство в iPhone Developer Program;
- исходные коды (приведенные в книге исходные коды приложений доступны по адресу http://code.google.com/p/iphone-sdk-programming-book-code-samples/downloads/list).

Структура книги

Глава 1. Описывает основные свойства языка Objective-C в среде Cocoa. Вы ознакомитесь с основными идеями в управлении классами в Objective-C и научитесь объявлять, определять и использовать новый класс. Вас также познакомят с основными классами и типами данных Cocoa. Еще вы узнаете об управлении памятью в iPhone OS, научитесь создавать и удалять новые объекты и изучите правила применения объектов из Cocoa-платформы или других платформ. Вы ознакомитесь с темой протоколов в Objective-C, научитесь использовать существующие протоколы и объявлять новые. Эта глава также описывает такие свойства языка, как атрибуты, категории и позиционирование. Здесь же представлены исключения и техники обработки ошибок. Еще вы ознакомитесь с концепцией программирования «ключ-значение» (key-value coding, KVC) и научитесь использовать многопоточность в вашем iPhone-приложении.

Глава 2. Посвящена коллекциям в Cocoa. В ней рассматриваются массивы, множества и словари. Вы узнаете о неизменяемых и изменяемых коллекциях, различных подходах, используемых для наполнения коллекций, а также некоторых методах сортировки.

Глава 3. В ней рассматриваются основные шаги при построении простейшего iPhone-приложения — описана основная структура простого iPhone-приложения и этапы разработки программы с применением Xcode.

Глава 4. Здесь разъясняются основные концепции видов. Вы узнаете о геометрии и иерархии видов, multitouch-интерфейсе, анимации и основах рисования в Quartz 2D.

Глава 5. Из нее вы узнаете о базовом классе для всех элементов управления — `UIControl` — и важном механизме «цель-действие» (target-action). Эта глава также представит некоторые важные графические элементы управления, которые можно использовать для построения интересных iPhone-приложений.

Глава 6. Описывает доступные контроллеры представлений, имеющиеся в iPhone SDK. Можно создать iPhone-приложение, не используя этих контроллеров, однако делать этого не стоит — контроллеры представлений упрощают приложение. Эта глава предлагает введение в использование контроллеров представлений и подробное описание применения контроллеров панелей закладок, контроллеров навигации и модальных контроллеров.

Глава 7. Познакомит вас с несколькими важными подклассами класса `UIView`. Вы узнаете об элементах выбора и их использовании для выбора каких-либо значений. Мы рассмотрим компоненты отображения прогресса, компоненты индикации деятельности и текстовые компоненты, применяемые для отображения многострочного текста. Вы узнаете, как использовать компоненты уведомлений для отображения уведомительных сообщений пользователю. Будут также рассмотрены диалоговые окна.

Глава 8. Эта глава шаг за шагом проведет вас через мир таблиц. Мы начнем с обзора основных концепций табличных представлений данных, затем рассмотрим простейшее приложение с таблицей и обязательные методы накопления и обработки данных взаимодействия пользователя с таблицей. Вы узнаете, как добавлять изображения в строки таблицы, и ознакомитесь с концепцией секций и приложением с таблицей, содержащей секции с собственными заголовками и подвалами. В этой главе также представлен метод редактирования таблиц и описана программа, позволяющая удалять строки в таблице. Мы обратимся к способам вставки новых строк в таблицу и обсудим приложение, дающее возможности просмотра отдельной записи данных и добавления в таблицу новых записей. Мы продолжим обсуждение режима редактирования и рассмотрим приложение с возможностью перестановки записей в таблице. Затем обсудим механизм отображения пользователю информации в иерархическом виде, а также приложение, использующее таблицу для представления трех уровней иерархии. На примерах займемся сгруппированными таблицами, а затем ознакомимся с основными идеями касательно индексированных таблиц.

Глава 9. Эта глава раскрывает тему управления файлами. Вы научитесь использовать высоко- и низкоуровневые методы сохранения и извлечения данных из файлов. Мы поговорим о домашнем каталоге приложения, а затем вы научитесь перечислять содержимое директории, используя высокоуровневые методы файлового менеджера NSFileManager. Вы узнаете больше о структуре домашнего каталога и местах возможного хранения файлов. Вы научитесь создавать и удалять каталоги, ознакомитесь с методами создания файлов и изучите атрибуты файлов и каталогов. В этой главе мы расскажем, как считывать и устанавливать атрибуты, а также использовать пакеты приложений и низкоуровневого доступа к файлам.

Глава 10. В ней рассматриваются основы механизма баз данных SQLite, доступного в iPhone SDK. SQLite является встроенной базой данных, то есть отдельно работающего сервера нет — весь механизм баз данных встроен в ваше приложение. Вы изучите основные выражения языка SQL, их реализацию с использованием вызовов функций SQLite и обработку результирующих множеств, генерируемых выражениями SQL. Мы также обратимся к темам предкомпилированных выражений, расширений SQLite API на примере пользовательских функций и хранения и считывания BLOB-полей из базы данных.

Глава 11. Здесь вы научитесь эффективному использованию XML в вашем iPhone-приложении. Глава развивает предшествующую тематику и описывает принципы работы iPhone-приложения для чтения RSS-ленты. Мы рассмотрим синтаксический анализ DOM и SAX, табличное приложение для чтения RSS-ленты, а в качестве итога будут описаны основные этапы, которые нужно пройти для эффективного использования XML в вашем iPhone-приложении.

Глава 12. В ней мы обратимся к теме определения местоположения — рассмотрим технологию Core Location и ее использование для построения приложений, определяющих местоположение, и обсудим простейшее приложение с определением местности. Затем вы ознакомитесь с геокодированием. Вы научитесь переводить почтовые адреса в географические, определять движение устройства и отражать эту информацию на картах и узнаете о способах соотношения почтовых индексов и географической информации.

Глава 13. В этой главе вы ознакомитесь с использованием некоторых внутренних устройств iPhone. Вы узнаете, как применять акселерометр, проигрывать небольшие звуковые и видеофайлы, получать техническую информацию об устройстве iPhone/iPod Touch и использовать встроенную камеру и фотобиблиотеку.

Приложение А. В нем описано использование списков свойств для сохранения и восстановления состояния приложения. Это создает видимость, что ваше приложение не завершается, когда пользователь нажимает кнопку Home (Домой).

Приложение Б. Здесь вы научитесь программно вызывать приложения iPhone из вашего собственного и публиковать службы, которые могут быть использованы другими приложениями iPhone.

Глава 1
Objective-C и Cocoa

Эта глава представляет основные свойства языка программирования Object-C в среде Cocoa.

Структура данной главы следующая.

В разд. 1.1 описывается основная концепция классов в Objective-C. Вы научитесь объявлять, определять и использовать новый класс, а также ознакомитесь с наиболее важными классами и типами данных Cocoa.

В разд. 1.2 вы изучите управление памятью в ОС iPhone. Вы научитесь создавать и удалять объекты, а также узнаете правила использования объектов платформы Cocoa и других платформ.

Разд. 1.3 раскрывает тему протоколов Objective-C. Вы научитесь применять существующие протоколы и объявлять новые.

Из разд. 1.4 вы узнаете об отличительной черте языка Objective-C, которая позволяет получать доступ к переменным экземпляра класса через точку (dot notation).

Концепция категорий — предмет изучения в разд. 1.5. Категории позволяют расширять существующие классы посредством добавления новых методов.

Позиционирование — это метод, незначительно отличающийся от категорий. Оно позволяет замещать класс одним из его потомков. Это обсуждается в разд. 1.6.

Исключения и обработка ошибок присущи любому современному языку программирования. Разд. 1.7 описывает эти обе техники и их использование.

В разд. 1.8 освещена концепция программирования «ключ-значение» (key-value coding, KVC). KVC — это важный и широко используемый в Cocoa метод. KVC позволяет получить непрямой доступ к свойствам объекта.

Затем вы научитесь использовать многопоточность в вашем iPhone-приложении (разд. 1.9). Cocoa упрощает применение многопоточности, и шаг за шагом вы узнаете, как выполнять задачу в фоновом режиме.

Итоги этой главы будут подведены в разд. 1.10.

Нам предстоит изучить многое, поэтому давайте начнем.

1.1. Классы

В объектно-ориентированных языках, таких как Java, объект инкапсулирует атрибуты и предоставляет методы. Эти методы могут использоваться окружающей средой (например, другими объектами), чтобы изменить состояние объекта или взаимодействовать с ним, чего можно достичь, не раскрывая окружающей среде настоящей реализации поведения объекта.

В Objective-C для создания нового класса его сначала нужно объявить, используя ключевое слово `interface`, а затем – определить посредством ключевого слова `implementation`. Объявление и определение обычно описываются в двух разных файлах. Объявление, как правило, содержится в файле с расширением `H` и названием, совпадающим с именем класса, а реализация (также имеющая то же название, что и класс) – в файле с расширением `M`. Части, отвечающие за объявление и определение, используют директивы компилятора. Директива компилятора – это инструкция компилятору Objective-C; она предваряется знаком `@`. Объявление помечается для компилятора директивой `@interface`, а определение класса – директивой `@implementation`.

1.1.1. Объявление класса

Чтобы объявить класс `MyClassName` в качестве наследника класса `MyParentName`, вы можете написать следующее:

```
@ interface   MyClassName : MyParentName
{
    объявление атрибутов
}
    объявление методов
@end
```

Здесь мы говорим компилятору, что объявляется новый тип класса `MyClassName`. Он является наследником класса `MyParentName`. Мы также приводим список всех переменных экземпляра класса внутри фигурных скобок. Методы объявляются после закрывающей фигурной скобки перед директивой компилятора `@end`.

Существует несколько важных аспектов объявления `@interface`.

1. Атрибуты, стоящие внутри фигурных скобок, являются переменными экземпляра класса. В процессе выполнения каждый класс имеет уникальный объект класса и ноль или более экземпляров. Каждый экземпляр (объект) `MyClassName` имеет собственные значения этих атрибутов. Уникальный объект класса доступа к этим переменным экземпляра не имеет.

2. Объявленные методы могут быть методами экземпляра либо класса. Метод экземпляра вызывается отправлением сообщения фактическому

экземпляру (объекту) класса. Метод класса не требует наличия экземпляра объекта. Вы вызываете метод класса, посылая сообщение уникальному объекту класса. В Objective-C каждый класс в ходе выполнения программы имеет только один объект. Метод экземпляра объявляется/определяется посредством префикса «-», а метод класса объявляется/определяется с помощью префикса «+».

Например, `-(Address *) getAddress;` является методом экземпляра, а `+(id)getANewInstance;` — методом класса.

3. Objective-C не поддерживает переменных класса, однако вы можете использовать уже знакомое вам ключевое слово `static` в файле реализации данного класса. Это позволит методам экземпляра (с префиксом «—» в определении) получать доступ к единственному значению переменной, общей для всех экземпляров данного класса. Если вы определите статическую переменную внутри метода, то этот метод будет единственным, имеющим доступ к этой переменной. Если вы поместите определение статической переменной снаружи реализации класса, то все методы будут иметь доступ к данной переменной.

1.1.2. Как использовать другие объявления

Как Cocoa-разработчику вам понадобится возможность использования классов, написанных другими разработчиками. К тому же, если объявление и определение ваших классов находятся в разных файлах, вам потребуется проинформировать компилятор о местонахождении объявления класса внутри файла реализации.

Если вы используете имя класса без доступа к его методам или переменным экземпляра, вы можете просто применить директиву `@class`. Это дает компилятору достаточно информации, чтобы успешно создать код. Обычно директива `@class` используется в объявлениях класса. Например, взгляните на следующее объявление:

```
@class Address;
@interface Person
{
    Address *address;
}
@end
```

Здесь есть объявление класса `Person`, которое использует класс `Address`. Компилятору нужно знать только, что тип `Address` является классом. Никаких деталей относительно реальных методов и атрибутов не требуется, пока мы используем только тип.

Если вы используете методы и/или атрибуты данного класса, вам потребуется указать компилятору местонахождение файла, содержащего объявление. Это можно сделать одним из следующих способов —

используя #include или #import. Эти варианты почти идентичны, за тем исключением, что #import подгружает нужный файл только однажды в процессе компиляции. Директива #import намного проще, полностью поддерживается Apple и создает потенциально меньше проблем. Отсюда вывод — используйте #import.

1.1.3. Определение класса

Чтобы фактически определить класс, вам нужно указать действительную реализацию методов класса/экземпляра, объявленных в части @interface. Чтобы определить класс, вам нужно написать следующее:

```
#import  "MyClassName.h"
@implementation MyClassName
    определения методов
@end
```

Обратите внимание, что нам потребовалось импортировать файл объявлений. Этот импорт позволяет опустить повторное описание как имени класса, так и переменных экземпляра — и то, и другое компилятор может логически отследить, поэтому в повторе нет необходимости.

1.1.4. Определение и вызов методов

В Java метод вызывается посредством использования его имени, за которым следуют левая и правая круглые скобки. Если методу требуются параметры, то их значения вставляются внутри скобок и разделяются запятыми. Например, если aPoint является Java-объектом, представляющим точку в 3D-пространстве, setLocation(float x, float y, float z) может представлять собой метод для изменения положения объекта данной точки. Выражение aPoint.setLocation(3, 6, 9) указывает объекту aPoint изменить координаты на (3, 6, 9). Проблема такой записи — читаемость. Если вы столкнетесь с подобным выражением, написанным другим программистом, вы не сможете с точностью сказать, что представляют собой данные значения. Вам нужно попасть в интерфейс данного класса и выяснить, что значит каждая позиция в списке параметров.

Objective-C является объектно-ориентированным языком. Он также предоставляет инкапсуляцию данных. Внешняя среда взаимодействует с объектом, посылая ему сообщения. Чтобы послать сообщение aMessage объекту aObject, вы должны использовать квадратные скобки и написать [aObject aMessage]. Сообщение состоит из двух частей — ключевых слов и параметров. Каждое сообщение имеет, по крайней мере, одно ключевое слово. Ключевое слово является идентификатором, за которым следует двоеточие.

Конкретизируем данные определения, написав вызов метода `setLocation` на Objective-C. В Objective-C вы запишете выражение следующего вида:

```
[aPoint setLocationX:3 andY:6 andZ:9];
```

Обратите внимание: вызов метода стал более читаемым. Взглянув на него, мы сразу поймем, что 6 используется для изменения координаты `y`. Это сообщение имеет три ключевых слова — `setLocationX:`, `andY:` и `andZ:`. Такое представление называется селектором. Селектор — это уникальное имя (в пределах класса) метода, используемого окружением для локализации кода, реализующего данный метод. Метод определен в интерфейсе следующим образом:

```
-(void)setLocationX: (float) x andY: (float) y andZ: (float) z;
```

Все выражение `[aPoint setLocationX:3 andY:6 andZ:9]` называется выражением сообщения. Если это выражение превращается в объект, то оно также может получать сообщения. Objective-C позволяет осуществлять вложенные вызовы сообщений. Например, вы можете написать следующее:

```
[[addressBook getEntryAtIndex:0] printYourself];
```

Сначала объекту `addressBook` посылается сообщение `getEntryAtIndex:0`. Метод, идентифицируемый селектором `getEntryAtIndex:`, возвращает объект. Затем этому объекту посылается сообщение `printYourself`.

Если у метода ноль параметров, при его вызове можно не использовать двоеточия. Такая запись может показаться сложной, но с течением времени она становится сама собой разумеющейся.

Методы в Objective-C всегда `public`. Здесь нет методов `private`. Переменные экземпляра по умолчанию `protected` — настройка, которая всегда будет хорошо работать.

1.1.5. Важные типы

Как уже упоминалось ранее, каждый класс в Cocoa-приложении имеет объект класса типа `singleton`. Тип этого объекта класса — `Class`. Нулевой указатель на класс имеет тип `Nill`. По существу, `Nill` — это `(Class)0`. Мы также знаем, что можно создать экземпляр класса. Экземпляр класса `A` объявляется так:

```
A *anObject;
```

Тем не менее в Cocoa определен тип, который может представлять собой случайный объект. Этот тип называется `id`. Если `anObject` не указывает

ни на какой объект, его значение равно nil. Значение nil — это то же, что и (id)0.

Тип SEL — предопределенный. Он представляет собой селектор. Чтобы получить SEL метода aMethod:, используйте директиву @selector следующим образом:

```
SEL  mySelector  =  @selector(aMethod:);
```

Если вы хотите, чтобы mySelector указывал на пустой селектор, присвойте ему значение NULL.

Вы также можете получить SEL метода из строкового представления его имени. Для этого существует функция NSSelectorfromString(), которая объявлена как

```
SEL NSSelectorFromString (
    NSString *aSelectorName
);
```

Функция NSSelectorFromName всегда вернет SEL по не-nil имени aSelectorName, даже если селектор не найден. Если селектора с именем aSelectorName не существует, будет зарегистрирован и возвращен новый селектор с таким именем. Если параметр aSelectorName равен nil или у функции возникли проблемы с памятью, то она вернет NULL (по существу, (SEL)0).

1.1.6. Важные классы Cocoa

Существует несколько наиболее важных классов Cocoa, которые вы часто будете использовать в своем iPhone-приложении. В этом подразделе мы обсудим только те, которые понадобятся нам в данной главе. Другие классы будут рассмотрены позже. Ниже приведены некоторые важные классы Cocoa.

- NSObject — базовый класс для всех классов Cocoa. Объект не является Cocoa-объектом, если он не является экземпляром класса NSObject или любого класса-наследника NSObject. Этот класс определяет методы окружения, необходимые для создания и удаления объектов.
- NSString — главный класс, представляющий строки в Cocoa. Используя его, вы можете хранить любой текст. Однако однажды сохранив значение в объект этого типа, вы не можете изменить его — этот тип класса значится как неизменяемый. Чтобы иметь возможность менять строковое значение (например, прибавлять к нему текст и т. д.), вам нужен изменяемый строковый класс NSMutableString. Вы можете создать строковую константу, используя знак @. Например, @"Piano" представляет собой экземпляр NSString.

- `NSArray` — экземпляры данного класса представляют собой объекты массивов Cocoa. Изменяемая версия этого класса — `NSMutableArray` (более подробную информацию по массивам в Cocoa вы найдете в разд. 2.1).
- `NSSet` — экземпляры данного класса представляют собой объекты множеств Cocoa. Изменяемая версия этого класса — `NSMutableSet` (для получения дополнительной информации по множествам в Cocoa обратитесь к разд. 2.2).

1.2. Управление памятью

Современные языки программирования используют «сборщики мусора». «Сборщик мусора» — это runtime-алгоритм, который сканирует созданные в вашей программе объекты и утилизирует те из них, с которыми вы потеряли связь. Например, если вы создали объект и сохранили его в переменную-указатель `ptr`, а позже установили `ptr` в `nil`, то созданный блок памяти, чей адрес был сохранен в `ptr`, в вашем коде более недоступен. «Сборщик мусора» по своему выбору вмешивается и высвобождает эту память за вас, и будущие запросы на выделение памяти могут ее использовать.

Cocoa под управлением ОС iPhone не имеет «сборщика мусора». Вместо этого Cocoa-приложения должны использовать управляемую память. Программы, исполняемые на этой платформе, должны сами очищать за собой память. Приложения iPhone выполняются в среде с ограниченной памятью, поэтому вы как разработчик должны уделять повышенное внимание вопросу использования памяти.

Из предыдущих подразделов вы узнали, что `NSObject` является корневым классом в среде программирования Cocoa. Кроме прочего, `NSObject` определяет методы управления памятью. Одним из наиболее важных методов класса `NSObject` является `alloc`. Когда вы посылаете сообщение `alloc` объекту класса, тот выделяет память для нового экземпляра объекта класса и обнуляет значения его атрибутов. Чтобы экземпляр класса начал принимать сообщения, обычно требуется вызвать метод `init`. Каждый класс на протяжении всей цепочки наследования явным или неявным образом реализует, по крайней мере, один метод `init`. Если вы переопределяете `init`, то сначала вы должны вызвать метод `init` родительского класса, а затем — выполнить собственную инициализацию. Этого можно достигнуть путем использования ключевого слова `super`. Слово `super` ищет метод (в данном случае — `init`) начиная с родительского класса, тогда как переменная `self` производит поиск начиная с класса, где выполняется это выражение. Изменить значение переменной `self` в процессе выполнения программы вы можете, а значение `super` — нет. Чтобы изменить `super`, вы должны внести изменения в интерфейс родительского класса, а затем скомпилировать ваш код.

Мы рассмотрели создание нового объекта, а как от него можно избавиться? Решение проблемы простое — хранить счетчик объектов. Он сообщает, сколько других объектов используют текущий объект. Когда другому объекту потребуется ваш объект, он инкрементирует счетчик. Когда ваш объект уже не нужен, счетчик декрементируется.

Этот счетчик реализуется классом `NSObject` и называется счетчиком захватов. Когда вы создаете объект, счетчик захватов становится равным `1`. В любой момент, когда другой объект собирается использовать ваш объект, он посылает сообщение на удержание, таким образом увеличивая счетчик на 1. Если объект хочет освободить ваш объект, он посылает сообщение об освобождении, в результате — уменьшение на 1 счетчика захватов. Когда значение счетчика захватов становится равным `0`, объект уничтожается системой.

Чтобы предотвратить утечки памяти в программе, вы должны высвобождать любой объект, за память которого вы отвечаете, если он больше не нужен. В основном вы ответственны за память объекта в следующих трех случаях.

1. Вы создали объект, используя метод `alloc`. Если вы создали объект, в конце вы должны уничтожить его.

2. Объект является копией, сделанной вами. Если вы создали объект копированием из другого объекта, вы ответственны за его удаление в конце.

3. Объект был вами удержан. Если в ваших интересах поддерживать этот объект существующим, то в ваших же интересах уничтожить его, когда он станет ненужным.

Осталась одна проблема, которую можно проиллюстрировать следующим кодом:

```
//  В одном из ваших методов
//  Запросите объект создать для вас новый объект
NSMutableString  *aString  =  [anObject  giveMeANewString];
```

В этом коде вы просите `anObject` создать для вас принципиально новый объект типа `NSMutableString`. Возникает вопрос, кто будет следить за высвобождением нового объекта, когда он будет не нужен.

Первое решение проблемы — отложить удаление объекта. В принципе, метод `giveMeANewString` создает новый объект `NSMutableString` (например, используя `alloc`) и сохраняет его в пуле объектов для последующего высвобождения. Когда приходит время и вы хотите высвободить немного памяти, вы высвобождаете весь пул, и он пробежит по всем объектам и пошлет сообщение о высвобождении каждому содержащемуся в нем объекту.

Платформа `Foundation` для обработки отложенных высвобождений предоставляет класс `NSAutoReleasePool`. Каждый поток в вашей программе должен содержать по крайней мере один экземпляр данного класса.

Вы создаете экземпляр класса следующим образом:

```
NSAutoreleasePool *pool = [[NSAutoReleasePool alloc] init];
```

Эта строка делает `pool` активным самовысвобождающимся пулом для нижеследующего кода. В любое время, когда ваш код либо любая функция платформы Cocoa посылает объекту сообщение о самовысвобождении, ссылка на этот объект добавляется в пул. Можете представлять себе пул как список записей. Каждая запись содержит ссылку на объект (например, его адрес) и целочисленное значение сообщений о высвобождении, посланных этому объекту в данном пуле.

Чтобы очистить весь пул, вы посылаете ему сообщение о высвобождении, например `[pool release]`. После этого пул пошлет сообщение о высвобождении каждому объекту, на который хранятся ссылки. Количество уведомлений о высвобождении равно количеству сообщений о самовысвобождении, посланных данному объекту в пуле (переменная, хранящаяся в записи об объекте в этом пуле). Вследствие этого счетчик удержаний каждого объекта уменьшится на соответствующее количество отложенных высвобождений, которые он получил.

Самовысвобождающиеся пулы могут быть вложенными. Когда вы создаете пул, он помещается в стек. В любой момент, когда объект получает сообщение о самовысвобождении, окружение переместит ссылку на этот объект в пул, находящийся на вершине стека.

Использование вложенных пулов позволяет разработчику оптимизировать использование памяти. В качестве примера представим, что `giveMeANewString` создает большое количество временных объектов, чтобы вычислить возвращаемое значение. В конце исполнения этого метода такие временные объекты становятся ненужными. Если в вашем приложении только один самовысвобождающийся пул, эти объекты останутся до конца текущего вычислительного цикла и только потом высвободятся.

Чтобы иметь возможность возвращать память, занятую этими временными объектами, вы можете создать новый самовысвобождающийся пул в начале данного метода. Все самовысвобождающиеся объекты, сгенерированные кодом этого метода (или вызовы, сделанные этим методом), поместятся в этот новый пул, так как он находится на вершине стека пулов. Перед окончанием работы метода вы высвобождаете локальный пул, таким образом заставляя все временные объекты также высвободиться.

Листинг 1.1 показывает, как может быть реализован метод `giveMeANewString`. Здесь мы допустим, что создание строки является достаточно сложным процессом, требующим большого объема памяти для временных объектов.

Листинг 1.1. Демонстрация локальных самовысвобождающихся пулов

```
-(NSMutableString*) giveMeANewString
{
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
    NSMutableString *returnString = [[NSMutableString alloc] init];
    // код, который генерирует большое количество
    // самовысвобождающихся объектов
    .
    .
    .
    // дополним returnString вычисленными данными
    [returnString appendString : computedData];
    [pool release];
    return [returnString autorelease];
}
```

Все временные объекты, созданные в теле метода, будут уничтожены. Значение, возвращенное методом, является самовысвобождающимся и будет доступно вызывающему, пока не высвободится его собственный пул.

Если ваш класс использует или создает другие объекты, вам нужно быть уверенным, что эти объекты высвободятся одновременно с вашим экземпляром. Для этой цели предназначен метод `dealloc`. Он вызывается перед высвобождением экземпляра объекта. Вы должны высвободить любой объект, созданный вами, и распространить высвобождение вверх по цепочке наследования, вызывая `[super dealloc]` в качестве последнего выражения в вашем методе.

1.3. Протоколы

Протоколы — важное отличительное свойство Objective-C. Помимо прочего, они предоставляют возможность реализовать множественное наследование в языке с одинарным наследованием.

Можете думать о протоколе как об интерфейсе в языке Java. Как классы в Java могут реализовывать множественные интерфейсы, так и классы в Cocoa могут принимать множественные протоколы.

Протокол — это всего лишь список методов. Каждый метод этого списка может быть помечен как обязательный (`@required` по умолчанию), так и как необязательный (`@optional`). Если класс принимает протокол, он должен реализовать, по крайней мере, все обязательные методы данного протокола. Например, чтобы определить протокол `Litigating`, вы должны написать приблизительно следующее:

```
@protocol Litigating
-(int)sue:(id<Litigating>)someone;
-(int)getSuedBy:(id<Litigating>)someone;
@end
```

Любой класс может принять этот протокол, просто приведя его с использованием угловых скобок в своем объявлении после имени суперкласса, например:

```
@interface  Citizen: Human <Litigating>
```

Класс может принимать несколько протоколов. Например, можно написать так[1]:

```
@interface  Citizen: Human <Litigating ,  MilitaryService ,  TransferFunds>
```

Два класса с различными цепочками наследования могут реализовывать один и тот же протокол, например:

```
@interface  DemocraticCountry: Country<Litigating>
```

Демократическая страна принимает протокол `Litigating` («судебное дело»); она может преследоваться судом и может подавать в суд на других. Диктатура или фашистское государство такого протокола не принимает.

Представим, что вы хотите смоделировать систему, где граждане могут путешествовать в разные страны и потенциально могут предстать перед судом страны либо подать на страну в суд. Как проверить, можно ли подать в суд на страну? Нельзя же просто послать сообщение в страну в надежде на любезный ответ[2].

Objective-C предоставляет элегантное решение данной проблемы — протоколы. Вы можете на лету проверить, является ли объект экземпляром класса, который принимает данный протокол. Вы реализуете это, посылая объекту сообщение `conformsToProtocol:`. Например, чтобы проверить, можно ли подать в суд на страну `aCountry`, вы пишете следующее:

```
if ([aCountry  conformsToProtocol: @protocol(Litigating)]) {
     [aCountry  getSuedBy:self];
else
     [self  letItGo];
}
```

Метод `conformsToProtocol:` определен в `NSObject` дважды: первый раз — как метод класса (со знаком +), второй — как метод экземпляра (со знаком –). Они идентичны и существуют только для удобства. Например, версия класса определена следующим образом:

```
+ (BOOL)conformsToProtocol:(Protocol  *)aProtocol
```

Она принимает один аргумент — объект протокола. Мы можем использовать `@protocol(имя_протокола)`, чтобы получить экземпляр протокола `имя_протокола`. Этот экземпляр в большинстве своем является уникальным все время работы программы. Тем не менее будет безопаснее предположить, что это не так, и не кэшировать объект протокола.

---

[1] *В США даже не граждане обязаны служить в случае мобилизации.*

[2] *В реальной жизни это может быть очень опасно!*

Функция `conformsToProtocol:` возвращает `YES`, если класс объекта принимает имя_протокола, и `NO` — в противном случае. Важно отметить, что это соответствие определяется на основе объявления класса-приемника, а не на том, какие методы протокола были реализованы на самом деле. Например, если у вас есть

```
@interface  DemocraticCountry: Country<Litigating>
@end
@implementation DemocraticCountry
@end
```

а также

```
DemocraticCountry  *aCountry  =  [[DemocraticCountry  alloc]  init];
```

то выражение

```
[aCountry  conformsToProtocol: @protocol(Litigating)]
```

вернет `YES(1)`. Тем не менее выражение:

```
[aCountry  getSuedBy : self];
```

приведет к краху приложения.

Одному протоколу ничего не стоит включать в себя другие протоколы, например:

```
@protocol  LegalEntity  <Taxable>
```

Класс, который принимает протокол, должен реализовывать все обязательные методы этого протокола, как и обязательные методы протоколов, которые включает в себя текущий протокол (и т. д., рекурсивно). Более того, методы протокола наследуются подклассами, то есть подкласс соответствует данному протоколу, если его суперкласс также соответствует этому протоколу.

1.4. Свойства

Свойство — это изящная отличительная черта Objective-C, которая позволяет генерировать методы `setter/getter` для ваших переменных экземпляра. Эти методы могут быть вызваны даже без необходимости их реализовывать. Чтобы получить/установить значение переменной экземпляра, используйте обращение через точку. Например, если вы определили свойство `name` типа `NSString *` в объекте `aObject`, вы можете написать `aObject.name = @"Piano"`. На самом деле это выражение транслируется компилятору в нечто подобное: `[aObject setName: @"Piano"]`. Обратите внимание, что переменная экземпляра, `name`, до сих пор является скрытой, но кажется, что мы получаем доступ к этой переменной напрямую извне.

Чтобы объявить свойство в объявлении класса, вы используете директиву `@property`. Чтобы на самом деле сгенерировать методы `getter` и/или `setter`, используйте директиву `@synthesize` в определении класса (то есть в его реализации). Это свойство Objective-C позволяет запрашивать только методы `getter` для тех переменных экземпляра, которые находятся в режиме «только чтение».

Свойства объявляются в разделе методов (то есть после фигурных скобок) части `@interface` вашего класса. Формат объявления свойства следующий:

```
@property (атрибуты-свойства)  тип-свойства  имя-свойства;
```

Атрибуты свойства используются чтобы влиять на то, как компилятор будет генерировать методы `getter/setter`. Вы можете использовать следующие атрибуты:

- `nonatomic` — используя этот атрибут, вы сообщаете компилятору, что не нужно генерировать избыточный код для гарантии потоковой безопасности (thread-safety). Если вы не укажете `nonatomic`, компилятор сгенерирует дополнительный код. Если вы знаете, что доступ к данному свойству будет из одного потока, то, указывая `nonatomic`, вы сможете повысить производительность. Наличие атомарного метода доступа значит, что методы `getter/setter` являются потоково безопасными. Тем не менее это не обязательно означает, что ваш код в целом правильный. Наличие потоково безопасного кода требует от вас гораздо большей работы, чем гарантия атомарности одной операции — такой, как метод `getter` или `setter`. Более подробную информацию по многопоточности вы найдете в разд. 1.9;

- `readonly` — говорит компилятору, что свойство может быть считано, но не может быть задано. Компилятор сгенерирует только метод `getter`. Если вы попытаетесь написать код, который будет задавать свойству значение с помощью обращения через точку (dot-notation), компилятор сгенерирует предупреждение;

- `readwrite` — это значение по умолчанию. Если вы используете директиву `@synthesize`, компилятор сгенерирует для вас как метод `getter`, так и `setter`;

- `assign` — значение, используемое вами для задания свойства, будет напрямую задано переменной экземпляра. Это — значение по умолчанию;

- `copy` — вы используете этот атрибут, когда хотите сохранить копию объекта, связанную с вашей переменной экземпляра, а не ссылку на этот объект. Задаваемое значение должно быть объектом, который знает, как копировать себя (то есть реализовывать протокол NSCopying);

- `retain` — задается, когда вы заинтересованы в монопольном владении данным объектом. Компилятор вызовет захват этого объекта и назначит его переменной экземпляра. Если вызывающий позднее освободил этот объект, он не будет уничтожен, пока вы им владеете. Вы сами должны удалить его, когда закончите с ним работу либо вызвав метод `dealloc`;
- `getter=имяМетода`, `setter=имяМетода` — по умолчанию именем автоматически сгенерированного метода `setter` для свойства `prob` является `setProb`, а метода `getter` — `prob`. Вы можете изменять этот принцип именования как одного отдельного метода, так и обоих сразу.

После объявления свойства у вас есть выбор — попросить компилятор сгенерировать методы `getter/setter`, используя директиву `@synthesize`, или реализовать эти методы самому, используя директиву `@dynamic`.

В качестве примера рассмотрим класс `Employee`, объявленный и определенный в листинге 1.2.

Листинг 1.2. Объявление и определение класса Employee, демонстрирующее свойства в Objective-C

```
@interface  Employee  :  NSObject
{
    NSString   *name;
    NSString   *address;
    NSMutableArray   *achievements;
    BOOL       married;
    Employee   * manager;
    NSString   *_disability;
}
@property   (nonatomic,  copy)     NSString*   name;
@property   (nonatomic,  retain)   Employee*   manager;
@property   (nonatomic,  assign)   NSString*   address;
@property   (nonatomic,  copy)     NSMutableArray*   achievements;
@property   (nonatomic,  getter = isMarried) BOOL married;
@property   (nonatomic,  copy)     NSString*   disability;
@end

@implementation  Employee
@synthesize  name, address, manager, achievements, married,
             disability = _disability;
@end
```

Первое объявление свойства

```
@property   (nonatomic,   copy)   NSString* name
```

может быть представлено компилятором в следующем виде:

```
-(NSString*)  name
{
    return  name;
```

```
}
-(void)  setName : (NSString*)  aName
{
    if (name  !=  aName)
    {
       [name  release];
       name  =  [aName  copy];
    }
}
```

Метод `getter` возвращает ссылку на переменную экземпляра `name`. Метод `setter` сначала проверяет, не является ли новое значение переменной `name` таким же, что и текущее значение. Если они различаются, то старый объект освобождается и копия (как гласит директива `@property`) объекта `aName` создается и сохраняется в переменной экземпляра `name`. Обратите внимание, что в методе `dealloc` нужно освободить `name`.

Второе объявление свойства

```
@property (nonatomic, retain) Employee *manager
```

может быть интерпретировано компилятором в следующем виде:

```
-(Employee*)  manager
{
    return  manager;
}

-(void)  setManager : (Employee*)  theManager
{
    if(manager != theManager){
      [manager release];
       manager = [theManager retain];
    }
}
```

Метод `setter` сначала проверяет, не является ли `theManager` тем же самым, что и переменная экземпляра `manager`. Если это разные объекты, старый объект `manager` высвобождается и переменная экземпляра `manager` устанавливается в захваченный `theManager`. Обратите внимание, что требуется освободить `manager` в методе `dealloc`.

Третье объявление свойства

```
@property  (nonatomic,  assign)  NSString*  address
```

может быть представлено компилятором в следующем виде:

```
-(NSString*)  address
{
    return  address;
}

-(void)  setAddress : (NSString *)anAddress
{
    address  =  anAddress;
}
```

Обратите внимание, что, так как директивой свойства является assign, метод setter просто сохраняет адрес anAddress в переменной экземпляра.

Четвертым объявлением свойства является

```
@property  (nonatomic,  copy)  NSMutableArray*  achievements
```

При работе с изменяемыми коллекциями, такими как NSMutableArray, методы setter/getter, сгенерированные компилятором, могут оказаться неподходящими. Рассмотрим возможный синтез свойства achievements.

```
-(NSMutableArray*)  achievements {
    return  achievements;
}
-(void) setAchievements : (NSMutableArray*)  newAchievements {
    if(achievements != newAchievements) {
      [achievements  release];
      achievements = [newAchievements  copy];
    }
}
```

При таком подходе есть две проблемы.

Вызывающий метод getter получит ссылку на реальный массив achievments. Это означает, что вызывающий метод получит возможность модифицировать состояние экземпляра Employee. В некоторых случаях вы не захотите допустить этого.

Вы можете самостоятельно переписать метод getter следующим образом:

```
-(NSArray *)  achievements {
    return  achievements;
}
```

Однако это не решит проблемы, так как возвращаемая ссылка, несмотря на то, что она переопределена как неизменяемый массив, все еще является NSMutableArray и может изменяться вызывающим методом. Первое решение данной проблемы — вернуть самовысвобождающуюся копию коллекции следующим образом:

```
-(NSArray *)  achievements {
    return  [[achievements  copy]  autorelease];
}
```

В этом случае вызывающий метод получит неизменяемый массив. Заметьте, что, следуя соглашениям по управлению памятью, вызывающий метод не несет ответственности за высвобождение возвращаемого значения. Соответственно, мы автоматически высвободили ее перед возвращением. Если в данный момент это приводит вас в замешательство, рассмотрите эту проблему после прочтения гл. 2.

Синтезированный метод `setter` назначит неизменяемую копию переменной экземпляра, являющейся изменяемым массивом. Вы не сможете добавлять/удалять объекты из этого массива, поэтому вы должны написать метод `setter` самостоятельно. Возможной валидной реализацией является следующий код:

```
-(void)  setAchievements : (NSMutableArray*)
    newAchievements {
    if(achievements != newAchievements) {
      [achievements   release ];
      achievements  =  [newAchievements   mutableCopy];
    }
}
```

Обратите внимание на использование метода `mutableCopy` вместо `copy`. Для получения подробной информации по массивам и коллекциям обратитесь к гл. 2.

Пятое свойство

```
@property   (nonatomic, getter = isMarried) BOOL married
```

предписывает компилятору изменить имя метода `getter` на `isMarried` вместо принятого `married`. Далее следует возможная реализация свойства:

```
-(BOOL)  isMarried
{
    return  married;
}
-(void)  setMarried : (BOOL) newMarried
{
    married  =  newMarried;
}
```

Шестое свойство

```
@property   (nonatomic, copy)  NSString*  disability
```

имеет директиву синтеза `@synthesize disability=_disability`. Оно будет сгенерировано так же, как мы видели у первого свойства, за тем исключением, что мы говорим компилятору ассоциировать свойство `disability` с переменной экземпляра `_disability`.

Далее следует возможная генерация данного свойства:

```
-(NSString*)  disability
{
    return  _disability;
}
-(void)  setDisability : (NSString*)newDisability
{
    if(_disability != newDisability) {
      [_disability  release];
      _disability  =  [newDisability  copy];
    }
}
```

Вы узнали, как можно реализовать различные типы объявлений свойств. Конечно, в основном для генерации методов доступа вы будете полагаться на компилятор, а не писать их сами. Однако в особых случаях, таких как изменяемые коллекции, или в ситуациях, зависящих от потребностей вашего приложения, вы, возможно, захотите написать некоторые методы доступа самостоятельно.

1.5. Категории

Категория — это особенность Objective-C, позволяющая расширять возможности класса. Это свойство работает, даже если вы не имеете доступа к исходному коду расширяемого класса.

Когда вы расширяете какой-то класс с помощью категории, расширение наследуется всеми его подклассами. Конечно, добавленные методы, определенные в категории, видны только в пределах вашей программы.

Чтобы продемонстрировать это мощное свойство, в качестве примера расширим класс `NSObject`, добавив к нему метод экземпляра:

```
@interface NSObject(EnhancedObject)
-(NSComparisonResult) rankSelf : (NSObject*)anotherObject;
@end

@implementation NSObject(EnhancedObject)
-(NSComparisonResult) rankSelf:(NSObject*) anotherObject
{
    if ([self retainCount] > [anotherObject retainCount]) {
       return NSOrderedDescending;
    }
    else if ([self retainCount] < [anotherObject retainCount]) {
       return NSOrderedAscending;
    }
    else return NSOrderedSame;
}
@end
```

Чтобы объявить категорию к существующему классу, такому как `NSObject`, вам нужно добавить имя категории в круглых скобках после названия класса. Реальное определение категории осуществляется в той же форме. Вы определяете метод категории в секции методов так же, как определяете обычные методы.

Следующий пример иллюстрирует использование категории. Все объекты являются наследниками `NSObject`, поэтому все объекты в вашем приложении получат возможность определять свой ранг.

```
NSMutableString *string =
  [[NSMutableString alloc] initWithString:@"string"];
Employee *empl = [[Employee alloc] init];
[empl retain ];
NSComparisonResult result = [empl rankSelf: string];
```

Здесь мы просим объект `empl` типа `Employee` определить свой ранг по отношению к строковому объекту типа `NSMutableString`.

Классы Employee и NSMutableString не имеют определения метода rankSelf. Категория EnhancedObject, определенная для NSObject, являющегося родительским классом для обоих, тем не менее определяет подобный метод. Когда empl получает сообщение rankSelf:, это сообщение распространяется по всей цепочке наследования до NSObject.

Это свойство широко используется в Cocoa. Например, в файле UIStringDrawing.h определяется категория UIStringDrawing (листинг 1.3) для NSString, таким образом давая любому объекту NSString возможность отрисовывать самого себя.

Листинг 1.3. Пример категории Cocoa, определенной для NSString с целью отрисовки

```
@interface  NSString(UIStringDrawing)
-(CGSize)
drawAtPoint:(CGPoint) point
withFont:(UIFont *)font;
-(CGSize)
drawAtPoint:(CGPoint) point
forWidth:(CGFloat) width
withFont:(UIFont *) font
lineBreakMode:(UILineBreakMode)lineBreakMode;
.
.
.
@end
```

1.6. Позиционирование

Позиционирование — это отличительная черта Objective-C, позволяющая поменять местами класс A с классом B. Перестановка приведет к тому, что все активные экземпляры, являющиеся подклассами A, и будущие экземпляры A или его подклассы будут использовать B вместо A, поэтому после позиционирования все сообщения, посланные A, вместо этого будут посылаться B. Это требует, чтобы B был наследником класса A. Класс B может переопределять существующие методы и добавлять новые, но он не может добавлять новые переменные экземпляра.

В отличие от категорий (где один и тот же метод, определенный в категории, заменяет метод, определенный в классе) класс, осуществляющий позиционирование, может вызвать переопределенный метод, используя ключевое слово super. Обычно позиционирование используется в сценариях тестирования.

Позиционирование осуществляется посредством единственного вызова метода класса NSObject, определенного следующим образом:

```
+ (void) pose AsClass:(Class)aClass
```

Например:

```
[B poseAsClass: [A class]];
```

Это должно быть сделано в начале программы — до того, как будет создан какой-либо экземпляр A.

1.7. Исключения и ошибки

Как разработчик вы столкнетесь с тем, что даже самое простое ваше приложение когда-нибудь встретится с непредсказуемым событием, которое приведет к изменениям в обычном выполнении кода. Этим событием может быть простое деление на ноль, отправка неопределенного сообщения объекту или добавление элемента к неизменяемой коллекции. Вне зависимости от типа ошибки ваше приложение должно быть готово к ней, чтобы изящно обработать ее, когда та появится.

Cocoa делит неожиданные события на две категории — относящиеся к ошибкам разработчика и пользователя. Проблемы, возникающие по вине программиста, называются исключениями, а происходящие по вине пользователя — ошибками. В Cocoa исключения применяются в процессе разработки приложения, а ошибки — во время работы программы. Cocoa-платформы используют как исключения, так и ошибки, поэтому вы как Cocoa-разработчик должны овладеть обеими техниками.

1.7.1. Исключения

Современные языки программирования, такие как Java и C++, предоставляют конструкции для обработки исключений. Objective-C не исключение и тоже предоставляет подобные возможности. Чтобы отловить возможное исключение, вы должны окружить проблематичный код блоком `try`. Чтобы обработать фактическое исключение, используется блок `catch()`. Если вы хотите выполнить некоторые выражения вне зависимости от того, появится исключение или нет (например, при очистке памяти и т. д.), вы заключаете эти выражения в блок `finally()`.

Что такое исключение и как его опознать? Исключением может быть любой объект Cocoa. Тем не менее как Cocoa-разработчик вы должны использовать `NSException` или любой его подкласс. Исключение опознается, будучи брошенным или поднятым. Objective-C предоставляет директиву `@throw` для выброса исключения, а класс `NSException` определяет метод экземпляра `raise` для поднятия исключения. Используя директиву `@throw`, вы можете бросить любой объект Cocoa, а не только экземпляр `NSException`.

Используя метод raise, вы можете бросить только NSException. В остальном эти обе техники выполняют одно и то же действие.

Структура обработки исключений следует такому шаблону:

```
@try  {
      //выражения, которые могут вызвать исключение
}
@catch  (NSException  *e) {
    //выражения, обрабатывающие исключение
    @throw; //опционально, бросаем исключение еще раз
}
@finally {
    //выражения, которые должны быть выполнены в любом случае
}
```

В основном вы окружаете потенциально проблематичный код директивой @try. Чтобы действительно обработать исключение, вы используете блок @catch. Блок catch принимает объект исключения в качестве единственного параметра. Как было сказано ранее, объект исключения не обязан быть экземпляром NSException; любой Cocoa-объект может быть брошен/пойман. Например, код, где ловится экземпляр NSString, может быть следующим:

```
@catch(NSString  *str) {
.
.
.
}
```

Тем не менее, как уже упоминалось, вы должны оперировать NSException или любыми его подклассами.

Вы можете использовать блок finally, чтобы поместить туда любой код, который необходимо выполнить вне зависимости от появления исключения. Этот код обычно освобождает память и закрывает открытые файлы.

Вы можете также заново бросить исключение на следующий уровень стека вызовов. Для этого используйте директиву @throw. Можете не указывать бросаемый объект исключения, но тем не менее его наличие предполагается.

Рассмотрим вышеописанное на конкретном примере:

```
#import <Foundation/Foundation.h>
int  main(int  argc ,  char  *argv[])  {
    NSAutoreleasePool * pool = [[NSAutoreleasePool  alloc]  init];
    NSMutableArray *myArray = [[NSMutableArray  alloc]  initWithCapacity :0];
    [my Array  addObject:@"an object"];
    [my Array  replaceObjectAtIndex : 1 withObject:@"another object"];
    [myArray release];
    [pool release];
    return 0;
}
```

Приведенный выше код создает массив, затем добавляет к нему элемент, а потом пытается заменить этот элемент другим объектом. Если мы попытаемся запустить этот код, то получим исключение, после чего программа завершится со следующим сообщением об ошибке:

```
Exception Type:  EXC_BREAKPOINT (SIGTRAP)
Exception Codes: 0x0000000000000002, 0x0000000000000000
Crashed Thread:  0

Application Specific Information:
*** Terminating app due to uncaught exception 'NSRangeException',
reason: '*** -[NSCFArray replaceObjectAtIndex:withObject:]:
index (1) beyond bounds (1)'
```

То, что произошло, является результатом нашего использования некорректного индекса (1) массива размером 1. Метод `replaceObjectAtIndex:withObject:` поднял исключение, встретив этот некорректный индекс. Сам метод определен как

```
-(void) replaceObjectAtIndex:(NSUInteger) index
   withObject:(id)anObject
```

Посмотрев документацию по этому методу, вы заметите, что потенциально метод может поднять два исключения: он поднимает `NSRangeException`, если индекс превышает границы получателя, и поднимает `NSInvalidArgumentException`, если `anObject` равен `nil`. Перепишем функцию `main`, добавив обработку исключений.

```
int
main(int  argc ,  char  *argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool  alloc]  init];
    NSMutableArray  *myArray = nil;
    @try {
      myArray = [[NSMutableArray  alloc]  initWithCapacity:0];
      [myArray  addObject:@"an object"];
      [myArray  replaceObjectAtIndex:1
            withObject:@"another object"];
    }
    @catch  (NSException  *e) {
      printf("Exception Name: %s. Reason: %s",
            [[e  name] cString],
            [[e  reason] cString]);
    }
    @finally {
      [my Array  release ];
      [pool  release ];
    }
    return  0;
}
```

Мы окружили проблематичный код блоком `try`. Поймав исключение, мы просто печатаем сообщение об ошибке. Блок `finally` важен, так как мы должны освободить занятую память. Вместо того чтобы прерваться,

приложение выводит следующее полезное сообщение, но, главное, оно изящно завершает работу.

```
Exception Name: NSRangeException.
Reason: *** -[NSCFArray replaceObjectAtIndex:withObject:]:
index (1) beyond bounds (1)
```

Каждый экземпляр NSException несет три полезные порции информации.

1. name — строка, идентифицирующая исключение. Это имя должно быть уникальным, относительно коротким и никогда не равным nil. Вы никогда не должны называть свои исключения именами, начинающимися с NS. Названия должны начинаться с чего-либо уникального, присущего вашему приложению или организации.

2. reason — также обязательный строковый атрибут. Он содержит понятное объяснение исключения.

3. userInfo — этот атрибут является пользовательским словарем (разд. 2.3). Используя этот словарь, код, который создает исключение, может общаться с обработчиком исключения.

Если вы пишете метод и хотели бы общаться с вызывающим посредством исключений, вы должны иметь возможность создавать исключения и затем бросать их. Чтобы создать исключение, вы можете использовать метод класса NSException exceptionWithName:reason:userInfo, объявленный следующим образом:

```
+ (NSException  *)exceptionWithName:(NSString  *)name
            reason:(NSString  *)reason
            userInfo:(NSDictionary  *)userInfo
```

Этот метод возвращает самовысвобождающийся объект NSException с заданными атрибутами. Он возвращает nil, если создать подобное исключение невозможно.

Вот пример для создания и выброса исключения:

```
-(void)  myMethod:(NSString *)  string {
  if (string == nil) {
    NSException *anException =
      [NSException
        exception WithName:@"MNSInvalidArgument"
        reason:@"Argument is nil"
        userInfo:nil];
        @throw  anException; //  или  [anException raise];
  }
  else {
    // продолжаем, все в порядке
  }
}
```

Вложенные исключения. Обработчик исключений может по своему усмотрению заново бросать существующее или вновь созданное

исключение, поэтому исключения могут быть вложенными. Рассмотрим сценарий, где у вас есть метод, добавляющий запись в базу данных. Этот метод вызывает другой высокоуровневый метод, который производит физическое добавление записи в файл. Следующие два листинга показывают методы `addRecord:` и `insertRecord:`.

```
-(void) addRecord:(Record*) record {
  @try {
    [file insertRecord:record];
  }
  @catch (NSException * e) {
    // создаем новое исключение, db,
    // name=MYDBException
    @throw db;
  }
  @finally {
    // закрываем файлы и т. п.
    // освобождаем память
  }
}
-(void) insertRecord:(Record*) record {
    @try {
      // открываем файл
      // устанавливаем указатель
      // вставляем запись
    }
    @catch (NSException * e) {
      // локально обрабатываем исключение
      @ throw;
    }
    @finally {
      // закрываем файл
      // освобождаем память
    }
}
```

Здесь мы видим вложенные исключения. Если исключение возникает при доступе к файлу, оно отлавливается в методе `insertRecord:`, обрабатывается локально и заново бросается. В методе `addRecord:` имеется обработчик исключений, который ловит это исключение. Он создает новое исключение, называемое `MYDBException`, и бросает его. Последнее исключение общается с вызывающим в методе `addRecord:`. В случае сбоя тот, кто вызывает метод `addRecord:`, видит более информативное исключение базы данных, чем низкоуровневое исключение при доступе к файлу. Следует отметить, что уровни вложенности могут быть произвольными.

1.7.2. Ошибки

В качестве С-программиста вы, должно быть, использовали коды ошибок, чтобы доставить ошибки вызывающему. Этот подход достаточно ограничен, так как вы можете доставить только один элемент информации (номер) вызывающему.

В качестве основного механизма для доставки текущих ошибок пользователю Cocoa использует объекты типа `NSError` (или его подклассы).

В доставке ошибок Cocoa придерживается следующего шаблона.

1. Значение, возвращаемое методом, используется для обозначения сбоя или успешного выполнения. Если возвращается значение типа `BOOL`, то `NO` указывает на ошибку. С другой стороны, если возвращается значение типа `id`, то `nil` указывает на ошибку.

2. В качестве второстепенного механизма для детальной обработки ошибки пользователь может передать указатель на объект `NSError` в качестве последнего параметра метода. Если этот параметр не `NULL`, метод сохраняет в нем новый самовысвобождающийся объект `NSError`, используя этот указатель.

Объект `NSError` хранит три важных атрибута.

1. `domain` — строка, представляющая собой домен ошибки. Разные платформы, библиотеки и даже классы имеют различные домены ошибок. Примерами доменов ошибок могут быть `NSPOSIXErrorDomain` и `NSCocoaErrorDomain`. Приложения могут и должны создавать собственные домены ошибок. Если вы создаете свой домен ошибок, убедитесь в его уникальности, предваряя имя домена названием приложения и вашей организации.

2. `code` — целочисленный код ошибки, который имеет значение внутри домена. Два объекта `NSError` с одинаковым кодом ошибки, но разными доменами отличаются друг от друга.

3. `userInfo` — словарь (разд. 2.3), содержащий объекты, имеющие отношение к ошибке.

Рассмотрим обработку ошибок в Cocoa. Следующий пример намеренно вызывает ошибку. Он обрабатывает ее, отображая три вышеописанных атрибута ошибки.

```
NSError *myError = nil;
NSURL *myUrl =
    [NSURL  URLWithString:@"http://fox.gov"];
NSString *str = [NSString
            stringWithContentsOfURL:myUrl
            encoding:NSUTF8StringEncoding
            error:&myError];
if(str == nil) {
    printf("Domain: %s. Code: %d \n",
      [[myError  domain] cString],
      [myError  code]);
    NSDictionary *dic = [myError userInfo];
    printf("Dictionary: %s\n", [[dic  description] cString]);
}
```

На данном этапе вам необязательно знать о загрузке через URL, так как позднее мы подробно рассмотрим этот вопрос. Все, что вам нужно знать, —

это то, что мы вызываем метод Cocoa с целью получить новый объект (в данном случае этим объектом является интернет-страница в качестве объекта NSString). Метод возвращает nil в случае ошибки, а также дает пользователю возможность задать указатель на объект NSError для более подробной информации об ошибке. Мы передаем указатель на объект NSError и вызываем метод. Сайт http://fox.gov не существует, поэтому метод возвращает значение nil, создается объект NSError и самовысвобождается методом Cocoa.

На выходе этого фрагмента кода будет домен ошибки, код и содержимое словаря.

```
Domain: NSCocoaErrorDomain. Code: 260
Dictionary: {
    NSURL = http://fox.gov;
    NSUnderlyingError = Error Domain=NSURLErrorDomain
      Code=-1003
      UserInfo=0x4183a0 "can\325t find host";
}
```

Домен ошибки — NSCocoaErrorDomain с кодом 260. Словарь userInfo содержит две записи — NSURL со значением http://fox.gov и NSUnderlyingError со значением Error Domain=NSURLErrorDomain Code=-1003 UserInfo=0x4183a0 "can't find host".

Создание экземпляра NSError. Мы узнали, как можно обрабатывать объекты ошибок, созданные для нас. Однако часто нам необходимо писать методы для наших клиентов, которые возвращают самовысвобождающиеся объекты ошибок. Чтобы создать объект NSError, вы можете использовать один из нескольких доступных методов класса/экземпляра. Например, метод класса errorWithDomain:code:userInfo возвращает самовысвобождающийся объект ошибки. Другой метод получения объекта ошибки — это создать его, используя alloc, и инициализировать с помощью initWithDomain:code:userInfo. Например, представив, что последним аргументом вашего метода является error типа NSError**, следующий код создает для вызывающего новый объект NSError.

```
*error = [[NSError alloc]
    initWithDomain:CompanyCustomDomain
    code:12  userInfo:dictionary];
```

1.8. Кодирование «ключ-значение»

До настоящего момента вы видели два способа организации доступа к переменным экземпляра объекта — используя методы доступа либо напрямую. Cocoa определяет третий путь, позволяющий получать косвен-

ный доступ к переменным экземпляра класса. Эта техника называется кодированием «ключ-значение» (key-value coding, KVC).

KVC объявлено в протоколе `NSKeyValueCoding`. Этот протокол реализуется корневым объектом всех объектов Cocoa — классом `NSObject`. В глубине этого протокола находятся два метода, которые вы будете использовать, — `setValue:forKey:` (устанавливает значение данного ключа) и `valueForKey:` (возвращает значение данного ключа).

Метод `valueForKey:` объявлен в протоколе как

```
-(id) valueForKey:(NSString *)key
```

где `key` — строка ASCII-формата, которая начинается со строчной буквы и не содержит пробелов.

Метод `setValue:forKey:` объявлен в протоколе как

```
-(void) setValue:(id)value forKey:(NSString *)key
```

где `value` — это объект Cocoa (например, подкласс `NSObject`), а `key` — экземпляр `NSString` с идентичными описанным выше ограничениями.

Для корректной работы KVC вы должны следовать некоторым соглашениям Cocoa для именования методов доступа. Если есть ключ `xyz`, то в вашем классе должен быть определен метод `xyz` или `isXyz`, чтобы вы могли использовать метод `valueForKey:`. Аналогичным образом, чтобы использовать метод `setValue:forKey:`, в вашем классе должен быть определен метод `setter setXyz`.

Несколько ключей могут быть разделены точкой, формируя так называемую цепочку ключей. Цепочка ключей определяет последовательность прохождения свойств объекта. Например, цепочка `key1.key2.key3` сообщает следующее: извлечь из получателя объект, заданный ключом `key1`, затем извлечь объект, заданный ключом `key2`, из только что полученного объекта и, наконец, извлечь объект, заданный ключом `key3`, из последнего объекта, полученного с помощью ключа `key2`.

Пример, иллюстрирующий KVC. Рассмотрим KVC на примере. Возьмем класс `Person`, объявленный и определенный следующим образом:

```
@interface  Person:  NSObject {
    NSString  *name;
    NSArray   *allies;
    Person    *lover;
}
@property  NSString  *name;
@property  NSArray   * allies;
@property  Person  *lover;
```

```
-(id) initWithName:(NSString *) theName;
@end

@implementation Person
@synthesize name, allies, lover;
-(id)initWithName:(NSString*) theName {
    if(self = [super init]) {
      name = theName;
    }
    return self;
}
@end
```

Дополнительно рассмотрим класс `Community`, объявленный и определенный так:

```
@interface Community : NSObject
{
    NSArray *population;
}
@property NSArray *population;
@end

@implementation Community
@synthesize population;
@end
```

Перед изучением примера обратите внимание, как реализован метод `initWithName:`. Во-первых, посмотрите, как мы вызвали метод `init` класса `super` и использовали результат как значение переменной `self`. Во-вторых, посмотрите, как мы напрямую задали значение переменной `name`. Причина этих действий не имеет ничего общего с нашим примером KVC. Это просто иллюстрация того, что, если при задании значения вы хотите использовать синтезированный метод `setter`, вы должны присвоить свойству `name` объекта следующее значение:

```
self.name = theName
```

Метод `setter` имеет директиву `assign` (по умолчанию), поэтому мы просто пропускаем его и присваиваем значение переменной экземпляра напрямую. Если вы не используете метод `setter`, вы скорее просто присвоите значение, чем будете вызывать метод `setter`. Будьте осторожны, реализуя метод `setter` самостоятельно. Если в вашем методе вы зададите значение переменной, используя `self`, это приведет к бесконечному циклу, переполнению стека и краху приложения.

Используем вышеупомянутые классы, чтобы привести в действие механизм KVC. Листинг 1.4 содержит функцию `main`, демонстрирующую KVC. Сначала создадим и инициализируем семь экземпляров `Person` и один экземпляр `Community`. Затем используем KVC, чтобы задать массив `allies`. После этого применим KVC, чтобы установить атрибут `lover`. Затем зададим экземпляр `Community`, представляющий, например, сообщество сериала «Остаться в живых» массивом, содержащим семь экземпляров `Person`.

Затем мы бы хотели использовать KVC для извлечения значений, используя ключи и цепочки ключей. Строка

```
[lost valueForKeyPath:@"population"];
```

извлекает массив `population` объекта `lost`. Ключ `population`, примененный к экземпляру `lost`, производит массив экземпляров `Person`, возвращаемый вызывающему. На рис. 1.1 результат представлен графически.

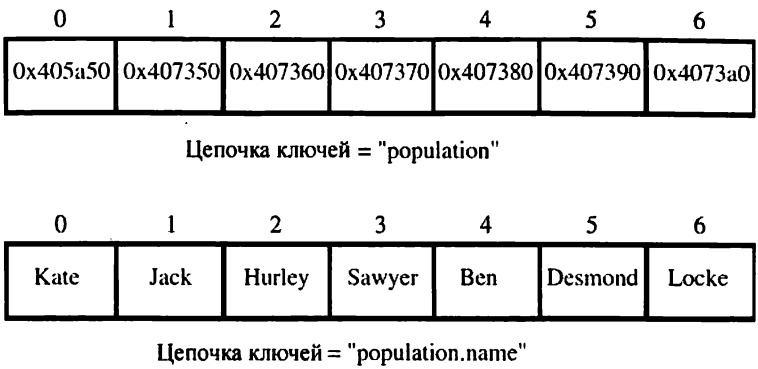

Рис. 1.1. Использование ключей и цепочек ключей для извлечения массива population и массива имен из population из экземпляра lost

Далее строка

```
[lost valueForKeyPath:@"population.name"];
```

извлекает массив имен из переменной `population` экземпляра `lost`. Это пример, иллюстрирующий цепочки ключей. Сначала ключ `population` применяется к получателю `lost`. Это произведет массив экземпляров `Person`. Затем ключ `name` применяется к каждой записи этого массива, результатом чего станет экземпляр `NSString`. Массив экземпляров `NSString` будет возвращен в качестве результата (графическое представление результата — см. рис. 1.1).

Интерес представляет строка

```
[lost valueForKeyPath:@"population.allies"];
```

Рассмотрим ее подробнее, чтобы понять, к какому результату она приведет. Сначала применяем ключ `population` к получателю `lost`. Это произведет массив экземпляров `Person`. Далее ключ `allies` применяется к каждому экземпляру `Person` в этом массиве. Это также произведет массив экземпляров `Person`. Теперь у нас есть массив массивов экземпляров

Person. Это и будет являться результатом и будет возвращено вызывающему. Графически результат представлен на рис. 1.2.

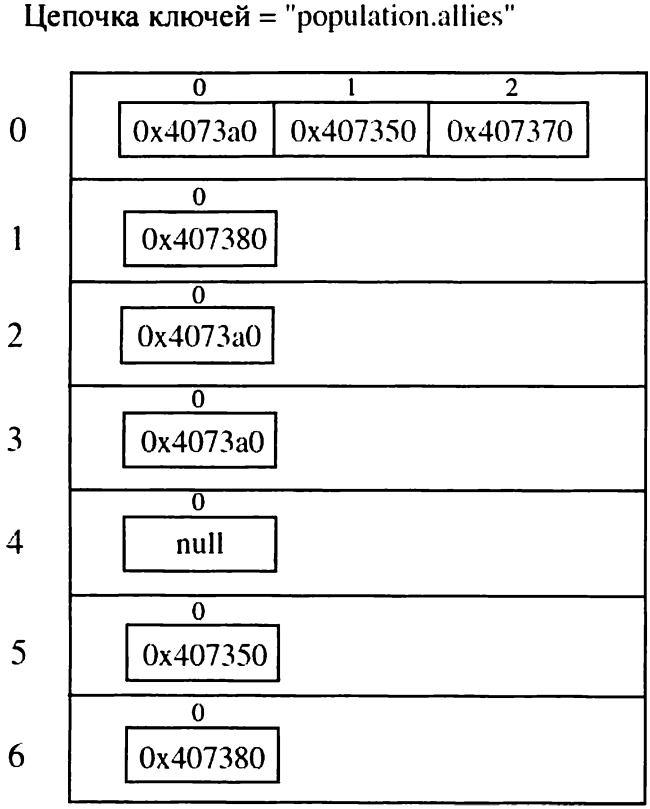

Рис. 1.2. Графическое представление результата, полученного от применения цепочки ключей population.allies к экземпляру lost

Строка

```
[lost valueForKeyPath:@"population.allies.allies"];
```

идет еще дальше. Цепочка подключей `population.allies` дает аналогичный описанному выше результат, но теперь мы применяем к результату другой ключ `allies`. Это произведет массив массивов массивов экземпляров Person, как показано на рис. 1.3.

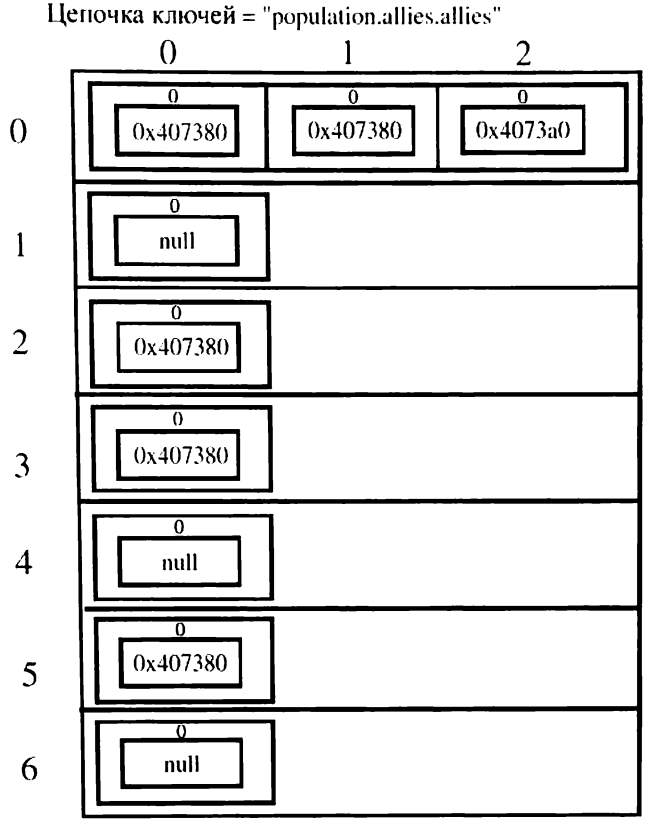

Рис. 1.3. Графическое представление результата применения цепочки ключей population.allies.allies к экземпляру lost

Строка

```
[lost valueForKeyPath:@"population.allies.allies.name"];
```

делает то же, что было описано выше, за тем исключением, что далее применяется ключ `name` к каждому экземпляру `Person` в массиве массивов массивов экземпляров `Person`.

Код

```
theArray =
    [lost valueForKeyPath:
            @"population.allies.name"];
```

```
NSMutableSet  *uniqueAllies =
        [NSMutableSet  setWithCapacity:5];
for(NSArray  *a  in  theArray){
  if(![a  isMemberOfClass:[NSNull  class]]){
    for(NSString  *n  in  a){
        printf ("%s ",  [n  cString]);
        [uniqueAllies  addObject:n];
    }
    printf("\n");
  }
}
```

демонстрирует структуру результата применения цепочки ключей `population.allies.allies.name`. Она перечисляет все имена и производит множество уникальных названий. Для более подробной информации по массивам и множествам обратитесь к гл. 2.

Следует остерегаться проблемы `nil`. Некоторые переменные экземпляров могут быть `nil`, а коллекции в Cocoa не могут иметь нулевых записей, поэтому Cocoa использует класс `NSNull` для представления нулевых записей. В приведенном выше коде мы проверяем, не является ли запись экземпляром `NSNull`. Если это так, мы просто пропускаем ее.

Некоторые могут прийти в замешательство от коллекций и цепочек ключей, думая, что результатом цепочки ключей всегда является экземпляр коллекции. Это неверно, так как эти две концепции ортогональны. Результатом выражения

```
NSString *luckyPerson =
    [jack  valueForKeyPath:@"lover.lover.lover.name"];
```

будет экземпляр `NSString` со значением `@"Hurley"`.

Листинг 1.4. Демонстрационный код кодирования «ключ-значение»

```
int  main(int  argc ,  char  *argv[]) {
    NSAutoreleasePool  *   pool  =
            [[NSAutoreleasePool  alloc]  init];
    Person  *kate =
            [[Person alloc]  initWithName:@"Kate"];
    Person  *jack=
            [[Person alloc]  initWithName:@"Jack"];
    Person  *hurley =
            [[Person alloc]  initWithName:@"Hurley"];
    Person  *sawyer=
            [[Person alloc]  initWithName:@"Sawyer"];
    Person  *ben =
            [[Person alloc]  initWithName:@"Ben"];
    Person  *desmond=
            [[Person alloc]  initWithName:@"Desmond"];
```

```
    Person *locke=
           [[Person alloc] initWithName:@"Locke"];
    Community *lost = [Community alloc] init];
    [kate setValue : [NSArray
       arrayWithObjects:locke, jack, sawyer, nil] forKey:@"allies"];
    [hurley setValue:[NSArray
       arrayWithObjects:locke, nil] forKey:@"allies"];
    [sawyer setValue:[NSArray
       arrayWithObjects:locke, nil] forKey:@"allies"];
    [desmond setValue:[NSArray
       arrayWithObjects:jack, nil] forKey:@"allies"];
    [locke setValue:[NSArray
       arrayWithObjects:ben, nil] forKey:@"allies"];
    [jack setValue:[NSArray
       arrayWithObjects:ben, nil] forKey:@"allies"];
    [jack setValue:kate forKey:@"lover"];
    [kate setValue:sawyer forKey:@"lover"];
    [sawyer setValue:hurley forKey:@"lover"];
    [lost setValue:[NSArray
      arrayWithObjects: kate, jack, hurley,
           sawyer, ben, desmond, locke, nil] forKey:@"population"];
      NSArray *theArray =
        [lost valueForKeyPath:@"population"];
    theArray = [lost valueForKeyPath:@"population.name"];
    theArray = [lost valueForKeyPath:@"population.allies"];
    theArray = [lost valueForKeyPath:@"population.allies.allies"];
    theArray = [lost valueForKeyPath:@"population.allies.allies.name"];
    theArray = [lost valueForKeyPath:@"population.allies.name"];
    NSMutableSet *uniqueAllies =
      [NSMutableSet setWithCapacity:5];
    for(NSArray *a in theArray) {
      if(![a isMemberOfClass:[NSNull class]]){
        for(NSString *n in a){
           printf("%s ", [n cString]);
           [uniqueAllies add Object:n];
        }
        printf("\n");
      }
    }
    NSString *luckyPerson =
      [jack valueForKeyPath:@"lover.lover.lover.name"];
    [kate release];
    [jack release];
    [hurley release];
    [sawyer release];
    [ben release];
    [desmond release];
    [locke release];
    [pool release];
    return 0;
}
```

1.9. Многопотоковость

Многопотоковость — это важный предмет в вычислительном процессе. В одноядерных системах многопотоковость создает у пользователя иллюзию параллельной обработки. Это позволяет разработчику создавать приложения с интерактивным пользовательским интерфейсом, выполняя в фоновом режиме задачи, требующие длительного времени на обработку. В многоядерных системах многопотоковость еще важнее. Разработчики хотят создавать приложения, использующие многоядерные компьютеры наиболее эффективно. Даже если компьютерная система одноядерная, они стремятся придумать наиболее гибкие и дружественные пользователю приложения.

Достичь многопотоковости в Cocoa несложно. Все, что вам нужно сделать, это удостовериться, что вы создаете многопоточные задачи[1], которые будут минимально взаимодействовать как с главным потоком, так и с остальными. Когда потоки взаимодействуют друг с другом, используя общие структуры данных, возникают проблемы в виде поврежденных данных или сложных для нахождения ошибок.

Простой подход к созданию многопотоковости — это использование операционных объектов. Вы можете использовать операционные объекты, наследуя класс `NSOperation` либо используя конкретный его подкласс, называемый `NSInvocationOperation`. С помощью последнего варианта сделать приложение многопотоковым проще.

Представим, что у нас есть метод класса, возможно, вызывающий другие методы, и мы хотим, чтобы этот метод работал в фоновом режиме. Без многопотоковости структура вашего кода выглядела бы, например, так.

В одном из методов какого-либо вашего объекта вы имеете следующее:

```
[myComputationallyIntensiveTaskObject  compute:data];
```

В классе, который на самом деле выполняет полезную работу (например, класс `myComputationallyIntensiveTaskObject`), есть определение метода `compute:`, где

```
-(void)compute:(id)data {
    // выполняем интенсивную вычислительную работу с data,
    // сохраняем промежуточные или конечные результаты
    // в какой-нибудь структуре данных, ds, для дальнейшего использования
}
```

Метод `compute:` обрабатывает данные `data` и выполняет затратные и интенсивные вычисления. Он сохраняет частичные результаты в пере-

[1] *Задача — это фрагмент кода, выполняющий специфическую работу (например, поиск квадратного корня числа).*

менной экземпляра для нужд остальных потоков либо ждет окончания вычислений, чтобы представить конечные результаты потребителю — все зависит от приложения.

Рассмотрим шаги, которые нужно предпринять, чтобы метод `compute:` работал в фоновом режиме, позволяя главному потоку взаимодействовать с пользователем, пока он сам будет выполнять задачу.

1. Создайте запускающий метод. Создайте метод в классе `myComputationallyIntensiveTaskObject`. Этот метод будет использоваться другими объектами, если они решат запускать задачу в фоновом режиме. Назовите его осмысленно, например, `initiateCompute:` или `computeInBackground:`.

2. В `computeInBackground:` создайте операционную очередь. Операционная очередь — это объект типа `NSOperationQueue`, который содержит операционные объекты. Необязательно создавать операционную очередь здесь — ее можно создать в любом месте программы.

3. Создайте объект `NSInvocationObject`. Это и будет ваш операционный объект. Сконфигурируйте этот объект, снабдив его достаточной информацией, чтобы новый поток знал, в каком месте начинать выполняться.

4. Добавьте созданный операционный объект в очередь, чтобы он начал выполняться.

5. Каждый поток нуждается в собственном самовысвобождающемся пуле объектов, поэтому в оригинальном методе `compute:` добавьте новый самовысвобождающийся пул в начале и высвободите его в конце.

6. Если метод `compute:` создает данные, которые будут использоваться другими потоками, синхронизируйте доступ к этим данным, используя блокировки. Применяйте блокировку во всех местах программы, где используются (считываются или записываются) разделяемые данные.

. Вот и все. Теперь применим эти шаги к нашему примеру и посмотрим, как на самом деле просто использовать многопоточность в Cocoa. Листинг 1.5 содержит обновленный код.

Мы добавили две переменные экземпляра к нашему классу, одну — для самой операции, вторую — для операционной очереди. Мы также добавили три метода: `computeInBackground:` для инициализации фоновых вычислений, `computationFinished` для проверки готовности конечного результата и `computationResult` для получения конечного результата. Это простейший пример межпоточного взаимодействия. В зависимости от требований вашего приложения вы можете предпочесть

более сложные протоколы. В инициализирующем фоновый поток методе `computeInBackground:` мы начинаем работу с создания операционной очереди. Далее мы создаем `NSInvocationObject` и инициализируем его с помощью объектов задачи, главным методом и начальными данными. Инициализационный метод `initWithTarget:selector:object` объявлен следующим образом:

```
- (id)initWithTarget:(id)target  selector:(SEL) sel object:(id)arg
```

Здесь `target` — это объект, определяющий селектор `sel`. Селектор `sel` — это метод, вызываемый при выполнении операции. Вы можете передать селектору максимум один объект-параметр через аргумент `arg`. Заметьте, что селектор имеет точно один параметр. В случае если вам не нужно передавать аргумент, можете передать `nil`.

После настройки операционной очереди, создания и инициализации операционного объекта мы добавляем операционный объект в очередь, в результате чего он начинает исполняться. Это происходит с использованием метода `addOperation:` объекта `NSInvocationOperation`.

Как было сказано ранее, самовысвобождающиеся пулы не являются общими для всех нитей. Наш метод `compute:` будет выполняться в собственном потоке, поэтому мы создаем и инициализируем объект `NSAutoreleasePool` в начале и высвобождаем в конце метода. Оригинальный метод `compute:` оставляем нетронутым.

В любой момент времени, когда производится доступ к общим данным `ds`, мы используем механизм блокировок для сохранности целостности данных. Общими данными может быть переменная экземпляра, определяющего метод `compute:`, или другого объекта. Вне зависимости от типа общих данных, если к ним осуществляется доступ из более чем одной нити, используйте блокировку.

Блокировку легко осуществить с помощью директивы `@synchronized`. Директива `@synchronized` используется для эксклюзивного доступа к блоку кода только для одного потока. Она принимает один объект в качестве аргумента. Этот объект будет выступать в качестве блокировки для этого фрагмента кода. Необязательно использовать защищаемые данные как блокировку — в качестве ее вы можете применить `self`, другой объект или даже сам объект `Class`. Следует отметить, что, если чувствительные структуры данных, которые вы пытаетесь защитить, используются из других частей программы, блокирующий объект должен быть того же типа. Чтобы улучшить параллелизм вашей программы, отложите доступ к разделяемым данным до самого конца (то есть до того, как их нужно будет записать) и используйте разные блокирующие объекты для различных несвязанных разделов вашего кода.

Листинг 1.5. Многопоточное приложение, использующее операционные объекты

```
// Изменения в интерфейсе
@interface MyComputationallyIntensiveTask {
…
    NSInvocationOperation *computeOp;
    NSOperationQueue      *operationQueue;
}
…
-(void) computeInBackground:(id)data;
-(BOOL) computationFinished;
-(DS*) computationResult;
@end

@implementation MyComputationallyIntensiveTask
…
// дополнительные методы
-(void) computeInBackground:(id)data{
    operationQueue = [[NSOperationQueue alloc] init];
    computeOp = [[[NSInvocationOperation alloc]
           initWithTarget:self
           selector:@selector(compute:)
           object:data] autorelease];
    [operationQueue addOperation: computeOp];
}
-(BOOL) computationFinished {
    @ synchronized(ds) {
      // если ds готово, возвращаем YES, иначе возвращаем NO
    }
}
-(DS*) computationResult{
    if([self computationFinished] == YES){
      return ds;
    }
    else
      return nil;
}
// изменения в оригинальном методе
-(void) compute:(id)data {
NSAutoreleasePool * threadPool =
    [[NSAutoreleasePool alloc] init];
    // производим интенсивные вычисления
    // над данными, хранящими (частичные или окончательные) результаты
    // в некоторой структуре данных ds
    @ synchronized(ds) {
    // сохраняем результат в ds
    }
    [threadPool release];
}
// Использование из другого объекта
-(void) someOtherMethod {
…
    [myComputationallyIntensiveTaskObject computeInBackground:data];
    // взаимодействуем с пользовательским интерфейсом

    //если требуются промежуточные или конечные результаты
    if (my ComputationallyIntensiveTaskObject computationFinished] == YES) {
      result = [myComputationallyIntensiveTaskObject computationResult];
    }
}
@end
```

1.10. Резюме

В этой главе вы ознакомились с большим количеством информации. В разд. 1.1 вы изучили механизм объявления и определения классов. Затем вы узнали, как объект взаимодействует с другими объектами посредством обмена сообщениями. В разд. 1.2 мы рассмотрели тему управления памятью. Вы узнали, как создавать и инициализировать объекты. Мы обсудили концепцию счетчика захватов и то, как каждый объект поддерживает данный счетчик. Мы также рассмотрели тему самовысвобождающихся пулов объектов и подчеркнули ответственность, которую несут клиенты при высвобождении объектов. В разд. 1.3 вы ознакомились с протоколами в Objective-C. Протоколы были представлены как мощное средство, которое, кроме прочего, позволяет реализовать множественное наследование в языке с одинарным наследованием. В разд. 1.4 мы обсудили свойства. Свойство — это отличительная черта Objective-C, позволяющая декларативно генерировать методы `setter/getter` для переменных экземпляра. В разд. 1.5 была раскрыта тема категорий, используя которые можно расширять возможности существующих классов даже без их исходных кодов. Позиционирование было рассмотрено в разд. 1.6. Оно облегчает замену одного класса другим классом-наследником и используется, в основном для тестирования. Исключения и ошибки были описаны в разд. 1.7. Исключения обычно применяются разработчиком для поиска ошибок, тогда как ошибки используются в конечном коде для информирования о сбоях, возникающих во время выполнения программы, окружению пользователя. В разд. 1.8 мы рассмотрели концепцию кодирования «ключ-значение» (KVC). KVC предоставляет возможность косвенного доступа к свойствам. KVC широко используется в Cocoa, и вы подробно изучили данный предмет. Наконец, мы обсудили многопотоковость, а также простой подход к ее организации с использованием операционных объектов.

Задачи

1. Рассмотрите следующее объявление, определение и использование класса:

```
@interface A
-(int)doSomething;
@end
@implementation A
-(int)doSomething {
    return 1;

@end
int main(int argc, char *argv[])
```

```
{
    A *a  =  [[A alloc]  init];
    int v  =  [a  doSomething];
    [a  release];
}
```

Изучите код и объясните, что будет на выходе функции `main`.

2. Изучите следующий класс и его использование в функции `main()`. Какое выражение будет выполнено последним и почему?

```
1   @interface B : NSObject {
2           NSString *myString;
3   }
4   @property(nonatomic)  NSString * myString;
5   -(unichar) getFirstCharacter;
6   @end
7
8   @implementation B
9   @synthesize  myString;
10  -(unichar) getFirstCharacter {
11          return  [myString  characterAtIndex:0];
12  }
13  @end
14
15  int  main(int  argc ,  char *argv[])
16  {
17          NSAutoreleasePool  *  pool1  =
18                  [[NSAutoreleasePool  alloc]  init];
19          NSMutableString  *str  =
20                  [NSMutableString  stringWithString:@"Where am I?"];
21            B *b  =  [[B  alloc]  init];
22            b. myString  =  str;
23            [pool1  release];
24            unichar  x  =  [b  getFirstCharacter];
25            [b  release];
26  }
```

3. Следующий код объявляет и определяет `Person` и `Dog`, а затем использует эти два класса.

```
@interface  Person : NSObject{}
-(void)bark;
@end

@implementation  Person
-(void)bark {
    printf("Woof\n");
}
@end

@interface Dog : NSObject{}
-(void)bark;
-(void)bark : (NSString*)a;
@end
```

```
@implementation Dog
-(void)bark : (NSString*)a {
    printf("Woof\n");
}
-(void) bark {
    printf("Woof woof\n");}
@end

int main(int argc, char *argv[]) {
    NSAutoreleasePool * pool =
            [[NSAutoreleasePool alloc] init];
    SEL sel = @selector(bark);
    SEL sel1 = @selector(bark:);
    Person *aPerson = [[Person alloc] init];
    Dog *aDog = [[Dog alloc] init];
    .
    .
    .
}
```

Ответьте на следующие вопросы.

Какое значение примет equal после следующего выражения:

```
BOOL equal = sel == sel1; ?
```

Что случится, когда выполнится следующее выражение:

```
[aPerson performSelector : sel]; ?
```

Что случится, когда выполнятся следующие два выражения:

```
[aDog bark];
[aDog bark:@""]; ?
```

Что будет результатом следующего выражения:

```
[aPerson
    performSelector : NSSelectorFromString (@"bark:")]; ?
```

Что делает следующее выражение:

```
[aDog bark:]; ?
```

4. Рассмотрите следующий код. Опишите результат каждой строки. Какая строка выполнится последней?

```
1  NSAutoreleasePool * pool1 =
2          [[NSAutoreleasePool alloc] init];
3  NSMutableArray *arr;
4  arr = [NSMutableArray arrayWithCapacity:0];
5  NSAutoreleasePool * pool2 =
6          [[NSAutoreleasePool alloc] init];
7  NSMutableString *str =
8          [NSMutableString stringWithString:@"Howdy!"];
9  [arr addObject:str];
```

```
10  [pool2  release];
11  int  n  =  [arr  count];
12  str  =  [arr  objectAtIndex:0];
13  [arr  release];
14  n  =  [str  length];
15  [pool1  release];
```

5. Следующая функция приведет к краху приложения. Почему? Подсказка: большой процент аварийных завершений iPhone-приложений является следствием некорректного доступа к памяти.

```
1   void  function () {
2           NSAutoreleasePool  *pool  =
3                   [[NSAutoreleasePool  alloc]  init];
4           NSString  *myString =
5                           [[[NSString  alloc]
6                           initWithString:@"Hello!"]  autorelease];
7           @try  {
8             [myString  appendString:@"Hi!"];
9           }
10          @catch  (NSException * e) {
11            @throw;
12          }
13          @finally {
14            [pool  release];
15          }
16  }
```

Глава 2
Коллекции

Вам как Cocoa-разработчику предоставляется множество классов, которые помогут различными способами группировать некоторые объекты. В этой главе мы обсудим основные доступные вам классы коллекций.

Глава построена следующим образом. Разд. 2.1 посвящен теме массивов. Вы научитесь использованию неизменяемых и изменяемых массивов, различным способам их копирования, а также некоторым методам сортировки. Разд. 2.2 раскрывает тему множеств. Множества — это коллекции, которые не упорядочивают объекты, хранящиеся в них. Вы научитесь интересным операциям с неизменяемыми и изменяемыми множествами. В разд. 2.3 мы рассмотрим словари. Словари позволяют хранить объекты и извлекать их посредством ключей. Как вы уже знаете из разд. 1.7, словари широко используются в Cocoa-платформах, поэтому их изучение необходимо. В разд. 2.4 мы подведем итоги.

2.1. Массивы

Чтобы хранить/получать доступ к объектам в упорядоченном виде, используется `NSArray` и `NSMutableArray`. `NSArray` — это неизменяемая версия массива, позволяющая сохранять объекты только однажды (во время инициализации). `NSMutableArray` — это подкласс `NSArray`, позволяющий добавлять/удалять объекты даже после инициализации коллекции.

Чтобы проиллюстрировать основные идеи использования этих двух классов, применим простой класс `Person` (листинг 2.1).

Листинг 2.1. Класс Person в примере массивов

```
#import <Foundation/Foundation.h>

@interface  Person : NSObject
{
    NSString  *name;
    NSString  *address;
}
@property(copy)  NSString  *name;
```

```
@property(copy)  NSString  *address;
-(id)initWithName : (NSString*)  theName
    andAddress : (NSString *)   theAddress;
-(id)init;
@end

@implementation  Person
@synthesize  name;
@synthesize  address;
-(id)initWithName : (NSString *)  theName
andAddress : (NSString*)  theAddress {
    self  =  [super  init];
    if(self) {
      self.name  =  theName;
      self.address  =  theAddress;
    }
    return  self;
}
-(id)init {
  return  [self  initWithName:@""
            andAddress:@""
    ];
}
-(void)dealloc{
    [name  release];
    [address  release];
    [super  dealloc];
}
@end
```

Листинг 2.2 показывает, как сконфигурировать статический массив, используя класс NSArray. Мы начинаем с создания пяти экземпляров класса Person. На рис. 2.1 изображено состояние этих пяти объектов перед добавлением в массив.

Листинг 2.2. Создание простейшего неизменяемого массива

```
int  main(int  argc ,  char *argv[])
{
    NSAutoreleasePool *pool =
          [[NSAutoreleasePool  alloc]  init];
    Person *a = [[Person  alloc]  init];
    Person *b = [[Person  alloc]  init];
    Person *c = [[Person  alloc]  init];
    Person *d = [[Person  alloc]  init];
    Person *e = [[Person  alloc]  init];
    NSArray *arr1 = [NSArray arrayWithObjects: a, b, c, d, e, nil];
    [pool release];
    [a release];
    [b release];
    [c release];
    [d release];
    [e release];
    return 0;
}
```

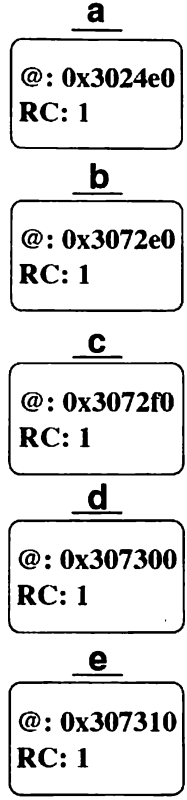

Рис. 2.1. Пять экземпляров Person перед добавлением в экземпляр NSArray; @ означает адрес, RC – счетчик захватов

Есть несколько способов создания и инициализации экземпляра `NSArray`. Так, мы используем метод `arrayWithObjects:` и разделенный запятыми список объектов, с которыми мы хотим инициализировать массив. Последний параметр должен быть `nil`. Это означает, что экземпляр `NSArray` не должен содержать элементов `nil`. Данный метод создания в своем имени не содержит `alloc` или `new`, а также мы не используем `copy` для получения нового экземпляра, поэтому мы не владеем этим экземпляром `NSArray` и не ответственны за его уничтожение. После инициализации экземпляра `NSArray` вы не можете добавлять/удалять его элементы. Он остается статическим до высвобождения.

Помните: если объект добавляется в коллекцию, его счетчик захватов увеличивается этой коллекцией. В нашем примере все пять объектов будут иметь счетчик захватов, равный 2, сразу после инициализации экземпляра `NSArray` (рис. 2.2). Обратите внимание, как массив хранит указатели на его элементы.

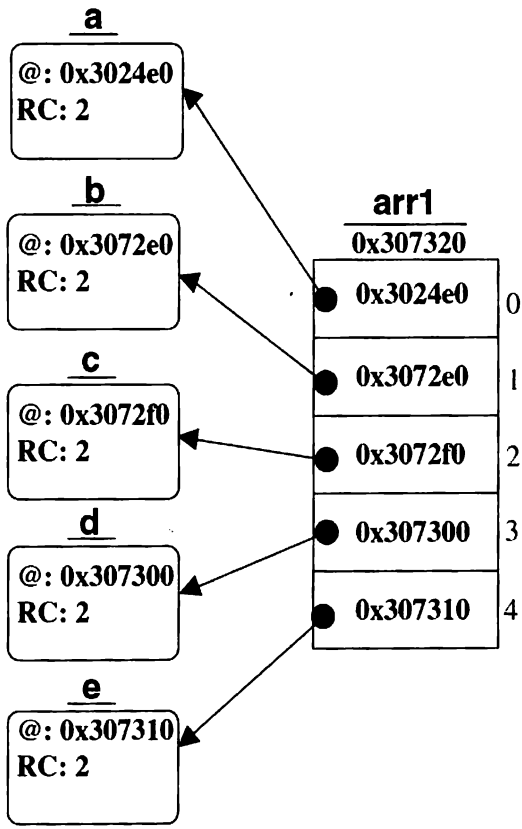

Рис. 2.2. Состояние пяти экземпляров Person после добавления к экземпляру NSArray; @ означает адрес, RC – счетчик захватов

По окончании работы с коллекцией мы можем высвободить ее. Когда вы высвобождаете экземпляр `NSArray`, текущий экземпляр посылает сообщение о высвобождении каждому своему элементу (помните, что коллекции захватывают объекты при добавлении). Мы не являемся владельцем

arr1, поэтому освобождение самовысвобождающегося пула высвободит arr1 и пять наших экземпляров Person. Однако мы должны послать сообщение release этим пяти объектам, так как являемся их владельцем.

2.1.1. Неизменяемое копирование

Вы ознакомились с NSArray, теперь рассмотрим синтаксис и семантику копирования экземпляра NSArray. В листинге 2.3 приведен пример кода, демонстрирующего такое поведение. Мы, как и ранее, создаем и добавляем в массив пять экземпляров Person. После этого мы просим arr1 сделать собственную копию и сохраняем ее в локальную переменную типа NSArray*.

Листинг 2.3. Неизменяемое копирование экземпляра NSArray

```
int main(int argc , char *argv[])
{
    NSAutoreleasePool *pool =
            [[NSAutoreleasePool alloc] init];
    Person *a = [[Person alloc] init];
    Person *b = [[Person alloc] init];
    Person *c = [[Person alloc] init];
    Person *d = [[Person alloc] init];
    Person *e = [[Person alloc] init];
    NSArray *arr1 =
            [NSArray arrayWithObjects: a, b, c, d, e, nil];
    NSArray *arr2 = [arr1 copy];

    Person *aPerson = [arr1 objectAtIndex:0];
    aPerson.name = @"Marge Simpson";
    aPerson.address = @"Springfield";

    // Результатом следующей строки будет:
    // Person at 0 is: Name: Marge Simpson, Addr: Springfield
    printf("Person at %d is: Name: %s, Addr: %s\n",
            0,
            [[[arr2 objectAtIndex:0] name]
                    cStringUsingEncoding : NSUTF8StringEncoding],
            [[[arr2 objectAtIndex:0] address]
                    cStringUsingEncoding : NSUTF8StringEncoding]);
    // Должны высвободить arr1, так как мы создали его, используя копирование
    [arr2 release];
    // должны высвободить все объекты
    [a release];
    [b release];
    [c release];
    [d release];
    [e release];
    [pool release];
    return 0;
}
```

После копирования массива мы изменяем первый элемент исходного массива — arr1. Помните, что NSArray и NSMutableArray являются упорядоченными коллекциями. Каждый хранящийся элемент имеет

специальное место, или индекс. Чтобы извлечь первый элемент, мы используем метод `objectAtIndex:` с индексом `0`.

Когда мы изменили первый элемент в исходном массиве, то рассматриваем первый элемент в копии и обнаруживаем, что этот элемент также изменился. Документированная семантика того, как `NSArray` копирует себя, следующая. Во-первых, `NSArray` делает плоскую копию своих элементов. Это означает, что копируются только указатели на элементы. Во-вторых, новый экземпляр не самовысвобождающийся и владельцем его является вызывающий. Рис. 2.3 демонстрирует состояние пяти наших объектов после копирования `NSArray`.

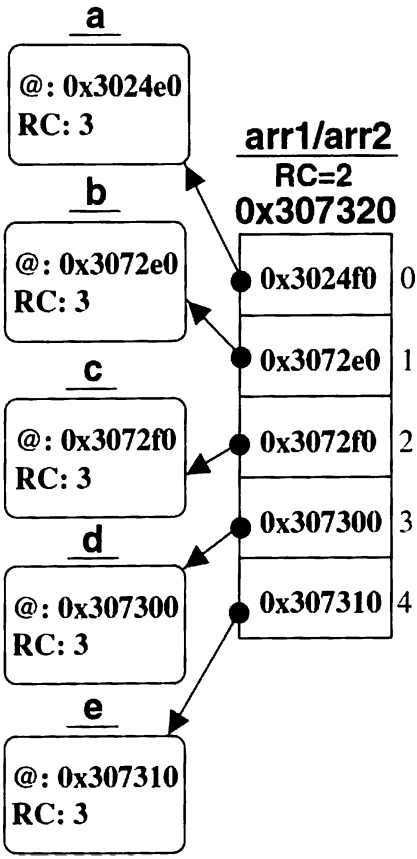

Рис. 2.3. Состояние NSArray и его неизменяемой копии

Нужно отметить следующее: мы даже не получили нового экземпляра NSArray. Все, что случилось, – arr2 получил копию адреса arr1 (что, в сущности, является простым присваиванием), счетчик захватов arr1 увеличился на 1 и счетчики захватов всех пяти объектов увеличились на 1. Такое поведение неизменяемой копии имеет смысл, так как исходный объект массива статичен и не меняется.

Несмотря на то что arr2 является просто arr1, мы получили его, используя метод, в котором содержится операция копирования, поэтому мы должны высвободить arr2 по окончании работы с ним.

2.1.2. Изменяемое копирование

До сих нор мы работали исключительно с NSArray и его статическим поведением. NSMutableArray является изменяемой версией этого класса, и количество объектов, хранящихся в нем, может расти и расширяться в процессе жизни программы. Как показано в листинге 2.4, мы можем получить изменяемую копию неизменяемого источника. Тем не менее вместо использования copy мы применим mutableCopy.

Листинг 2.4. Получение изменяемой копии неизменяемого экземпляра NSArray

```
int main(int argc , char *argv[])
{
    NSAutoreleasePool *pool =
           [[NSAutoreleasePool alloc] init];
    Person *a = [[Person alloc] init];
    Person *b = [[Person alloc] init];
    Person *c = [[Person alloc] init];
    Person *d = [[Person alloc] init];
    Person *e = [[Person alloc] init];
    NSArray *arr1 =
           [NSArray arrayWithObjects: a, b, c, d, e, nil];
    NSMutableArray *arr2 = [arr1 mutableCopy];
    Person *f = [[Person alloc] init];
    [arr2 addObject:f];
    [arr2 removeObject:a];
    [arr2 release ];
    [a release];
    [b release ];
    [c release];
    [d release];
    [e release];
    [f release];
    [pool release];
    return 0;
}
```

Наличие экземпляра NSMutableArray позволяет динамически удалять и добавлять в него объекты. В примере кода мы добавили новый объ-

ект f и удалили существующий объект a[1]. Кроме изменений в позициях текущих элементов внутри массива, а также их счетчиков захватов, счетчик захвата f увеличился на 1, а счетчик захватов a уменьшился на 1. На рис. 2.4 изображено состояние объектов на этом этапе. Обратите внимание, что, хотя оба массива сейчас различны, они все еще разделяют указатели на одни и те же элементы.

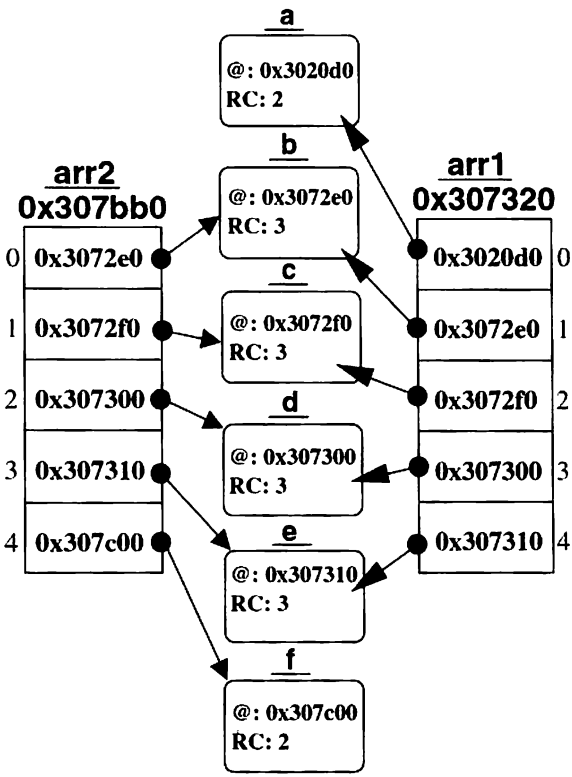

Рис. 2.4. Изменяемая копия неизменяемого NSArray

2.1.3. Расширенное копирование

До сих пор мы имели дело с плоскими копиями. Плоское копирование — это поведение по умолчанию `NSArray` и `NSMutableArray` безот-

[1] *Метод* `removeObject:` *удаляет все вхождения объекта в коллекцию.*

носительно к типу копирования (то есть является копия изменяемой или неизменяемой). Однако вам может понадобиться копия объекта, которая будет независимой от оригинала. Такое копирование называется расширенным.

Чтобы объект имел возможность клонировать себя, его класс должен реализовывать протокол `NSCopying`. Этот протокол определяет только один метод — `copyWithZone:`. Этот метод должен быть реализован заимствующим классом.

Он должен возвращать независимую, функционально эквивалентную копию объекта, чьи переменные должны иметь значения, идентичные значениям переменных начального объекта во время копирования. Копия создастся следующим образом.

1. Ваш класс должен быть наследником `NSObject` и заимствовать протокол `NSCopying`.

2. Он должен реализовывать метод `copyWithZone:`. Несмотря на то что суперклассом является `NSObject`, вызов метода `copyWithZone:` суперкласса (с той же зоной, что вы получили) должен быть первым выражением в вашей реализации `copyWithZone:`.

3. Внутри `copyWithZone:` вы создаете новый экземпляр класса и инициализируете его в то же состояние, как и ваш экземпляр. В зависимости от того, насколько расширенное копирование вы хотите произвести, можете продолжить копирование требуемого количества переменных экземпляра. Все зависит от требований вашего кода.

4. Метод `copyWithZone:` возвращает вызывающему новую копию экземпляра. Вызывающий владеет этой копией, и он единственный отвечает за ее своевременное высвобождение.

Вот основные шаги, которые вы должны предпринять для создания новой копии оригинального объекта.

Остается один вопрос: каким образом отправка экземпляру сообщения `copy` на самом деле оказывается отправкой сообщения `copyWithZone:`? Ответ прост: `NSObject` имеет особым образом устроенный метод `copy`, вызывающий `copyWithZone:` с зоной `nil`. Обратите внимание, что `NSObject` не принимает сам протокола `NSCopying`. Очевидно, если ни один из классов в цепочке наследования не реализует метода `copyWithZone:`, мы получим исключение (см. подразд. 1.7.1), поэтому важно понимать, что отправка сообщения `copy` объекту массива всегда будет иметь на выходе плоские копии вне зависимости от того, принимают ли объекты, хранящиеся в нем, протокол `NSCopying`.

Листинг 2.5 показывает обновленный класс `Person` с добавленным заимствованием протокола `NSCopying`. Вы можете заметить, что этот класс реализует подход расширенного копирования. Мы создаем новый экземпляр

объекта, содержащий новые переменные экземпляра, а не просто копируем указатели. Новые переменные экземпляра получаются от свойства атрибута copy, используемого для обеих переменных экземпляра — name и address.

Листинг 2.5. Улучшенный класс Person, заимствующий протокол NSCopying

```
#import <Foundation/Foundation.h>

@interface  Person : NSObject<NSCopying>
{
    NSString  *name;
    NSString  *address;
}
@property(copy)  NSString  *name;
@property(copy)  NSString  *address;
-(id)initWithName:(NSString *)  theName
        andAddress:(NSString *)  theAddress;
-(id)init;
-(id)copyWithZone:(NSZone *)zone;
@end

@implementation  Person
@synthesize  name;
@synthesize  address;
-(id)initWithName:(NSString *)  theName
        andAddress:(NSString*)  theAddress {
    self  =  [super  init];
    if(self) {
      self.name = theName;
      self.address = theAddress;
    }
    return  self;
}
-(id)init {
    return  [self  initWithName:@""  andAddress:@""];
}
-(void) dealloc {
    [name release];
    [address release];
    [super dealloc];
}

// Реализация copyWithZone: объявленного в NSCopying
-(id)copyWithZone:(NSZone *)zone
{
    Person  *aPerson  =  [[[self  class]  allocWithZone:  zone]
          initWithName : [self  name]
          andAddress : [self  address]];
    return  aPerson;
}
@end
```

В листинге 2.6 приведен демонстрационный код для получения расширенной копии объекта NSArray. Мы используем тот же метод класса, arrayWithObject:, который до этого применяли для получения нового экземпляра NSArray. Теперь объекты являются копиями оригинальных объектов a, b, c , d и e.

Ссообщение copy возвращает новый объект, которым мы владеем, поэтому мы посылаем сообщение autorelease копии объекта перед ее добавлением в новый массив.

Листинг 2.6. Реализация расширенного копирования массива объектов

```
int main(int argc , char *argv[])
{
    NSAutoreleasePool *pool =
        [[NSAutoreleasePool alloc] init];
    Person *a = [[Person alloc] init];
    Person *b = [[Person alloc] init];
    Person *c = [[Person alloc] init];
    Person *d = [[Person alloc] init];
    Person *e = [[Person alloc] init];

    NSArray *arr1 =
        [NSArray arrayWithObjects: a, b, c, d, e, nil];
    NSArray *arr2 =
        [NSArray arrayWithObjects:
            [[a copy] autorelease],
            [[b copy] autorelease],
            [[c copy] autorelease],
            [[d copy] autorelease],
            [[e copy] autorelease],
            nil
        ];

    Person *aPerson = [arr1 objectAtIndex:0];
    aPerson.name = @"Marge Simpson";
    aPerson.address = @"Springfield";

    // Результатом следующей строчки будет:
    // Person at 0 is: Name: , Addr:
    printf("Person at %d is: Name: %s, Addr: %s\n",
        0,
        [[[arr2 objectAtIndex:0] name]
            cStringUsingEncoding : NSUTF8StringEncoding],
        [[[arr2 objectAtIndex:0] address]
            cStringUsingEncoding : NSUTF8StringEncoding]);
    // must release all objects
    [a release];
    [b release];
    [c release];
    [d release];
    [e release];
    [pool release];
    return 0;
}
```

В конце создания и инициализации нового массива состояние объектов следующее (рис. 2.5). Как показывает рисунок, массивы независимы друг от друга. Каждый задействованный объект уникален. Если мы изменим элемент в arr1, с arr2 ничего не произойдет, и наоборот. Это проиллюстрировано в коде, когда мы изменяем состояние объекта по индексу 0 в arr1 и не наблюдаем никаких изменений ни в arr2, ни в содержащихся в нем объектах.

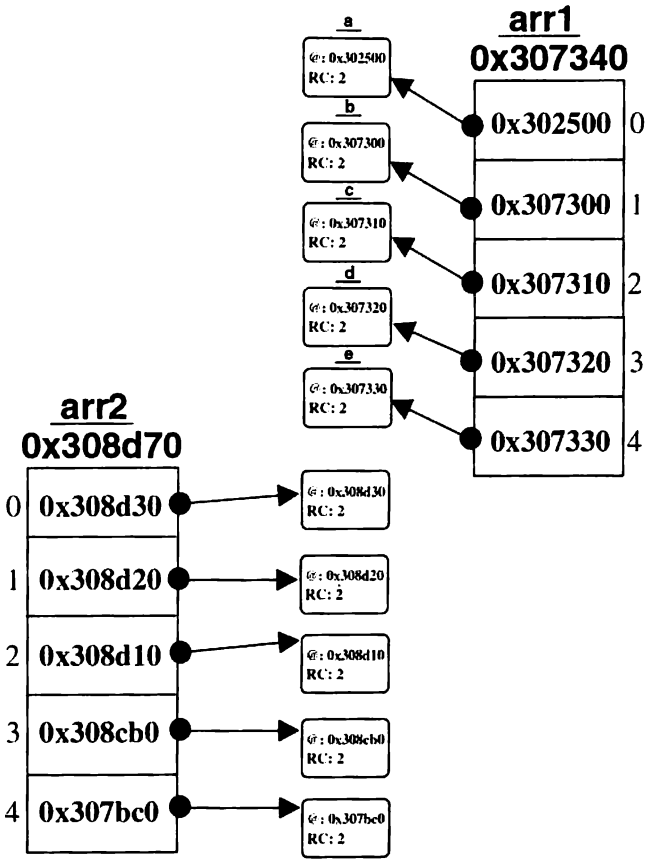

Рис. 2.5. Иллюстрация состояний объектов, задействованных в расширенном копировании содержимого массива объектов

Чтобы все очистить, мы просто должны высвободить объекты `a`, `b`, `c`, `d`, `e` и `pool`. Все остальные объекты высвободятся сами, когда мы высвободим в конце самовысвобождающийся пул.

2.1.4. Сортировка массива

Сортировать массив можно несколькими способами. Два основных — написать функцию, которая находит надлежащий порядок двух

объектов, или добавить вашему классу метод, который позволяет экземпляру сравнивать себя с другим экземпляром того же класса.

В листинге 2.7 класс `Person` усовершенствован следующим образом. Во-первых, мы добавили новую переменную экземпляра `personID` типа `NSInteger` (что, по существу, `int`). Во-вторых, мы ввели новый метод `nameAscCompare:`, который позволяет объекту `Person` сравнивать себя с другим экземпляром `Person`.

Листинг 2.7. Усовершенствованный класс Person с методом сортировки

```
#import <Foundation/Foundation.h>

@interface  Person : NSObject
{
    NSString  *name;
    NSString  *address;
    NSInteger  personID;
}
@property(copy)  NSString  *name;
@property(copy)  NSString  *address;
@property  NSInteger  personID;
-(id)initWithName : (NSString *)  theName
    andAddress:(NSString*)  the Address
    andID:(NSInteger)  theID;
-(id)init;
-(NSComparisonResult)nameAscCompare:(Person *)aPerson;
@end

@implementation  Person
@synthesize  name;
@synthesize  address;
@synthesize  personID;

-(id)initWithName : (NSString *)  theName
    andAddress:(NSString*)   the Address
    andID:(NSInteger)   theID {
  self = [super  init];
  if(self) {
    self.name = theName;
    self.address  =  theAddress;
    personID = theID;
  }
  return  self;
}
-(id)init {
  return  [self  initWithName:@""  andAddress:@""  andID:0];
}
-(void) dealloc {
    [name  release];
    [address  release];
    [super  dealloc];
}
- (NSComparisonResult)nameAscCompare:(Person *)aPerson {
  return  [name  caseInsensitiveCompare:[aPerson  name]];
}
@end
```

Предположим, вы хотите отсортировать массив, содержащий объекты Person, по увеличению personID. Объект NSArray предоставляет метод sortedArrayUsingFunction: context:. Этот метод объявлен следующим образом:

```
- (NSArray *) sortedArrayUsingFunction:
            (NSInteger  (*)(id, id, void *))comparator
            context:(void *)context
```

Первый параметр — это указатель на функцию, которая принимает два произвольных объекта и (void *) в качестве третьего параметра. Она возвращает NSInteger. Второй параметр — обычный С-указатель, поэтому при желании его можно использовать в реализации в качестве контекста. Этот второй указатель на самом деле является третьим указателем, используемым в каждом вызове, производимом вашей реализацией функции сравнения. Она должна возвращать:

- NSOrderedAscending, если первый объект меньше второго;
- NOrderedSame, если два объекта равны;
- NSOrdereDescending, если первый объект больше второго.

В листинге 2.8 показана функция intSort(), которая будет использоваться в качестве функции сравнения. Реализация данной функции зависит от требований приложения. Здесь же мы просто сравниваем personID двух объектов и возвращаем подходящий результат сравнения.

В листинге 2.8 мы посылаем arr1 следующее сообщение:

```
sortedArrayUsingFunction:intSort context:NULL
```

Затем мы сохраняем ссылку на новый массив в intSortedArray. Обратите внимание, что новый массив самовысвобождающийся и объекты в нем являются такими же, как в оригинальном массиве arr1.

Второй способ сортировки массива — оснащение объекта методом сравнения себя самого с такими же объектами. Выше (см. листинг 2.7) приведено добавление нового метода nameAscCompare:, который сравнивает значение переменной экземпляра name с соответствующей переменной экземпляров других объектов. Он использует метод caseInsensitiveCompare:.

В листинге 2.8 показано, как отсортировать массив вторым способом. Мы используем метод sortedArrayUsingSelector:. Обратите внимание, что никакого контекста в качестве параметра не требуется, так как есть self и указатели на объекты класса.

Листинг 2.8. Две разные схемы сортировки массива

```
NSInteger intSort(id p1, id p2, void *context)
{
    int v1 = [(Person*)p1 personID];
    int v2 = [(Person*)p2 personID];
    if (v1 < v2)
        return NSOrderedAscending;
    else if(v1 > v2)
        return NSOrderedDescending;
    else
        return NSOrderedSame;
}

int main(int argc , char *argv[])
{
    NSAutoreleasePool *pool =
        [[NSAutoreleasePool alloc] init];
    Person *a = [[Person alloc]
        initWithName:@"Kate Austen"
        andAddress:@"" andID:5];
    Person *b = [[Person alloc]
        initWithName :@"Sayid Jarrah"
        andAddress:@"" andID:4];
    Person *c = [[Person alloc]
        initWithName:@"Sun Kwon"
        andAddress:@"" andID:1];
    Person *d = [[Person alloc]
        initWithName:@"Hurley Reyes"
        andAddress:@"" andID:2];
    Person *e = [[Person alloc]
        initWithName:@"Jack Shephard"
        andAddress:@"" andID:3];
    NSArray *arr1 =
        [NSArray arrayWithObjects: a, b, c, d, e, nil];
    NSArray *intSortedArray =
        [arr1 sortedArrayUsingFunction:intSort context:NULL];
    NSArray *strSortedArray =
        [arr1 sortedArrayUsingSelector:
            @selector(nameAscCompare:)
        ];
    // нужно высвободить все объекты
    [a release];
    [b release];
    [c release];
    [d release];
    [e release];
    [pool release];
    return 0;
}
```

На рис. 2.6 изображено состояние трех объектов массива и их элементов. Заметьте, что счетчик захватов элементов массива теперь равен 4 (1 на каждый массив и 1 на `alloc`).

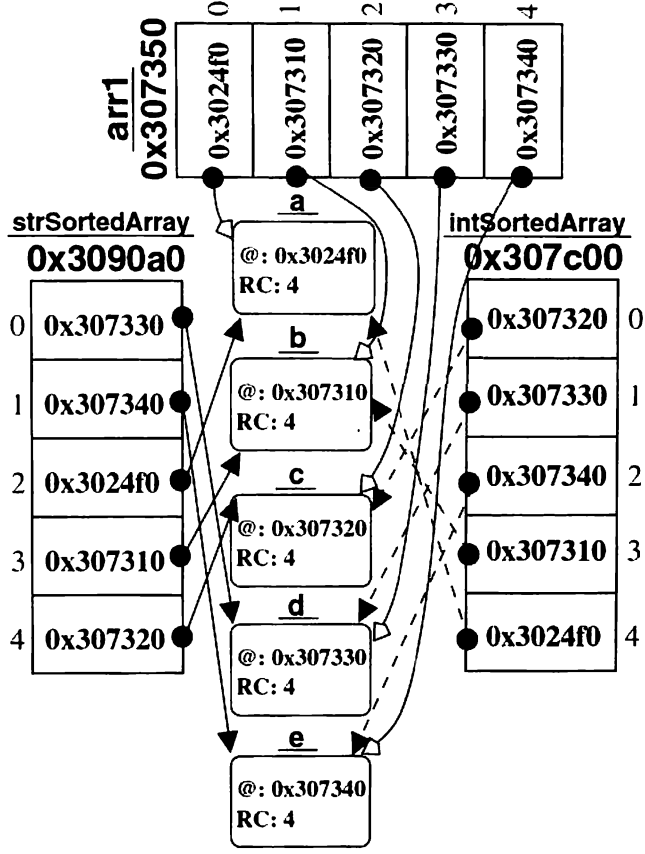

Рис. 2.6. Состояние оригинального массива и двух его по-разному отсортированных копий

2.2. Множества

В предыдущем разделе вы увидели, как можно хранить упорядоченные объекты в массиве. Однако некоторые сценарии не требуют упорядочивания объектов, а заинтересованы в управлении уникальными объектами и предоставлении быстрого механизма проверки объекта на принадлежность.

Множества предоставляют именно такое поведение. Как и массивы, множества бывают двух типов — статические, представленные классом `NSSet`, и динамические, или изменяемые, представленные классом `NSMutableSet`.

2.2.1. Неизменяемые множества

В листинге 2.9 приведен код, демонстрирующий использование неизменяемых множеств. Для создания неизменяемого множества вы должны указать его составляющие во время фазы инициализации. Именно это делает `initWithObjects:`. Этот метод схож с таким же в классе `NSArray`. Приведенный ниже код создает три множества.

Листинг 2.9. Неизменяемые множества

```
NSSet *favoritShows = [[NSSet alloc]
    initWithObjects:
    @"Everybody Loves Raymond",
    @"Lost",
    @"Nova",
    @"Late Show",
    @"Tom and Jerry",
    nil
    ];
NSSet *reallyFavoritShows =
    [NSSet alloc] initWithObjects:
    @"Everybody loves Raymond",
    @"Lost",
    nil
    ];
NSSet *hatedShows = [[NSSet alloc] initWithObjects:
    @"Heroes",
    @"60 minutes",
    @"The Tonight Show",
    @"Deal or no deal",
    nil
    ];
printf ("Number of elements = %d\n", [favoritShows count]);
if([favoritShows
    intersectsSet:hatedShows] == YES) {
      printf("makes no sense!\n");
}
if([reallyFavoritShows
    isSubsetOfSet:favoritShows] == YES) {
      printf("makes sense!\n");
}
if([reallyFavoritShows
    isEqualToSet:favoritShows] == YES) {
      printf("makes some sense!\n");
}
```

```
NSString *anyShow = [favoritShows anyObject];
if([hatedShows containsObject: anyShow] == YES) {
  printf("makes no sense!\n");
}
```

Чтобы найти количество элементов во множестве, используйте метод экземпляра `count`. Класс `NSSet` также предоставляет методы, реализующие математические операции, определенные над множеством. Например, приведенный выше код демонстрирует, как можно определить, пересекается ли (имеет ли общие элементы) экземпляр множества с другим множеством. Метод `intersectsSet:` объявлен следующим образом:

```
- (BOOL)intersectsSet:(NSSet *)otherSet
```

Он возвращает `YES`, если у приемника и `otherSet` существует по крайней мере один общий элемент, и `NO`, если нет.

Вы также можете проверить, является ли экземпляр множества подмножеством другого множества, используя `isSubsetOfSet:`, объявленный следующим образом:

```
(BOOL) isSubsetOfSet:(NSSet *) otherSet
```

Чтобы этот метод вернул `YES`, каждый элемент приемника должен присутствовать в `otherSet`.

Если вы хотите проверить, присутствуют ли все элементы приемника в другом множестве и наоборот, используйте метод `isEqualToSet:`.

Метод `anyObject` возвращает произвольный (но необязательно случайный) элемент множества. Метод `containsObject:` возвращает `YES`, если данный элемент присутствует во множестве, и `NO` в противном случае. Он объявлен следующим образом:

```
- (BOOL)containsObject:(id)anObject
```

2.2.2. Изменяемые множества

Листинг 2.10 является продолжением листинга 2.9. Мы продолжаем наш пример и рассматриваем динамическую версию множества `NSMutableSet`.

Чтобы создать изменяемый экземпляр множества, используйте `alloc` для установки начального объема множества. Этот начальный объем не является ограничением размера, так как он может расширяться динамически. Выражение `[[NSMutableSet alloc] initWithCapacity:2]` создает новое изменяемое множество с начальным объемом в два элемента.

Чтобы добавить элемент во множество, используйте `addObject:`. Как вы видели в случае с массивами, множество не копирует добавляемый вами объект, а создает на него заявку, посылая сообщение о захвате. Если вы добавляете элемент, который уже находится во множестве, метод не имеет влияния ни на множество, ни на элемент. Чтобы удалить элемент из множества, используйте метод `removeObject:`. Если элемент, который вы пытаетесь удалить, отсутствует во множестве, сторонних эффектов не будет.

После добавления и удаления объектов из множества мы выводим множество на экран, используя метод `description:`. Результатом первого `printf` будет следующее:

```
dynamicSet = {(
    Heroes,
    Lost
)}
```

Вы можете объединить два множества, используя `unionSet:`, объявленный так:

```
- (void)unionSet:(NSSet *) otherSet
```

Метод `unionSet:` добавляет каждый элемент `otherSet`, не являющийся элементом приемника, к последнему. После операции объединения `dynamicSet` будет показан как:

```
dynamicSet = {(
    Everybody Loves Raimons,
    Heroes,
    Lost
)}
```

Метод `minusSet:` объявлен следующим образом:

```
- (void)minusSet:(NSSet *) otherSet
```

Он удаляет все составляющие `otherSet` из множества-приемника. Содержимым `dynamicSet` после выполнения выражения `[dynamicSet minusSet: reallyFavoritShows]` будет следующее:

```
dynamicSet = {(
    Heroes
)}
```

Листинг 2.10. Демонстрация изменяемых множеств

```
NSMutableSet  *dynamicSet =
    [[NSMutableSet  alloc]  initWithCapacity:2];
[dynamicSet  addObject:@"Lost"];
[dynamicSet  addObject:@"Nova"];
[dynamicSet  addObject:@"Heroes"];
```

```
[dynamicSet  addObject:@"Lost"];
[dynamicSet  removeObject:@"Everybody Loves Raymond"];
[dynamicSet  removeObject:@"Nova"];
printf("dynamicSet  =  %s\n",
    [[dynamicSet  description]  cString]);
[dynamicSet  unionSet:reallyFavoritShows];
printf("dynamicSet  =  %s\n",
    [[dynamicSet  description]  cString]);
[dynamicSet  minusSet:reallyFavoritShows];
printf("dynamicSet  =  %s\n",
    [[dynamicSet  description]  cString]);
```

2.2.3. Дополнительные важные методы

Чтобы удалить все элементы из множества, вы можете использовать

```
- (void)removeAllObjects
```

Для удаления всех элементов множества и последующего их добавления в другое множество запишите

```
- (void)setSet:(NSSet *)otherSet
```

Если у вас есть массив элементов и вы хотели бы добавить все его элементы во множество, используйте

```
- (void)addObjectsFromArray:(NSArray *)anArray
```

Чтобы послать каждому объекту множества сообщение, примените

```
- (void)makeObjectsPerformSelector:(SEL)aSelector
```

Метод, заданный селектором `aSelector`, не должен иметь аргументов.

Если вы хотите общаться с элементами множества посредством объекта, используйте

```
- (void) makeObjectsPerformSelector :(SEL) aSelector
    withObject:(id)anObject
```

Этот метод будет использовать селектор, представляющий метод, который имеет только один аргумент типа `id`. В обоих методах селектор не должен изменять сам экземпляр множества.

2.3. Словари

Неизменяемый `NSDictionary` и его изменяемый подкласс `NSMutableDictionary` дают возможность хранить объекты и извлекать их, используя ключи. Каждая запись в словаре состоит из ключа

и значения. Ключом может быть любой объект, заимствующий протокол NSCopying. Обычно используются экземпляры класса NSString, но это может быть любой класс.

Ключи в словаре должны быть уникальными. Ключи и значения не могут быть nil. Для хранения нулевых объектов платформа предоставляет класс NSNull. Когда вы добавляете в словарь записи, класс словаря делает копию ключа и использует ее в качестве ключа для хранения объекта-значения. Само хранилище следует модели хэша, но классы экранируют от вас детали реализации. Объект-значение не копируется, а только захватывается.

Как и с классами массивов, после инициализации неизменяемого словаря вы не можете добавлять/удалять элементы. Вы можете получить изменяемый словарь из неизменяемого, и наоборот. Листинг 2.11 демонстрирует работу со словарями.

Листинг 2.11. Работа со словарями

```
#import <Foundation/Foundation.h>

int main(int argc , char *argv[])
{
    NSAutoreleasePool * pool =
          [[NSAutoreleasePool alloc] init];
    NSMutableArray *kArr = [NSMutableArray arrayWithObjects:
          @"1", @"2", @"3", @"4", nil];
    NSMutableArray *aArr =
          [NSMutableArray arrayWithObjects:
                @"2", nil];
    NSDictionary *guide =
          [NSDictionary dictionaryWithObjectsAndKeys:
          kArr, @"Kate", aArr, @"Ana-Lucia", nil];

    NSEnumerator *enumerator = [guide keyEnumerator];
    id key;
    while ((key = [enumerator nextObject])) {
      if ([[key substringToIndex:1] isEqual:@"K"]) {
          [[guide objectForKey:key] addObject:@"5"];
      }
    }

    NSMutableDictionary *dynaGuide = [guide mutableCopy];
    for(key in dynaGuide) {
      if ([[key substringToIndex:1] isEqual:@"A"]) {
          [[dynaGuide objectForKey:key] addObject:@"5"];
      }
    }
    NSArray *keys = [dynaGuide allKeys];
    for(key in keys) {
      if ([[key substringToIndex:1] isEqual:@"A"]) {
```

```
            [dynaGuide  removeObjectForKey:key];
        }
    }
    [dynaGuide  release];
    [pool  release];
    return  0;
}
```

В приведенном выше коде мы создаем неизменяемый словарь, где ключами являются строки, а значениями — изменяемые массивы. Словарь создается посредством метода `dictionaryWithObjectsAndKeys:`. Методу передаются список переменных значений и ноль-терминированных ключей.

Для доступа к хранящимся в словаре значениям вы можете использовать нумератор ключей. Метод `keyEnumerator` возвращает такой нумератор из объекта словаря. Данный код использует нумератор для проверки всех объектов, чьи ключи начинаются с «A». Для каждого такого ключа он обновляет значение данной записи. Чтобы получить значение по данному ключу, используется метод `objectForKey:`.

Для создания изменяемой копии словаря используйте метод `mutableCopy`. Он создаст новый изменяемый словарь, инициализированный значениями словаря-источника. Внутри метода используется `copy`, поэтому вы являетесь владельцем этого объекта и по завершении работы должны высвободить его.

Другой способ прохождения по словарю — использование быстрой нумерации. Строка `for(key in dynaGuide)` проходит по ключам в словаре `dynaGuide`.

Вы можете добавлять/удалять записи в изменяемый словарь. Чтобы удалить запись, используйте метод `removeObjectForKey:`. Нельзя удалять запись во время прохождения по элементам словаря. Вместо этого сделайте копию ключей и затем с их помощью удалите выбранные записи. Чтобы получить экземпляр `NSArray` всех ключей, используйте метод `allKeys`. Сделав это, вы можете проходить по ключам и изменять словарь по своему усмотрению.

Дополнительные методы. Рассмотрим несколько важных дополнительных методов.

```
- (BOOL)isEqualToDictionary:(NSDictionary *)otherDictionary
```

Возвращает `YES`, если приемник имеет то же количество записей и для каждого ключа соответствующие значения двух словарей равны (то есть `isEqual` возвращает `YES`).

```
- (NSArray *)allValues
```

Создает новый массив значений записей, содержащихся в словаре.

```
- (NSArray *)keysSortedByValueUsingSelector:(SEL)comparator
```

Генерирует массив ключей, упорядоченных по отсортированным с использованием `comparator` значениям.

```
- (void)addEntriesFromDictionary:(NSDictionary *)otherDictionary
```

Записи в `otherDictionary` добавляются к получателю данного сообщения. Если получатель уже имеет запись с таким же ключом, перед заменой эта запись получает сообщение о высвобождении.

2.4. Резюме

В этой главе мы рассмотрели тему коллекций. Коллекции — это Cocoa-объекты, служащие контейнерами для других объектов Cocoa. Мы изучили три типа коллекций, определенных в платформе Foundation. В разд. 2.1 были описаны массивы. Неизменяемые массивы являются экземплярами класса `NSArray`. Этот класс позволяет создать статический массив, который нельзя изменить после инициализации. Изменяемая версия `NSArray` — это `NSMutableArray`. Экземпляр `NSMutableArray` можно модифицировать посредством добавления и удаления элементов во время работы программы. Класс `NSArray` эффективнее, чем `NSMutableArray`. Экземпляры `NSArray` следует использовать, если массив не требует изменений после инициализации.

Вы ознакомились с основами плоского и расширенного копирования. Плоская копия объекта является новым экземпляром, разделяющим ссылки (указатели) на другие объекты, хранящиеся в оригинальном объекте. Расширенная копия копирует и хранящиеся объекты. Коллекции реализуют плоское копирование в независимости от типов хранящихся объектов. Чтобы класс реализовывал расширенное копирование, ему требуется заимствовать протокол `NSCopying` и реализовывать метод `copyWithZone:`. Чтобы произвести расширенное копирование всей коллекции, нужно действовать в следующем порядке: пройти по всем объектам коллекции, послать сообщение о копировании, сделать объект самовысвобождающимся и затем добавить его в коллекцию.

Вы также ознакомились с сортировкой и ее двумя способами — с использованием функции для сравнения двух экземпляров и добавлением метода так, чтобы экземпляр мог сравнивать себя с другим. Первая схема используется в методе экземпляра массива `sorted-`

ArrayUsingFucntion: context:, вторая — в методе экземпляра массива sortedArrayUsingSelector:.

В разд. 2.2 мы изучали множества. Неизменяемые множества являются экземплярами класса NSSet, изменяемые множества — экземплярами NSMutableSet. Мы рассмотрели некоторые методы, реализующие математические операции над множествами.

В разд. 2.3 рассматривались словари. Словари позволяют извлекать объекты, используя ключи. Вы увидели несколько примеров, иллюстрирующих работу с неизменяемыми и изменяемыми словарями.

Глава 3
Анатомия iPhone-приложения

В этой главе мы рассмотрим основные шаги, необходимые для построения простейшего iPhone-приложения. Мы изучим основную структуру простейшего iPhone-приложения и этапы разработки приложения с использованием XCode.

3.1. Приложение HelloWorld

В этом разделе мы рассмотрим основную структуру простого iPhone-приложения. Наша программа выводит пользователю сообщение Hello World. Написание такого приложения делится на следующие этапы.

1. Создайте функцию `main.m`. Как в любой С-программе, исполнение приложений Objective-C начинается с `main()`. Вам нужно создать функцию `main()` в файле `main.m` следующим образом:

```
#import <UIKit/UIKit.h>

int main(int argc, char *argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil,
                                   @"HelloWorldAppDelegate");
    [pool release];
    return retVal;
```

Функция `main()` начинается с создания самовысвобождающегося пула и заканчивается его высвобождением. В середине она вызывает функцию `UIApplicationMain()`, объявленную следующим образом:

```
int
  UIApplicationMain(int argc, char *argv[],
    NSString *principalClassName,
    NSString  *delegateClassName
)
```

Эта функция имеет четыре параметра. Первые два являются параметрами, передаваемыми в функцию `main()`. Они должны быть знакомы любому С-программисту. Третий параметр — имя класса приложения. Если задан `nil`, для создания уникального объекта приложения используется класс `UIApplication`. Четвертый, последний параметр — имя класса-делегата приложения.

`UIApplicationMain()` представляет функционал приложения и объектов-делегатов программы. Она устанавливает свойство делегата

на экземпляр делегата программы и основной цикл выполнения приложения. С этого момента события, такие как касания пользователя, собираются системой в очередь, а объект приложения удаляет их из этой очереди и доставляет соответствующим объектам в вашей программе (обычно основному окну).

2. Создайте класс-делегат приложения. Экземпляр класса-делегата будет получать важные сообщения от объектов программы во время его работы. Следующий код является типичным классом-делегатом приложения:

```
#import <UIKit/UIKit.h>

@class MyView;
@interface  HelloWorldAppDelegate:NSObject <UIApplicationDelegate> {
            UIWindow * window;
            MyView *view;
}
@end
```

Обратите внимание, что класс-делегат приложения заимствует протокол `UIApplicationDelegate`. Ссылки на объекты пользовательского интерфейса, которые будут созданы и представлены пользователю, сохраняются в переменных экземпляра. В большинстве случаев вы будете располагать одним объектом окна и несколькими присоединенными к нему представлениями. В приведенном примере переменная `window` хранит ссылку на объект главного окна, а `view` используется для хранения ссылки на собственное представление типа `MyView`.

Одним из главных методов протокола `UIApplicationDelegate` является `applicationDidFinishLaunching:`. Он вызывается объектом приложения, чтобы информировать делегата, что приложение закончило запуск. Обычно вы реализуете этот метод для инициализации программы и создания пользовательского интерфейса. Далее приведен листинг реализации класса-делегата приложения. В методе `applicationDidFinishLaunching:` мы сначала создаем главное окно приложения.

Окна являются экземплярами класса `UIWindow`. Он является подклассом класса `UIView`, который, в свою очередь, — базовый класс объектов пользовательского интерфейса. Он предоставляет доступ к обработке касаний и рисунков пользователя. Как и другому объекту, экземпляру `UIWindow` выделяется память, после чего он инициализируется. Инициализатор (как вы узнаете из гл. 4) задает визуальные рамки объекта окна. Затем создается и инициализируется экземпляр класса `MyView` со своими рамками. После конфигурирования экземпляра `MyView` и установки цвета его фона мы добавляем его как дочернее представление к окну и делаем объект окна ключевым и видимым.

```
#import  "HelloWorldAppDelegate.h"
#import  "MyView.h"
```

```
@implementation HelloWorldAppDelegate
-(void)
applicationDidFinishLaunching:(UIApplication *) application {
    window = [[UIWindow alloc]
          initWithFrame : [[UIScreen mainScreen] bounds]];
    CGRect frame = [UIScreen mainScreen].applicationFrame;
    view = [[MyView alloc] initWithFrame : frame];
    view.message = @"Hello World!";
    view.backgroundColor = [UIColor whiteColor];
    [window addSubview : view];
    [window makeKeyAndVisible];
}
-(void) dealloc {
 [view release];
 [window release];
 [super dealloc];
}
@end
```

3. Создайте подклассы пользовательского интерфейса. Чтобы обрабатывать пользовательские события (например, касания) и рисовать внутри представления, нужно создать подкласс `UIView` и переопределить его методы, ответственные за обработку событий и рисование. Ниже приведено объявление класса `MyView`, используемого в нашем приложении `Hello World`.

```
#import <UIKit/UIKit.h>

@interface MyView: UIView {
    NSString *message;
}
@property(nonatomic, copy) NSString *message;
@end
```

Далее приведена реализация класса `MyView`. Он переопределяет методы-обработчики событий для определения конечных точек касания (мы рассмотрим multitouch-интерфейс в следующей главе) и метод `drawRect:` для рисования в пользовательской области. Для рисования мы просто отрисовываем содержимое переменной экземпляра `message` с заданным шрифтом. В момент когда палец пользователя поднимается с поверхности экрана, это событие сохраняется в очереди приложения. Объект программы извлекает событие из очереди и посылает его главному окну. Окно производит поиск по своим дочерним представлениям того представления, которое должно получить это событие, и доставляет это событие нужному представлению. В нашем примере экземпляр `MyView` развернут во весь экран, поэтому все события касаний будут доставлены ему и вызовется метод `touchesEnded:withEvent:`. Можете добавить свой код в этот метод, чтобы обновить состояние приложения, или изменить его внешний вид, или и то и другое.

```
#import "MyView.h"

@implementation MyView
@synthesize message;
```

```
- (void)
touchesEnded : (NSSet *)touches  withEvent:(UIEvent *)event {
    if([(UITouch*) [touches  anyObject]  tapCount]  ==  2) {
      //обрабатываем двойное касание
    }
}
- (void)drawRect:(CGRect)rect {
  [message  drawAtPoint:CGPointMake(100,100)
    withFont:[UIFont  systemFontOfSize:32]];
}
@end
```

3.2. Создание приложения HelloWorld

Для реализации приложения `HelloWorld` вам необходимо пройти следующие этапы.

1. Создайте новый проект в XCode — выполните команду **File** (Файл) ▸ **New Project** (Новый проект), затем выберите шаблон **Window-Based Application** (Приложение с оконным интерфейсом) (рис. 3.1).

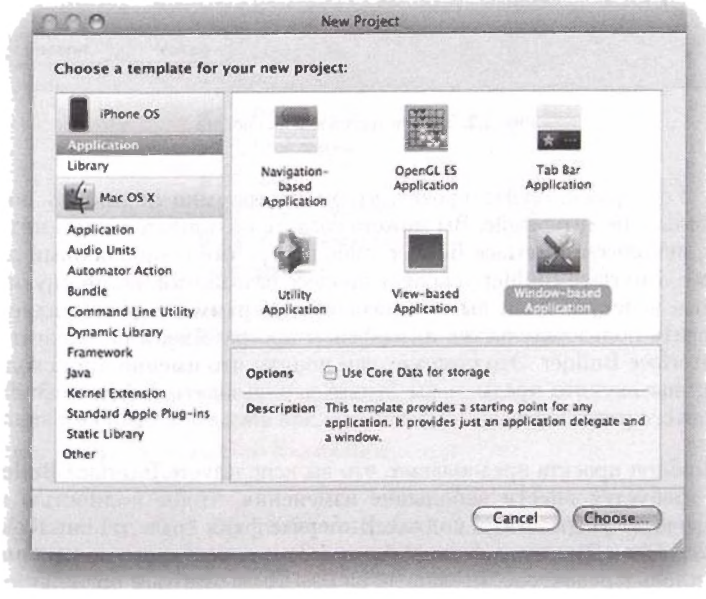

Рис. 3.1. Выбор шаблона нового проекта

Назовите проект `HelloWorld` и нажмите **Save** (Сохранить) (рис. 3.2).

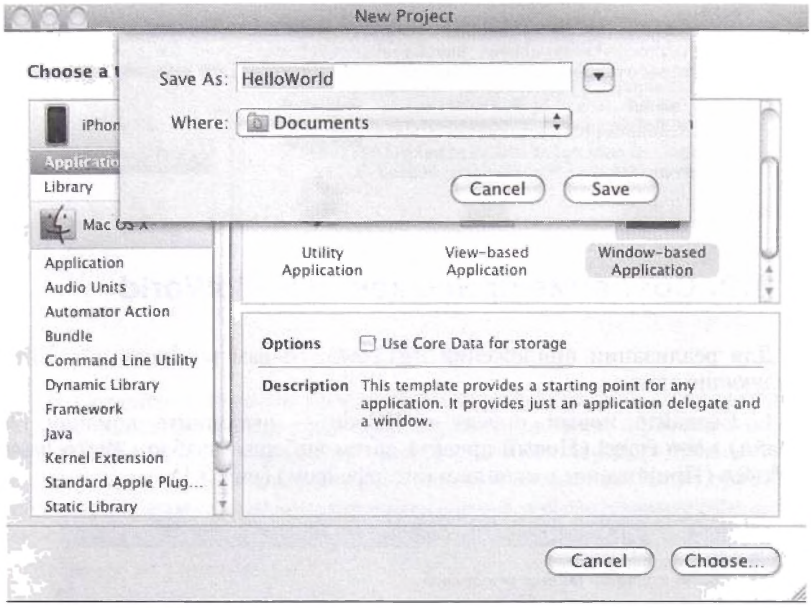

Рис. 3.2. Задание имени нового проекта

2. Усовершенствуйте проект, чтобы программно построить пользовательский интерфейс. Вы можете создать пользовательский интерфейс, используя Interface Builder, либо программно, либо обоими способами. Interface Builder ускоряет процесс разработки, но он опускает важные аспекты. Если вы начинающий программист, рекомендуется создавать пользовательские интерфейсы программно, а не полагаться на Interface Builder. Это поможет вам понять, что именно происходит. Когда вы изучите предмет, вы будете использовать Interface Builder в процессе разработки.

Шаблон проекта предполагает, что вы используете Interface Builder. Вам требуется внести небольшие изменения, чтобы полностью использовать программный подход. Выберите файл `Info.plist` в окне **Groups & Files** (Группы и файлы) (рис. 3.3) так, чтобы его содержимое появилось в редакторе. Щелкните на **Main nib file base name** правой кнопкой мыши и выполните команду **Cut** (Вырезать). Щелкните правой кнопкой мыши на файле `MainWindow.xib` в окне **Groups & Files** (Группы и файлы) и выберите **Delete** (Удалить), а затем — **Also Move to Trash** (Также переместить в мусор).

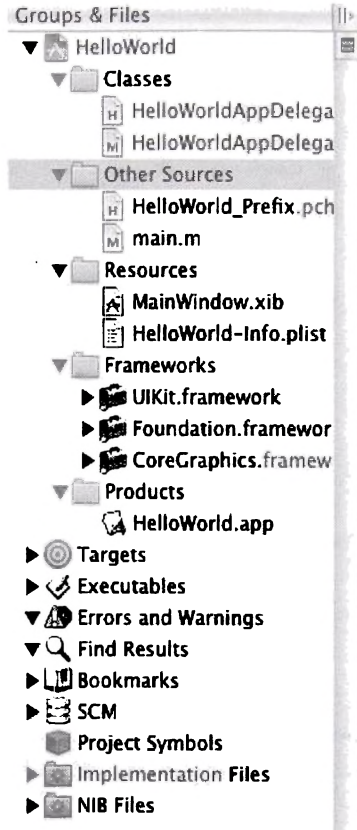

Рис. 3.3. Окно Groups & Files (Группы и файлы)

Выберите файл `main.m` и измените вызов `UIApplicationMain()`, добавив имя класса-делегата приложения, как показано ниже:

```
int retVal = UIApplicationMain(argc, argv, nil,
    @"HelloWorldAppDelegate");
```

3. Напишите свой код. Выберите файл `HelloWorldAppDelegate.h` и замените его содержимое листингом, описанным в предыдущем разделе. То же сделайте для файла `HelloWorldAppDelegate.m`.

4. Создайте подкласс `UIView`. Для этого выполните команду **File** (Файл) ▸ **New File** (Новый файл) ▸ **UIView subclass** (Подкласс UIView) и нажмите **Next** (Далее) (рис. 3.4).

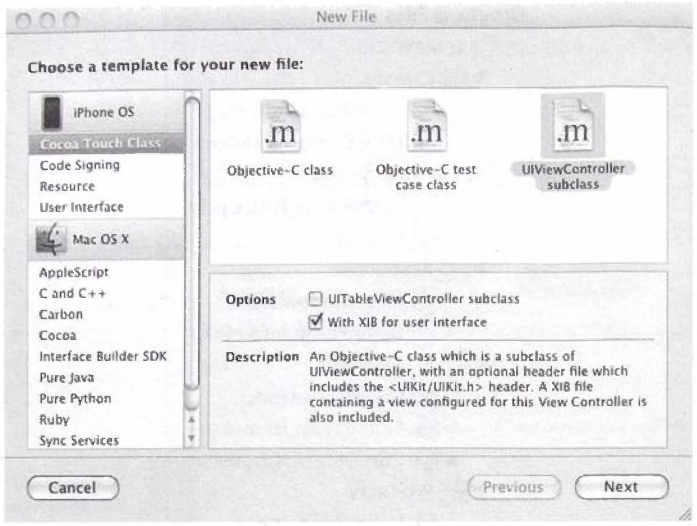

Рис. 3.4. Выбор шаблона нового файла

Назовите файл `MyView.m` и нажмите Finish (Готово) (рис. 3.5). Будет создан подкласс `UIView`. Замените содержимое `MyView.h` и `MyView.m` листингами, приведенными в предыдущем разделе.

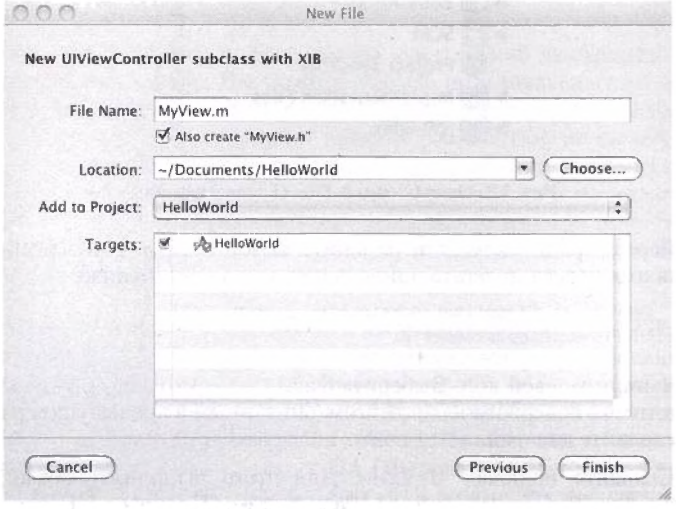

Рис. 3.5. Задание имени нового подкласса UIView

5. Скомпилируйте и запустите приложение. Нажмите кнопку Build and Go (Создать и запустить) (рис. 3.6) для компиляции и запуска приложения (рис. 3.7).

Рис. 3.6 Панель инструментов XCode

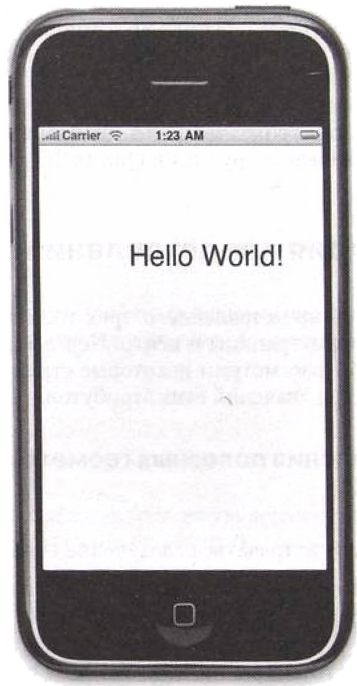

Рис. 3.7. Приложение HelloWorld

Глава 4
Представление

Эта глава разъясняет основные концепции представлений. Из разд. 4.1 вы узнаете о геометрии представлений. В разд. 4.2 мы рассмотрим тему иерархии представлений. В разд. 4.3 будет подробно описан multitouch-интерфейс. В этом разделе вы изучите множественные касания. В разд. 4.4 будут рассмотрены некоторые приемы анимации, а в разд. 4.5 — использование функций Quartz 2D для рисования внутри представлений.

4.1. Геометрия представления

Необходимо иметь представление о трех геометрических атрибутах класса `UIView` — фрейм, границы и центр. Перед тем как перейти к описанию этих атрибутов, рассмотрим некоторые структуры и функции, используемые для задания значений этих атрибутов.

4.1.1. Определения полезных геометрических типов данных

Далее в тексте будут встречаться следующие типы.

- `CGFloat` представляет числа с плавающей точкой и объявлен как

```
typedef float CGFloat;
```

- `CGPoint` — это структура, представляющая геометрическую точку. Она объявлена как

```
struct CGPoint {
    CGFloat x;
    CGFloat y;
};
typedef struct CGPoint CGPoint;
```

Здесь `x` представляет x-координату точки, а `y` — ее y-координату.

Вы будете использовать `CGPoint` достаточно часто. `CGPointMake()` — удобная функция для инициализации `CGPoint` из пары значений `x` и `y`. Она объявлена следующим образом:

```
CGPoint CGPointMake (
    CGFloat x,
    CGFloat y
);
```

- `CGSize` — структура, используемая для представления значений длины и высоты, объявлена как

```
struct CGSize {
    CGFloat width;
    CGFloat height;
};
typedef struct CGSize CGSize;
```

Здесь `width` — значение длины, а `height` — высоты.

Чтобы создать `CGSize` из значений длины и высоты, используйте вспомогательную функцию `CGSizeMake()`, объявленную следующим образом:

```
CGSize CGSizeMake (
    CGFloat width,
    CGFloat height
);
```

- `CGRect` используется для определения положения и размеров прямоугольника и объявлена следующим образом:

```
struct CGRect {
    CGPoint origin;
    CGSize size;
};
typedef struct CGRect CGRect;
```

Значение `origin` представляет левую верхнюю точку прямоугольника, а `size` — его размеры (длину и высоту).

Для создания структуры `CGRect` можно использовать вспомогательную функцию `CGRectMake()`, объявленную как

```
CGRect CGRectMake (
    CGFloat x,
    CGFloat y,
    CGFloat width,
    CGFloat height
);
```

4.1.2. Класс UIScreen

Класс `UIScreen` предоставляется для получения размеров экрана устройства. Разрешение его экрана — 320 × 480 (рис. 4.1).

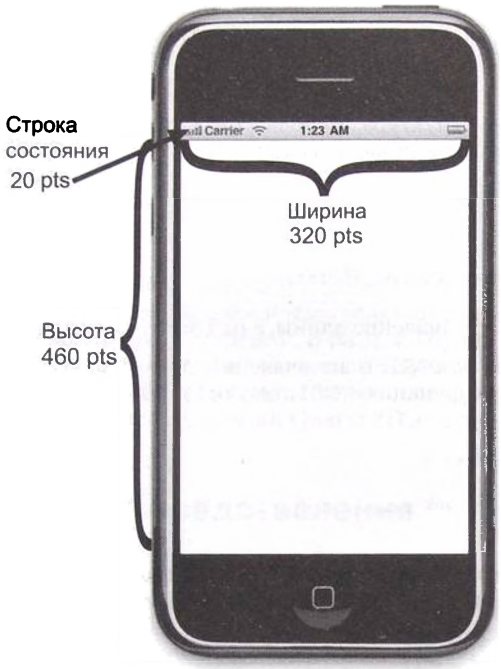

Рис. 4.1. Размеры экрана устройства

Панель статуса занимает 20 точек от общей высоты, оставляя 460 точек приложению. Вы можете отключить панель статуса с помощью следующего выражения:

```
[UIApplication sharedApplication].statusBarHidden = YES;
```

Получить размер экрана устройства можно таким образом:

```
[[UIScreen mainScreen] bounds].size
```

В данном выражении мы сначала получаем экземпляр синглетона `UIScreen`, а затем — размеры ограничивающего его прямоугольника.

Окно приложения располагается сразу под панелью статуса. Чтобы получить фрейм приложения, используйте следующее выражение:

```
CGRect frame = [[UIScreen mainScreen] applicationFrame]
```

Если есть панель статуса, размеры фрейма приложения будут 320 × 460. В противном случае они будут равны размерам всего экрана.

4.1.3. Атрибуты frame и center

В классе `UIView` объявлено свойство `frame`, используемое для получения позиции и размеров экземпляра `UIView` внутри другого экземпляра `UIView`. Свойство объявлено следующим образом:

```
@property(nonatomic) CGRect frame
```

Обычно вы задаете фрейм представления во время фазы инициализации. Например, следующий код создает экземпляр `UIView`, чей левый верхний угол расположен в точке родительского представления с координатами (50, 100), и его длина и высота равны 150 и 200.

```
CGRect  frame = CGRectMake(50, 100, 150, 200);
aView = [[UIView  alloc]  initWithFrame:frame];
[window  addSubview:aView];
```

Ниже показан результат добавления экземпляра `UIView` к полноэкранному (за вычетом панели статуса) окну (рис. 4.2). На иллюстрации изображен левый верхний угол данного представления (50, 100) и его центр (125, 200), все в координатах родительского представления (окна).

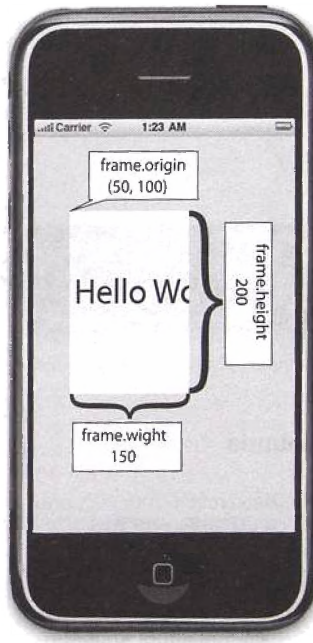

Рис. 4.2. Геометрические атрибуты frame и center дочернего представления главного окна

Изменение `center` повлечет за собой изменение позиции левого верхнего угла фрейма. Точно так же изменения позиции левого верхнего угла или размера фрейма изменят и центр. Если в приведенном выше примере увеличить *x*-координату атрибута `center` на 80 точек, то координаты левого верхнего угла станут (130, 100), что повлечет за собой сдвиг представления вправо на 80 точек (рис. 4.3).

Рис. 4.3. Смещение позиции представления путем изменения атрибута center

4.1.4. Атрибут bounds

Атрибут `bounds` используется для задания левого верхнего угла и размера представления в его собственных координатах. Свойство объявлено как

```
@property(nonatomic) CGRect bounds
```

При инициализации представления координаты левого верхнего угла `bounds` устанавливаются равными `(0,0)`, а его размеры — равными

frame.size. Изменения bounds.origin не влияют на его атрибуты frame и center. Тем не менее изменение bounds.size повлечет за собой изменение свойств frame и center.

Взгляните еще раз на рис. 4.2: bound.origin равен (0,0). Представление отображает строковое значение следующим образом:

```
-(void)drawRect:(CGRect) rect {
  int x = 0;
  int y = self.bounds.size.height/3;
  [@"Hello World!"
    drawAtPoint:CGPointMake(x, y)
    withFont:[UIFont systemFontOfSize:40]];
}
```

Координата *x* точки, с которой начинается отрисовка строки Hello World, равна 0. Если мы изменим значение bounds.origin.x с 0 на 50, нарисованная строка сдвинется на 50 точек влево (рис. 4.4).

Рис. 4.4. Изменение свойства bounds.origin влияет на содержимое представления, но не на его размеры/положение

4.2. Иерархия представлений

В большинстве случаев у вас будет одно окно приложения и несколько представлений и элементов управления с различными размерами и позициями. Главное окно (экземпляр `UIWindow`, являющийся подклассом `UIView`) выступает в качестве корня дерева. Чтобы добавить к приложению еще одно представление, вы добавляете последнее к окну или уже существующему представлению. В итоге вы получите древовидную структуру, корнем которой будет исходное окно. Каждое представление будет иметь только одного родителя и ни одного или более дочерних представлений. Чтобы получить доступ к экземпляру родительского представления, используйте атрибут `superview`, объявленный следующим образом:

```
@property(nonatomic, readonly) UIView *superview
```

Чтобы получить потомков данного представления, используйте атрибут `subviews`, объявленный так:

```
@property(nonatomic, readonly, copy) NSArray *subviews
```

Чтобы добавить представление к уже существующему, для начала выделите для него память, инициализируйте его, сконфигурируйте, а затем добавьте его как дочернее. Следующие два выражения создают представление, занимающее весь экран (за вычетом панели статуса).

```
CGRect frame = [UIScreen mainScreen].applicationFrame;
view1 = [[UIView alloc] initWithFrame:frame];
```

В качестве инициализатора обычно применяется метод `initWithFrame:`. Чтобы добавить дочернее представление, используйте метод `addSubview:`, объявленный следующим способом:

```
- (void)addSubview:(UIView *) view
```

После вызова этого метода родительское представление захватит новый экземпляр представления. Чтобы удалить представление из иерархии, используйте метод `removeFromSuperview`. Кроме удаления представления из дерева, этот метод также высвобождает его.

4.3. Multitouch-интерфейс

Когда пользователь дотрагивается до экрана, он ожидает от приложения отклика. Программа состоит из нескольких представлений, дочерних представлений и элементов управления, поэтому система должна определять, какой объект является приемником касаний пользователя.

Каждое приложение имеет единственный объект `UIApplication` для обработки пользовательских касаний. Когда пользователь касается экра-

на, система пакует касания в объект события и помещает его в очередь событий приложения. Этим объектом события является класс `UIEvent`.

Объект события содержит все одновременные касания экрана. Каждый палец на экране имеет собственный объект касания, экземпляр класса `UITouch`. Как вы увидите позже, каждый объект касания может находиться в различных фазах, например, «только что коснулся», «движется», «стационарный» и т. д. Каждый раз, когда пользователь дотрагивается до экрана, объект события и объекты касаний изменяются, чтобы отразить текущую ситуацию.

Уникальный экземпляр `UIApplication` извлекает объект события из очереди и посылает его объекту ключевого окна. Объект окна посредством механизма, называемого проверкой попадания, определяет, какое дочернее представление должно получить данное событие, и передает это событие ему. Этот объект называется первым откликнувшимся. Если объекту требуется обработать событие, он делает это, и событие помечается как доставленное. Если объекту не нужно обрабатывать это событие, он передает его через связанный лист объектов, называемый цепочкой ответчиков.

Цепочка ответчиков данного объекта начинается с него самого и заканчивается объектом приложения. Если какой-либо объект из цепочки принимает сообщение, то продвижение события в сторону экземпляра приложения останавливается. Если экземпляр программы получает событие и не знает его корректного адресата, он отбрасывает это событие.

4.3.1. Класс UITouch

Каждый палец, касающийся экрана, инкапсулируется объектом класса `UITouch`. Рассмотрим некоторые важные свойства и методы класса.

- `phase` — это свойство используется для определения текущей фазы касания. Оно объявлено следующим образом:

```
@property(nonatomic, readonly) UITouchPhase phase
```

Доступны несколько значений `UITouchPhase`:

- `UITouchPhaseBegan`, показывающее, что палец дотронулся до экрана;
- `UITouchPhaseMoved`, означающее, что палец движется по экрану;
- `UITouchPhaseStationary`, показывающее, что палец не двигался по экрану со времени последнего события;
- `UITouchPhaseEnded`, означающее, что палец убрали с экрана;
- `UITouchPhaseCancelled`, показывающее, что касание было отменено системой.

- `timestamp` — время, когда касание сменило фазу. Объект `UITouch` продолжает меняться в течение всего события. Это значение ссылается на последнее изменение.
- `tapCount` — количество постукиваний пользователя, когда он дотронулся до экрана. Удачное постукивание по одному и тому же месту будет иметь результатом значение данной переменной больше 1. Свойство объявлено следующим образом:

```
@property(nonatomic, readonly) NSUInteger tapCount
```

- `locationInView:` — этот метод возвращает положение касания в данном представлении. Метод объявлен следующим образом:

```
- (CGPoint)locationInView:(UIView *)view
```

Значение возвращается в координатах представления. Если вы передадите `nil`, возвратится значение в координатах главного окна.

- `previousLocationInView:` — с помощью этого метода в данном представлении можно получить предыдущее положение касания. Метод объявлен следующим образом:

```
- (CGPoint)previousLocationInView:(UIView *)view
```

4.3.2. Класс UIEvent

Последовательность касаний отслеживается объектом класса `UIEvent`. Приложение будет получать все тот же объект `UIEvent` в течение всей своей работы. Этот объект будет изменяться во время выполнения программы. Вы можете получить текущий момент времени этого объекта, используя свойство `timestamp`. Чтобы получить касания, которые представляет этот объект, используйте метод `allTouches`, объявленный следующим образом:

```
- (NSSet *) allTouches
```

4.3.3. Класс UIResponder

Объекты интерфейса, которых касается пользователь, такие как экземпляры `UIView`, являются подклассами класса `UIResponder`. Чтобы разобраться в multitouch-интерфейсе, вы должны иметь представление о классе `UIResponder` и его четырех главных методах обработки множественных касаний.

Ниже приведены основные методы, которые нужно переопределить наследникам класса `UIResponder` (таким как подклассы `UIView`), чтобы обрабатывать жесты пользователя.

- `touchesBegan: withEvent:` — вызывается для извещения реагирующего объекта о том, что один или более пальцев только что коснулись экрана. Метод объявлен следующим образом:

```
-(void) touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
```

Первый параметр — множество объектов `UITouch`, которые описывают касания экрана. Второй параметр — событие, с которым ассоциированы данные касания.

- `touchesMoved: withEvent:` — вызывается для оповещения реагирующего объекта о том, что один или более пальцев только что двигались по экрану. Метод объявлен так:

```
-(void) touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event
```

Первый параметр — множество объектов `UITouch`, которые описывают двигавшиеся касания экрана. Второй параметр — событие, с которым ассоциированы эти касания.

- `touchesEnded: withEvent:` — вызывается для извещения реагирующего объекта о том, что один или более пальцев поднялись с экрана. Метод объявлен следующим образом:

```
-(void) touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event
```

Первый параметр — множество объектов `UITouch`, которые описывают места на экране, откуда пользователь убрал пальцы. Второй параметр — событие, с которым ассоциированы данные касания.

- `touchesCancelled: withEvent:` — вызывается системой для извещения реагирующего объекта о том, что событие было отменено. Метод объявлен так:

```
-(void) touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event
```

Первый параметр — множество, содержащее единственный объект `UITouch`, чья фаза — `UITouchPhaseCancel`. Второй параметр — событие, которое было отменено.

Понять механизм множественных касаний проще всего на подробном примере. Представим три пальца F1, F2 и F3, касающиеся экрана, двигающиеся и поднимающиеся с него в различные моменты времени. Изучим вызовы методов как результаты действий пальцев. Для каждого вызова мы рассмотрим содержимое множеств `touches` и `allTouches` объекта события. Начнем с пустого экрана.

1. Два пальца, F1 и F2, коснулись экрана.

Происходит вызов `touchesBegan: withEvent:`.

`touches`: множество из двух элементов.

Касание T1, представляющее F1: <UITouch: 0x14a360>, состояние — начало.

Касание T2, представляющее F2: <UITouch: 0x14a0f0>, состояние — начало.

event:<UIEvent: 0x143ae0>. Множество allTouches:

T1: <UITouch: 0x14a360>, состояние — начало.

T2: <UITouch: 0x14a0f0>, состояние — начало.

2. Пальцы F1 и F2 сдвинулись.

Происходит вызов touchesMoved: withEvent:.

touches: множество из двух элементов.

Касание T1, представляющее F1: <UITouch: 0x14a360>, состояние — сдвинут.

Касание T2, представляющее F2: <UITouch: 0x14a0f0>, состояние — сдвинут.

event:<UIEvent: 0x143ae0>. Множество allTouches:

T1: <UITouch: 0x14a360>, состояние — сдвинут.

T2: <UITouch: 0x14a0f0>, состояние — сдвинут.

3. Палец F1 сдвинулся.

Происходит вызов touchesMoved: withEvent:.

touches: множество из одного элемента.

Касание T1, представляющее F1: <UITouch: 0x14a360>, состояние — сдвинут.

event:<UIEvent: 0x143ae0>. Множество allTouches:

T1: <UITouch: 0x14a360>, состояние — сдвинут.

T2: <UITouch: 0x14a0f0>, состояние — неподвижен.

4. Палец F2 сдвинулся.

Происходит вызов touchesMoved: withEvent:.

touches: множество из одного элемента.

Касание T2, представляющее F2: <UITouch: 0x14a0f0>, состояние — сдвинут.

event:<UIEvent: 0x143ae0>. Множество allTouches:

T1: <UITouch: 0x14a360>, состояние — неподвижен.

T2: <UITouch: 0x14a0f0>, состояние — сдвинут.

5. Палец F3 дотронулся до экрана, палец F2 сдвинулся.

Происходит вызов `touchesMoved: withEvent:`.

`touches`: множество из одного элемента.

Касание T3, представляющее F3: `<UITouch: 0x145a10>`, состояние — начало.

`event:<UIEvent: 0x143ae0>`. Множество `allTouches`:

T1: `<UITouch: 0x14a360>`, состояние — неподвижен.

T2: `<UITouch: 0x14a0f0>`, состояние — сдвинут.

T3: `<UITouch: 0x145a10>`, состояние — начало.

Происходит вызов `touchesMoved: withEvent:`.

`touches`: множество из одного элемента.

T2: `<UITouch: 0x14a0f0>`, состояние — сдвинут.

`event:<UIEvent: 0x143ae0>`. Множество `allTouches`:

T1: `<UITouch: 0x14a360>`, состояние — неподвижен.

T2: `<UITouch: 0x14a0f0>`, состояние — сдвинут.

T3: `<UITouch: 0x145a10>`, состояние — начало.

6, Пальцы F2 и F3 сдвинулись.

Происходит вызов `touchesMoved: withEvent:`.

`touches`: множество из одного элемента.

Касание T2, представляющее F2: `<UITouch: 0x14a0f0>`, состояние — сдвинут.

Касание T3, представляющее F3: `<UITouch: 0x145a10>`, состояние — сдвинут.

`event:<UIEvent: 0x143ae0>`. Множество `allTouches`:

T1: `<UITouch: 0x14a360>`, состояние — неподвижен.

T2: `<UITouch: 0x14a0f0>`, состояние — сдвинут.

T3: `<UITouch: 0x145a10>`, состояние — сдвинут.

7. Палец F3 поднялся с экрана, палец F2 сдвинулся.

Происходит вызов `touchesMoved: withEvent:`.

`touches`: множество из одного элемента.

Касание T2, представляющее F2: `<UITouch: 0x14a0f0>`, состояние — сдвинут.

event:<UIEvent: 0x143ae0>. Множество allTouches:

T1: <UITouch: 0x14a360>, состояние — неподвижен.

T2: <UITouch: 0x14a0f0>, состояние — сдвинут.

T3: <UITouch: 0x145a10>, состояние — конец.

Происходит вызов touchesEnded: withEvent:.

touches: множество из одного элемента.

Касание T3, представляющее F3: <UITouch: 0x145a10>, состояние — конец.

event:<UIEvent: 0x143ae0>. Множество allTouches:

T1: <UITouch: 0x14a360>, состояние — неподвижен.

T2: <UITouch: 0x14a0f0>, состояние — сдвинут.

T3: <UITouch: 0x145a10>, состояние — конец.

8. Палец F2 сдвинулся.

Происходит вызов touchesMoved: withEvent:.

touches: множество из одного элемента.

Касание T2, представляющее F2: <UITouch: 0x14a0f0>, состояние — сдвинут.

event:<UIEvent: 0x143ae0>. Множество allTouches:

T1: <UITouch: 0x14a360>, состояние — неподвижен.

T2: <UITouch: 0x14a0f0>, состояние — сдвинут.

9. Палец F1 поднялся с экрана, палец F2 сдвинулся.

Происходит вызов touchesMoved: withEvent:.

touches: множество из одного элемента.

Касание T2, представляющее F2: <UITouch: 0x14a0f0>, состояние — сдвинут.

event:<UIEvent: 0x143ae0>. Множество allTouches:

T1: <UITouch: 0x14a360>, состояние — конец.

T2: <UITouch: 0x14a0f0>, состояние — сдвинут.

Происходит вызов touchesEnded: withEvent:.

touches: множество из одного элемента.

Касание T1, представляющее F1: <UITouch: 0x14a360>, состояние — конец.

event:<UIEvent: 0x143ae0>. Множество allTouches:

T1: <UITouch: 0x14a360>, состояние — конец.

T2: <UITouch: 0x14a0f0>, состояние — сдвинут.

10. Палец F2 сдвинулся.

Происходит вызов touchesMoved: withEvent:.

touches: множество из одного элемента.

Касание T2, представляющее F2: <UITouch: 0x14a0f0>, состояние — сдвинут.

event:<UIEvent: 0x143ae0>. Множество allTouches:

T2: <UITouch: 0x14a0f0>, состояние — сдвинут.

11. Палец F2 поднялся с экрана.

Происходит вызов touchesEnded: withEvent:.

touches: множество из одного элемента.

Касание T2, представляющее F2: <UITouch: 0x14a0f0>, состояние — конец.

event:<UIEvent: 0x143ae0>. Множество allTouches:

T2: <UITouch: 0x14a0f0>, состояние — конец.

Листинг 4.1 показывает подкласс UIView, который переопределяет три метода отклика и отслеживает касания и события всех трех фаз. Используйте это в приложении, чтобы проверить уровень вашего понимания multitouch-интерфейса.

Листинг 4.1. Подкласс UIView, переопределяющий три метода отклика и отслеживающий касания и события трех фаз

```
@interface ViewOne: UIView ()
@end

@implementation ViewOne

-(void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
  for(UITouch *t in touches)
    NSLog(@"B: touch: %@", t);
    NSLog(@"B: event: %@", event);
}

-(void) touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event {
  for(UITouch *t in touches)
    NSLog(@"M: touch: %@", t);
    NSLog(@"M: event: %@", event);

-(void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event {
  for(UITouch *t in touches)
    NSLog(@"E: touch: %@", t);
    NSLog(@"E: event: %@", event);
}
@end
```

4.3.4. Обработка скольжений

В этом подразделе мы рассмотрим перехват различных фаз пользовательских касаний для определения скользящего жеста. Приложение, которое нам предстоит построить, будет распознавать правое/левое скольжение и выводить его скорость (в точках в секунду) в представление.

Листинг 4.2 показывает объявление класса-делегата приложения. Делегат приложения `SwipeAppDelegate` использует представление `SwipeDemoView` в качестве основного представления программы.

Листинг 4.2. Объявление класса-делегата приложения SwipeAppDelegate

```
#import <UIKit/UIKit.h>
#import "SwipeDemoView.h"

@interface SwipeAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow * window;
    SwipeDemoView *viewOne;
}

@property (nonatomic, retain) UIWindow *window;
@end
```

Листинг 4.3 демонстрирует реализацию класса-делегата приложения. Метод `applicationDidFinishLaunching:` создает экземпляр класса представления `SwipeDemoView` и делает его доступным для обработки множественных касаний, устанавливая свойство `multiTouchEnabled` в `YES`. Если этого не сделать, то множество `touches` в четырех методах будет всегда иметь размер 1.

Листинг 4.3. Реализация класса-делегата приложения SwipeAppDelegate

```
#import  "SwipeAppDelegate.h"

@implementation  SwipeAppDelegate
@synthesize  window;
- (void) applicationDidFinishLaunching:(UIApplication *)application {
    Window = [[UIWindow  alloc]
              initWithFrame : [[UIScreen mainScreen]  bounds]];
    CGRect frame = [UIScreen mainScreen].applicationFrame;
    viewOne = [[SwipeDemoView  alloc]  initWithFrame:frame];
    viewOne.multipleTouchEnabled  = YES;
    viewOne.backgroundColor = [UIColor  whiteColor];
    [window  addSubview:viewOne];
    [window  makeKeyAndVisible];
}

- (void)dealloc {
    [viewOne release];
    [window release];
    [super dealloc];
}
@end
```

Представление будет продолжать отслеживать время и место двух касаний. Оно также использует переменную state для распознавания скольжения. Если представление находится в state S0, то мы не получили никаких касаний. Если оно находится в state S1, то мы получили одно касание и ожидаем поднятия пальца. Листинг 4.4 демонстрирует объявление класса представления SwipeDemoView. Обратите внимание, что мы располагаем двумя переменными экземпляра для местонахождения и двумя переменными экземпляра для времени. Время задается в NSTimeInterval (double), которое измеряется в секундах.

Листинг 4.4. Объявление класса представления SwipeDemoView

```
#import <UIKit/UIKit.h>

typedef enum {
    S0,
    S1
}   STATE;

@interface  SwipeDemoView: UIView {
    CGPoint startLocation, endLocation;
    NSTimeInterval  startTime,  endTime;
    STATE  state;
}
@end
```

Проанализируем логику распознавания скольжения и вывода его скорости. Листинг 4.5 показывает метод touchesBegan:withEvent: класса UIResponder, переопределенного классом SwipeDemoView. В этом методе мы сначала хотим удостовериться, что до этого момента мы не получили никаких касаний (то есть находимся в состоянии S0). Мы также желаем убедиться, что количество касаний в touches объекта event и количество элементов в объекте touches одинаково и равняется 1. Удостоверившись, что это состояние сохраняется, мы запомним начальное время и местоположение касания и перейдем в состояние S1.

Листинг 4.5. Метод touchesBegan:withEvent, который используется в приложении, определяющем скольжения

```
- (void)touchesBegan:(NSSet *)touches  withEvent:(UIEvent *)event {
   int noTouchesInEvent = ((NSSet *)[event  allTouches]).count;
   int noTouchesBegan = touches.count;
   NSLog(@"began %i, total %i",
      noTouchesBegan,  noTouchesInEvent);
   if((state == S0) &&
    (noTouchesBegan == 1) &&
    (noTouchesInEvent == 1)){
    startLocation =
            [(UITouch*)[touches  anyObject]
                    locationInView:self];
    startTime = [(UITouch*)[touches  anyObject]  timestamp];
    state = S1;
   }
```

```
    else {
     state = S0;
     [self  setNeedsDisplay];
    }
}
```

Листинг 4.6 демонстрирует метод `touchesEnded:withEvent:`. В этом методе мы убеждаемся, что находимся в состоянии `S1` (то есть мы начали с одного касания, и оно заканчивается). Мы также удостоверимся, что это последнее касание экрана. Сделаем это путем проверки того, что количество касаний в объекте `event` равно количеству элементов в `touches` и составляет 1. Получив данные условия, мы записываем местоположение и время касания и выводим результат пользователю.

Листинг 4.6. Метод touchesEnded:withEvent, который используется в приложении, определяющем скольжения

```
- (void) touchesEnded:(NSSet *)touches  withEvent:(UIEvent *)event {
    int noTouchesInEvent = ((NSSet*)[event  allTouches]).count;
    int noTouchesEnded = touches.count;
    NSLog(@"ended %i %i", touches.count,
     ((NSSet*)[event allTouches]).count);
    if((state == S1) && (noTouchesEnded == 1) &&
    (noTouchesInEvent == 1)){
       endLocation =
            [(UITouch *)[touches  anyObject]  locationInView:self];
       endTime =
            [(UITouch*)[touches  anyObject]  timestamp];
       [self  setNeedsDisplay];
    }
}
```

Листинг 4.7 показывает остаток определения класса `SwipeDemoView`. Метод `drawRect:` представляет пользователю информацию о скольжении. Если состояние равно `S0`, мы очищаем статистику предыдущего скольжения. Если состояние `S1`, мы проверяем, был ли жест скольжением. Следующее выражение проверяет, во-первых, абсолютная разность y-координат начала и конца касания меньше или равна значению `Y_TOLERANCE`, и, во-вторых, абсолютная разность x-координат начала и конца касания больше либо равна значению `X_TOLERANCE`.

```
if((fabs(startLocation.y - endLocation.y)  <=  Y_TOLERANCE)  &&
   (fabs(startLocation.x - endLocation.x)  >=  X_TOLERANCE)
```

Значения чувствительности определены следующим образом:

```
#define Y_TOLERANCE 20
#define X_TOLERANCE 100
```

Вы можете задать наиболее подходящие для вашего приложения значения.

Распознав скольжение, мы определяем его направление, используя следующее выражение:

```
direction = (endLocation.x > startLocation.x) ? "right" : "left";
```

Наконец, мы определяем скорость скольжения с помощью следующего выражения:

```
fabs (endLocation.x - startLocation.x) /(endTime-startTime)
```

Результат изображен на рис. 4.5.

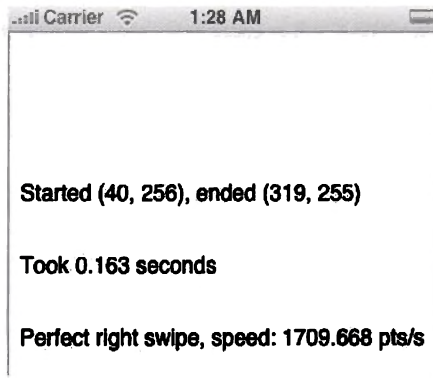

Рис. 4.5. Снимок экрана приложения, распознающего скольжения, показывает превосходное скольжение вправо

Листинг 4.7. Остаток определения класса SwipeDemoView

```
- (id)initWithFrame:(CGRect)frame {
   if(self = [super initWithFrame : frame]) {
    state = S0;
   }
   return self;
}

- (void)drawRect:(CGRect)rect {
   NSString *message;
   if(state == S0) {
    [@"\t\t\t\t\t\t\t\t\t\t" drawAtPoint:CGPointMake(10,100)
      withFont:[UIFont systemFontOfSize:16]];
    [@"\t\t\t\t\t\t\t\t\t\t" drawAtPoint:CGPointMake(10,150)
      withFont:[UIFont systemFontOfSize:16]];
    [@"\t\t\t\t\t\t\t\t\t\t" drawAtPoint:CGPointMake(10,200)
      withFont:[UIFont systemFontOfSize:16]];
   }
   if(state == S1) {
    message = [NSString
      stringWithFormat:
```

```
        @"Started (%.0f, %.0f), ended (%.0f, %.0f)",
      startLocation.x, startLocation.y,
      endLocation.x, endLocation.y
      ];
    [message drawAtPoint:CGPointMake(10,100)
      withFont:[UIFont systemFontOfSize:16]];
    message = [NSString
      stringWithFormat:
        @"Took %4.3f seconds", endTime-startTime
      ];
    [message drawAtPoint:CGPointMake(10,150)
      withFont:[UIFont systemFontOfSize:16]];
    if((fabs(startLocation.y - endLocation.y) <= Y_TOLERANCE)
    &&
      (fabs(startLocation .x - endLocation .x) >= X_TOLERANCE)
      ){
      char *direction;
      direction =
         (endLocation.x > startLocation.x) ? "right" : "left";
      message = [NSString stringWithFormat:
           @"Perfect %s swipe, speed: %4.3f pts/s", direction,
           (endTime-startTime) > 0 ?
                 fabs(endLocation.x - startLocation.x) /
                       (endTime-startTime):0
      ];
      [message drawAtPoint:CGPointMake(10,200)
           withFont:[UIFont systemFontOfSize:16]];
    }
    else {
      [@"\t\t\t\t\t\t\t\t\t\t" drawAtPoint:CGPointMake(10,200)
           withFont:[UIFont systemFontOfSize:16]];
    }
    state = S0;
  }
}
```

Ничего страшного, что этот алгоритм распознавания жестов не принимает во внимание промежуточные движения касания. Для этого вам нужно переопределить метод `touchesMoves:withEvent:` и удостовериться, что переменная `Y_TOLERANCE` не нарушается.

4.3.5. Более совершенное распознавание жестов

В этом подразделе мы рассмотрим еще одно приложение, работающее с жестами со множественными касаниями. Оно распознает следующий жест: два пальца касаются экрана одновременно или, по крайней мере, в течение двух секунд. Далее оба пальца двигаются вместе или отдельно. В конце два пальца поднимаются с экрана одновременно. Приложение выводит следующую статистику: во-первых, каков процент времени, когда оба пальца двигались вместе, и, во-вторых, какова средняя дистанция (в точках) между двумя пальцами.

Делегат приложения идентичен изученному вами в предыдущем разделе. Единственное различие — собственный класс представления `ResponderDemoView`. Листинг 4.8 показывает объявление класса пред-

ставления. Мы определяем три состояния: S0 (начальное состояние), S1 (когда мы получили два касания в приемлемое время и можем обрабатывать статистику) и S2 (когда мы получили только одно касание и ждем секунду). Мы отслеживаем текущее состояние с помощью переменной экземпляра `state`. Переменные `moveTogether` и `moveSeperately` записывают, соответственно, количество движений обоими пальцами вместе и количество движений пальцами отдельно. Общее расстояние между пальцами накапливается в переменной `accDistance`. Дополнительно информация о первом касании (в случае задерживающегося второго) кэшируется в двух переменных — `firstTouchLocInView` и `firstTouchTimeStamp`.

Листинг 4.8. Объявление класса представления ResponderDemoView

```
#import <UIKit/UIKit.h>
typedef enum {
    S0,
    S1,
    S2
}   STATE;
@interface ResponderDemoView:UIView {
    STATE  state;
    float movedTogether, movedSeperate;
    float accDistance;
    CGPoint firstTouchLocInView;
    NSTimeInterval firstTouchTimeStamp;
}
@end
```

Листинг 4.9 демонстрирует метод `touchesBegan:withEvent:` для приложения с усовершенствованным отслеживанием жестов. У этого метода есть три главных раздела. Первый проверяет, коснулись ли экрана одновременно оба пальца. В этом случае метод меняет состояние на S1 и инициализирует переменные для сбора статистики. Начальное расстояние также подсчитывается и используется для инициализации переменной, накапливающей дистанцию. Расстояние в точках подсчитывается с использованием функции `distance()`, приведенной ниже.

```
float  distance (CGPoint a,  CGPoint b) {
    return  sqrt(pow((a.x - b.x), 2) + pow((a.y - b.y), 2));
}
```

Если оба пальца не используются одновременно, мы проверяем, является ли это одиночным касанием, и если да, это будет первым полученным касанием. В этом случае мы переходим в состояние S2 (означающее, что мы получили одно касание и ожидаем одну секунду) и кэшируем информацию о касании.

Если же мы находимся в состоянии S2 и объект `event` насчитывает два касания, мы проверяем, получено ли второе касание в приемлемое время. Следующее выражение проверяет, ниже ли пороговая разница в получении обоих касаний.

```
if ((aTouch.timestamp - firstTouchTimeStamp)  <=
    MAX_ELAPSED_TIME)
```

В этом случае мы переходим в состояние S1, иначе касание расценивается как первое и мы ждем следующего. Значение максимального времени ожидания MAX_ELAPSED_TIME равно двум секундам.

```
#define MAX_ELAPSED_TIME  2
```

Листинг 4.9. Метод touchesBegan:withEvent: для приложения с усовершенствованным отслеживанием жестов

```
- (void) touchesBegan:(NSSet *)touches  withEvent:(UIEvent *)event {
    int noTouchesInEvent = ((NSSet*)[event  allTouches]).count;
    int noTouchesBegan = touches.count;
    NSLog(@"began %i,   total  %i",
          noTouchesBegan, noTouchesInEvent);
    if ((noTouchesBegan ==  2) && (noTouchesInEvent == 2)){
          NSArray  *touchArray = [touches  allObjects];
          state = S1;
          movedTogether = 1;
          movedSeperate = 0;
          accDistance  =
                distance ([[touchArray  objectAtIndex:0]
                      locationInView:self],
                        [[touchArray  objectAtIndex:1]
                      locationInView:self]
                );
    }
    else if((state != S2) && ((noTouchesBegan == 1) &&
          (noTouchesInEvent == 1))) {
          state = S2; // S2 означает, что мы получили первое касание
          UITouch  *aTouch = (UITouch*)[touches anyObject];
          firstTouchTimeStamp = aTouch.timestamp;
          firstTouchLocInView = [aTouch  locationInView:self];
    }
    else if ((state == S2) && (noTouchesInEvent == 2)) {
          UITouch *aTouch = (UITouch*)[touches anyObject];
      if ((aTouch.timestamp - firstTouchTimeStamp)   <=
          MAX_ELAPSED_TIME) {
          // S1 значит, что мы получили второе касание в приемлемое время
          state = S1;
          movedTogether = 1;
          movedSeperate = 0;
          accDistance = distance([aTouch  locationInView:self],
                firstTouchLocInView
                );
      }
      else {
          firstTouchTimeStamp = aTouch.timestamp;
          firstTouchLocInView = [aTouch locationInView:self];
      }
    }
    else state = S0;
}
```

Листинг 4.10 показывает метод touchesMoved:withEvent:. Если количество касаний равно двум и мы находимся в состоянии S1 (собираем статистику), мы увеличиваем на один счетчик movedTogether и обновляем расстояние в accDistance. Если мы получили только одно движение, то инкрементируем счетчик movedSeperate.

Листинг 4.10. Метод touchesMoved:withEvent: для приложения с усовершенствованным отслеживанием жестов

```
- (void) touchesMoved:(NSSet *)touches  withEvent:(UIEvent *)event {
    NSLog(@"moved %i %i", touches.count,
          ((NSSet*)[event  allTouches]).count);
    NSArray  *allTouches = [touches allObjects];
    if((state  == S1) && ([touches  count]  ==  2)) {
      movedTogether++;
      accDistance  +=
            distance ([[allTouches  objectAtIndex:0]
                        locationInView:self],
                      [[allTouches  objectAtIndex:1]
                        locationInView:self]
                      );
    }
    else if((state == S1) && ([touches count] == 1)) {
      movedSeperate++;
    }
}
```

Листинг 4.11 демонстрирует метод `touchesEnded:withEvent:`. Метод удостоверяется, что оба пальца поднялись в одно и то же время, и запрашивает вывод статистики, посылая экземпляру представления сообщение `setNeedsDisplay`. В итоге это приведет к выполнению метода `drawRect:` в листинге 4.13.

Листинг 4.11. Метод touchesEnded:withEvent: для приложения с усовершенствованным отслеживанием жестов

```
- (void)touchesEnded:(NSSet *)touches  withEvent:(UIEvent *)event {
    NSLog(@"ended %i %i",  touches.count,
          ((NSSet*)[event allTouches]).count);
    if((state  ==  S1) && ([touches  count]  ==  2)) {
          NSLog(@"started together and ended together,"
                "moved together %.0f%% "
                "of the time. AVG distance:%4.2f",
                (movedSeperate+movedTogether) ?
                100*(movedTogether/(movedTogether + movedSeperate)):100.0,
                movedTogether ? accDistance/movedTogether: 0.0
          );
          [self  setNeedsDisplay];
    }
    state = S0;
}
```

Если система отменяет событие, то мы сбрасываем переменные, как показано в листинге 4.12.

Листинг 4.12. Переопределенный метод touchesCancelled:withEvent: для приложения с усовершенствованным отслеживанием жестов

```
- (void)touchesCancelled:(NSSet *)touches  withEvent:(UIEvent *)event {
    state = S0;
    movedTogether = movedSeperate = 0;
    accDistance = 0;
}
```

Листинг 4.13 показывает остаток определения класса представления. Инициализатор `initWithFrame:` устанавливает статистику и переменные состояния в начальные значения. Метод `drawRect:`, вызываемый, когда представление получает сообщение `setNeedToDisplay`, выводит процент времени, когда оба пальца двигались вместе, и среднее расстояние между ними во время их совместного движения. Далее приведен снимок экрана приложения (рис. 4.6).

Листинг 4.13. Остаток реализации класса представления, используемого в приложении с усовершенствованным отслеживанием жестов

```
- (id) initWithFrame:(CGRect)frame {
    if (self = [super  initWithFrame:frame]) {
            state = S0;
            movedTogether = movedSeperate = 0;
            accDistance = 0;
    }
    return  self;
}

-(void)drawRect:(CGRect)rect {
    NSString *message =
            [NSString
                    stringWithFormat:@"Moved together %.0f%% of the time.",
                    (movedSeperate + movedTogether) ?
                    100*(movedTogether/(movedTogether+movedSeperate)) :
                    100.0
            ];
            [message  drawAtPoint:CGPointMake(10,100)
                    withFont:[UIFont  systemFontOfSize:16]];
            message = [NSString stringWithFormat:@"Average distance:%4.2f.",
            movedTogether ?  accDistance/movedTogether: 0.0
            ];
            [message drawAtPoint:CGPointMake(10,150)
                    withFont:[UIFont systemFontOfSize:16]];
}
```

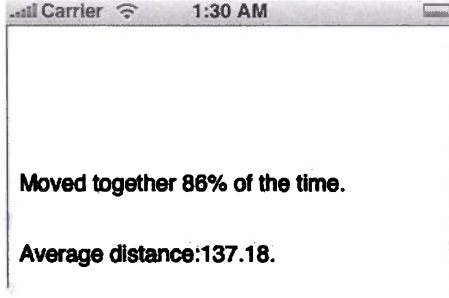

Рис. 4.6. Снимок экрана приложения с усовершенствованным отслеживанием жестов

4.4. Анимация

Анимация — одна из главных особенностей ОС iPhone. В этом разделе мы рассмотрим примеры анимации. Они не требуют знаний принципов обработки изображений. Мы начнем с использования класса `UIView` для анимации свойства представлений, а затем рассмотрим пример, в котором осуществляются переходы между представлениями.

4.4.1. Использование поддержки анимации в классе UIView

Геометрические свойства представления можно легко анимировать. Класс `UIView` предоставляет некоторые методы класса, которые можно использовать для выполнения простых анимаций, таких как движение представления в другую позицию или масштабирование.

Анимацию свойств представлений следует производить между двумя вызовами класса `UIView` — `beginAnimation:context:` и `commitAnimations`. Внутри блока анимации вы задаете характеристики анимации (например, ее длительность, временные функции) и изменяете свойства представления (к примеру, его центр) на конечные значения. Когда вы фиксируете анимацию, свойства представления анимируются, принимая новые значения.

Начнем с создания приложения, которое позволяет пользователю передвигать представление по экрану двойным касанием на новой позиции. Движение представления анимируется, изменяя при этом его центр. Мы создадим подкласс `UIView` и назовем его `AnimView`. Подкласс `AnimView` добавляет дочернее представление и ждет касаний пользователя. Когда последний обозначает двойным касанием местоположение внутри экземпляра `AnimView`, свойство центра дочернего представления анимируется и меняет местоположение на то, где пользователь дотронулся дважды.

Листинг 4.14 демонстрирует класс-делегат приложения. Метод `applicationDidFinishLaunching:` создает главное окно и добавляет к нему экземпляр класса `AnimView`. Экземпляр `AnimView` занимает весь доступный пользователю экран и имеет фон серого цвета.

Листинг 4.14. Класс-делегат приложения для анимирования свойства центра представления

```
#import <UIKit/UIKit.h>
#import "AnimView.h"

@interface  AnimationApplAppDelegate:
    NSObject <UIApplicationDelegate> {
    UIWindow * window;
}
@end
```

```
@implementation AnimationAppIAppDelegate
- (void) applicationDidFinishLaunching:
  (UIApplication *) application {
    window = [[UIWindow alloc]
            initWithFrame:[[UIScreen mainScreen] bounds]];
    CGRect frame =
            [UIScreen  mainScreen].applicationFrame;
    AnimView *view = [[AnimView alloc]
            initWithFrame:frame];
    view.backgroundColor = [UIColor grayColor];
    [window addSubview:view];
    [view release];
    [window makeKeyAndVisible];
}
- (void) dealloc {
    [window release];
    [super dealloc];
}
@end
```

Листинг 4.15 показывает класс AnimView. Класс хранит ссылку на дочернее представление в переменной экземпляра childView. Инициализатор initWithFrame: создает экземпляр дочернего представления, конфигурирует его, устанавливая белый цвет фона, и добавляет как дочернее представление.

Логика передвижения дочернего представления на новое место находится в методе touchesEnded:withEvent:. Сначала метод проверяет получение двойного касания от пользователя. В этом случае он начинает блок анимации следующим выражением:

```
+ (void)
beginAnimations:(NSString *)animationID
    context:(void *)context
```

Два параметра этого метода могут быть NULL. animationID и context могут использоваться для коммуникации с делегатами анимации. Наш пример не использует делегата анимации, поэтому мы передаем значения NULL.

После начала блока анимации метод устанавливает кривую скорости анимации. Следующее выражение переопределяет кривую скорости анимации по умолчанию и устанавливает ее в UIAnimationCurveEaseOut:

```
[UIView  setAnimationCurve:
    UIViewAnimationCurveEaseOut];
```

Метод setAnimationCurve: определен следующим образом:

```
+ (void) setAnimationCurve:(UIViewAnimationCurve)curve
```

Доступны следующие кривые скорости анимации:

- UIViewAnimationCurveEaseInOut — означает, что анимация должна замедляться в начале и в конце; это кривая по умолчанию;

- `UIViewAnimationCurveEaseIn` — она значит, что анимация должна замедляться только в начале;
- `UIViewAnimationCurveEaseOut` — означает, что анимация должна замедляться только в конце;
- `UIViewAnimationCurveEaseLinear` — она значит, что скорость анимации должна быть постоянной.

Длительность анимации устанавливается с помощью метода `setAnimationDuration:`, который объявлен следующим образом:

```
+ (void)
    setAnimationDuration:(NSTimeInterval)duration
```

Параметр `duration` задается в секундах. Значение по умолчанию — 0,2 секунды.

После настройки анимации метод изменяет свойства представления (в нашем случае это одно свойство (центр) и одно представление (`childView`)) и фиксирует анимацию. Свойство центра изменяется в следующем выражении:

```
childView . center  =  [touch  locationInView : self]
```

Листинг 4.15. Класс AnimView, используемый в анимации свойства центра дочернего представления

```
#import <UIKit/UIKit.h>
#import <QuartzCore/QuartzCore.h>

@interface AnimView : UIView {
    UIView  *childView;
}
@end

@implementation AnimView

- (id) initWithFrame:(CGRect)frame {
    if (self = [super initWithFrame:frame]) {
            childView  =  [[UIView  alloc] initWithFrame:
                    CGRectMake(100, 150, 100, 150)];
            childView.backgroundColor  =  [UIColor  whiteColor];
            [self addSubview:childView];
    }
    return  self;
}

- (void) touchesEnded:(NSSet *)touches
            withEvent:(UIEvent   *)event{
    if([[UITouch*)[touches  anyObject]  tapCount] == 2) {
            UITouch  *touch = [touches  anyObject];
            [UIView beginAnimations : nil  context:NULL];
            [UIView  setAnimationCurve:UIViewAnimationCurveEaseOut];
            [UIView  setAnimationDuration:1];
```

```
            childView.center = [touch locationInView:self];
            [UIView commitAnimations];
    }
}
- (void) dealloc {
    [childView release];
    [super dealloc];
}
@end
```

Иногда вам может понадобиться получить сообщение, когда анимация закончится. Вы можете установить делегат анимации, используя метод `setAnimationDelegate:`. Производятся два вызова этого делегата — `animationDidStart:` и `animationDidStop:finished:`. Эти методы определены с использованием категории `NSObject`.

Усовершенствуем наше анимационное приложение, чтобы изменялся цвет фона дочернего представления и анимировались его размеры. Когда анимация закончится, мы вернемся к начальным размерам и цвету. Далее приведен усовершенствованный метод `touchesEnded:withEvent:`.

```
- (void)
    touchesEnded:(NSSet *)touches
withEvent:(UIEvent *) event {
    if([[(UITouch *)[touches anyObject] tapCount] == 2){
            childView.backgroundColor = [UIColor blueColor];
            [UIView beginAnimations:nil context:NULL];
            [UIView setAnimationCurve:UIViewAnimationCurveEaseOut];
            [UIView setAnimationDuration:0.5];
            [UIView setAnimationDelegate:self];
            childView.transform = CGAffineTransformMakeScale(1.5, 1.5);
            [UIView commitAnimations];
    }
}
```

В приведенном выше фрагменте кода по двойному касанию мы изменяем цвет дочернего представления снаружи блока анимации. Это приведет к немедленной смене цвета, а не к анимированной, как если это было бы сделано внутри блока анимации. После этого начинается блок анимации, устанавливаются кривая скорости, длительность и делегат. Чтобы масштабировать дочернее представление, увеличив его на 50 %, метод обновляет свойство представления `transform`. Свойство объявлено следующим образом:

```
@property(nonatomic) CGAffineTransform transform
```

Трансформация производится с использованием матрицы 3 × 3, необходимой для вращения, масштабирования или перемещения представления. Функция `CGAffineTransform` хранит первые два столбца этой матрицы. Третий столбец всегда равен `[0,0,1]`. Чтобы увеличить дочернее представление на 50 %, мы используем следующее выражение:

```
childView.transform = CGAffineTransformMakeScale(1.5, 1.5)
```

В приведенном выше выражении мы получаем аффинную трансформацию для увеличения на 50 %, используя функцию CGAffineTransformMakeScale, и устанавливаем значение свойства transform. После того как анимация закончится и дочернее представление увеличится на 50 %, вызывается метод animationDidStop:finished:, определенный в классе AnimView следующим образом:

```
- (void)
animationDidStop:(CAAnimation *)theAnimation
finished:(BOOL)flag {
    childView.transform = CGAffineTransformIdentity;
    childView.backgroundColor = [UIColor whiteColor];
}
```

Приведенный выше метод изменяет цвет фона дочернего представления на белый и мгновенно трансформирует измерения на значения до масштабирования.

4.4.2. Анимация перехода

Класс UIView фактически является оберткой, которая берет через цепочку наследования свойства обработки событий у класса NSResponder, а анимационные свойства — у его уникальной переменной экземпляра CALayer. Свойство layer, экземпляр CALayer, является объектом Core Animation, инкапсулирующим информацию об анимации, которая должна быть отрисована на дисплее.

Когда вы изменяете экземпляр UIView, например, добавляя или удаляя дочерние представления, изменения происходят мгновенно. Чтобы анимировать эти изменения, вы создаете объект анимации, конфигурируете его и добавляете к свойству layer. В этом разделе мы рассмотрим, как можно анимировать замещение одного представления другим через анимацию перехода. Приложение, демонстрирующее это, создаст два дочерних представления главного окна и добавит одно из них к окну. Когда пользователь дважды коснется активного представления, приложение заместит представление другим неактивным представлением и анимирует изменение, передвинув новое представление справа налево.

Анимация производится в классе-делегате представления. Листинг 4.16 показывает объявление класса-делегата приложения. Класс оперирует двумя ссылками на экземпляры AnimView — двумя представлениями. Метод showOtherView: используется для анимации замещения одного представления другим.

Листинг 4.16. Объявление класса-делегата приложения, использующегося для анимации перехода между представлениями

```
#import <UIKit/UIKit.h>
@class AnimView;
@interface  AnimationApp2AppDelegate:
```

```
            NSObject <UIApplicationDelegate> {
    UIWindow *window;
    AnimView *view1,*view2;
}

-(void)showOtherView:(UIView *) oldView;
@end
```

Листинг 4.17 демонстрирует реализацию класса-делегата приложения. Метод `applicationDidFinishLaunching:` создает главное окно и два дочерних представления. Он добавляет одно представление к окну и делает его ключевым и видимым.

Когда текущее представление запрашивает делегат приложения переключиться на другое, вызывается метод `showOtherView:` со ссылкой на активное дочернее представление. Текущее представление удаляется из окна, а другое — добавляется. Чтобы анимировать это изменение, мы создаем объект анимации и добавляем его к свойству окна `layer`.

Объекты анимации являются экземплярами класса `CAAnimation`. Подкласс `CATransition` является подклассом `CAAnimation`, упрощающим анимацию переходов. Сначала мы получаем новый объект анимации, используя метод класса `animation`. Далее мы настраиваем тип, длительность и временные характеристики анимации. Тип анимации — появление справа, выбранная длительность — 0,5 секунды. Используется также функция замедления в начале и в конце. Чтобы добавить анимацию, мы используем метод `addAnimation:forKey:`, объявленный следующим образом:

```
- (void)
    addAnimation:(CAAnimation *)anim
    forKey:(NSString *)key
```

Параметр `anim` — экземпляр класса `CAAnimation`, представляющий анимацию. Второй параметр (может быть `nil`) используется, чтобы отличать разные анимации данного слоя. Параметр `anim` копируется методом, поэтому после настройки объекта анимации вам требуется вызывать этот метод.

Листинг 4.17. Реализация класса-делегата приложения, использующегося для анимации перехода между представлениями

```
#import <QuartzCore/QuartzCore.h>
#import "AnimationApp2AppDelegate.h"
#import "AnimView.h"

@implementation  AnimationApp2AppDelegate

- (void) applicationDidFinishLaunching:(UIApplication *) application {
    window = [[UIWindow  alloc]
            initWithFrame:[[UIScreen  mainScreen] bounds]];
    CGRect  frame = [UIScreen  mainScreen].applicationFrame;
    view1 = [[AnimView  alloc]  initWithFrame:frame];
    view1.message = @"View 1";
```

```
    view1.backgroundColor = [UIColor  whiteColor];
    [window  addSubview:view1];
    view2 = [[AnimView  alloc]  initWithFrame:frame];
    view2.message = @"View 2";
    view2.backgroundColor = [UIColor  yellowColor];
    [window  makeKeyAndVisible];
}

- (void) showOtherView:(UIView *)  oldView {
    if(oldView == view1) {
            [view1  removeFromSuperview];
            [window  addSubview:view2];
    } else {
            [view2  removeFromSuperview];
            [window  addSubview:view1];
    }
    CATransition  *animation = [CATransition  animation];
    [animation  setType:kCATransitionMoveIn];
    [animation  setSubtype:kCATransitionFromRight];
    [animation  setDuration:0.5];
    [animation  setTimingFunction:
            [CAMediaTimingFunction functionWithName:
              kCAMediaTimingFunctionEaseInEaseOut]];
    [[window  layer]  addAnimation:animation  forKey:@"mykey"];
}

- (void) dealloc {
    [view1 release];
    [view2 release];
    [window release];
    [super dealloc];
}
@end
```

Листинг 4.18 показывает класс AnimView. Класс оперирует переменной экземпляра message, чье содержимое отрисовывается на экране. Это будет отличительной чертой двух меняющихся представлений.

Листинг 4.18. Класс AnimView, использующийся в приложении с переходами между представлениями

```
#import <UIKit/UIKit.h>
#import <QuartzCore/QuartzCore.h>

@interface AnimView:UIView {
    NSString *message;
}

@property(nonatomic,  copy)  NSString  *message;
@end

@implementation AnimView
@synthesize  message;
- (void)
    touchesEnded:(NSSet *)touches  withEvent:(UIEvent *)event {
    if([[(UITouch *)[touches  anyObject]  tapCount] == 2) {
            [[ UIApplication  sharedApplication ].delegate
                    showOtherView:self];
    }
```

```
}
- (void)drawRect:(CGRect)rect {
    [message  drawAtPoint:CGPointMake(100,100)
            withFont:[UIFont  systemFontOfSize:32]];
}
@end
```

4.5. Рисование

Метод `drawRect:` — идеальное место для рисования внутри представления. Представление настроит для вас свое окружение, упрощая рисование. Используйте функции Quartz 2D для рисования простых и сложных фигур. В качестве первого параметра эти функции требуют графического контекста. Вы можете получить его, используя `UIGraphicsGetCurrentContext()`.

Как только вы получили графический контекст, можете рисовать контуры. Контур — это коллекция одной или нескольких фигур. Изобразив контур, можете его заштриховать, залить либо и то и другое.

Листинг 4.19 показывает метод `drawRect:`, который рисует несколько контуров. Результат вы видите на рис. 4.7. После нахождения графического контекста мы устанавливаем толщину линий контура в 5 единиц (по умолчанию 1). Затем мы указываем новое положение контура, используя функцию `CGContextMoveToPoint()`. Она применяется, чтобы добавить линию к контуру, начиная с `(50, 100)` и заканчивая `(200, 100)`. На этом этапе в контуре есть только одна фигура (прямая линия). Чтобы нарисовать его, мы используем функцию `CGContextStrokePath()`, которая нарисует новый контур и очистит текущий.

Чтобы нарисовать эллипс, используйте функцию `CGContextAddEllipseInRect()`. Когда вы следом за ней вызовете `CGContextStrokePath()`, появится эллипс. Если вы хотите заполнить его, используйте функцию `CGContextFillPath()`.

Вы можете установить цвет штриха, используя функцию `CGContextSetRGBStrokeColor()`. В этой функции вы задаете RGB- и альфа (прозрачность)-компоненты. Цвет заливки можно установить с помощью функции `CGContextSetRGBFillColor()`. Кроме линий и эллипсов, вы можете рисовать прямоугольники, кривые, дуги и т. д.

Листинг 4.19. Метод drawRect:, рисующий различные фигуры

```
- (void) drawRect:(CGRect) rect {
    CGContextRef  context = UIGraphicsGetCurrentContext();
    CGContextSetLineWidth(context, 5.0);
    CGContextMoveToPoint(context, 50, 100);
    CGContextAddLineToPoint(context, 200, 100);
    CGContextStrokePath(context);
    CGContextAddEllipseInRect(context,
```

```
        CGRectMake(70.0, 170.0, 50.0, 50.0));
    CGContextStrokePath(context);
    CGContextAddEllipseInRect(context,
        CGRectMake(150.0, 170.0, 50.0, 50.0));
    CGContextFillPath(context);
    CGContextSetRGBStrokeColor(context, 0.0, 1.0, 0.0, 1.0);
    CGContextSetRGBFillColor(context, 0.0, 0.0, 1.0, 1.0);
    CGContextAddRect(context,
        CGRectMake(30.0, 30.0, 60.0, 60.0));
    CGContextFillPath(context);
    CGContextAddArc(context,
        260, 90, 40, 0.0*M_PI/180, 270*M_PI/180, 1);
    CGContextAddLineToPoint(context, 280, 350);
    CGContextStrokePath(context);
    CGContextMoveToPoint(context, 130, 300);
    CGContextAddLineToPoint(context, 80, 400);
    CGContextAddLineToPoint(context, 190, 400);
    CGContextAddLineToPoint(context, 130, 300);
    CGContextStrokePath(context);
}
```

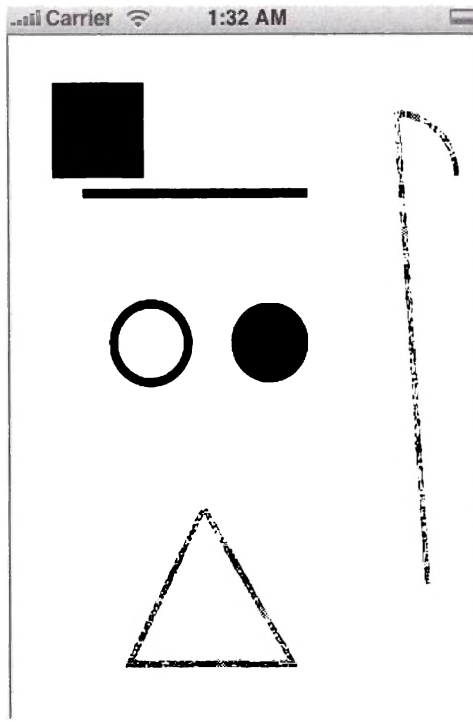

Рис. 4.7. Рисование различных фигур с использованием Quartz 2D

Глава 5
Элементы управления

Элементы управления — это графические объекты, предназначенные для того, чтобы пользователь приложения мог объяснить свою цель. Например, элемент «слайдер» применяется для тонкой настройки определенного значения, «переключатель» — для включения/выключения определенного параметра. В этой главе мы рассмотрим некоторые важные графические элементы управления, с помощью которых можно построить привлекательные iPhone-приложения.

Вы изучите основные доступные элементы управления и их использование. Перед тем как начать говорить о специализированных элементах управления, разберемся с основным классом для всех элементов управления `UIControl`, а также важным механизмом «цель-действие».

5.1. Основа всех элементов управления

Элементы управления являются подклассами класса `UIControl`. Место `UIControl` в иерархии классов изображено на рис. 5.1. В этом классе содержатся основы поведения всех элементов управления, поэтому его понимание является неотъемлемым для использования базовых подклассов, таких как `UITextField`, `UISlider`, `UIDatePicker` и т. д.

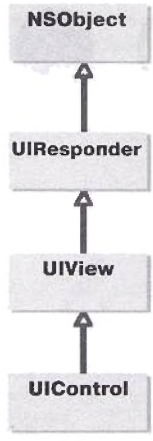

Рис. 5.1. Иерархия наследования UIControl

5.1.1. Атрибуты UIControl

Как суперкласс для всех элементов управления `UIControl` имеет несколько общих атрибутов, которые могут быть сконфигурированы, используя методы доступа. Эти атрибуты включают следующие.

- `enabled` — булев атрибут, указывающий, задействован ли элемент. Свойство определено как

```
@property(nonatomic, getter=isEnabled) BOOL enabled
```

Если значением `enabled` является `NO`, касания пользователя игнорируются.

- `highlighted` — это булево значение контролирует, подсвечен ли элемент управления. По умолчанию значение атрибута `NO`. Когда пользователь касается элемента управления, атрибут принимает значение `YES` и элемент подсвечивается. Когда пользователь отпускает элемент, значение становится `NO` и элемент не подсвечивается. Объявление свойства этого атрибута:

```
@property(nonatomic, getter=isHighlighted) BOOL highlighted
```

- `selected` — этот булев атрибут показывает, выбран ли элемент управления. Большинство подклассов `UIControl` его не использует. Тем не менее его может применять подкласс `UISwitch`. Объявление этого свойства следующее:

```
@property(nonatomic, getter=isSelected) BOOL selected
```

- `state` — это доступный только для чтения атрибут типа `UIControlState`. `UIControlState` определен как беззнаковое целое (`NSUInteger`); `state` — битовая маска, представляющая более одного состояния. Примеры определенных состояний — `UIControlStateHighlighted`, `UIControlStateDisabled` и `UIControlStateNormal`.

Свойство определено следующим образом, но помните, что атрибут доступен только для чтения:

```
@property(nonatomic, readonly) UIControlState state
```

5.1.2. Механизм «цель-действие»

Класс `UIControl` и его подклассы для информирования заинтересованных сторон о появлении изменений элемента управления используют механизм «цель-действие». В основном объект, обычно называемый контроллером, посылает сообщение элементу управления, информируя его, что он заинтересован в мониторинге некоторых событий, имеющих

отношение к элементу управления. Когда это событие случится, элемент управления проинформирует этот объект (например, контроллер).

Объекту, регистрирующему самого себя, используя механизм «цель-действие», требуется указать три вида сведений: указатель на объект («цель»), который должен получить сообщение (обычно он сам); селектор («действие»), представляющий метод обработки; событие элемента управления, в котором объект заинтересован.

Когда элемент управления получает сообщение о регистрации, он сохраняет эту информацию во внутренней таблице управления. Обратите внимание, что одна и та же цель может быть зарегистрирована для различных событий с разными селекторами. Более того, различные цели могут быть зарегистрированы на одно и то же событие.

Когда появляется событие (такое как изменение значения элемента управления), элемент посылает самому себе сообщение `sendActionsForControlEvents:`. Затем этот метод сверится с внутренней таблицей управления (таблицей, строящейся последовательно как результат регистрационных сообщений), чтобы найти все действия-обработчики для этого события. После этого элемент управления посылает экземпляру синглетона `UIApplication` сообщение `sendAction:to:from:forEvent:` для каждой записи. Экземпляр `UIApplication` единственный ответственен за рассылку целям сообщений о действиях.

Объект регистрирует элемент управления с помощью метода экземпляра `UIControl`, объявленного следующим образом:

```
- (void)addTarget:(id)target  action:(SEL) action
    forControlEvents:(UIControlEvents) controlEvents
```

Целью обычно является экземпляр, зарегистрированный для этого события (например, контроллер). Действие — это селектор, который идентифицирует сообщение действия цели (то есть метод, который вызывается экземпляром `UIApplication`, когда появляется событие). Селектор принимает одну из следующих трех форм:

```
- (void) action
- (void) action:(id)sender
- (void) action:(id)sender forEvent:(UIEvent *)event
```

Часть `controlEvents` — это битовая маска, определяющая управляющие события, которые инициализируют посылку сообщений действий к цели. В `UIControl.h` определено несколько таких управляющих событий. Некоторые примеры содержат:

- `UIControlEventValueChanged` — используется для указания, что значение элемента управления изменилось, например, слайдер передвинулся;

- `UIControlEventEditingDidBegin` — началось редактирование элемента управления (например, `UITextField`);
- `UIControlEventEditingDidEnd` — касание прервало редактирование поля, покинув его границы;
- `UIControlEventTouchDown` — одиночное касание внутри границ элемента управления.

Ниже изображена диаграмма последовательностей для сценария «цель-действие» (рис. 5.2). Итак, у нас есть два контроллера, `ctrl1` и `ctrl2`, следящих за событием элемента управления `UIControlEventValueChanged`. Диаграмма иллюстрирует роль, которую играет синглетон экземпляра `UIApplication` в доставке сообщений о действиях.

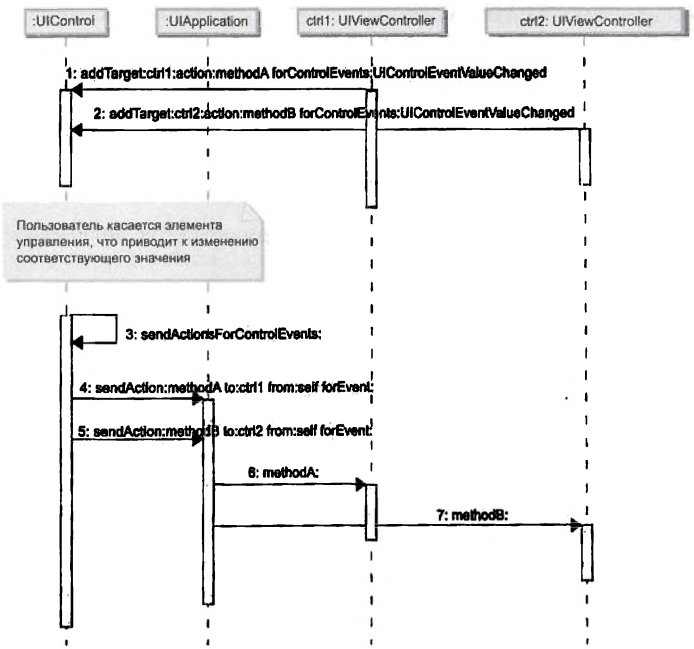

Рис. 5.2. Диаграмма последовательностей, иллюстрирующая механизм «цель-действие»: два контроллера, ctrl1 и ctrl2, добавляют самих себя в качестве целей для управляющего события UIControlEventValueChanged

Есть и другие важные методы, доступные в классе `UIControl` и относящиеся к механизму «цель-действие».

- `removeTarget:actionforControlEvents:` — этот метод используется для удаления заданной записи о цели и действии из

управляющей таблицы для конкретного события элемента управления. Метод объявлен как

```
- (void)removeTarget:(id)target action:(SEL)action
    forControlEvents:(UIControlEvents)controlEvents
```

- `allTargets` — метод возвращает все объекты целей, связанных с объектом элемента управления. Объявлен как

```
- (NSSet *)allTargets
```

- `allControlEvents` — этот метод возвращает все управляющие события, связанные с объектом элемента управления. Метод объявлен как

```
- (UIControlEvents)allControlEvents
```

- `actionsForTarget:forControlEvent:` — метод возвращает действия, ассоциированные с заданной целью и конкретным событием элемента управления. Он объявлен как

```
- (NSArray *) actionsForTarget:(id)target
    forControlEvent:(UIControlEvents)controlEvent
```

Возвращаемое значение — массив `NSArray` объектов типа `NSString`, представляющих имена селекторов.

Теперь, когда мы разобрались с классом `UIControl`, рассмотрим некоторые его конкретные реализации.

5.2. UITextField

`UITextField` (рис. 5.5) инкапсулирует элемент управления для редактирования текста, позволяющий пользователю ввести небольшой объем информации. Элемент управления предоставляет справа опциональную кнопку очистки для удаления текста. `UITextField` использует протокол `UITextFieldDelegate` для взаимодействия с классом-делегатом (обычно контроллером). Сам `UITextField` заимствует протокол `UITextInputTraits`. Этот протокол должен быть реализован любым элементом управления, использующим клавиатуру. Вы создаете экземпляр `UITextField` и добавляете его к представлению в качестве дочернего. У этого элемента управления есть следующие важные свойства.

- `text` — используя это свойство, вы можете получать и устанавливать текст, отображаемый элементом управления. Свойство объявлено как

```
@property(nonatomic, copy) NSString *text
```

- `textAlignment` — применяется для контроля над методом, используемым для выравнивания текста внутри элемента управления. Объявлено как

```
@property(nonatomic) UITextAlignment textAlignment
```

Свойство `textAlignment` может быть установлено в одно из следующих значений: `UITextAlignmentLeft` (по умолчанию), `UITextAlignmentCenter` и `UITextAlignmentRight`.

`textColor` — цвет текста внутри элемента управления. Свойство объявлено как

```
@property(nonatomic, retain) UIColor *textColor
```

Установка значения в `nil` (по умолчанию) приведет к черному непрозрачному цвету текста.

- `background` — изображение, представляющее фон элемента управления. Объявлено как

```
@property(nonatomic, retain) UIImage *background
```

По умолчанию равно `nil`.

- `clearButtonMode` — управляет внешним видом кнопки очистки. Свойство объявлено как

```
@property(nonatomic) UITextFieldViewMode clearButtonMode
```

Вы можете установить его значение в одно из следующих: `UITextFieldViewModeNever` (кнопка очистки никогда не появляется), `UITextFieldViewModeWhileEditing` (появляется, только когда пользователь редактирует текст), `UITextFieldViewModeUnlessEditing` (появляется, только когда пользователь не редактирует текст) и `UITextFieldViewModeAlways` (появляется всегда).

- `borderStyle` — используется для установки стиля окантовки элемента управления. Оно объявлено как

```
@property(nonatomic) UITextBorderStyle borderStyle
```

Значение может быть одним из следующих: `UITextBorderStyle None` (по умолчанию), `UITextBorderStyleLine`, `UITextBorderStyleBezel` и `UITextBorderStyleRoundedRect`.

- `delegate` — используйте это свойство для назначения делегата элемента управления. Объявление свойства следующее:

```
@property(nonatomic assign) id<UITextFieldDelegate> delegate
```

Если значение не `nil`, элемент управления будет посылать специальные сообщения делегату, информируя его о важных изменениях

в редактировании, например, что пользователь нажал кнопку Return (Возврат) на клавиатуре. Позже мы вкратце ознакомимся с протоколом UITextFieldDelegate.

- disabledBackground — если значение этого атрибута не равно nil, значение disabledBackground будет использовано как фон заблокированного элемента управления. Свойство объявлено как

```
@property(nonatomic, retain) UIImage *disabledBackground
```

- editing — это атрибут только для чтения, показывающий, находится ли элемент управления в состоянии редактирования. Свойство объявлено как

```
@property(nonatmic, readonly, getter=isEditing) BOOL editing
```

- font — значение представляет шрифт текста. Свойство объявлено следующим образом:

```
@property(nonatomic, retain) UIFont *font
```

- placeholder — используется для вывода в текстовом элементе управления, если в поле ввода отсутствует текст. Объявление свойства следующее:

```
@property(nonatomic, copy) NSString *placeholder
```

Значение по умолчанию — nil (то есть строка заменителя отсутствует). Если значение не nil, строка прорисовывается 70 %-м серым цветом.

5.2.1. Взаимодействие с клавиатурой

Выше упоминалось, что элемент управления UITextField согласовывается с протоколом UITextinputTraits. Этот протокол должен быть реализован любым элементом управления, который должен взаимодействовать с пользователем посредством клавиатуры. Протокол определяет следующие свойства.

- keyboardType — контролирует стиль клавиатуры, ассоциированной с текстовым полем. Свойство объявлено как

```
@property(nonatomic) UIKeyboardType keyboardType
```

Существует несколько типов клавиатур:

- UIKeyboardTypeDefault — клавиатура по умолчанию;
- UIKeyboardTypeAlphabet — представляет стандартную алфавитно-цифровую клавиатуру (Qwerty);
- UIKeyboardTypeNumbersAndPunctuation — клавиатура с цифрами и знаками препинания;

- `UIKeyboardTypeURL` — стиль клавиатуры, облегчающий ввод URL;
- `UIKeyboardTypeNumberPad` — цифровая клавиатура для ввода PIN;
- `UIKeyboardTypePhonePad` — клавиатура, разработанная для ввода телефонных номеров;
- `UIKeyboardTypeNamePhonePad` — клавиатура для ввода имени и телефонного номера;
- `UIKeyboardTypeEmailAddress` — стиль клавиатуры для ввода адреса электронной почты.

Ниже изображены некоторые доступные стили клавиатуры (рис. 5.3, 5.4).

Рис. 5.3. Два типа клавиатуры — UIKeyboardTypeDefault и UIKeyboardTypePhonePad

- `secureTextEntry` — используется для указания, что вводимый текст должен быть спрятан (например, каждый символ должен замещаться *). Свойство объявлено как

```
@property(nonatomic, getter=isSecureTextEntry) BOOL secureTextEntry
```

- `returnKeyType` — применяется для задания заголовка клавиши Return (Возврат). Свойство объявлено следующим образом:

```
@property(nonatomic) UIReturnKeyType returnKeyType
```

Атрибут `returnKeyType` может принимать одно из 3 следующих значений: `UIReturnKeyDefault`, `UIReturnKeyGo`, `UIReturnKeyGoogle`, `UIReturnKeyJoin`, `UIRetumKeyNext`, `UIReturnKeyRoute`, `UIReturnKeySearch`, `UIReturnKeySend`, `UIReturnKeyYahoo`, `UIReturnKeyDone` и `UIReturnKeyEmergencyCall`.

- `keyboardAppearance` — используется для различия между вводом текста внутри приложения и набором текста внутри панели сообщений. Свойство объявлено как

```
@property(nonatomic) UIKeyboardAppearance keyboardAppearance
```

Значение может быть UIKeyboardAppearanceDefault (по умолчанию) или UIKeyboardAppearanceAlert.

- enablesReturnKeyAutomatically — если значение этого свойства YES, клавиша Return (Возврат) на клавиатуре будет заблокирована, пока пользователь не введет какой-нибудь текст. Значение по умолчанию — NO. Свойство объявлено как

```
@property(nonatomic) BOOL enablesReturnKeyAutomatically
```

- autocorrectionType — используется для управления автокоррекцией пользовательского ввода. Объявлено как

```
@property(nonatomic) UITextAutocorrectionType autocorrectionType
```

Свойство может принимать одно из следующих значений: UITextAutocorrectionTypeDefault (выбирает подходящую автокоррекцию), UITextAutocorrectionTypeNo (нет автокоррекции) и UITextAutocorrectionTypeYes (автокоррекция включена).

- autocapitalizationType — определяет, когда клавиша Shift будет автоматически нажиматься для ввода заглавных букв. Объявлено следующим образом:

```
@property(nonatomic) UITextAutocapitalizationType autocapitalizationType
```

Свойство может принимать одно из следующих значений: UITextAutocapitalizationTypeNone (не включать автоматической подстановки заглавных букв), UITextAutocapitalizationTypeWords (автоматически переводит в заглавную первую букву каждого слова), UITextAutocapitalizationTypeSentences (автоматически переводит в заглавную первую букву каждого предложения) и UITextAutocapitalizationTypeAllCharacters (автоматически переводит в заглавные все символы).

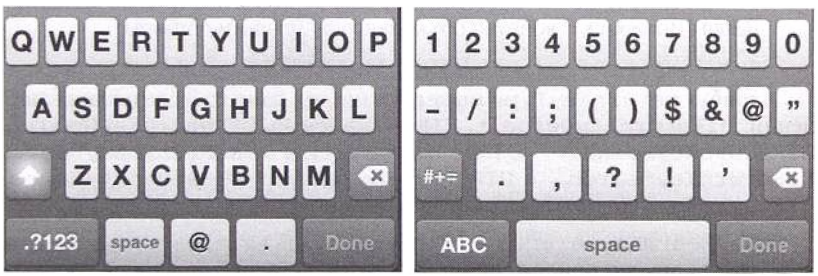

Рис. 5.4. Два типа клавиатуры — UIKeyboardTypeEmailAddress и UIKeyboardTypeNumbersAndPunctuation

5.2.2. Делегат

Как уже упоминалось, элемент управления использует делегат для взаимодействия с важными событиями редактирования. Используемый протокол делегата — `UITextFieldDelegate`. Он объявляет следующие методы.

- `textFieldShouldReturn` — этот метод делегата объявлен следующим образом:

```
- (BOOL) textFieldShouldReturn (UITextField *)textField
```

Он вызывается, когда пользователь нажимает кнопку Return (Возврат). В случае однострочного текстового поля вы можете использовать это событие как сигнал конца редактирования поля. Вы должны также выступить в роли первого реагирующего. Возврат `YES` или `NO` не влияет на скрытие клавиатуры и не имеет значения в контексте однострочного текстового поля. В качестве примера изучите код, представленный в листинге 5.2.

- `textFieldShouldClear:` — этот метод вызывается при нажатии кнопки очистки. Объявлен как

```
- (BOOL) textFieldShouldClear:(UITextField *)textField
```

Если вы вернете `YES`, содержимое текстового поля очистится, в противном случае — нет.

- `textFieldDidBeginEditing:` — вызывается, когда текстовое поле начинает выступать в качестве первого реагирующего, готового к пользовательскому вводу. Метод объявлен как

```
- (void) textFieldDidBeginEditing:(UITextField *)textField
```

- `textField:shouldChangeCharactersInRange:replacementString:` — вызывается при запросе разрешения у делегата на замену заданного текста. Объявлен как

```
- (BOOL) textField:(UITextField *)textField
shouldChangeCharactersInRange:(NSRange)range
replacementString:(NSString *)string
```

Здесь `range` — это диапазон заменяемых символов, а `string` — замещающая строка.

- `textFieldDidEndEditing:` — метод вызывается по окончании редактирования текстового поля. Объявлен как

```
- (void) textFieldDidEndEditing:(UITextField *)textField
```

- `textFieldShouldBeginEditing:` — вызывается, запрашивая разрешение у делегата, после чего текстовое поле может начать редактирование. Объявлен следующим образом:

```
- (BOOL) textFieldShouldBeginEditing:(UITextField *)textField
```

Возвратите YES, чтобы начать редактирование, и NO в противном случае.

- `textFieldDidBeginEditing:` — вызывается, когда начинается редактирование текстового поля. Объявлен как

```
- (void)textFieldDidBeginEditing:(UITextField *)textField
```

5.2.3. Создание и работа с UITextField

Рассмотрим, как можно создать экземпляр `UITextField` и добавить его к представлению. Листинг 5.1 показывает, как создать экземпляр текстового поля, сконфигурировать его и привязать к представлению.

Листинг 5.1. Создание и настройка экземпляра UITextField как дочернего представления

```
CGRect  rect = CGRectMake(10, 10, 150, 30);
myTextField = [[UITextField  alloc]
    initWithFrame:rect];
myTextField.textColor = [UIColor  blackColor];
myTextField.font = [UIFont  systemFontOfSize:17.0];
myTextField.placeholder = @"<enter text>";
myTextField.backgroundColor = [UIColor  whiteColor];
myTextField.borderStyle = UITextBorderStyleBezel;
myTextField.keyboardType = UIKeyboardTypeDefault;
myTextField.returnKeyType = UIReturnKeyDone;
myTextField.clearButtonMode = UITextFieldViewModeAlways;
myTextField.delegate = self;
[theView  addSubview:myTextField];
```

Обычно вы создаете эти элементы управления в методе `loadView:` контроллера представления. Здесь мы делаем `self` (то есть контроллер представления) делегатом. У вас есть выбор, какой из методов делегата реализовать. В данном примере мы реализуем метод `textFieldShouldReturn:`, приведенный в листинге 5.2.

Листинг 5.2. Реализация метода textFieldShouldReturn: экземпляра UITextField

```
- (BOOL) textFieldShouldReturn:(UITextField *)textField {
    if (myTextField = textField) {
      if ((([myTextField  text]  isEqualToString:@"Hillary"]) == NO) {
            [myTextField  resignFirstResponder]; // спрятать клавиатуру
            return NO; // это текстовое поле, не требуется перевода строки
      }
      else return NO;   // это текстовое поле, не требуется перевода строки
    }
    return NO; // это текстовое поле, не требуется перевода строки
}
```

В приведенном выше коде мы сначала проверяем, является ли текстовое поле экземпляром myTextField. Если является, то мы проверяем введенный текст. Если это слово Hillary, мы не передаем первого реагирующего и возвращаем NO. В противном случае мы посылаем текстовому элементу управления сообщение resignFirstResponder, чтобы он перестал выступать в роли первого реагирующего. Клавиатура исчезнет, после чего мы возвращаем NO. Как уже было сказано, возвращаемое значение не влияет на однострочное текстовое поле.

5.3. Слайдеры

Элемент управления UISlider (рис. 5.5) — это привычный горизонтальный элемент управления, используемый для выбора единственного значения из непрерывного диапазона.

Рис. 5.5. Экземпляры UITextField, UISlider и UISwitch

Далее приведены основные свойства, необходимые для настройки слайдера.

- value — этот атрибут содержит текущее значение, отображаемое слайдером. Свойство объявлено как

```
@property(nonatomic) float value
```

Вы можете считывать и записывать это значение. Если вы измените это значение, слайдер перерисуется сам.

- minimumValue — содержит минимальное значение слайдера. Свойство объявлено следующим образом:

```
@property(nonatomic) float minimumValue
```

- maximumValue — содержит максимальное значение слайдера. Свойство объявлено как

```
@property(nonatomic) float maximumValue
```

- continuous — булев атрибут, контролирующий частоту, с которой слайдер посылает обновленные текущие значения ассоциированному с ним целевому действию. Если его значение YES (по умолчанию), слайдер постоянно посылает обновления текущего значения, пока пользователь перетаскивает ползунок слайдера. Если значение NO, он посылает обновление только однажды — когда пользователь отпускает ползунок слайдера. Объявляется как

```
@property(nonatomic, getter=isContinuous) BOOL continuous
```

Листинг 5.3 показывает, как можно настроить экземпляр слайдера и добавить его к представлению в качестве дочернего.

Листинг 5.3. Создание и настройка экземпляра UISlider

```
CGRect  rect = CGRectMake(10, 60, 200, 30);
mySlider = [[UISlider  alloc]  initWithFrame:rect];
[mySlider  addTarget:self
    action:@selector(sliderValueChaged:)
    forControlEvents:UIControlEventValueChanged];
mySlider.backgroundColor = [UIColor  clearColor];
mySlider.minimumValue = 0.0;
mySlider.maximumValue = 10.0;
mySlider.continuous = YES;
mySlider.value = 5.0;
[theView addSubview: mySlider];
```

Диапазон слайдера — от `0.0` до `10.0`. Он постоянно посылает обновления методу-действию `sliderValueChanged:`, когда пользователь изменяет его значение.

Для получения обновлений текущего значения слайдера мы используем механизм «цель-действие». Мы добавляем целевой метод `sliderValueChanged:` (листинг 5.4) для управляющего события `UIComtrolEventValueChanged`.

Листинг 5.4. Метод sliderValueChanged:

```
-(void) sliderValueChaged:(id) sender
{
    UISlider *slider = sender;
    if(mySlider == slider) {
      printf ("Value of slider is %f \n", [mySlider value]);
    }
}
```

5.4. Переключатели

`UISwitch` — это элемент управления, позволяющий представить пользователю переключатель «вкл/выкл». Класс `UISwitch` определяет свойство для считывания и установки текущего состояния переключателя. Свойство объявлено как

```
@property(nonatomic, getter=isOn) BOOL on
```

Вы также можете использовать метод `setOn: animated:`, позволяющий устанавливать состояние переключателя, опционально анимируя изменение. Метод объявлен как

```
- (void) setOn:(BOOL)on animated:(BOOL)animated
```

Если `animated` установлено в `YES`, изменение состояния анимируется.

Как и в любом другом элементе управления, вы можете настроить целевое действие и ассоциировать его с событием. Как разработчика вас больше всего интересует событие, когда пользователь щелкает переключателем. Для этого вы можете использовать событие `UIControlEventValueChanged`. Листинг 5.5 демонстрирует создание и настройку экземпляра `UISwitch`, а листинг 5.6 — целевой метод для события `UIControlEventValueChanged`.

Листинг 5.5. Создание и настройка экземпляра UISwitch

```
rect = CGRectMake(10, 90, 100, 30);
mySwitch = [[UISwitch  alloc]  initWithFrame:rect];
[mySwitch addTarget:self
    action:@selector(switchValueChanged:)
    forControlEvents:UIControlEventValueChanged];
mySwitch.backgroundColor = [UIColor  clearColor];
[theView addSubview: mySwitch];
```

Листинг 5.6. Метод действия для примера экземпляра UISwitch

```
- (void) switchValueChanged:(id)sender
{
    UISwitch *aSwitch = sender;
    if(mySwitch == aSwitch) {
      if ([mySwitch isOn] == YES) {
          printf("The switch is on\n");
      } else
          printf("The switch is off\n");
    }
}
```

5.5. Кнопки

Класс `UIButton` — это элемент управления, инкапсулирующий поведение кнопок. Вы создаете кнопку, используя метод `buttonWithType:` класса `UIButton`. Затем вы настраиваете «цель-действие», чтобы обрабатывать пользовательские нажатия.

Ниже приведены некоторые доступные типы кнопок:

- `UIButtonTypeRoundedRect` — этот стиль используется для создания кнопок со скругленными краями;
- `UIButtonTypeDetailDisclosure` — используется для создания кнопок для показа деталей;
- `UIButtonTypeInfoLight` — этот стиль создает кнопку информации со светлым фоном;

- `UIButtonTypeInfoDark` — используется для создания кнопки информации с темным фоном;
- `UIButtonTypeContactAdd` — создает кнопку для добавления контакта.

Листинг 5.7 демонстрирует создание и настройку экземпляра `UIButton` (рис. 5.6).

Рис. 5.6. Пример элемента управления типа «кнопка»

Листинг 5.7. Создание и настройка экземпляра UIButton

```
myButton  =
    [[UIButton  buttonWithType:UIButtonTypeRoundedRect]  retain];
myButton.frame = CGRectMake(40.0, 100.0, 100, 50);
[myButton  setTitle:@"Click Me"  forState:UIControlStateNormal];
[myButton  addTarget:self
    action:@selector(buttonClicked:)
    forControlEvents:UIControlEventTouchUpInside];
[theView  addSubview: myButton];
```

Приведенный выше код настраивает «цель-действие» для события нажатия. Листинг 5.8 показывает метод действия, обрабатывающий нажатие кнопки. Как можно заметить, это является базовым механизмом, унаследованным от `UIControl`, и никакого нового механизма нет.

Листинг 5.8. Метод действия для примера экземпляра UIButton

```
- (void) buttonClicked:(id)sender {
    UIButton *button = sender;
    if(myButton == sender){
      printf("The button was tapped\n");
    }
 }
```

5.6. Сегментированные элементы управления

Сегментированный элемент управления — это горизонтальный элемент, управляющий элементами, похожими на кнопки. Каждый элемент может быть текстовой строкой либо изображением. Пользователь нажимает элемент внутри сегментированного элемента управления, и данный элемент выбирается и подсвечивается.

Чтобы создать сегментированный элемент управления, нужно создать экземпляр класса `UISegmentedControl`. После создания экземпляра вы инициализируете элемент управления с помощью массива элементов. Этот массив может состоять из объектов `NSString` или `UIImage`. В одном сегментированном элементе управления можно комбинировать текст и изображения.

После создания и инициализации сегментированного элемента управления нужно добавить «цель-действие» для управляющего события `UIControlEventValueChanged:`. Когда изменится выбранный элемент, будет вызван целевой метод.

Чтобы считать порядковый номер выбранного элемента, используйте свойство `selectedSegmentIndex`, объявленное следующим образом:

```
@property(nonatomic) NSInteger selectedSegmentIndex
```

Значение данного элемента по умолчанию — `UISegmentedControlNoSegment`, показывающее, что ни один сегмент не выбран. Если вы установите свойство в это значение, не будет выбран (то есть подсвечен) ни один элемент.

Далее следует задать фрейм для сегментированного элемента управления, определяющий местоположение и измерения. Наконец, сегментированный элемент управления должен быть добавлен к существующему представлению. Листинг 5.9 отображает основные шаги, необходимые для отображения сегментированного элемента управления.

Листинг 5.9. Создание и настройка экземпляра UISegmentedControl

```
NSArray  *textOptionsArray = [NSArray  arrayWithObjects:
    @"Bart", @"Lisa", @"Maggie", nil];
segmentedCtrl = [[UISegmentedControl  alloc]
    initWithItems:textOptionsArray];
segmentedCtrl.frame = CGRectMake(20.0, 100.0, 280, 50);
[segmentedCtrl addTarget:self
    action:@selector(segmentChanged:)
    forControlEvents:UIControlEventValueChanged];
[theView  addSubview:segmentedCtrl];
```

Элемент управления в приведенном выше листинге содержит три текстовых элемента. Действие, вызываемое при нажатии элемента пользователем, — `segmentChanged:` (листинг 5.10).

Листинг 5.10. Метод действия segmentChanged:, вызываемый, когда сегментированный элемент управления изменит выбранный элемент

```
-(void)segmentChanged:(id)sender {
    if(segmentedCtrl == sender) {
      printf("The segment was changed to %d\n",
            [segmentedCtrl  selectedSegmentIndex]);
    }
}
```

На рис. 5.7 изображен текстовый сегментированный элемент управления, на рис. 5.8 — он же, но с выбранным средним элементом.

Рис. 5.7. Сегментированный элемент управления из текстовых элементов

Рис. 5.8. Сегментированный элемент управления с выбранным элементом

Чтобы создать сегментированный элемент управления, основанный на изображениях, следуйте той же процедуре за исключением фазы инициализации. Вы можете инициализировать элемент управления с помощью массива изображений следующим образом:

```
segmentedCtrl = [[UISegmentedControl  alloc]  initWithItems:
    [NSArray arrayWithObjects:
    [UIImage imageNamed:@"bart.png"],
    [UIImage imageNamed:@"lisa.png"],
    [UIImage imageNamed:@"maggie.png"],
    nil]];
```

На рис. 5.9. изображен сегментированный элемент управления, построенный на изображениях, а на рис. 5.10 — он же, но с выбранным средним элементом.

Рис. 5.9. Сегментированный элемент управления, построенный на изображениях

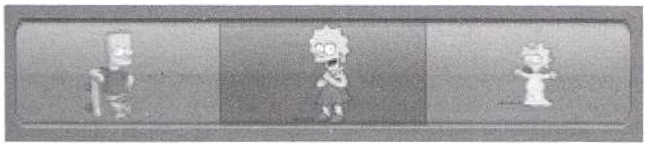

Рис. 5.10. Сегментированный элемент управления, состоящий из изображений, с выбранным элементом

В предыдущих примерах мы принимали внешний вид сегментированного элемента управления по умолчанию. Свойство `segmentedControlStyle` позволяет выбирать стиль элемента управления. Оно объявлено как

```
@property(nonatomic) UISegmentedControlStyle segmentedControlStyle
```

Доступные стили:

- `UISegmentedControlStylePlain` — стиль по умолчанию, который мы уже видели;
- `UISegmentedControlStyleBordered` — это стиль с границами; его пример приведен на рис. 5.11;

Рис. 5.11. Сегментированный элемент управления, состоящий из изображений, имеющий стиль UISegmentedControlStyleBordered

- `UISegmentedControlStyleBar` — стиль панели инструментов, пример на рис. 5.12.

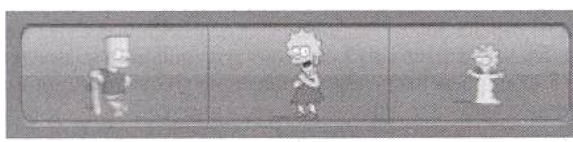

Рис. 5.12. Сегментированный элемент управления, состоящий из изображений, стиль UISegmentedControlStyleBar

В данном сегментированном элементе управления вы можете динамически изменять элементы. Можно использовать `setTitle:forSegmentAtIndex:`, чтобы изменить текстовый элемент, или `setImage:forSegmentAtIndex:` для изменения изображения элемента.

Вы также можете добавить к элементу управления новый сегмент. Если вы хотите добавить строковый элемент, используйте `insertSegmentWithTitle:atIndex:animated:`, объявленный как

```
- (void)insertSegmentWithTitle:(NSString *)title
atIndex:(NSUInteger)segment animated:(BOOL)animated
```

Если вы хотите добавить элемент-изображение, используйте `insertSegmentWithImage:atIndex:animated:`, объявленный следующим образом:

```
-(void)insertSegmentWithImage:(UIImage *)image
atIndex:(NSUInteger)segment animated:(BOOL)animated
```

Вы также можете удалять элементы, используя `removeSegmentAtIndex:animated:`, объявленный как

```
- (void)removeSegmentAtIndex:(NSUInteger)segment
animated:(BOOL)animated
```

Чтобы удалить все элементы, вызовите `removeAllSegments`.

5.7. Страничные элементы управления

Страничный элемент управления (рис. 5.13) отображается пользователю в виде множества горизонтальных точек, означающих страницы. Текущая страница представлена белой точкой. Пользователь может перейти с текущей страницы на следующую или предыдущую.

Рис. 5.13. Страничный элемент управления с 15 страницами

Чтобы представить страничный элемент управления пользователю, создайте новый экземпляр `UIPageControl` и инициализируйте его с помощью фрейма. Затем установите количество страниц (максимум 20). Чтобы реагировать на изменения в текущей странице, добавьте «цель-действие» для управляющего события `UIControlEventValueChanged`. Наконец, добавьте страничный элемент управления к представлению. В листинге 5.11 приведены основные шаги, необходимые для представления страничного элемента управления пользователю.

Листинг 5.11. Создание и настройка экземпляра UIPageControl

```
pageCtrl = [[UIPageControl  alloc]
    initWithFrame:CGRectMake(20.0, 100.0, 280, 50)];
[pageCtrl  addTarget:self
    action:@selector (pageChanged:)
    forControlEvents:UIControlEventTouchUpInside];
pageCtrl.numberOfPages = 15;
[theView addSubview:pageCtrl];
```

Метод действия приведен ниже. Чтобы считать текущую страницу, используйте свойство `currentPage`.

```
-(void)pageChanged:(id)sender {
    if(pageCtrl == sender) {
      printf ("The page was changed to %d\n",
            [pageCtrl  currentPage]);
    }
}
```

Вы также можете изменять текущую страницу программно и обновлять визуальный индикатор страницы с помощью `updateCurrentPageDisplay`.

5.8. Элементы выбора даты

`UIDatePicker` — это элемент управления, позволяющий пользователю выбирать время и дату, используя вращающиеся колесики. На рис. 5.14 вы видите четыре примера экземпляра `UIDatePicker`.

Рис. 5.14. Четыре примера UIDatePicker

Далее приведено несколько важных свойств и методов, объявленных этим классом.

- `calendar` — календарь, используемый в экземпляре UIDatePicker. Свойство объявлено как

```
@property(nonatomic, copy) NSCalendar *calendar
```

Если значение равно `nil`, применяется пользовательский календарь.

- `date` — это свойство представляет дату, используемую для отображения в элементе выбора даты. Объявлено следующим образом:

```
@property(nonatomic, copy) NSDate *date
```

- `setDate: animated:` — этот метод используется для изменения даты. Метод объявлен как

```
- (void) setDate:(NSDate *)date animated:(BOOL)animated
```

Если `animated` имеет значение `YES`, изменения анимируются.

- `datePickerMode` — используя это свойство, вы можете выбирать режим элемента выбора даты. Определено как

```
@property(nonatomic) UIDatePickerMode datePickerMode
```

Экземпляр `UIDatePicker` может быть настроен для выбора даты, времени, даты и времени либо действовать как обратный счетчик (см. рис. 5.14).

Листинг 5.12 показывает, как настроить элемент выбора даты и добавить его к представлению как дочерний.

Листинг 5.12. Создание и настройка экземпляра UIDatePicker

```
myDatePicker = [[UIDatePicker alloc] initWithFrame:CGRectZero];
myDatePicker.autoresizingMask = UIViewAutoresizingFlexibleWidth;
```

```
myDatePicker.datePickerMode = UIDatePickerModeDate;
CGSize  pickerSize = [myDatePicker  sizeThatFits:CGSizeZero];
rect = CGRectMake(0, 150, pickerSize.width, pickerSize.height);
myDatePicker.frame = rect;
[myDatePicker  addTarget:self
    action:@selector(datePickerChanged:)
forControlEvents:UIControlEventValueChanged];
[theView  addSubview: myDatePicker];
```

Экземпляры `UIDatePicker` отличаются от других элементов управления тем, что они внутренне оптимизируют свое расположение. Вам нужно только задать исходную точку в представлении — размер будет подсчитан автоматически. Приведенный выше код создает экземпляр и настраивает метод действия для получения измененного значения элемента управления.

Когда пользователь вращает колесико, элемент управления вызывает действующий метод сразу после остановки колесика. Листинг 5.13 демонстрирует действующий метод `datePickerChanged:`, который начинает работу по изменению значения.

Листинг 5.13. Действующий метод для изменения значения элемента выбора даты

```
- (void)datePickerChanged:(id)sender
{
    UIDatePicker *datePicker = sender;
    if(myDatePicker == datePicker) {
      printf("Value of picker is  %s\n",
            [[[myDatePicker  date]  description]  cString]);
    }
}
```

5.9. Резюме

В этой главе мы рассмотрели тему элементов управления в ОС iPhone. Мы начали с представления базового класса для всех элементов управления `UIControl` и его основных свойств. Затем мы поговорили о важном механизме «цель-действие», используемом для сообщения клиентам об изменениях. В данной главе были описаны текстовое поле, слайдер, переключатель, кнопка, страничный элемент управления, элемент выбора даты и сегментированные элементы управления.

Глава 6
Контроллеры представлений

«Модель-представление-контроллер» (Model-view-controller, MVC) – это популярный шаблон программирования, используемый при создании ПО, чтобы изолировать бизнес-логику от графического интерфейса пользователя. В MVC контроллер используется для координации модели (где находится бизнес-логика) и представления (где происходит взаимодействие с пользователем).

Из данной главы вы узнаете о доступных в iPhone SDK контроллерах представления. Можно создавать iPhone-приложения, не используя этих контроллеров, однако делать этого не стоит. Как вы узнаете далее, контроллеры представления существенно упрощают приложение.

Структура главы следующая. Разд. 6.1 представляет собой небольшое введение в тему контроллеров представления с примером простого приложения с единственным контроллером представления. Эта программа продемонстрирует важные концепции контроллеров представления. В разд. 6.2 мы поговорим о контроллерах панелей закладок и их использовании для создания радиоинтерфейсов. В разд. 6.3 будут рассмотрены контроллеры навигации, используемые в основном для представления иерархической информации пользователю. В разд. 6.4 будут описаны контроллеры модальных представлений и приведен подробный пример их использования. Итоги главы будут подведены в разд. 6.5.

6.1. Простейший контроллер представления

В этом разделе мы рассмотрим простейший контроллер представления. Приложение будет состоять из представления, контроллера представления и модели данных. Эта программа просто выводит сообщение, описывающее ориентацию устройства. Представление запрашивает у контроллера строку сообщения, контроллер определяет положение устройства и извлекает соответствующий текст из модели данных.

6.1.1. Контроллер представления

Листинг 6.1 демонстрирует объявление контроллера представления. Класс `UIViewController` – это базовый класс для всех контроллеров представления. Создавая контроллер представления, вы наследуете этот

класс либо один из его подклассов. Модель данных представлена тремя строками, каждая из которых описывает положение устройства. Метод message используется экземпляром представления для извлечения подходящего текста, описывающего ориентацию устройства.

Листинг 6.1. Объявление простейшего контроллера представления CDAViewController в файле CDAViewController.h

```
#import <UIKit/UIKit.h>
@interface CDAViewController : UIViewController {
    NSString *strPortraitNormal, *strPortraitUpSideDown, *strLandscape;
}
-(NSString*)message ;
@end
```

В листинге 6.2 приведена реализация контроллера представления CDAViewController. Мы переопределяем инициализационный метод initWithNibName:bundle:, чтобы инициализировать модель данных. Значение трех строк устанавливается согласно предназначению.

Мы создаем представление программно, поэтому нам требуется переопределить метод loadView. Наш метод создает экземпляр представления класса CDAUIView (который мы рассмотрим позднее) и настраивает его, чтобы он имел переменные высоту и ширину, путем установки свойства autoresizingMask в соответствующее значение. Представлению необходима ссылка на контроллер, поэтому мы также присваиваем свойству myController нашего представления значение соответствующего экземпляра контроллера представления.

Метод shouldAutorotateToInterfaceOrientation: также переопределен. Он вызывается в случае изменения ориентации устройства. Если вы возвращаете YES, то ориентация устройства изменяется; в противном случае изменений ориентации устройства не происходит. Наше приложение требует смены ориентации, поэтому мы возвращаем YES. Метод message — это метод, используемый представлением для получения текста, который требуется отобразить. Метод просто запрашивает текущую ориентацию устройства и возвращает подходящую строку, используя простейшую модель данных. Свойство interfaceOrientation класса UIViewController используется для получения текущей ориентации устройства. Свойство объявлено следующим образом:

```
@property(nonatomic, readonly)

UIInterfaceOrientation interfaceOrientation
```

Есть четыре ориентации типа UIInterfaceOrientation:

- UIInterfaceOrientationPortrait — показывает, что iPhone находится в портретной ориентации, когда кнопка Home (Домой) расположена внизу;

- `UIInterfaceOrientationPortraitUpsideDown` — означает, что iPhone находится в портретной ориентации, когда кнопка Home (Домой) расположена вверху;
- `UIInterfaceOrientationLandscapeLeft` — показывает, что iPhone находится в альбомной ориентации, когда кнопка Home (Домой) расположена слева;
- `UIInterfaceOrientationLandscapeRight` — означает, что iPhone находится в альбомной ориентации, когда кнопка Home (Домой) расположена справа.

Листинг 6.2. Реализация простейшего контроллера представления CDAViewController в файле CDAViewController.m

```
#import "CDAViewController.h"
#import "CDAUIView.h"
@implementation  CDAViewController
- (id)initWithNibName:(NSString *)nibNameOrNil
    bundle:(NSBundle *)nibBundleOrNil {
   if (self = [super initWithNibName:nibNameOrNil
            bundle:nibBundleOrNil]) {
            strPortraitNormal = @"Portrait";
            StrPortraitUpSideDown = @"Portrait UpSideDown";
            strLandscape = @"Landscape";
   }
   return   self;
}
- (void)loadView {
    CGRect  rectFrame = [UIScreen  mainScreen].applicationFrame;
    // Создает главное представление
    CDAUIView *theView =
            [[CDAUIView alloc]  initWithFrame:rectFrame];
    theView.backgroundColor = [UIColor   whiteColor ];
    theView.myController = self;
    theView.autoresizingMask =
            UIViewAutoresizingFlexibleHeight  |
            UIViewAutoresizingFlexibleWidth;
    self.view = theView;
    [theView  autorelease];
}
- (BOOL)shouldAutorotateToInterfaceOrientation:
    (UIInterfaceOrientation)interfaceOrientation {
   return YES;
}
- (NSString *)message {
    switch  (self.interfaceOrientation) {
     case  UIInterfaceOrientationPortrait:
        return  strPortraitNormal;
     case  UIInterfaceOrientationPortraitUpsideDown:
        return  strPortraitNormal;
     default:
        return  strLandscape;
    }
}
@end
```

6.1.2. Представление

Представление, управляемое контроллером представления, является экземпляром класса CDAUIView, объявленного в листинге 6.3. Представление имеет свойство, используемое для ассоциирования с управляющим контроллером представления.

Листинг 6.3. Объявление представления CDAUIView в CDAUIView.h, используемое для демонстрации простейшего контроллера представления

```
#import <UIKit/UIKit.h>
@class CDAViewController;
@interface CDAUIView : UIView {
    CDAViewController *myController;
}
@property(nonatomic, assign) CDAViewController* myController;
@end
```

Листинг 6.4 показывает реализацию класса представления. Класс переопределяет метод drawRect:. Метод просто запрашивает у контроллера текстовое сообщение и прорисовывает это сообщение в представлении.

Листинг 6.4. Реализация представления CDAUIView в CDAUIView.h, используемая для демонстрации простейшего контроллера представления

```
#import "CDAUIView.h"
@implementation CDAUIView
@synthesize myController;
-(void) drawRect:(CGRect) rect {
 [[myController message] drawAtPoint:CGPointMake(80, 30)
    withFont:[UIFont systemFontOfSize:50]];
}
@end
```

6.1.3. Делегат приложения

Листинг 6.5 демонстрирует объявление делегата приложения. Он содержит две переменные экземпляра — экземпляр окна и экземпляр контроллера представления.

Листинг 6.5. Объявление делегата приложения CDAAppDelegate в CDAAppDelegate.h, демонстрирующее простейший контроллер представления

```
@class CDAViewController;
@interface CDAAppDelegate : NSObject {
    UIWindow *window;
    CDAViewController *viewController;
}
@end
```

В листинге 6.6 приведена реализация класса делегата приложения. Как обычно, мы инициализируем объекты пользовательского интерфейса в методе `applicationDidFinishLaunching:`. Сначала мы создаем окно приложения. Затем создается и инициализируется экземпляр контроллера представления с помощью `initWithNibName:bundle:`. Мы создаем контроллер программно, поэтому передаем `nil` для обоих параметров. Затем представление контроллера добавляется к окну в качестве дочернего, и окно делается ключевым и видимым.

Класс `UIViewController` объявляет свойство `view` следующим образом:

```
@property(nonatomic, retain) UIView *view
```

Свойство `view` изначально равно `nil`. Когда к свойству производится доступ (как в выражении `viewController.view`), контроллер проверяет, не равняется ли оно `nil`. Если да, контроллер посылает самому себе сообщение `loadView`. Как вы видели ранее, представление создается в методе `loadView`, а значение свойства `view` устанавливается равным экземпляру, который мы создали, поэтому, когда представление контроллера добавляется к окну, в реальности в качестве дочернего к окну добавляется наш экземпляр `CDAUIView`.

Листинг 6.6. Реализация делегата приложения CDAAppDelegate в CDAAppDelegate.h, демонстрирующего простейший контроллер представления

```
#import  "CDAAppDelegate.h"
#import  "CDAViewController.h"
@implementation  CDAAppDelegate
-(void) applicationDidFinishLaunching:
    (UIApplication *) application {
  window = [[UIWindow  alloc]
            initWithFrame:[[UIScreen  mainScreen]  bounds]];
  viewController = [[CDAViewController  alloc]
            initWithNibName:nil  bundle:nil];
    [window  addSubview: viewController. view ];
    [window  makeKeyAndVisible];
}
-(void) dealloc {
 [window  release];
 [viewController  release];
 [super  dealloc];
}
@end
```

На рис. 6.1, 6.2 изображено приложение соответственно в портретной и альбомной ориентации.

Рис. 6.1. Приложение, демонстрирующее простейший контроллер представления, в портретной ориентации

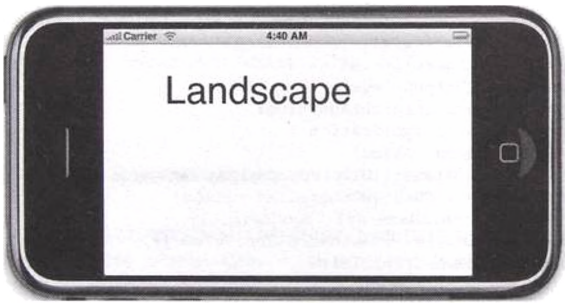

Рис. 6.2. Программа, показывающая простейший контроллер представления, в альбомной ориентации

6.1.4. Резюме

Обобщая материал по созданию простейшего MVC-приложения, вспомним основные пройденные нами шаги.

1. Создать подкласс `UIViewController`. Создать подкласс и переопределить следующие методы:

- `initWithNibName:bundle:` — инициализатор контроллера представления; можно инициализировать модель данных и экземпляр контроллера;
- `loadView` — этот метод используется для загрузки представления, управляемого контроллером; вы должны создать экземпляр представления, настроить его и установить его ссылку в свойство контроллера `view`;
- `shouldAutorotateToInterfaceOrientation:` — если ваше приложение позволяет смену ориентации, вы должны переопределить этот метод и возвратить `YES` для приемлемых ориентаций.

2. Создать подкласс `UIView`. Если вашему приложению требуется специализированное представление, вы должны унаследовать класс `UIView`. Опционально добавьте к контроллеру свойство, которое контроллер мог бы изменять для коммуникации с изменениями в модели данных, касающимися контроллера. С другой стороны, контроллер может взаимодействовать с делегатом приложения.

3. Создать класс делегата приложения. В методе `application-DidFinishLaunching:` вы должны создать главное окно и контроллер представления, после чего добавить свойство `view` контроллера представления как дочернее представление к главному окну.

6.2. Радиоинтерфейсы

Вам может потребоваться создать приложение, обладающее несколькими функциональностями либо работающее в параллельных режимах. Интерфейс такого приложения иногда называется радиоинтерфейсом. Каждая функциональность или режим будет управляться контроллером представления, и множество этих контроллеров будет определять приложение. Вы можете использовать контроллер панели закладок для управления несколькими контроллерами представления, схожими с изученными вами в предыдущем разделе. Каждый контроллер представляется в виде кнопки. Набор кнопок доступен внизу экрана на панели закладок, управляемой контроллером закладок. Когда пользователь нажимает кнопку, представление соответствующего контроллера становится видимым, а кнопка меняет внешний вид на сигнализирующий об ее активном режиме. Добавить панель закладок к вашему приложению несложно: вы просто создаете контроллер панели закладок, добавляете к нему множество контроллеров представления и прибавляете представление панели закладок к уже существующему. В этом разделе мы рассмотрим простое приложение, демонстрирующее основные шаги, необходимые для разработки программ, базирующихся на радиоинтерфейсах.

6.2.1. Детальный пример

В этом подразделе мы создадим простое приложение, использующее панель закладок. Снимок экрана приложения приведен на рис. 6.3.

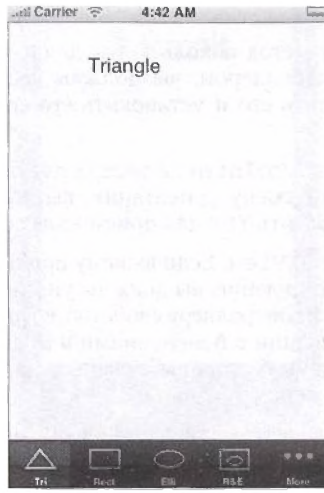

Рис. 6.3. Снимок экрана простейшего приложения с панелью закладок

Приложение представляет пользователю набор геометрических фигур. Каждому элементу соответствует контроллер представления, отображающий в своем представлении название фигуры при выборе какого-либо элемента.

Сначала нужно написать классы для контроллеров представлений каждого элемента. В этом примере для преставления каждого элемента мы используем единственный контроллер представления. Каждый экземпляр контроллера будет сконфигурирован для вывода сообщения. Обратите внимание: это обычная ситуация, когда каждый элемент (или режим) имеет собственный класс контроллера представления. Класс контроллера представления — `CDBViewController`. Он объявлен в листинге 6.7

Листинг 6.7. Объявление контроллера представления, реализующего каждый элемент в приложении с панелью закладок

```
#import <UIKit/UIKit.h>
@interface  CDBViewController : UIViewController {
    NSString  *message;
}
@property (nonatomic, retain)  NSString  *message;
- (id)initWithMessage:(NSString *)theMessage
    andImage:(UIImage*) image;
@end
```

Контроллер представления использует собственный инициализатор, `initWithMessage: andImage:`, инициализирующий контроллер с индивидуальным текстовым сообщением, которое отображается, когда контроллер становится активным, и изображением, представляющим этот элемент в списке на панели закладок.

Реализация класса контроллера представления показана в листинге 6.8. Инициализатор сначала вызывает метод инициализации суперкласса `initWithNibName: bundle:`. Мы собираемся строить наш графический интерфейс программно, поэтому передаем значения `nil` обоим параметрам. Затем инициализатор сохраняет пользовательское сообщение в свойстве `message` для дальнейшего использования управляемым представлением, а изображение, представляющее контроллер, — в свойстве `image` атрибута `tabBarItem` экземпляра контроллера представления.

Свойство `tabBarItem` класса `UIViewController` объявлено следующим образом:

```
@property (nonatomic, readonly, retain) UITabBarItem *tabBarItem
```

Значение свойства — объект (возможно, `nil`) класса `UITabBarItem`, являющийся контроллером представления панели закладок. Класс `UITabBarItem` — это подкласс `UIBarItem`. Он наследует у своего суперкласса свойства `image` и `title`. Значение изображения по умолчанию — `nil`. В то же время значение `title`, если оно не установлено, становится равным значению `title` контроллера представления. Класс `UITabBarItem` добавляет дополнительное свойство `badgeValue`, являющееся экземпляром `NSString`, и его значение отображается внутри красного овала справа от соответствующего элемента закладки (позже мы рассмотрим это подробнее).

Контроллер, как обычно, переопределяет метод `loadView` для настройки его управляющего представления. Этот метод идентичен изученному вами в предыдущем разделе. Класс представления — это пользовательский класс `CDBUIView`, который вы скоро увидите. Мы планируем добавить контроллер представления к контроллеру панели закладок, поэтому представление, управляемое контроллером, должно иметь возможность изменять размеры, чтобы вмещаться в область над панелью закладок. Важно установить свойство `autoresizingMask` управляемого представления так, как показано в методе.

Листинг 6.8. Реализация контроллера представления, используемого в приложении с панелью закладок

```
#import  "CDBViewController.h"
#import  "CDBUIView.h"
@implementation  CDBViewController
@synthesize  message;
-(id)initWithMessage:(NSString *)theMessage
    andImage:(UIImage*) image {
```

```
    if (self = [super initWithNibName:nil bundle:nil]) {
      self.message = theMessage;
      self.tabBarItem.image = image;
    }
    return self;
}
-(void)loadView {
    CGRect rectFrame =
          [UIScreen mainScreen].applicationFrame;
    CDBUIView *theView =
          [[CDBUIView alloc] initWithFrame:rectFrame];
    theView.backgroundColor = [UIColor whiteColor];
    theView.myController = self;
    theView.autoresizingMask =
          UIViewAutoresizingFlexibleHeight |
          UIViewAutoresizingFlexibleWidth;
    self.view = theView;
    [theView release];
}
@end
```

Класс представления `CDBUIView` объявлен в листинге 6.9. Он содержит свойство `myController` для хранения ссылки на его контроллер. Эта ссылка необходима, чтобы представление могло получать соответствующее сообщение, которое будет выведено в нем.

Листинг 6.9. Объявление класса представления, используемого в приложении с панелью закладок

```
#import <UIKit/UIKit.h>
@class CDBViewController;
@interface CDBUIView : UIView {
    CDBViewController *myController;
}
@property(nonatomic, assign) CDBViewController* myController;
@end
```

Реализация класса `CDBUIView` показана в листинге 6.10. Класс переопределяет метод `drawRect:` его суперкласса для прорисовки сообщения, полученного от контроллера.

Листинг 6.10. Реализация класса представления, используемого в приложении с панелью закладок

```
#import "CDBUIView.h"
@implementation CDBUIView
@synthesize myController;
- (void) drawRect:(CGRect)rect {
    [[myController message] drawAtPoint:CGPointMake(80, 30)
withFont:[UIFont systemFontOfSize:20]];
}
@end
```

Теперь, когда мы сконструировали контроллер представления и необходимый ему класс представления, можно использовать делегат при-

ложения, чтобы представить окно приложения пользователю. Объявление класса делегата приложения показано в листинге 6.11 В дополнение к ссылке на главное окно он содержит шесть ссылок на контроллеры представлений, все типа CDBViewController. Эти контроллеры добавятся к контроллеру панели закладок типа UITabBarController, ссылкой на который является переменная экземпляра tabBarController.

Листинг 6.11. Объявление класса делегата приложения, используемого для главного окна в приложении с панелью закладок

```
#import <UIKit/UIKit.h>
@class  CDBViewController;
@interface  CDBAppDelegate : NSObject {
    UIWindow  *window;
    CDBViewController *viewController1, *viewController2,
            *viewController3, *viewController4,
            *viewController5, *viewController6;
    UITabBarController *tabBarController;
}
@end
```

Листинг 6.12 демонстрирует реализацию класса делегата приложения. Делегат приложения переопределяет метод application-DidFinishLaunching: для настройки главного окна. Сначала мы создаем шесть экземпляров класса контроллера представления CDBViewController. Каждый из этих экземпляров инициализируется сообщением, которое будет отображаться в их представлении и изображении на панели закладок. В дополнение устанавливается заголовок контроллера представления, который влечет за собой то же значение заголовка элемента на панели закладок.

Файлы изображений хранятся в пакете приложения. Используя метод класса imageNamed:, мы получаем изображения, инкапсулируемые классом UIImage (более подробную информацию по этому вопросу вы найдете в гл. 9).

После создания контроллеров представления мы создаем контроллер панели закладок, который будет управлять ими. Это класс типа UITabBarController. Чтобы добавить контроллеры представлений, мы устанавливаем его свойство viewControllers. Оно объявлено следующим образом:

```
@property(nonatomic, copy) NSArray *viewControllers
```

Все, что нужно сделать, — это создать экземпляр NSArray с шестью контроллерами представления в качестве элементов и использовать его как значение свойства viewControllers.

Наконец, мы должны добавить представление панели закладок в качестве дочернего к главному окну. Вместе с главным окном появится и представление панели закладок, которое будет состоять из самой панели закладок

(внизу) и текущего выбранного представления над ним. Изначально будет выбран первый контроллер.

Листинг 6.12. Реализация класса делегата приложения, представляющего главное окно в приложении с панелью закладок

```
#import  "CDBAppDelegate.h"
#import  "CDBViewController.h"
@implementation  CDBAppDelegate
- (void) applicationDidFinishLaunching:(UIApplication *)
          application {
    window = [[UIWindow  alloc]
          initWithFrame:[[UIScreen  mainScreen]  bounds]];
    viewController1 = [[CDBViewController  alloc]
          initWithMessage:@"Triangle"
          andImage:[UIImage  imageNamed:@"tri.png"]];
    viewController1.title = @"Tri";
    viewController2 = [[CDBViewController  alloc]
          initWithMessage:@"Rectangle"
          andImage:[UIImage  imageNamed:@"rect.png"]];
    viewController2.title = @"Rect";
    viewController3 = [[CDBViewController alloc]
          initWithMessage:@"Ellipse"
          andImage:[UIImage  imageNamed:@"ellipse.png"]];
    viewController3.title = @"Elli";
    viewController4 = [[CDBViewController  alloc]
          initWithMessage:@"Rectangle+Ellipse"
          andImage:[UIImage  imageNamed:@"rect-elli.png"]];
    viewController4.title = @"R&E";
    viewController5 = [[CDBViewController  alloc]
          initWithMessage:@"Rectangle+Triangle"
          andImage:[UIImage  imageNamed:@"rect-tri.png"]];
    viewController5.title = @"R&T";
    viewController6 = [[CDBViewController  alloc]
          initWithMessage:@"Rectangle+Rectangle"
          andImage:[UIImage  imageNamed:@"two-tri.png"]];
    viewController6.title = @"R&R";
    tabBarController = [[UITabBarController  alloc] init];
    tabBarController.viewControllers =
          [NSArray  arrayWithObjects: viewController1,
                viewController2,
                viewController3,
                viewController4,
                viewController5,
                viewController6, nil];
    [window  addSubview:tabBarController.view];
    [window  makeKeyAndVisible];
}
- (void) dealloc {
    [window  release];
    [viewController1  release];
    [viewController2  release];
    [viewController3  release];
    [viewController4  release];
    [viewController5  release];
    [viewController6  release];
    [tabBarController  release];
    [super  dealloc];
}
@end
```

6.2.2. Некоторые комментарии к контроллерам панелей закладок

Остановимся на некоторых дополнительных аспектах контроллеров панелей закладок.

- **Список More (Еще).** Если требуется управлять более чем пятью контроллерами представления, панель закладок будет отображать первые четыре контроллера и дополнительный элемент More (Еще), добавляемый в качестве пятого контроллера. Нажатие More (Еще) выведет список оставшихся контроллеров. На рис. 6.4 показано, как изменится представление при нажатии More (Еще).

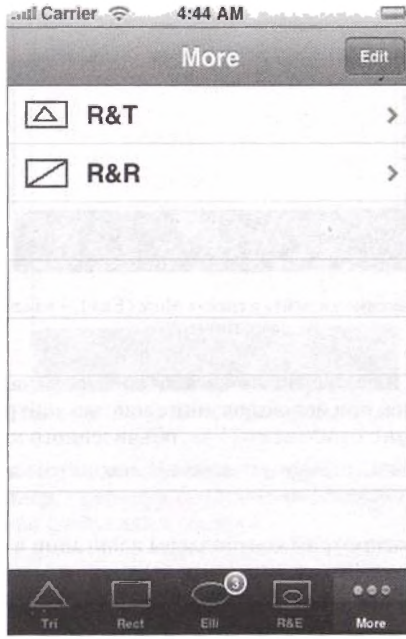

Рис. 6.4. Вывод на экран дополнительных элементов панели закладок при нажатии More (Еще)

Затем пользователь может нажать любой элемент для активации соответствующего контроллера представления. На рис. 6.5 вы видите, что происходит при нажатии пользователем элемента R&T. Пользователь может нажать кнопку More (Еще), чтобы вернуться к списку дополнительных элементов.

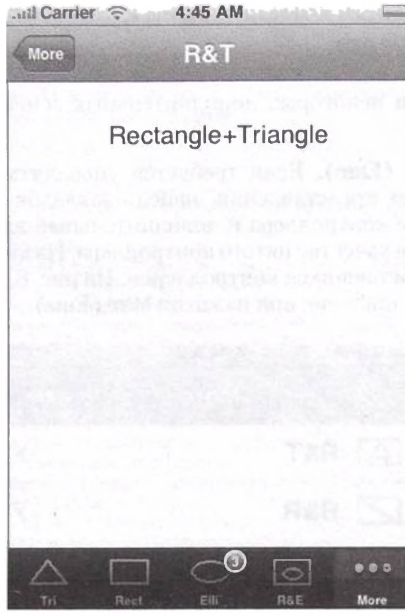

Рис. 6.5. Отображение элемента в списке More (Еще) — нажатие More (Еще) выводит список

Элемент More (Еще) управляется контроллером навигации, который может быть доступен при использовании свойства контроллера панели закладок `moreNavigationController`, объявленного как

```
@property(nonatomic, readonly) UINavigationController *
    moreNavigationController
```

Мы подробно рассмотрим контроллеры навигации в следующем разделе. Сейчас же необходимо отметить, что контроллеры навигации позволяют разработчику приложения представлять пользователю иерархическую информацию в привычном виде.

- **Бэйдж.** Каждый элемент панели закладок может иметь опциональное значение, отображаемое в его правом верхнем углу в красном овале. Свойство, контролирующее это, называется `badgeValue`, и объявлено следующим образом:

```
@property(nonatomic, copy) NSString *badgeValue
```

Значение этого свойства по умолчанию равно `nil`. Вы можете задать ему любое строковое значение, но обычно это короткое сообщение (напри-

мер, номер). К примеру, чтобы добавить значение бэйджа третьему контроллеру, вы можете написать:

```
viewController3.tabBarItem.badgeValue = @"3";
```

На рис. 6.6 вы видите эффект от этого выражения на панели закладок.

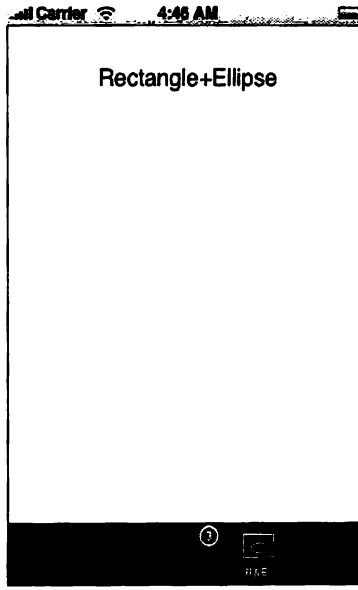

Рис. 6.6. Отображение значения бэйджа для контроллера представления на панели закладок

- **Выбранный контроллер.** Вы можете получать либо изменять выбранный контроллер, оперируя свойством `selectedViewController`. Оно объявлено следующим образом:

```
@property(nonatomic, assign) UIViewController *selectedViewController
```

Обратите внимание, что, если выбран элемент More (Еще), возвращается контроллер представления списка More (Еще). Заметьте также, что вы можете изменять выбранный контроллер представления для элементов, отображаемых на панели закладок. В нашем примере подобное выражение приведет к исключению `NSRangeException`:

```
tabBarController.selectedViewController = viewController5;
```

Вы также можете получать/устанавливать выбранный контроллер представления, используя свойство `selectedIndex`, объявленное как

```
@property(nonatomic) NSUInteger selectedIndex
```

Индекс 0 (выбран первый контроллер) — это значение по умолчанию. Если вы попытаетесь выбрать контроллер представления, элемент которого отсутствует на панели закладок, то получите `NSRangeException`. Свойства `selectedViewController` и `selectedIndex` связаны, и изменение одного повлечет за собой изменения другого.

- **Настройка.** Если вы располагаете более чем пятью контроллерами, управляемыми контроллером панели закладок, то можете дать пользователю возможность изменять позиции этих контроллеров. Только первые четыре контроллера появятся на главном экране, остальные же отобразятся в таблице, а пользователь может захотеть переместить некоторые из контроллеров из таблицы в главное окно.

Вы можете указать, что контроллер представления может быть настроен, поместив ссылку на него в массив `customizableViewControllers`, объявленный следующим образом:

```
@property(nonatomic, copy) NSArray *customizableViewControllers
```

Чтобы изменить позицию заданного контроллера, пользователь нажимает элемент More (Еще), а затем — кнопку Edit (Редактировать). Затем он жмет изображение контроллера и перетаскивает его на новую позицию. На рис. 6.7 вы видите элемент R&R в процессе перемещения на первую позицию, а на рис. 6.8 — состояние сразу после перемещения.

Рис. 6.7. Упорядочивание элементов панели закладок путем перетаскивания элемента R&R в начало

Рис. 6.8. Состояние панели закладок после перемещения элемента R&R в начало, но все еще находящееся в режиме редактирования

Контроллер, утративший свое место (в нашем примере это контроллер Tri), будет перемещен в таблицу (рис. 6.9).

Рис. 6.9. Вид панели закладок после перемещения элемента R&R и выхода из режима редактирования

По умолчанию, когда вы устанавливаете свойство `viewControllers`, те же ссылки на объекты помещаются в атрибут `customizableViewControllers`. Это означает, что все контроллеры представлений — настраиваемые. Если вы хотите закрепить один или несколько контроллеров представлений, вы должны изменить это свойство. Например, чтобы сделать настраиваемыми только первый, второй и пятый контроллеры, вы должны написать приблизительно следующее:

```
tabBarController.customizableViewControllers =
    [NSArray  arrayWithObjects:viewController1,
    viewController2, viewController5, nil];
```

6.3. Контроллеры навигации

Вам может понадобиться представить пользователю иерархическую информацию. Пользователь начинает с верхнего уровня иерархии. Он нажимает заданный элемент, после чего выводится следующий уровень иерархии. Процесс углубления продолжается, пока пользователь не достигнет нужного уровня.

Для управления иерархическими представлениями предназначен класс `UINavigationController`. Как вы видели в предыдущем разделе, контроллер управляет контроллерами представления, и каждый контроллер управляет текущим представлением для данного уровня. Этот класс представляет пользователю панель навигации и представление текущего уровня иерархии. Из разд. 8.8 вы узнаете, насколько хороши для такого отображения табличные представления данных. В этом разделе мы будем использовать контроллер навигации с табличными представлениями для вывода иерархической информации в дружественной пользователю манере. В этом разделе мы изучим основные механизмы класса `UINavigationController`. Сначала мы рассмотрим подробный пример, показывающий поведение этого класса по умолчанию, а затем — некоторые доступные параметры настроек.

6.3.1. Пример поведения класса навигации

В этом подразделе мы рассмотрим подробный пример с контроллером навигации. Приложение имеет три уровня иерархии — `Level I`, `Level II` и `Level III`. Для простоты примера пользователь будет перемещаться на следующий уровень иерархии касанием представления предыдущего уровня, все контроллеры представления, управляемые контроллером навигации, будут являться экземплярами одного и того же класса, а сообщение, отображаемое внутри каждого представления, будет идентифицировать уровень иерархии.

Первый уровень иерархии изображен на рис. 6.10 — это панель навигации и представление контроллера, отображенное под ним.

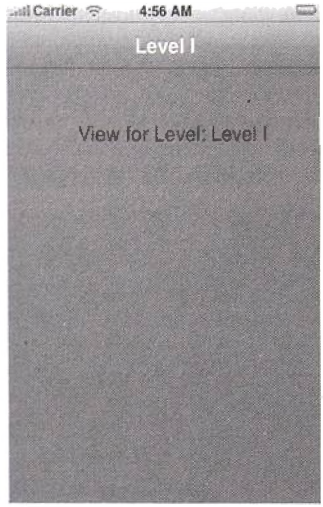

Рис. 6.10. Приложение контроллера навигации, показывающее первый уровень иерархии

Посередине панели навигации расположен заголовок. По умолчанию он идентичен свойству заголовка контроллера представления, находящегося посередине. На рис. 6.11 вы видите снимок экрана приложения в момент, когда пользователь касается представления первого уровня.

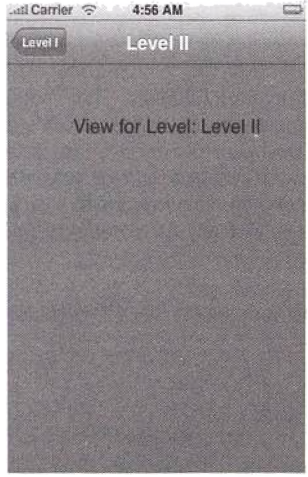

Рис. 6.11. Приложение контроллера навигации, отображающее второй уровень иерархии

Второе представление появляется, помещая новый контроллер представления на вершину стека навигации, управляемого контроллером навигации. Обратите внимание, что по умолчанию кнопка возврата, появляющаяся слева, имеет заголовок предыдущего уровня. Касание кнопки возврата приведет к извлечению из стека текущего контроллера представления и появлению представления предыдущего контроллера. На рис. 6.12 изображен третий, и последний, уровень иерархии.

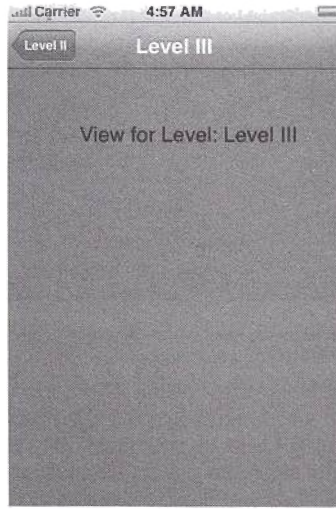

Рис. 6.12. Приложение контроллера навигации, показывающее третий уровень иерархии

Контроллер представления. Начнем с построения классов контроллеров представления, чьи экземпляры будут помещаться в стек навигации и извлекаться из него. Для простоты представим, что все контроллеры представления являются экземплярами одного класса `CDCViewController`. В листинге 6.13 приведено объявление этого класса. Он объявляет метод `showNextLevel`, используемый представлением для отображения следующего уровня иерархии.

Листинг 6.13. Объявление контроллера представления, используемое в примере контроллера навигации

```
#define  LEVELI @"Level I"
#define  LEVELII @"Level II"
#define  LEVELIII @"Level III"
@interface  CDCViewController : UIViewController {
}
-(void)showNextLevel;
@end
```

Листинг 6.14 представляет реализацию контроллера представления. Метод `showNextLevel` использует делегат приложения для помещения контроллера представления следующего уровня на вершину стека контроллера навигации (который также управляется делегатом приложения). Для получения ссылки на одиночный делегат приложения используйте метод `sharedApplication` класса `UIApplication`. Метод `loadView` идентичен изученному вами ранее. Для представления он использует класс `CDCUIView`.

Листинг 6.14. Реализация контроллера представления, используемая в примере контроллера навигации

```
#import  "CDCViewController.h"
#import  "CDCUIView.h"

@implementation  CDCViewController

- (void) showNextLevel {
    [[[UIApplication  sharedApplication]  delegate]
            showNextLevel:self.title];
}
- (void)loadView {
    CGRect rectFrame =
            [UIScreen  mainScreen].applicationFrame;
    CDCUIView *theView =
            [[CDCUIView  alloc]  initWithFrame:rectFrame];
    theView.backgroundColor = [UIColor  grayColor];
    theView.myController = self;
    theView.autoresizingMask =
            UIViewAutoresizingFlexibleHeight |
            UIViewAutoresizingFlexibleWidth;
    self.view = theView;
    [theView  release];
}
@end
```

Представление. Листинг 6.15 демонстрирует объявление класса представления `CDCUIView`, используемого контроллером представления. Представление содержит ссылку на собственный контроллер в свойстве `myController`. Далее мы увидим, как используется эта ссылка в методе, перехватывающем касания пользователя, чтобы перейти на следующий уровень.

Листинг 6.15. Объявление класса представления, используемого в примере контроллера навигации

```
@class  CDCViewController;

@interface CDCUIView : UIView   {
    CDCViewController *myController;
}
@property(nonatomic, assign)  CDCViewController*  myController;
@end
```

Листинг 6.16 показывает реализацию класса представления. Как было сказано ранее, для перехода на следующий уровень пользователь касается представления. Метод `touchesBegan: withEvent:` перехватывает касание и вызывает метод контроллера `showNextLevel`, который, в свою очередь, вызывает метод делегата приложения `showNextLevel:`. Метод `drawRect:` используется для вывода уникального сообщения в пространстве представления, специфического для каждого из трех уровней навигации.

Листинг 6.16. Реализация класса представления, используемого в примере контроллера навигации

```
#import  "CDCUIView.h"
#import  "CDCViewController.h"

@implementation CDCUIView

@synthesize  myController;

-(void)touchesBegan:(NSSet *)touches  withEvent:(UIEvent *)event {
 [myController showNextLevel];
}

-(void)drawRect:(CGRect)rect {
  NSString  *message;
  message = [NSString  stringWithFormat:@"View for Level: %@",
    [myController  title]];
    [message drawAtPoint:CGPointMake(70, 50)
    withFont:[UIFont  systemFontOfSize:20]];
}
@end
```

Делегат приложения. Листинг 6.17 представляет объявление класса делегата приложения. Он отслеживает положение окна и трех контроллеров представления. В дополнение он оперирует ссылкой на контроллер навигации. Делегат приложения также объявляет метод `showNextLevel:`, который будет вызван контроллером представления для перехода на следующий уровень.

Листинг 6.17. Объявление класса делегата приложения, используемого в примере контроллера навигации

```
@class  CDCViewController;

@interface  CDCAppDelegate : NSObject {
    UIWindow *window;
    CDCViewController *levelI, *levelII, *levelIII;
    UINavigationController *navController;
}

-(void)showNextLevel:(NSString*) level;
@end
```

Листинг 6.18 демонстрирует реализацию класса делегата приложения. Метод `applicationDidFinishLaunching:` используется для инициа-

лизации графического пользовательского интерфейса приложения. Он начинает с создания окна и контроллера представления первого уровня иерархии. Затем создается и инициализируется контроллер навигации, который является экземпляром класса `UINavigationController` подкласса `UIViewController`. Экземпляр контроллера навигации инициализируется методом `initWithRootViewController:`, который объявлен следующим образом:

```
- (id)initWithRootViewController:(UIViewController *) rootViewController
```

Инициализатор имеет единственный параметр — экземпляр контроллера представления, которое станет первым (корневым) уровнем иерархии. Контроллер помещается в стек (пустой) без анимации. После создания и инициализации контроллера навигации мы добавим его представление к окну в качестве дочернего. В результате панель навигации добавится ниже панели состояния, а представление корневого контроллера — под панелью навигации.

Метод `showNextLevel:` принимает в качестве параметра имя текущего уровня. Он сохраняет контроллер второго уровня, если текущий уровень является корневым, и контроллер третьего уровня, если текущий уровень — второй. Чтобы положить новый контроллер представления в стек, сначала нужно создать его, а затем, используя метод `pushViewController: animated:`, положить в стек. Этот метод объявлен следующим образом:

```
- (void)pushViewController:(UIViewController *)viewController
    animated:(BOOL)animated
```

Если `animated` равно `YES`, переход на следующий уровень анимируется. По умолчанию, когда контроллер представления кладется в стек, заголовок текущего контроллера представления становится заголовком левой кнопки. Заголовок в середине изменится в соответствии с недавно положенным контроллером представления. Когда пользователь касается левой кнопки, текущий контроллер представления извлекается из стека, представление предыдущего контроллера замещает представление внизу панели навигации, заголовок в середине панели навигации заменяется заголовком предыдущего контроллера представления и, соответственно, изменяется заголовок кнопки возврата. Метод для извлечения верхнего контроллера представления — `popViewControllerAnimated:`, объявленный следующим образом:

```
- (UIViewController *)popViewControllerAnimated:(BOOL)animated
```

Если `animated` равно `YES`, извлечение анимируется, в противном случае — нет. Обратите внимание, что метод также возвращает ссылку на извлеченный контроллер представления. Не проблема, если контроллер представления в стеке будет только один (корневой) — метод не сможет извлечь его и просто закончит работу без создания исключения.

Листинг 6.18. Реализация класса делегата приложения, используемого в примере контроллера навигации

```
#import "CDCAppDelegate.h"
#import "CDCViewController.h"

@implementation   CDCAppDelegate

-(void) showNextLevel:(NSString *)  level {
    if ([level  isEqualToString:LEVELI]) {
            levelII = [[CDCViewController  alloc]
                    initWithNibName: nil  bundle:nil];
            levelII.title = LEVELII;
            [navController  pushViewController:levelII animated:YES];
    }
    else if ([level isEqualToString:LEVELII]) {
            levelIII = [[CDCViewController  alloc]
                    initWithNibName:nil  bundle:nil];
            levelIII.title = LEVELIII;
            [navController  pushViewController:levelIII
                    animated:YES];
    }
}

- (void) applicationDidFinishLaunching:(UIApplication *)  application {
    window = [[UIWindow  alloc]
            initWithFrame:[[UIScreen  mainScreen]  bounds]];
    levelI = [[CDCViewController  alloc]
             initWithNibName:nil bundle:nil];
    levelI.title = LEVELI;
    navController = [[UINavigationController  alloc]
            initWithRootViewController:levelI];
    [window  addSubview:navController.view];
    [window  makeKeyAndVisible];
}

- (void) dealloc {
    [window  release];
    [levelI  release];
    [levelII  release];
    [levelIII  release];
    [navController  release];
    [super  dealloc];
}
@end
```

6.3.2. Настройка

В предыдущем подразделе мы рассмотрели поведение по умолчанию контроллеров навигации. Теперь мы изучим способы настройки внешнего вида и поведения панелей навигации.

Элемент навигации. Каждый контроллер представления ассоциируется с элементом навигации на панели навигации. Элемент навигации — это экземпляр класса `UINavigationItem`. Класс объявляет несколько свойств, определяющих внешний вид контроллера представления, когда он помещается в стек или когда другой контроллер помещается на вершину стека (то есть он становится непосредственно дочерним контроллером). По умолчанию, когда контроллер представления помещается в стек, заголовок в середине панели навигации ста-

новится идентичным заголовку контроллера представления. Когда другой контроллер помещается в стек, заголовок кнопки возврата становится таким же, как и заголовок текущего активного контроллера представления (который должен стать дочерним). Чтобы изменить такое поведение, вы можете получить доступ к элементу навигации каждого контроллера представления и определить значения различных его свойств вместо установленных по умолчанию. Чтобы получить экземпляр элемента навигации, используйте свойство `navigationItem`, объявленное в классе `UIViewController` следующим образом:

```
@property(nonatomic, readonly, retain) UINavigationItem *navigationItem
```

Рассмотрим основные свойства элемента навигации.

- **Заголовок.** Для установки заголовка панели навигации, когда контроллер представления находится на вершине стека, определите свойство `title` его элемента навигации. Оно объявлено следующим образом:

```
@property(nonatomic, copy) NSString *title
```

Например, для установки заголовка элемента навигации контроллера представления `ctrl1` измените свойство `title` так:

```
ctrl1.navigationItem.title = @"Nav Title 1";
```

Каждый раз, когда `ctrl1` будет находиться на вершине стека, заголовком панели навигации будет `Nav Title 1`.

- **Подсказка.** Существует опциональный текст, который может отображаться над кнопками панели навигации. Чтобы использовать эту возможность, установите свойство `prompt` элемента навигации, объявленное следующим образом:

```
@property(nonatomic, copy)  NSString  *prompt
```

Например, следующий код приведет к появлению подсказки, как на рис. 6.13:

```
ctrl1.navigationItem.prompt = @"The Prompt 1";
```

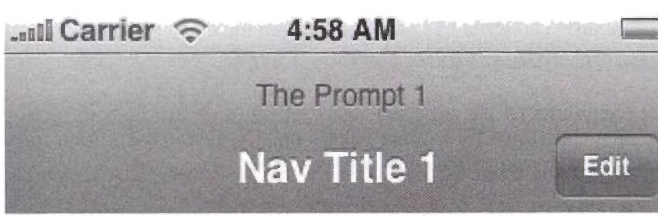

Рис. 6.13. Настройка панели навигации для вывода пользовательских кнопок Влево, Вправо, подсказки и заголовка

- **Правая/левая кнопки.** Вы можете добавить правую и/или левую кнопки на панель навигации. Чтобы создать правую кнопку, которая появится,

когда контроллер представления будет верхним в стеке, установите свойство `rightBarButtonItem`, объявленное следующим образом:

```
@property (nonatomic, retain) UIBarButtonItem  *rightBarButtonItem
```

Чтобы добавить левую кнопку на панель навигации, установите свойство `leftBarButtonItem`, объявленное как

```
@property (nonatomic, retain) UIBarButtonItem * leftBarButtonItem
```

Обратите внимание, что эта левая пользовательская кнопка заменит на панели обычную кнопку возврата, если таковая есть.

Например, для создания правой кнопки вы можете записать в инициализаторе контроллера представления приблизительно следующее:

```
UIBarButtonItem * rightButton =
    [[UIBarButtonItem  alloc]
    initWithTitle:@"Right"
            style:UIBarButtonItemStyleDone
            target:self
            action:@selector(rightButtonTapped:)
            ];
self.navigationItem.rightBarButtonItem = rightButton;
[rightButton  release];
```

В приведенном выше коде мы создаем экземпляр `UIBarButtonItem` и инициализируем его с параметрами заголовка, стиля и метода, который будет вызван при нажатии кнопки. Возможно, вам следует еще раз прочесть подразд. 5.1.2, чтобы лучше понять механизм «цель-действие».

Выше (см. рис. 6.13) изображена панель навигации с пользовательскими левой и правой кнопками. Правая создана с помощью приведенного выше кода и имеет стиль `UIBarButtonItemStyleDone`. Левая кнопка создана с использованием стиля `UIBarButtonItemStylePlain`. В вашем распоряжении есть еще один стиль — `UIBarButtonItemStyleBordered`.

- **Заглавное представление.** На выбор вы можете показывать представление вместо заголовка панели навигации. Чтобы задать представление, установите свойство `titleView`, объявленное следующим образом:

```
@property(nonatomic, retain) UIView *titleView
```

Для наглядности рассмотрим класс представления `MyBarView`, приведенный в листинге 6.19. Этот класс просто реализует представление с белым фоном и прописывает внутри текст `Hello`.

Листинг 6.19. Пользовательское представление, которое заменит заголовок в панели навигации

```
@interface MyBarView : UIView {}
@end

@implementation MyBarView
```

```
-(id)initWithFrame:(CGRect)frame {
    if (self = [super initWithFrame:frame]) {
      self.backgroundColor = [UIColor whiteColor];
    }
    return self;
}

- (void)drawRect:(CGRect)rect {
  [@"Hello" drawAtPoint:CGPointMake(55, 5)
    withFont:[UIFont systemFontOfSize:20]];
}
@end
```

Чтобы заменить текст заголовка представлением (рис. 6.14) для контроллера `ctrl1`, запишите приблизительно следующее:

```
MyBarView *titleView = [[[MyBarView alloc]
    initWithFrame:CGRectMake(0, 0, 150, 30)] autorelease];
ctrl1.navigationItem.titleView = titleView;
```

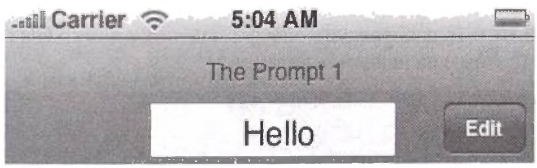

Рис. 6.14. Панель навигации с заглавным пользовательским представлением

- **Поддержка редактирования.** Некоторые подклассы `UIViewController`, такие как `UITableViewController`, поддерживают редактирование представления. Когда представление, управляемое контроллерами, может редактироваться, справа у него обычно есть кнопка Edit (Редактировать). Когда пользователь нажимает ее, предполагается, что представление переходит в режим редактирования, а заголовок кнопки меняется на Done (Готово). Закончив редактирование, пользователь нажимает кнопку Done (Готово), и контроллер представления должен сохранить изменения.

Этот механизм встроен, и его использование не требует больших усилий. Во-первых, класс `UIViewController` содержит метод для перехода в режим редактирования. Это метод `setEditing:animated:`, объявленный следующим образом:

```
- (void)setEditing:(BOOL)editing  animated:(BOOL)animated
```

Подклассы `UIViewController` переопределяют этот метод, чтобы реагировать соответствующим образом.

Кроме того, `UIViewController` объявляет свойство `editing`, которое можно установить для изменения режима редактирования контроллера представления. Свойство объявлено как

```
@property (nonatomic, getter=isEditing) BOOL editing
```

Когда вы меняете значение этого свойства, вызывается `setEditing:animated:` с соответствующими изменениями. К тому же когда пользователь нажимает кнопку Edit (Редактировать)/Done (Готово), значение свойства меняется соответственно.

Чтобы добавить справа кнопку Edit (Редактировать)/Done (Готово), когда `ctrl1` является активным контроллером представления, вы можете записать приблизительно следующее:

```
ctrl1.navigationItem.rightBarButtonItem = [ctrl1 editButtonItem];
ctrl1.editing = NO;
```

Обратите внимание, как мы получили кнопку, используя метод экземпляра контроллера представления `editButtonItem`, который объявлен следующим образом:

```
- (UIBarButtonItem *) editButtonItem
```

На рис. 6.15, 6.16 изображены, соответственно, кнопки Edit (Редактировать) и Done (Готово) на панели навигации.

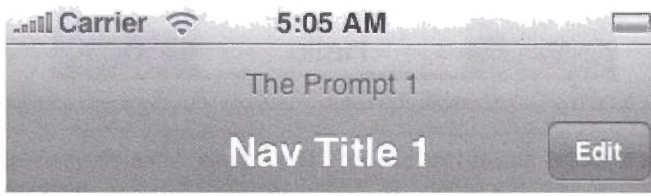

Рис. 6.15. Кнопка Edit (Редактировать) на панели навигации

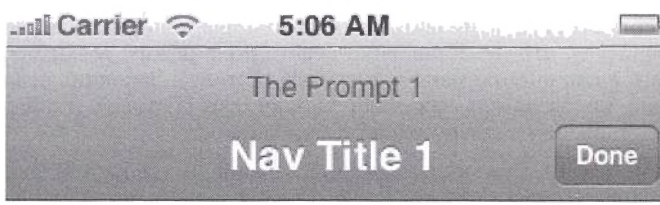

Рис. 6.16. Кнопка Done (Готово) на панели навигации

6.4. Модальные контроллеры представления

Модальный контроллер представления позволяет создавать перекрывающий контроллер представления поверх существующего. Когда модальный контроллер представления отображается пользователю, он занимает все пространство снизу от панели состояния. Как уровни навигации по-

являются с анимацией слева направо, так и модальные контроллеры представления появляются с анимацией снизу вверх.

Каждый контроллер представления может создать не более одного модального контроллера за один раз. Свойство `modalViewController` содержит ссылку на модальный контроллер представления. Свойство объявлено как

```
@property(nonatomic, readonly)  UIViewController *
    modalViewController;
```

Также контроллер представления, будучи созданным как модальный контроллер другим контроллером представления, содержит ссылку на родительский контроллер, используя свойство `parentViewController`, объявленное следующим образом:

```
@property(nonatomic, readonly)  UIViewController *
    parentViewController
```

Чтобы отобразить модальный контроллер, контроллер представления вызывает метод экземпляра `presentModalViewController:animated:`, который объявлен как

```
- (void)presentModalViewController:
    (UIViewController *)modalViewController
    animated:(BOOL)animated
```

Когда появится представление модального контроллера и пользователь закончит взаимодействовать с ним, используется механизм (обычно кнопка на панели навигации) для скрытия модального контроллера. Чтобы скрыть модальный контроллер представления, родительский контроллер должен вызвать метод `dismissModalViewControllerAnimated:`, объявленный как

```
- (void)dismissModalViewControllerAnimated:(BOOL)animated
```

Пример модального контроллера. Рассмотрим модальные контроллеры представления подробнее. В следующем примере мы создадим приложение, которое сначала отображает контроллер представления, управляемый контроллером навигации, а когда пользователь касается представления, другой контроллер навигации выводится модально (рис. 6.17).

Родительским классом контроллера представления является `MainViewController`, который объявлен в листинге 6.20.

Листинг 6.20. Объявление MainViewController, показывающего модальный контроллер представления

```
@class  SecondaryViewController;
@interface  MainViewController : UIViewController {
    Secondary ViewController  *secondaryCtrl1;
    UINavigationController  *secondaryNavigationCtrl;
}
-(void) showModalController;
-(void) dismissModelController;
@end
```

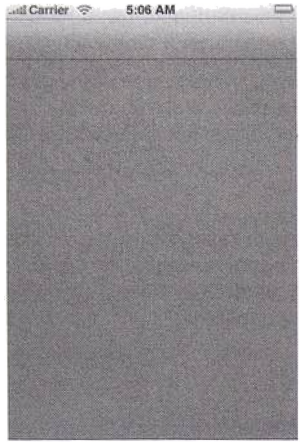

Рис. 6.17. Представление MainViewController и панель навигации

Контроллер содержит ссылку на модальный контроллер представления и создаст его, когда пользователь коснется представления. Хранится также ссылка на контроллер навигации, который будет создан в это же время. Как мы вскоре увидим, метод `showModalController` будет вызван из его представления, и метод `dismissModelController` является методом действия кнопки Dismiss (Отклонить) панели навигации, находящейся на модальном контроллере (рис. 6.18). Листинг 6.21 демонстрирует реализацию данного класса.

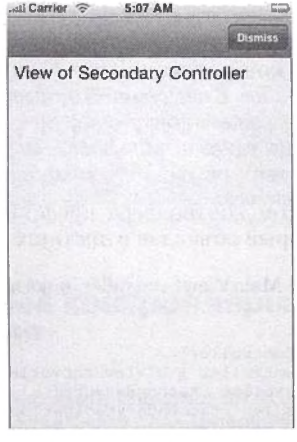

Рис. 6.18. Представление модального контроллера и панель навигации с кнопкой Dismiss (Отклонить)

Листинг 6.21. Реализация MainViewController, показывающего модальный контроллер представления

```
#import  "MainViewController.h"
#import  "SecondaryViewController.h"
#import  "MainView.h"

@implementation  MainViewController

- (void)loadView {
    CGRect  rectFrame = [UIScreen  mainScreen].applicationFrame;
    MainView  *theView =
            [[MainView  alloc ]  initWithFrame:rectFrame];
    theView.myController = self;
    theView.backgroundColor = [UIColor  grayColor];
    theView.autoresizingMask  =
            UIViewAutoresizingFlexibleHeight |
            UIViewAutoresizingFlexibleWidth;
    self.view = theView;
    [theView  release];
}
- (void) showModalController {
    secondaryCtrl1 = [[SecondaryViewController  alloc] initWithNibName: nil
            bundle:nil  parent:self];
    secondaryNavigationCtrl  = [[ UINavigationController   alloc ]
            initWithRootViewController:secondaryCtrl1];
    [self presentModalViewController:secondaryNavigationCtrl
            animated:YES];
}
-(void) dismissModelController {
    [secondaryCtrl1  release];
    [secondaryNavigationCtrl  release];
    [self  dismissModalViewControllerAnimated:YES];
}
@end
```

Метод `loadView` схож с методами `loadView`, которые вы видели до настоящего момента. Созданное представление имеет тип `MainView`, с которым вы скоро ознакомитесь. Когда пользователь касается представления, оно вызывает метод `showModalController`. Этот метод создает контроллер представления типа `SecondaryViewController` и делает его корневым для нового контроллера навигации. Затем он отображает контроллер навигации и панель навигации над собой, вызвав метод `presentModalViewController:animated:`, анимирующий появление снизу вверх. Как мы скоро увидим, `SecondaryViewController` добавляет кнопку Dismiss (Отклонить) к панели навигации, создает механизм «цель-действие» для данного экземпляра контроллера и метод `dismissModelController`. Метод просто уничтожает представление вместе с контроллерами навигации и вызывает метод `dismissModalViewControllerAnimated:`, анимирующий исчезновение контроллера навигации сверху вниз.

Класс представления родительского контроллера объявлен в листинге 6.22. Он схож с тем, что мы видели до настоящего момента.

Листинг 6.22. Объявление класса MainView, который используется в качестве представления для контроллера, выступающего в роли модального контроллера представления

```
@class  MainViewController;

@interface MainView : UIView {
     MainViewController *myController;
}
@property(nonatomic,  assign)  MainViewController *myController;
@end
```

Реализация данного класса показана в листинге 6.23. Представление переопределяет метод `touchesBegan: withEvent:` для запроса контроллера представления на отображение модального контроллера представления.

Листинг 6.23. Реализация класса MainView, используемого в качестве представления для контроллера, который выступает в роли модального контроллера представления

```
#import  "MainView.h"
#import  "MainViewController.h"

@implementation MainView

@synthesize  myController;
- (void) touchesBegan:(NSSet *)touches  withEvent:(UIEvent *)event {
    [myController  showModalController];
}
@end
```

Листинг 6.24 демонстрирует объявление класса `SecondaryViewController`. Инициализатор содержит ссылочный параметр на родительский контроллер представления, который станет целевым для кнопки Dismiss (Отклонить).

Листинг 6.24. Объявление класса SecondaryViewController

```
@class  MainViewController;

@interface  SecondaryViewController : UIViewController {
}
-(id) initWithNibName:(NSString *)nibNameOrNil
bundle:(NSBundle *)nibBundleOrNil
parent:(MainViewController*) my Parent;
@end
```

Листинг 6.25 показывает реализацию класса `SecondaryViewController`. Инициализатор добавляет правую кнопку Dismiss (Отклонить) на панель навигации.

Листинг 6.25. Реализация класса SecondaryViewController

```
#import  "SecondaryViewController.h"
#import  "SecondaryView.h"
#import  "MainViewController.h"

@implementation  SecondaryViewController
- (id) initWithNibName:(NSString *)nibNameOrNil
bundle:(NSBundle *)nibBundleOrNil
```

```
parent:(MainViewController*)  myParent {
    if (self = [super  initWithNibName:nibNameOrNil
        bundle:nibBundleOrNil]) {
      UIBarButtonItem  *rightButton = [[UIBarButtonItem  alloc]
            initWithTitle:@"Dismiss"
            style:UIBarButtonItemStyleDone
            target:myParent
            action:@selector(dismissModelController)
            ];
            self.navigationItem.rightBarButtonItem = rightButton;
            [rightButton  release];
    }
    return self;
}

- (void)loadView {
    CGRect  rectFrame = [UIScreen  mainScreen].applicationFrame;
    SecondaryView *theView =
            [[SecondaryView  alloc]  initWithFrame:rectFrame];
    theView.backgroundColor = [UIColor  yellowColor];
    theView.autoresizingMask =
            UIViewAutoresizingFlexibleHeight |
            UIViewAutoresizingFlexibleWidth;
    self.view = theView;
    [theView  release];
}
@end
```

Листинг 6.26 представляет класс SecondaryView, используемый контроллером представления SecopndaryViewController.

Листинг 6.26. Класс SecondaryView

```
@interface  SecondaryView : UIView {
}
@end

@implementation  SecondaryView
- (void)drawRect:(CGRect)rect {
  [@"View of Secondary Controller"
    drawAtPoint:CGPointMake(10, 5)
    withFont:[UIFont systemFontOfSize:20]];
}
@end
```

Чтобы объединить все предыдущие фрагменты, в листингах 6.27, 6.28 приведен класс делегата приложения. Делегат программы содержит ссылку на контроллер навигации и его единственный контроллер представления. Он создает контроллер навигации, инициализирует его с контроллером представления и отображает пользователю.

Листинг 6.27. Объявление класса делегата приложения для программы с модальным контроллером представления

```
@class  MainViewController;

@interface  CDEAppDelegate : NSObject <UIApplicationDelegate> {
```

```
    UIWindow  *window;
    MainViewController *ctrl1;
    UINavigationController  *navCtrl1 ;
}
@property (nonatomic, retain)  UIWindow  *window;
@end
```

Листинг 6.28. Реализация класса делегата приложения для программы с модальным контроллером представления

```
#import  "CDEAppDelegate.h"
#import  "MainViewController.h"

@implementation  CDEAppDelegate
@synthesize  window;
- (void) applicationDidFinishLaunching:(UIApplication *)
          application {
    window = [[UIWindow  alloc]
          initWithFrame:[[UIScreen mainScreen]  bounds]];
    ctrl1 = [[MainViewController  alloc]
          initWithNibName:nil  bundle:nil];
    navCtrl1 = [[UINavigationController  alloc]
          initWithRootViewController:ctrl1];
    [window  addSubview:navCtrl1.view];
    [window  makeKeyAndVisible];
}

- (void) dealloc {
    [window  release];
    [ctrl1  release];
    [navCtrl1  release];
    [super  dealloc];
}
@end
```

6.5. Резюме

В этой главе мы рассмотрели тему контроллеров представления. В разд. 6.1 было предоставлено введение в понятие контроллеров представления на примере простого приложения с единственным контроллером представления. Эта программа демонстрировала важные концепции контроллеров представления. Из разд. 6.2 вы узнали о контроллерах панели закладок и их использовании при создании радиоинтерфейсов. В разд. 6.3 были описаны контроллеры навигации, в основном применяемые для отображения иерархической информации пользователю. В разд. 6.4 мы рассмотрели модальные контроллеры представления и подробный пример их использования.

Глава 7
Специализированные представления

В этой главе мы рассмотрим несколько важных подклассов класса `UIView`. В разд. 7.1 мы обсудим представления подбора значений и их использование для выбора элементов. В разд. 7.2 будут описаны представления индикаторов выполнения и индикаторов деятельности, а в разд. 7.3 — текстовые представления для отображения многострочного текста. Из разд. 7.4 вы узнаете, как использовать представления предупреждений для отображения пользователю предупредительных сообщений. В разд. 7.5 будут рассмотрены списки действий, а в разд. 7.6 — некоторые аспекты веб-представлений.

7.1. Представления подбора значений

Класс `UIPickerView` можно использовать в качестве оригинального способа выбора элемента из набора значений. Этот класс позволяет оперировать несколькими множествами значений, каждое из которых представлено в виде колеса. Пользователь крутит колесо, чтобы выбрать определенное значение из множества. Значениями данного множества могут быть представления (такие как экземпляры `UILabel`, `UIImageView` и т. д.) или строки.

Каждое колесо является графическим представлением *компонента*. Компоненты нумеруются начиная с 0. Каждый компонент содержит *ряды* (например, множество значений). Ряды также нумеруются начиная с 0. Вы можете динамически изменять содержимое любого компонента. Чтобы изменить содержимое определенного компонента, требуется изменить нижележащую модель данных (например, массив), представляющую значения компонента, и вызвать метод `reloadComponent:` класса `UIPickerView`, передавая порядковый номер компонента. Для изменения всех компонентов вызовите метод `reloadAllComponents`.

7.1.1. Делегат

Чтобы использовать экземпляр `UIPickerView`, для него нужно настроить делегат. Этот делегат должен заимствовать протокол `UIPickerViewDelegate` и реализовывать специфические методы, определенные в этом протоколе.

Для создания содержимого представления экземпляру `UIPickerView` требуются значения для каждой пары «компонент-ряд». Делегат должен

реализовать метод, который возвращает экземпляр NSString либо экземпляр UIView. Метод, возвращающий экземпляр NSString, объявлен следующим образом:

```
- (NSString *)pickerView:(UIPickerView *)pickerView
    titleForRow:(NSInteger)row forComponent:(NSInteger)component
```

Здесь pickerView — это представление подбора значений, запрашивающее значение заголовка, а row — значение ряда внутри component.

Метод, который возвращает экземпляр UIView, объявлен так:

```
- (UIView *)pickerView:(UIPickerView *)pickerView
    viewForRow:(NSInteger)row  forComponent:(NSInteger)component
    reusingView:(UIView *)view
```

Здесь view — это ранее использовавшееся представление для пары component-row. Если его можно использовать заново, можете просто вернуть это значение. В противном случае верните новый экземпляр представления.

Делегат должен реализовать два следующих метода.

- Первый предоставляет количество компонентов (колес), которые должен создать экземпляр представления подбора значений:

```
- (NSInteger) numberOfComponentsInPickerView:(UIPickerView *)pickerView
```

- Второй дает количество рядов, которые содержит component:

```
- (NSInteger)pickerView:(UIPickerView *)pickerView
    numberOfRowsInComponent:(NSInteger)component
```

В дополнение к вышеуказанным обязательным методам по желанию вы можете реализовать следующие методы.

- pickerView: rowHeightForComponent: используется для информирования представления о высоте (в точках) данного компонента. Метод объявлен как

```
- (CGFloat)pickerView:(UIPickerView *)pickerView
    rowHeightForComponent:(NSInteger)component
```

- pickerView: widthForComponent: применяется, чтобы сообщить представлению длину (в точках) данного компонента. Объявлен следующим образом:

```
- (CGFloat)pickerView:(UIPickerView *)pickerView
    widthForComponent:(NSInteger)component
```

- pickerView:didSelectRow:inComponent: используется представлением для подбора значений для информирования делегата, что в заданном компоненте был выбран данный ряд. Метод вызывается однажды, когда колесо устанавливается на указанном ряде. Метод объявлен как

```
- (void)pickerView:(UIPickerView *)pickerView
    didSelectRow:(NSInteger)row inComponent:(NSInteger)component
```

7.1.2. Пример

Рассмотрим концепции класса `UIPickerView` на примере. Создадим представление, в котором пользователь выбирает название улицы и направление по ней. Пример использует два класса — делегат приложения `PVAppDelegate` и контроллер представления `PVViewController`.

Листинги 7.1 и 7.2 показывают, соответственно, объявление и определение класса делегата приложения.

Листинг 7.1. Файл PVAppDelegate.h, объявляющий делегат приложения

```
#import <UIKit/UIKit.h>
#import <Foundation/Foundation.h>
@class PVViewController;
@interface PVAppDelegate : NSObject {
    UIWindow  *window;
    PVViewController  *viewController;
}
@end
```

Делегат приложения создает контроллер представления и инициализирует его с помощью массива названий улиц.

Листинг 7.2. Файл PVAppDelegate.m, релизующий делегат приложения

```
#import "PVAppDelegate.h"
#import "PVViewController.h"
@implementation PVAppDelegate
- (void)applicationDidFinishLaunching:(UIApplication *)application {
    window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    NSArray *arr = [NSArray arrayWithObjects:
            @"Plano PKWY", @"Coit Road",
            @"Preston Road", @"Legacy",
            @"Independence", nil];
    viewController = [[PVViewController alloc] initWithStreetNames:arr];
    [window addSubview:viewController.view];
    [window makeKeyAndVisible];
}
- (void)dealloc {
    [window release];
    [viewController release];
    [super dealloc];
}
@end
```

Листинги 7.3 и 7.4 демонстрируют объявление и определение контроллера представления соответственно.

Листинг 7.3. Файл PVViewController.h, объявляющий контроллер представления

```
#import <UIKit/UIKit.h>
#import <Foundation/Foundation.h>
@interface PVViewController:UIViewController<UIPickerViewDelegate> {
    UIPickerView  *pickerView;
    NSArray  *streetNames;
    NSMutableArray  *directions;
}
-(id) initWithStreetNames:(NSArray*) streets;
@end
```

Листинг 7.4 Файл PVViewController.m, определяющий контроллер представления

```
#import "PVViewController.h"
@implementation PVViewController
-(id)initWithStreetNames:(NSArray*) streets {
  if (self = [super init]) {
    directions = [[NSMutableArray arrayWithObjects:@"East", @"West", nil] retain];
          streetNames = [streets copy];
  }
  return self;
}
-(id)init {
  return [self initWithStreetNames:
    [NSArray arrayWithObjects:@"Street Name", nil]];
}
-(void)loadView {
    //Создает основное представление
    UIView *theView = [[UIView alloc]
          initWithFrame:[UIScreen mainScreen].applicationFrame];
    theView.autoresizingMask =
          UIViewAutoresizingFlexibleHeight |
          UIViewAutoresizingFlexibleWidth;
    // Создает представление подбора значений
    pickerView = [[UIPickerView alloc] initWithFrame:CGRectZero];
    CGSize pickerSize = [pickerView sizeThatFits:CGSizeZero];
    pickerView.frame= CGRectMake(0,0, pickerSize.width, pickerSize.height);
    pickerView.autoresizingMask = UIViewAutoresizingFlexibleWidth;
    pickerView.delegate = self;
    pickerView.showsSelectionIndicator = YES;
    [theView addSubview:pickerView];
    self.view = theView;
}
-(void)dealloc {
    [pickerView release];
    [directions release];
    [streetNames release];
    [super dealloc];
}
// Методы делегата
-(void)pickerView:(UIPickerView *)pickerView
    didSelectRow:(NSInteger)row inComponent:(NSInteger)component {
    NSString *street, *direction;
    street =
          [streetNames objectAtIndex:[pickerView selectedRowInComponent:0]];
    direction =
          [directions objectAtIndex:[pickerView selectedRowInComponent:1]];
    if(component == 0) {
          if(([[street isEqual:@"Coit Road"] == YES) ||
          ([street isEqual:@"Preston Road"] == YES) ||
          ([street isEqual:@"Independence"] == YES)) {
                [directions removeAllObjects];
                [directions addObject:@"North"];
                [directions addObject:@"South"];
                [pickerView reloadComponent:1];
          }
          else {
                [directions removeAllObjects];
                [directions addObject:@"East"];
                [directions addObject:@"West"];
                [pickerView reloadComponent:1];
          }
          printf("Selected row in Component 0 is now %s.
```

```
            Selected row in Component 1 remains %s\n",
            [street cStringUsingEncoding:NSUTF8StringEncoding],
            [direction cStringUsingEncoding:NSUTF8StringEncoding]);
    }
    else {
            printf("Selected row in Component 1 is now %s.
            Selected row in Component 0 remains %s\n",
            [direction cStringUsingEncoding:NSUTF8StringEncoding],
            [street cStringUsingEncoding:NSUTF8StringEncoding]);
    }
}
-(NSString *)pickerView:(UIPickerView *)pickerView
    titleForRow:(NSInteger)row forComponent:(NSInteger)component {
    if (component == 0) {
            return [streetNames objectAtIndex:row];
    }
    else {
            return [directions objectAtIndex:row];
    }
}
-(CGFloat)pickerView:(UIPickerView *)pickerView
    widthForComponent:(NSInteger)component {
    if (component == 0)
            return 200.0;
    else
            return 100.0;
}
-(CGFloat)pickerView:(UIPickerView *)pickerView
    rowHeightForComponent:(NSInteger)component {
            return 40.0;
}
-(NSInteger)pickerView:(UIPickerView *)pickerView
    numberOfRowsInComponent:(NSInteger)component {
    if (component == 0) {
            return [streetNames count];
    }
    else {
            return [directions count];
    }
}
- (NSInteger)numberOfComponentsInPickerView:(UIPickerView *)pickerView {
    return 2;
}
@end
```

Контроллер использует два массива в качестве нижележащей модели данных для представления подбора значений. Массив `streetNames` содержит названия улиц. Изменяемый массив `directions` является динамическим и содержит направления, доступные на текущей выбранной улице. Инициализация модели данных производится в методе `initWithStreetNames:`.

Представление подбора значений создается в методе контроллера `loadView`. Объект `UIPickerView` — специфический: он самостоятельно рассчитывает свой оптимальный размер, поэтому вы должны использовать в инициализации `CGRectZero`. Чтобы подогнать представление с помощью внутреннего рассчитанного фрейма, используйте метод `sizeThatFits:` экземпляра `UIPickerView` и передайте ему `CGRectZero`.

Теперь, когда у вас есть оптимальная ширина и высота представления, нужно обновить фрейм представления подбора значений. Чтобы завершить настройку, установите свойство `showSelectionIndicator` в `YES`, сделайте контроллер делегатом и добавьте представление подбора значений к главному представлению.

Второй компонент, направления улиц, является функцией выбранной улицы, поэтому при изменении текущей выбранной улицы мы должны изменить нижележащую модель данных. Метод `pickerView:didSelectRow:inComponent:` является методом делегата, который вызывается при изменении выбранного ряда компонента. Это хорошее место для логики, изменяющей модель данных компонента направлений улиц.

Метод начинается с получения двух выбранных рядов. Чтобы получить выбранное значение компонента, используйте метод `selectedRowInComponent:`. Если выбранный ряд находится в первом компоненте (колесо с названиями улиц), метод определяет направления улиц и обновляет второе колесо. Для обновления компонента вы должны обновить модель данных (массив `directions`) и вызвать метод представления подбора значений `reloadComponent:`, передавая номер компонента.

Ниже изображено главное окно приложения (рис. 7.1).

Рис. 7.1. Представление подбора значений, позволяющее пользователю выбирать название улицы и направление движения

7.2. Представления индикаторов деятельности

Представления индикаторов деятельности (рис. 7.2) являются визуальными индикаторами, информирующими пользователя о выполнении какой-либо задачи. Если вы можете определить процентное соотношение выполнения задачи, используйте экземпляр `UIProgressView`. Если вы не можете определить, как долго будет выполняться задача, больше подходит экземпляр `UIActivityIndicatorView`.

Рис. 7.2. Представление визуальной информации пользователю в зависимости от степени завершенности данной задачи

У экземпляра `UIProgressView` есть только один главный атрибут – `progress`. Объявление свойства `progress` следующее:

```
@property(nonatomic) float progress
```

Значение, которое принимает свойство `progress`, изменяется от `0.0` до `1.0`. В любой момент при изменении значения `progress` представление обновляется само.

Чтобы значок на экземпляре UIActivityindicatorView начал вращаться, вызовите метод startAnimating. Для его остановки пошлите сообщение stopAnimating. Если булев атрибут hidesWhenStopped равен YES (по умолчанию), представление индикатора деятельности скрывается при получении сообщения stopAnimating. Существует несколько значений UIActivityIndicatorStyle, которые вы можете использовать для установки значения свойства activityIndicatorViewStyle — UIActivityIndicatorViewStyleGray, UIActivityIndicatorView StyleWhite и UIActivityIndicatorViewStyleWhiteLarge.

Пример. Рассмотрим концепции, касающиеся двух классов, на примере. В нем используются два класса — делегат приложения PVAppDelegate и контроллер представления PVViewController.

Листинги 7.5 и 7.6 демонстрируют объявление и определение класса делегата приложения соответственно.

Листинг 7.5. Файл PVAppDelegate.h, объявляющий делегат приложения

```
#import <UIKit/UIKit.h>
#import <Foundation/Foundation.h>
@class PVViewController;
@interface PVAppDelegate : NSObject {
    UIWindow  *window;
    PVViewController  *viewController;
}
@end
```

Листинг 7.6. Файл PVAppDelegate.m, определяющий делегат приложения

```
#import "PVAppDelegate.h"
#import "PVViewController.h"
@implementation PVAppDelegate
-(void)applicationDidFinishLaunching:(UIApplication *)application {
    window = [[UIWindow alloc] initWithFrame:[[UIScreen  mainScreen] bounds]];
    viewController = [[PVViewController alloc] init];
    [window addSubview:viewController.view];
    [window makeKeyAndVisible];
}
-(void)dealloc {
    [viewController release];
    [super dealloc];
}
@end
```

Листинги 7.7 и 7.8 показывают объявление и определение класса контроллера представления. Главное представление,theView содержит два дочерних — экземпляры UIProgressView и UIActivityIndicatorView.

Сначала мы создаем панель прогресса и ставим начальный прогресс в 0. Затем создается индикатор деятельности и начинается анимация. Далее в целях демонстрации мы создаем экземпляр NSTimer. Таймер устанавливается на вызов нашего метода updateProgress: каждую секунду.

Внутри метода updateProgress: мы обновляем значение атрибута progress, таким образом продвигая индикатор панели прогресса на 1/10. Если задача закончилась (то есть прошло 10 секунд), мы посылаем stopAnimating индикатору деятельности и останавливаем таймер.

Листинг 7.7. Файл PVViewController.h, объявляющий контроллер представления

```
#import <UIKit/UIKit.h>
#import <Foundation/Foundation.h>
@interface PVViewController : UIViewController {
    UIProgressView *progressBar;
    UIActivityIndicatorView *activityIndicator;
}
@end
```

Листинг 7.8. Файл PVViewController.m, определяющий контроллер представления

```
#import "PVViewController.h"
@implementation PVViewController
- (void)loadView {
    //Создаем главное представление
    UIView *theView = [[UIView alloc] initWithFrame:
            [UIScreen mainScreen].applicationFrame];
    theView.autoresizingMask = UIViewAutoresizingFlexibleHeight |
            UIViewAutoresizingFlexibleWidth;
    [theView setBackgroundColor:[UIColor grayColor]];
    //Создаем панель прогресса
    CGRect frame = CGRectMake(50.0, 100.0, 200, 40);
    progressBar = [[UIProgressView alloc] initWithFrame:frame];
    progressBar.progressViewStyle = UIProgressViewStyleDefault;
    progressBar.progress = 0.0;
    [theView addSubview:progressBar];
    //Создаем индикатор деятельности
    frame = CGRectMake(150.0, 200.0, 40, 40);
    activityIndicator =
            [[UIActivityIndicatorView alloc] initWithFrame:frame];
    activityIndicator.activityIndicatorViewStyle =
            UIActivityIndicatorViewStyleWhite;
    [activityIndicator startAnimating];
    [theView addSubview:activityIndicator];
    //Создаем таймер для демонстрационных целей
    [NSTimer scheduledTimerWithTimeInterval:1
            target:self
            selector:@selector(updateProgress:) userInfo:nil repeats:YES];
    self.view = theView;
}
- (void)updateProgress:(NSTimer*)theTimer {
    progressBar.progress += 0.1;
    if(progressBar.progress >= 1.0) {
            [theTimer invalidate];
            [activityIndicator stopAnimating];
    }
}
- (void)dealloc {
    [progressBar release];
    [activityIndicator release];
    [super dealloc];
}
@end
```

7.3. Текстовое представление

Класс UITextView является подклассом UIScrollView (который расширяет UIView). Вы можете использовать этот класс для отображения многострочного текста с заданными цветом, шрифтом и выравниванием.

Основные свойства данного класса следующие.

- text — его значение — текст, отображаемый внутри представления. Свойство объявлено как

```
@property(nonatomic, copy) NSString *text
```

- font — представляет шрифт текста. Обратите внимание: для всего текста можно выбрать только один шрифт. Объявлено следующим образом:

```
@property(nonatomic, retain) UIFont *font
```

- textColor — представляет цвет текста. Как и в случае со шрифтом, можно использовать только один цвет. Свойство объявлено как

```
@property(nonatomic, retain) UIColor *textColor
```

- textAlignment — его значение задает выравнивание текста в представлении. Объявлено следующим образом:

```
@property(nonatomic) UITextAlignment textAlignment
```

Выравнивание может быть по левому краю (UITextAlignmentLeft), по правому (UITextAlignmentRight) либо по центру (UITextAlignmentCenter). Значением по умолчанию является выравнивание по левому краю.

- editable — его значение определяет, является ли текст редактируемым. Свойство объявлено как

```
@property(nonatomic, getter=isEditable) BOOL editable
```

7.3.1. Делегат

Делегат этого класса должен заимствовать протокол UITextViewDelegate. Все методы этого протокола являются опциональными. Делегат должен получать сообщения, связанные с редактированием текста. Сообщения могут быть следующими:

- textViewShouldBeginEditing: — делегат опрашивается, должно ли текстовое представление начать редактирование. Это включается пользователем, касающимся области текста. Вы возвращаете YES, если хотите начать редактирование текста, либо NO в противном случае;

- `textViewDidBeginEditing:` — это сообщение получается сразу после начала редактирования, но перед фактическим изменением текста;
- `textViewShouldEndEditing:` — такое сообщение получается после того, как текстовое представление теряет фокус. Вы возвращаете YES, если хотите закончить редактирование текста, или NO в противном случае;
- `textViewDidEndEditing:` — данное сообщение информирует делегата, что текстовое представление закончило редактирование.

7.3.2. Пример

Для лучшего понимания концепций класса `UITextView` рассмотрим пример. Мы создадим область многострочного текста и позволим пользователю редактировать его. Пользователь сигнализирует об окончании редактирования нажатием кнопки Done (Готово).

Пример использует два класса — делегат приложения, показанный в листингах 7.9 и 7.10, и контроллер представления, показанный в листингах 7.11 и 7.12.

Листинг 7.9. Файл TVAppDelegate.h, объявляющий делегат приложения

```
#import <UIKit/UIKit.h>
#import <Foundation/Foundation.h>
@class TVViewController;
@interface TVAppDelegate : NSObject {
    UIWindow  *window;
    TVViewController  *viewController;
}
@end
```

Листинг 7.10. Файл TVAppDelegate.m, определяющий делегат приложения

```
#import "TVAppDelegate.h"
#import "TVViewController.h"
@implementation TVAppDelegate
- (void)applicationDidFinishLaunching:(UIApplication *)application {
    window =
            [[UIWindow alloc] initWithFrame: [[UIScreen mainScreen] bounds]];
    viewController = [[TVViewController alloc] init];
    [window addSubview:viewController.view];
    // Сделаем окно ключевым и видимым
    [window makeKeyAndVisible];
}
-(void)dealloc {
    [window release];
    [viewController release];
    [super dealloc];
}
@end
```

Внутри метода `loadView` контроллера представления мы создаем экземпляр `UITextView`, инициализируем его с размерами 320, 200 и позиционируем на 0, 50. Затем настраиваем цвет текста, шрифт и цвет фона текстовой области.

Теперь можно добавить к представлению начальный текст. Обратите внимание: `\n` используется для перехода на новую строку. Тип клавиши Return (Возврат) — `UIReturnkewDefault`. Тип клавиатуры связан со значением `UIKeyboardTypeDefault`.

После создания области текстового представления и добавления его к основному представлению `theView` метод продолжается созданием и настройкой кнопки Done (Готово). При нажатии эта кнопка перемещает фокус с экземпляра текстового представления, таким образом скрывая клавиатуру.

Листинг 7.11. Файл TVViewController.h, объявляющий контроллер представления

```
#import <UIKit/UIKit.h>
#import <Foundation/Foundation.h>
@interface TVViewController : UIViewController <UITextViewDelegate>{
    UITextView  *textView;
    UIButton  *doneButton;
}
@end
```

Листинг 7.12. Файл TVViewController.m, определяющий контроллер представления

```
#import "TVViewController.h"
@implementation TVViewController
-(void)loadView {
    //Создаем основное представление
    UIView *theView =
            [[UIView alloc] initWithFrame:[UIScreen mainScreen].applicationFrame];
    theView.autoresizingMask = UIViewAutoresizingFlexibleHeight |
            UIViewAutoresizingFlexibleWidth;
    CGRect frame = CGRectMake(0.0, 50.0, 320, 200.0);
    textView = [[UITextView alloc] initWithFrame:frame];
    textView.textColor = [UIColor blackColor];
    textView.font = [UIFont fontWithName:@"Arial" size:20];
    textView.delegate = self;
    textView.backgroundColor = [UIColor whiteColor];
    textView.text = @"Dear Sir/Madam, \n I would like ";
    textView.returnKeyType = UIReturnKeyDefault;
    textView.keyboardType = UIKeyboardTypeDefault;
    [theView addSubview: textView];
    doneButton =
            [[UIButton buttonWithType:UIButtonTypeRoundedRect] retain];
    doneButton.frame = CGRectMake(210.0, 5.0, 100, 40);
    [doneButton setTitle:@"Done" forState:UIControlStateNormal];
    [doneButton addTarget:self action:@selector(doneAction:)
            forControlEvents:UIControlEventTouchUpInside];
    doneButton.enabled = NO;
    [theView addSubview: doneButton];
    self.view = theView;
}
- (void)doneAction:(id)sender{
    [textView resignFirstResponder];
    doneButton.enabled = NO;
     // Методы делегата UITextView
-(BOOL)textViewShouldBeginEditing:(UITextView *)textView {
```

```
    printf("textViewShouldBeginEditing\n");
    return YES;
}
-(void)textViewDidBeginEditing:(UITextView *)textView {
    printf("textViewDidBeginEditing\n");
    doneButton.enabled = YES;
}
-(BOOL)textViewShouldEndEditing:(UITextView *)textView {
    printf("textViewShouldEndEditing\n");
    return YES;
}
-(void)textViewDidEndEditing:(UITextView *)textView {
    printf("textViewDidEndEditing\n");
}
-(void)dealloc {
    [textView release];
    [doneButton release];
    [super dealloc];
}
@end
```

Как уже было сказано, все методы делегата опциональны. Из них мы реализовали четыре. Когда начинается редактирование текстового представления, вызывается метод `textViewDidBeginEditing:`. В нем мы создаем кнопку Done (Готово), таким образом разрешая пользователю выходить из режима редактирования по завершении изменений. Три остальных метода приведены в демонстрационных целях и реализуют поведение по умолчанию. Ниже приведен снимок экрана приложения (рис. 7.3).

Рис. 7.3. Редактирование многострочного текста

7.4. Представление предупреждения

Представления предупреждения используются для отображения пользователю предупредительных сообщений. Такое уведомление включает заголовок, короткий текст и одну или более кнопок. Класс, используемый для представления предупреждения, — `UIAlertView`, который является подклассом `UIView`. Для инициализации представления предупреждения используйте удобный инициализатор `initWithTitle:message:delegate:cancelButtonTitle:otherButtonTitles:`, объявленный как

```
-(id)initWithTitle:(NSString *)title message:(NSString *)message
    delegate:(id<UIAlertViewDelegate>)delegate
    cancelButtonTitle:(NSString *)cancelButtonTitle
    otherButtonTitles:(NSString *)otherButtonTitles, ...
```

Здесь `title` — это строка, используемая в качестве заголовка представления предупреждения, `message` — краткий текст, более детально информирующий о назначении представления предупреждения, а `delegate` — это объект, заимствующий `UIAlertViewDelegate`, который будет получать сообщения от экземпляра `UIAlertView`. Свойство `cancelButtonTitle` — это заголовок кнопки отмены, предназначенной для закрытия представления предупреждения. Вам потребуется, по крайней мере, одна кнопка, чтобы пользователь мог закрыть представление. Вы можете добавить одну или несколько кнопок в `otherButtonTitles`, перечисляя их заголовки в терминирующем `nil` списке, разделенном запятой. После инициализации экземпляра представления предупреждения вы посылаете ему сообщение `show`, чтобы представление появилось на экране. Следующий код демонстрирует основные шаги, описанные выше:

```
UIAlertView *alert = [[UIAlertView alloc]
    initWithTitle:@"Host Unreachable"
    message:@"Please check the host address"
    delegate:self cancelButtonTitle:@"OK" otherButtonTitles: nil];
[alert show];
```

Свойство `UIAlertViewDelegate` имеет несколько объявленных методов. В основном вам понадобится метод `alertView:clickedButtonAtIndex:`. Он будет вызываться представлением предупреждения, информируя делегата о номере кнопки, используемой для скрытия представления. Метод объявлен как

```
-(void)alertView:(UIAlertView *)alertView
    clickedButtonAtIndex:(NSInteger)buttonIndex
```

Здесь `alertView` — это экземпляр вызвавшего представления, а `buttonIndex` — номер кнопки, использовавшейся для закрытия представления. Номер первой кнопки — 0. Наша реализация для этого примера просто выводит номер кнопки, нажатой для закрытия представления, как показано далее:

```
-(void)alertView:(UIAlertView *)alertView
    clickedButtonAtIndex:(NSInteger)buttonIndex {
    NSLog(@"The index of the alert button clicked is %d", buttonIndex);
    }
```

В нашем примере выводимый номер всегда будет 0, так как у нас только одна кнопка. Ниже приведен пример представления предупреждения (рис. 7.4).

Рис. 7.4. Представление предупреждения с одной кнопкой

Как уже было сказано, в представлении может быть больше одной кнопки. Следующее выражение добавляет к кнопке отмены еще две кнопки.

```
UIAlertView *alert = [[UIAlertView alloc]
                      initWithTitle:@"Disk Error"
                      message:@"Error reading sector 18"
                      delegate:self cancelButtonTitle:@"Abort"
                      otherButtonTitles:@"Retry", @"Fail", nil];
```

Номер кнопки **Abort** (Завершить) остается равным 0. Номера кнопок **Retry** (Повторить) и **Fail** (Неудача) – соответственно 1 и 2. Ниже изображено представление предупреждения с тремя кнопками (рис. 7.5).

Делегат определяет метод `alertViewCancel:`. Он вызовется, когда пользователь нажмет кнопку **Home** (Домашняя страница) на своем iPhone. Если делегат не реализует этот метод, будет симулироваться нажатие кнопки **Cancel** (Отмена) и представление предупреждения закроется.

Рис. 7.5. Представление предупреждения с тремя кнопками

7.5. Списки действий

Списки действий схожи с представлениями предупреждения в том, что они также выводят пользователю сообщение и одну либо несколько кнопок. Тем не менее списки действий отличаются внешним видом и способом представления. Чтобы отобразить список действий, создайте новый экземпляр класса `UIActionSheet` и инициализируйте его, используя инициализатор `initWithTitle:delegate:cancelButtonTitle:destructiveButtonTitle:otherButtonTitles:`, объявленный как

```
- (id)initWithTitle:(NSString *)title
    delegate:(id <UIActionSheetDelegate>)delegate
    cancelButtonTitle:(NSString *)cancelButtonTitle
    destructiveButtonTitle:(NSString )destructiveButtonTitle
    otherButtonTitles:(NSString *)otherButtonTitles, ...
```

Здесь `title` используется для установки заголовка списка действий. Делегат определяется с помощью параметра `delegate`. Заголовок кнопки отмены задается параметром `cancelButtonTitle`. Заголовок кнопки закрытия (отмеченной красным цветом) устанавливается с помощью `destructiveButtonTitle`. Дополнительные кнопки задаются терминирующим `nil` списком, разделенные запятыми, в параметре `otherButtonTitles`. После создания и инициализации списка действий вы выводите его на экран, используя метод `showInView:` и пе-

редавая ему экземпляр родительского представления. Простой список действий выводится на экран следующим образом:

```
UIActionSheet *
actionSheet = [[UIActionSheet alloc]
    initWithTitle:@"Are you sure you want to erase all data?"
    delegate:self cancelButtonTitle:@"Cancel"
    destructiveButtonTitle:@"ERASE" otherButtonTitles:nil];
[actionSheet showInView:self.view];
```

Ниже изображен список действий, созданный с помощью приведенного выше кода (рис. 7.6).

Делегат `UIActionSheetDelegate` определяет несколько опциональных методов. Если вы хотите узнать, какая кнопка была нажата пользователем, вы должны реализовать метод `actionSheet:clickedButtonAtIndex:`. Кнопки нумеруются начиная с 0. В приведенном выше примере кнопка **Cancel** (Отмена) имеет индекс 0, а кнопка **ERASE** (Стереть) — индекс 1.

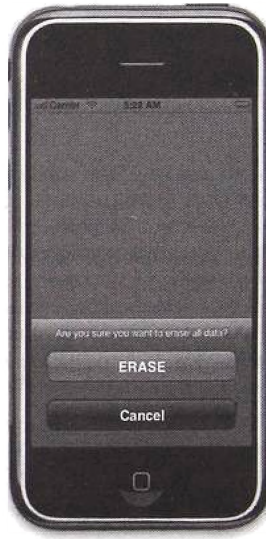

Рис. 7.6. Список действий с двумя кнопками

7.6. Веб-представления

В этом разделе мы рассмотрим класс `UIWebView`. Он является подклассом `UIView`, позволяющим представлять богатое внутреннее наполнение. Начнем с простого приложения с веб-представлением.

7.6.1. Простое приложение с веб-представлением

Рассмотрим простое приложение, использующее веб-представление. Приложение представляет массив персональных записей в форме HTML-таблицы.

Листинг 7.13 демонстрирует объявление класса делегата приложения. Класс хранит ссылку на записи, которые будут представлены в форме таблицы, в экземпляре `NSArray records`. Как вы увидите далее, метод `allRecordsInHTML` используется делегатом приложения для создания HTML-представления персональных записей, находящихся в массиве.

Листинг 7.13. Объявление класса делегата приложения, используемого в простейшем приложении с веб-представлением

```
#import <UIKit/UIKit.h>
@interface DWebViewAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    NSArray  *records;
}
@property (nonatomic, retain) UIWindow *window;
-(NSString*) allRecordsInHTML;
@end
```

Листинг 7.15 показывает реализацию класса делегата приложения. Метод `applicationDidFinishLaunching:` строит модель данных с помощью вызова метода `loadData`. В демонстрационных целях два экземпляра `Person` инициализируются и добавляются в массив `records`. После этого `applicationDidFinishLaunching:` создает и инициализирует экземпляр `UIWebView` так же, как и остальные подклассы `UIView`. Чтобы содержимое занимало весь экран и пользователь мог приближать/удалять его, свойство `scalesPageToFit` должно быть выставлено в `YES`.

Чтобы загрузить HTML-содержимое веб-представления, приложение использует метод `loadHTMLString:baseURL:`. Он объявлен следующим образом:

```
- (void)loadHTMLString:(NSString *)string baseURL:(NSURL *)baseURL
```

Первый параметр — HTML-код для отображения, второй — базовый URL для содержимого. Наше приложение конструирует HTML-содержимое (как вы скоро увидите) и использует `nil` в качестве базового URL. HTML-содержимое состоит из статической и динамической частей. Статическая состоит из HTML-тегов и форматирования, необходимого для отображения HTML-таблицы. Динамическая часть создается путем конкатенации вывода каждой персональной записи, как вы увидите далее. Листинг 7.14 показывает HTML-страницу, окончательно загруженную и визуализированную веб-приложением.

Чтобы текст страницы был приемлемого размера, вы должны использовать мета-тег viewport и задать ширину страницы содержимого 320:

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

Статическая часть HTML также включает тег таблицы и первой строки, задающей заголовки столбцов.

Листинг 7.14 HTML-страница, загруженная и визуализированная веб-представлением

```
<html>
    <meta name="viewport" content="width=320"/>
    <body>
            <h4> Records Found: </h4>
            <table border="6">
            <caption>Database</caption>
            <tr>
                    <td>Name</td>
                    <td>Address</td>
                    <td>Phone</td>
            </tr>
            <tr>
                    <td>John Doe</td>
                    <td>1234 Fake st</td>
                    <td>(555) 555-1234</td>
            </tr>
            <tr>
                    <td>Jane Doe</td>
                    <td>4321 Fake st</td>
                    <td>(555) 555-7898</td>
            </tr>
            </table>
    </body>
</html>
```

Листинг 7.15. Реализация класса делегата приложения, используемого в простейшем веб-представлении

```
#import "DWebViewAppDelegate.h"
#import "Person.h"
@implementation DWebViewAppDelegate
@synthesize window;
-(void)loadData {
    Person *a, *b;
    a = [[Person alloc] autorelease];
    a.name = @"John Doe";
    a.address = @"1234 Fake st";
    a.phone = @"(555) 555-1234";
    b = [[Person alloc] autorelease];
    b.name = @"Jane Doe";
    b.address = @"4321 Fake st";
    b.phone = @"(555) 555-7898";
    records = [NSArray arrayWithObjects:a, b, nil];
}
- (void)applicationDidFinishLaunching:(UIApplication *)application {
    [self loadData];
    CGRect  rectFrame = [UIScreen mainScreen].applicationFrame;
```

```
    window = [[UIWindow alloc] initWithFrame:rectFrame];
    UIWebView *webView =
            [[UIWebView alloc] initWithFrame:rectFrame];
    webView.scalesPageToFit = YES;
    NSMutableString *html = [[NSMutableString alloc] initWithCapacity:200];
    [html appendString: @"<html>"
            " <meta name=\"viewport\" content=\"width=320\"/>"
            " <body>"
            " <h4>"
            " Records Found:"
            " </h4>"
            " <table border=\"6\">"
            " <caption>Database</caption>"
            " <tr>"
            " <td>Name</td>"
            " <td>Address</td>"
            " <td>Phone</td>"
            " </tr>"];
    [html appendString:[self allRecordsInHTML]];
    [html appendString:
            @"</table>"
            " </body>"
            " </html>"
    ];
    [webView loadHTMLString:html baseURL:nil];
    [window addSubview:webView];
    [webView release];
    [html release];
    [window makeKeyAndVisible];
}
-(NSString*) allRecordsInHTML {
    NSMutableString *myRecords =
            [[[NSMutableString alloc] initWithCapacity:200] autorelease];
    for (Person *p in records) {
            [myRecords appendString:[p html]];
    }
    return myRecords;
}
- (void)dealloc {
    [window release];
    [super dealloc];
}
@end
```

Динамическая часть HTML-страницы производится методом `allRecordsInHTML`. Он проходит все элементы массива `records` и добавляет каждому HTML-содержимое. Элементы в массиве `records` являются экземплярами класса `Person`, показанного в листингах 7.16 и 7.17.

Каждый экземпляр `Person` содержит ссылки на строки `name`, `address` и `phone` каждой персоны. Метод `html` используется для создания HTML-кода для строки в таблице со значениями этих трех строк в качестве столбцов.

Листинг 7.16. Объявление класса Person, используемого в приложении с веб-представлением

```
#import <Foundation/Foundation.h>
@interface Person : NSObject {
    NSString *name, *address, *phone;
```

```
}
@property(nonatomic, assign) NSString  *name;
@property(nonatomic, assign) NSString  *address;
@property(nonatomic, assign) NSString  *phone;
-(NSString*)html;
@end
```

Листинг 7.17. Реализация класса Person, используемого в приложении с веб-представлением

```
#import "Person.h"
@implementation Person
@synthesize name, address, phone;
-(NSString*)html {
    NSMutableString *output =
            [[[NSMutableString alloc] initWithCapacity:50] autorelease];
    [output appendString:@"<tr> <td>"];
    [output appendString:name];
    [output appendString:@"</td> <td>"];
    [output appendString:address];
    [output appendString:@"</td> <td>"];
    [output appendString:phone];
    [output appendString:@"</td> </tr>"];
    return output;
}
@end
```

Ниже приведен снимок экрана простейшего приложения с веб-представлением, которое мы создали в этом разделе (рис. 7.7).

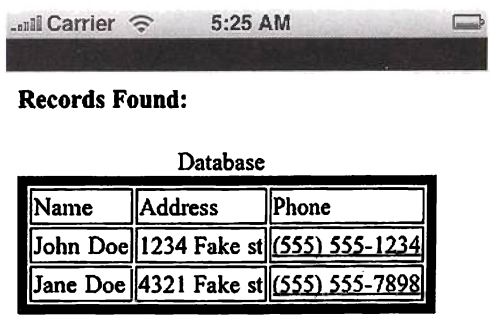

Рис. 7.7. Снимок экрана простейшего приложения с веб-представлением, отображающим данные в форме HTML-таблицы

7.6.2. Просмотр локальных файлов

В этом разделе вы научитесь встраивать изображения, хранящиеся в вашей директории Home (Домашняя страница), в веб-страницу и отображать эту страницу в веб-представлении. Листинги 7.18 и 7.19 показывают,

соответственно, объявление и определение класса делегата приложения. Делегат приложения создает контроллер представления типа `MyWebViewController` и добавляет его представление в качестве дочернего к главному окну.

Листинг 7.18. Объявление класса делегата приложения, используемого для просмотра изображений в веб-представлении

```
#import <UIKit/UIKit.h>
@class MyWebViewController;
@interface BWebViewAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow        *window;
    MyWebViewController *ctrl;
}
@property (nonatomic, retain) UIWindow *window;
@end
```

Листинг 7.19. Реализация класса делегата приложения, используемого для просмотра изображений в веб-представлении

```
#import "BWebViewAppDelegate.h"
#import "MyWebViewController.h"
@implementation BWebViewAppDelegate
@synthesize window;
-(void)applicationDidFinishLaunching:(UIApplication *)application {
    window = [[UIWindow alloc]
              initWithFrame:[[UIScreen mainScreen]  bounds]];
    ctrl = [[MyWebViewController alloc] initWithNibName:nil bundle:nil];
    [window addSubview:ctrl.view];
    [window makeKeyAndVisible];
}
- (void)dealloc {
    [ctrl release];
    [window release];
    [super dealloc];
}
@end
```

Листинг 7.20 демонстрирует объявление контроллера представления, используемого делегатом приложения. Контроллер представления содержит ссылку на экземпляр `UIWebView`. В дополнение метод `produceImageReference: withType:`, как вы скоро увидите, используется для внутренних нужд для генерации HTML-тега `IMG` для конкретного локального файла с изображением.

Листинг 7.20. Объявление контроллера представления, используемого делегатом приложения, показывающего локальные файлы изображений в веб-представлении

```
#import <UIKit/UIKit.h>
@interface MyWebViewController : UIViewController {
    UIWebView  *webView;
}
-(NSString*)produceImageReference:(NSString*) imgFileName
    withType:(NSString*) imgType;
@end
```

Листинг 7.22 показывает реализацию класса контроллера представле ния. Метод `loadView` начинает с создания объектов пользовательског интерфейса (как вы видели в предыдущем разделе). Затем он создает веб страницу, содержащую ссылки на изображения.

Чтобы встроить изображение в веб-страницу, вы можете использоват элемент `IMG`. У него есть два обязательных атрибута — `src`, являющийс URL-адресом, задающим местоположение изображения, и `alt`, определя ющий описание изображения.

В нашем примере изображение хранится в упаковке приложени (рис. 7.8 и гл. 9).

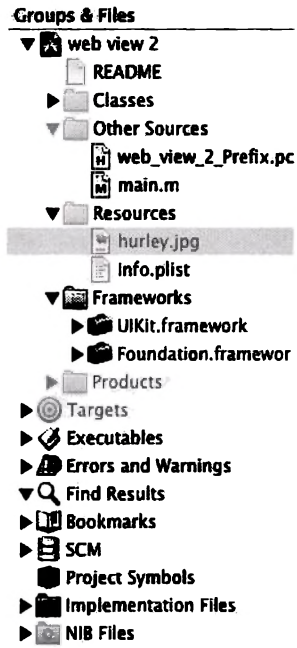

Рис. 7.8. Содержимое Groups & Files (Группы и файлы), показывающее изображение hurley.jpg в группе Resources (Ресурсы)

Для задания URL-адреса локального изображения можете использовать протокол `file` с полным путем к файлу изображения. Например, следующий HTML-код генерируется в устройстве динамически:

```
<IMG SRC="file:///var/mobile/Applications/
    8884F8E2-E466-4500-8FFF-6263C99016DB/web view 2.app/
    hurley.jpg" ALT="hurley"
>
```

Пока вам не нужно знать, как мы получили путь к изображению, — об этом будет рассказано в гл. 9. Изменяемая строка `htmlContents`, последовательно увеличиваясь, конструируется, чтобы содержать полный код веб-страницы. Для простоты мы встроим одно и то же изображение три раза с различными заголовками. Полная веб-страница, которая загружается в устройство, приведена в листинге 7.21.

Листинг 7.21. HTML-страница, загруженная и визуализированная веб-представлением для приложения со страницей, содержащей встроенные локальные изображения

```
<meta name="viewport" content="width=320"/>
<html>
    <body>
        <H2>Hurley</H2>
        <IMG SRC="file:///var/mobile/Applications/
            8884F8E2-E466-4500-8FFF-6263C99016DB/web view 2.app/
            hurley.jpg" ALT="hurley">
        <H1>Hugo</H1>
        <IMG SRC="file:///var/mobile/Applications/
            8884F8E2-E466-4500-8FFF-6263C99016DB/web view 2.app/
            hurley.jpg" ALT="hurley">
        <H3>Jorge Garcia</H3>
        <IMG SRC="file:///var/mobile/Applications/
            8884F8E2-E466-4500-8FFF-6263C99016DB/web view 2.app/
            hurley.jpg" ALT="hurley">
    </body>
</html>
```

Листинг 7.22. Реализация контроллера представления, используемого делегатом приложения, которое отображает локальные файлы изображений в веб-представлении

```
#import "MyWebViewController.h"
@implementation MyWebViewController
-(void)loadView {
    CGRect  rectFrame = [UIScreen mainScreen].applicationFrame;
    UIView  *view = [[UIView alloc] initWithFrame:rectFrame];
    view.autoresizingMask = UIViewAutoresizingFlexibleHeight |
    UIViewAutoresizingFlexibleWidth;
    webView = [[UIWebView alloc] initWithFrame:CGRectMake(0, 0, 320, 460)];
    webView.scalesPageToFit = YES;
    NSMutableString *htmlContents =
            [[NSMutableString alloc] initWithCapacity:100];
    [htmlContents appendString:
            @"<meta name=\"viewport\" content =\"width=320\"/>"
            "<html>"
            "<body>"];
    [htmlContents appendString:@"<H2>Hurley</H2>"];
    [htmlContents appendString:[self produceImageReference:
            @"hurley" withType:@"jpg"]];
    [htmlContents appendString:@"<H1>Hugo</H1>"];
    [htmlContents
            appendString:[self produceImageReference:@"hurley" withType:@"jpg"]];
    [htmlContents appendString:@"<H3>Jorge Garcia</H3>"];
    [htmlContents
```

```
            appendString:[self produceImageReference:@"hurley" withType:@"jpg"]];
    [htmlContents appendString:@
    "</body>"
    "</html>"];
    [webView loadHTMLString:htmlContents baseURL:nil];
    [view addSubview:webView];
    self.view = view;
    [view release];
    [htmlContents release];
}
-(NSString*)produceImageReference:(NSString*) imgFileName
  withType:(NSString*) imgType{
    NSMutableString *returnString =
            [[[NSMutableString alloc] initWithCapacity:100] autorelease];
    NSString *filePath =
            [[NSBundle mainBundle] pathForResource:imgFileName
            ofType:imgType];
    if (filePath){
            [returnString appendString:@"<IMG SRC=\"file://"];
            [returnString appendString:filePath];
            [returnString appendString:@"\" ALT=\""];
            [returnString appendString:imgFileName];
            [returnString appendString:@"\">"];
            return returnString;
    }
    else return @"";
}
- (void)dealloc {
    [webView release];
    [super dealloc];
}
@end
```

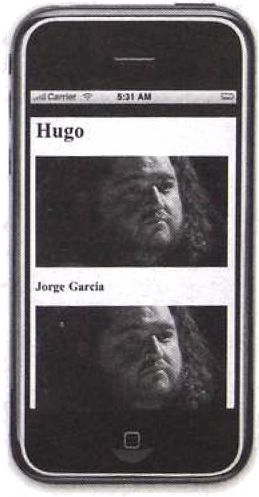

Рис. 7.9. Снимок экрана приложения, демонстрирующего локальное изображение в веб-представлении

7.6.3. Выполнение JavaScript

В этом подразделе мы создадим приложение, отображающее веб-страницу с формой и одним текстовым полем (рис. 7.10).

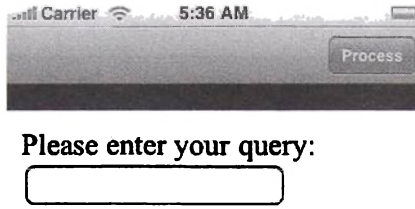

Рис. 7.10. Снимок экрана приложения, демонстрирующего взаимодействие JavaScript и Objective-C: HTML-форма предоставляется пользователю для ввода поискового слова

Пользователь вводит текст и нажимает кнопку Process (Обработка) на панели навигации. Если введен запрещенный текст (позже мы определим значение такого текста), приложение выводит представление предупреждения, информирующее пользователя, что текст нужно ввести заново (рис. 7.11).

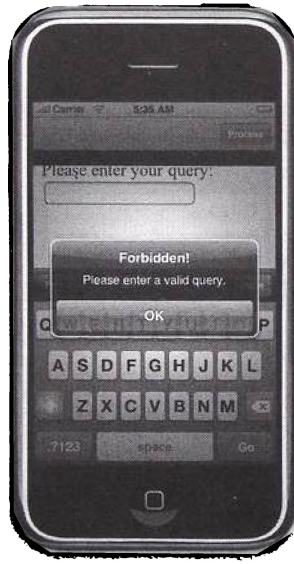

Рис. 7.11. Снимок экрана приложения, демонстрирующего выполнение JavaScript из кода Objective-C; это представление предупреждения при использовании запрещенного запроса

Текстовое поле очищается, и пользователю предоставляется возможность заново ввести текст. Если набранный текст не является запрещенным, приложение получает текстовое значение из поля ввода и обновляет веб-страницу ссылкой на Google-запрос с введенным словом (рис. 7.12).

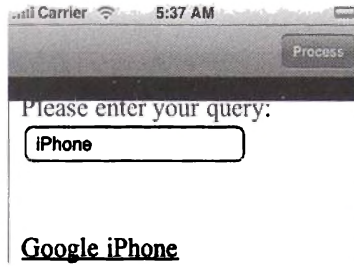

Рис. 7.12. Приложение, реагирующее на разрешенное поисковое слово обновлением страницы в веб-представлении

При нажатии пользователем созданной ссылки отображаются результаты поиска (рис. 7.13). Вот что должно выполнять приложение. Начнем его создание.

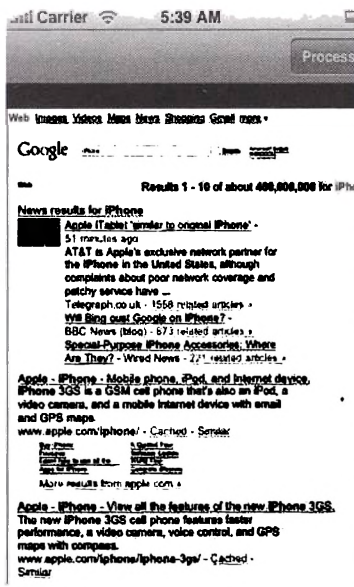

Рис. 7.13. Результат поиска разрешенного термина, просматриваемый в веб-представлении

Листинг 7.23 показывает объявление делегата приложения. Он оперирует контроллерами навигации и представления.

Листинг 7.23. Объявление класса делегата приложения, используемого для демонстрации выполнения JavaScript-кода в веб-представлении

```
#import <UIKit/UIKit.h>
@class MyViewController;
@interface AWebViewDelegate : NSObject <UIApplicationDelegate> {
    UIWindow  *window;
    MyViewController  *ctrl;
    UINavigationControiler  *navCtrl;
}
@property (nonatomic, retain) UIWindow *window;
@end
```

Листинг 7.24 демонстрирует реализацию класса делегата приложения. Метод `applicationDidFinishLaunching:` создает контроллер навигации, добавляет к нему экземпляр `MyViewController` и затем добавляет представление контроллера навигации в качестве дочернего к главному окну.

Листинг 7.24. Реализация класса делегата приложения, используемого для демонстрации выполнения JavaScript-кода в веб-представлении

```
#import "AWebViewDelegate.h"
#import "MyViewController.h"
@implementation AWebViewDelegate
@synthesize window;
- (void)applicationDidFinishLaunching:(UIApplication *)application {
    window = [[UIWindow alloc] initWithFrame:[[UIScreen  mainScreen] bounds]];
    ctrl = [[MyViewController alloc] initWithNibName:nil bundle:nil];
    navCtrl =
            [[UINavigationController alloc] initWithRootViewController:ctrl];
    [window addSubview:navCtrl.view];
    [window makeKeyAndVisible];
}
- (void)dealloc {
    [ctrl release];
    [navCtrl release];
    [window release];
    [super dealloc];
}
@end
```

Листинг 7.26 показывает объявление класса контроллера представления. Контроллер оперирует ссылкой на экземпляр веб-представления и правой кнопкой, реализующей кнопку Process (Обработка) на панели навигации. Листинг 7.27 демонстрирует реализацию этого класса. Метод `initWithNibName:bundle:` инициализирует экземпляр контроллера представления и добавляет кнопку навигации Process (Обработка). Методом действия для этой кнопки является `processJavaScript`, который мы рассмотрим позже.

Метод `loadView` создает и инициализирует экземпляр `UIWebView` и добавляет его к главному окну. Веб-представление загружается со строкой `loadHTMLString`. HTML-код в этой строке является статичным. Он приведен в листинге 7.25. HTML содержит форму с одним текстовым полем. Он также объявляет три следующие функции:

- `getQuery()` : — позволяет получить текстовое значение из поля ввода в форме;
- `clearQuery()` : — очищает содержимое текстового поля;
- `loseFocusOfField()` : — заставляет текстовое поле потерять фокус, после чего исчезает клавиатура.

Листинг 7.25. Статический HTML-код, который задает главную страницу приложения, демонстрирующего выполнение JavaScript в веб-представлении внутри Objective-C-кода

```
<html>
    <head>
            <title>Query Assistant</title>
            <meta name="viewport" content="width=320"/>
            <script>
            function getQuery(){
                    return document.queryform.query.value;
            }
            function clearQuery(){
                    return document.queryform.query.value="";
            }
            function loseFocusOfField(){
                    return document.queryform.query.blur();
            }
            </script>
    </head>
    <body>Please enter your query:
            <form name="queryform">
                    <input name="Query" type="text" value="" id="query" />
                    <br/>
                    <br/>
                    <br/>
                    <a id="anchor" href=""></a>
            </form>
    </body>
</html>
```

Класс `UIWebView` позволяет исполнять код JavaScript по требованию. Чтобы выполнить его, вы должны использовать метод `stringByEvaluatingJavaScriptFromString:`. Он объявлен следующим образом:

```
- (NSString *)stringByEvaluatingJavaScriptFromString:(NSString *)script;
```

Параметр `script` является экземпляром `NSString`, содержащим JavaScript-код, который вы хотите исполнить. Конечное значение выполненного кода JavaScript, если оно существует, возвращается как объект `NSString`.

Когда пользователь нажимает правую кнопку Process (Обработка) на панели навигации, выполняется метод `processJavaScript`. В этом методе мы сначала получаем текстовое значение поля ввода. Выполняется JavaScript-выражение `getQuery()`, и нам возвращается результат. По получении значения мы проверяем его на равность тексту `dissent`. Если это так, мы выводим пользователю представление предупреждения (см. рис. 7.11) и очищаем текстовое поле посредством вызова JavaScript-функции `clearQuery()`.

Если текстовое значение верно, мы обновляем веб-страницу, изменяя атрибут `href` элемента, чей ID равен `anchor`, на поисковый запрос Google. Ниже приведен код, созданный для термина поиска `iPhone`.

```
document.getElementById('anchor').href=
    "http://www.google.com/search?q=iPhone";
```

Мы также обновляем атрибут `innerHTML`, как показано ниже:

```
document.getElementById('anchor').innerHTML="Google iPhone";
```

Затем выполняются обновления веб-страницы. В дополнение мы вызываем JavaScript-функцию `loseFocusOfField()` для скрытия клавиатуры.

Листинг 7.26. Объявление класса контроллера представления, используемого для демонстрации выполнения JavaScript-кода в веб-представлении

```
#import <UIKit/UIKit.h>
@interface MyViewController : UIViewController {
    UIWebView  *webView;
    UIBarButtonItem  *rightButton;
}
@end
```

Листинг 7.27. Реализация класса контроллера представления, используемого для демонстрации выполнения JavaScript-кода в веб-представлении

```
#import "MyViewController.h"
@implementation MyViewController
-(id)initWithNibName:(NSString *)nibNameOrNil
  bundle:(NSBundle *)nibBundleOrNil {
    if (self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil]){
          rightButton = [[UIBarButtonItem alloc]
                initWithTitle:@"Process"
                style:UIBarButtonItemStyleDone
                target:self
                action:@selector(processJavaScript)];
          self.navigationItem.rightBarButtonItem = rightButton;
          [rightButton release];
    }
    return self;
```

```
}
-(void)processJavaScript {
    NSString* var =
            [webView stringByEvaluatingJavaScriptFromString:@"getQuery()"];
    if([var isEqualToString:@"dissent"] == YES){
            UIAlertView *alert = [[UIAlertView alloc]
                    initWithTitle:@"Forbidden!"
                    message:@"Please enter a valid query."
                    delegate:nil
                    cancelButtonTitle:@"OK"
                    otherButtonTitles:nil];
            [alert show];
            [webView stringByEvaluatingJavaScriptFromString:@"clearQuery()"];
            return;
    }
    NSMutableString *query=[[NSMutableString alloc] initWithCapacity:200];
    [query appendString:@"document.getElementById('anchor').href"
            "=\"http://www.google.com/search?q="];
    [query appendString:var];
    [query appendString:@"\";"];
    NSMutableString *innerHTML =
            [[NSMutableString alloc] initWithCapacity:200];
    [innerHTML appendString:
            @"document.getElementById('anchor').innerHTML=\"Google "];
    [innerHTML appendString:var];
    [innerHTML appendString:@"\";"];
    [webView
            stringByEvaluatingJavaScriptFromString:@"loseFocusOfField()"];
    [webView stringByEvaluatingJavaScriptFromString:innerHTML];
    [webView stringByEvaluatingJavaScriptFromString:query];
    rightButton.enabled = NO;
    [query release];
    [innerHTML release];
}
-(void)loadView {
    CGRect  rectFrame = [UIScreen mainScreen].applicationFrame;
    UIView  *view = [[UIView alloc] initWithFrame:rectFrame];
    view.autoresizingMask = UIViewAutoresizingFlexibleHeight |
    UIViewAutoresizingFlexibleWidth;
    webView =
            [[UIWebView alloc] initWithFrame:rectFrame];
    webView.scalesPageToFit = YES;
    [webView loadHTMLString:
            @"<html><head><title>Query Assistant</title>\n"
            "<meta name=\"viewport\" content=\"width=320\"/>"
            "<script>"
            "function getQuery(){"
            "return document.queryform.query.value;}"
            "function clearQuery(){"
            "return document.queryform.query.value=\"\";}"
            "function loseFocusOfField(){"
            "return document.queryform.query.blur();}"
            "</script>"
            "</head><body>Please enter your query: "
            "<form name=\"queryform\">"
```

```
            "<input name=\"Query\" type=\"text\" "
            "value=\"\" id=\"query\" />"
            "<br/>"
            "<br/>"
            "<br/>"
            "<a id=\"anchor\" href=\"\"></a>"
            "</form></body></html>" baseURL:nil];
    [view addSubview:webView];
    self.view = view;
    [view release];

}
-(void)dealloc {
    [webView release];
    [rightButton release];
    [super dealloc];
}
@end
```

7.6.4. Делегат веб-представления

В этом подразделе мы создадим приложение, перехватывающее навигацию пользователя. Если пользователь нажимает ссылку на PDF-файл, его спрашивают, хочет ли он скачать копию для дальнейшего использования. Мы не будем реализовывать реальное управление скачиванием/сохранением (это будет описано в других главах). Из этого подраздела вы узнаете, как перехватить важные изменения экземпляра веб-представления, вызванные взаимодействием с пользователем.

Класс `UIWebView` имеет важное свойство `delegate`, позволяющее перехватывать важные вызовы. Свойство делегата объявлено следующим образом:

```
@property(nonatomic, assign) id<UIWebViewDelegate> delegate
```

Протокол `UIWebViewDelegate` объявляет четыре опциональных метода.

1. `webView:shouldStartLoadWithRequest:navigationType:` — вызывается сразу перед загрузкой содержимого веб-страницы. Метод объявлен следующим образом:

```
- (BOOL)webView:(UIWebView *)webView
shouldStartLoadWithRequest:(NSURLRequest *)request
navigationType:(UIWebViewNavigationType)navigationType
```

Вы возвращаете `YES`, если хотите, чтобы веб-представление загрузило данный запрос, и `NO` в противном случае. Первый параметр — экземпляр веб-представления, второй — экземпляр `NSURLRequest`, представля-

ющий запрос, а третий параметр — тип навигации, приведший к загрузке содержимого.

Класс NSURLRequest определяет метод URL для получения экземпляра класса NSURL. Этот класс определяет absoluteString для получения экземпляра NSString, представляющего URL-адрес запроса. Как вы увидите позже, мы будем использовать эту строку для решения, выполнять какие-либо действия или нет.

Существуют следующие предопределенные значения типов навигации:

- UIWebViewNavigationTypeLinkClicked, показывающий, что пользователь щелкнул на ссылке на странице;
- UIWebViewNavigationTypeFormSubmitted — пользователь отправил форму;
- UIWebViewNavigationTypeBackForward — пользователь нажал кнопки возврата/перехода;
- UIWebViewNavigationTypeReload — пользователь нажал кнопку перегрузки страницы;
- UIWebViewNavigationTypeFormResubmitted — пользователь повторно отправил форму;
- UIWebViewNavigationTypeOther, показывающий какое-либо другое навигационное действие.

Этот метод будет реализован в нашем приложении. Если URL является запросом PDF-файла, мы спросим пользователя, желает ли он загрузить копию файла для дальнейшего использования.

2. webViewDidStartLoad: — используется для извещения, что веб-представление начало загрузку содержимого. Метод объявлен следующим образом:

```
- (void) webViewDidStartLoad:(UIWebView *) webView
```

3. webViewDidFinishLoad: — применяется для сообщения, что веб-представление закончило загрузку содержимого. Объявлен как

```
- (void) webViewDidFinishLoad:(UIWebView *) webView
```

4. webView:didFailLoadWithError: — используется для извещения, что веб-представление встретило ошибку в процессе загрузки содержимого. Метод объявлен следующим образом:

```
- (void) webView:(UIWebView *) webView didFailLoadWithError: (NSError *) error
```

Листинги 7.28 и 7.29 показывают, соответственно, объявление и реализацию класса делегата приложения. Делегат создает экземпляр контроллера

представления MyViewController и добавляет его в качестве дочернего к главному окну.

Листинг 7.28. Объявление класса делегата приложения, демонстрирующего перехват взаимодействия пользователя с веб-представлением

```
#import <UIKit/UIKit.h>
@class MyViewController;
@interface EWebViewAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    MyViewController *ctrl;
}
@property (nonatomic, retain) UIWindow *window;
@end
```

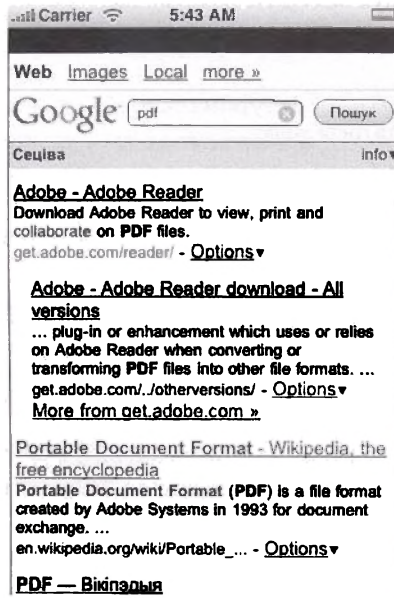

Рис. 7.14. Результат операции поиска в приложении, позволяющем локальное кэширование PDF-файлов

Листинг 7.29. Реализация класса делегата приложения, демонстрирующего перехват взаимодействия пользователя с веб-представлением

```
#import "EWebViewAppDelegate.h"
#import "MyViewController.h"
@implementation EWebViewAppDelegate
@synthesize window;
- (void)applicationDidFinishLaunching:(UIApplication *)application {
    window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    ctrl = [[MyViewController alloc] initWithNibName:nil bundle:nil];
```

```
    [window addSubview:ctrl.view];
    [window makeKeyAndVisible];
}
- (void)dealloc {
    [ctrl release];
    [window release];
    [super dealloc];
}
@end
```

Листинги 7.30 и 7.31 показывают, соответственно, объявление и реализацию класса контроллера представления. Класс оперирует ссылкой на веб-представление, созданное в методе `loadView`. Веб-представление создано с поддержкой масштабирования, а свойство `delegate` установлено равным экземпляру контроллера представления. Веб-представление стартует со страницы поиска Google (рис. 7.15). Контроллер представления реализует только метод `webView:shouldStartLoadWithRequest:navigationType:` протокола `UIWebViewDelegate`.

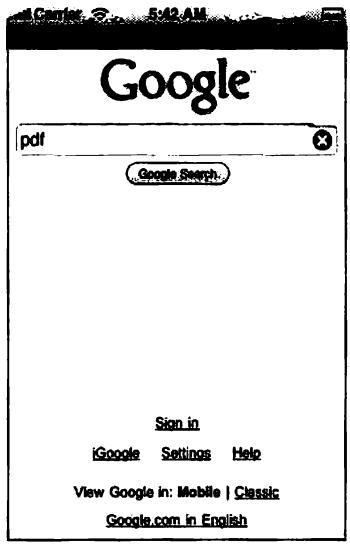

Рис. 7.15. Снимок экрана приложения, позволяющего локальное кэширование PDF-файлов

Метод `webView:shouldStartLoadWithRequest:navigationType:` сначала получает URL-строку запроса. Он проверяет, является ли результат запроса PDF-файлом, используя метод `hasSuffix:` класса `NSString`. Если да, пользователю выводится представление предупреждения с вопросом о загрузке файла в локальную директорию (рис. 7.16). Файл PDF всегда загружается в веб-представление.

Рис. 7.16. Снимок экрана приложения, позволяющего локальное кэширование PDF-файлов (представление предупреждения, спрашивающее пользователя, скачивать ли PDF-файл)

Листинг 7.30. Объявление класса контроллера представления, используемого в приложении, которое демонстрирует перехваты взаимодействия пользователя с веб-представлением

```
#import <UIKit/UIKit.h>
@interface MyViewController : UIViewController<UIWebViewDelegate> {
    UIWebView  *webView;
    NSString  *url;
}
@end
```

Листинг 7.31. Реализация класса контроллера представления, используемого в приложении, которое показывает перехваты взаимодействия пользователя с веб-представлением

```
#import "MyViewController.h"
@implementation MyViewController

- (void)loadView {
    CGRect  rectFrame = [UIScreen mainScreen].applicationFrame;
    UIView  *view = [[UIView alloc] initWithFrame:rectFrame];
    view.autoresizingMask = UIViewAutoresizingFlexibleHeight |
            UIViewAutoresizingFlexibleWidth;
    webView = [[UIWebView alloc]
            initWithFrame:rectFrame];
```

```
    webView.scalesPageToFit = YES;
    webView.delegate = self;
    [webView  loadRequest:[NSURLRequest requestWithURL:
            [NSURL URLWithString:@"http://www.google.com"]]];
    [view addSubview:webView];
    self.view = view;
    [view release];
}
- (void)dealloc {
    [webView release];
    [super dealloc];
}
-(BOOL)webView:(UIWebView *)webView
    shouldStartLoadWithRequest:(NSURLRequest *)request
    navigationType:(UIWebViewNavigationType)navigationType{
    url = [[request URL] absoluteString];
    NSLog(url);
    if([url hasSuffix:@".pdf"] == YES){
            UIAlertView *alert = [[[UIAlertView alloc]
                    initWithTitle:@"Do You Want To Also Save A Copy?"
                    message:@"Download PDF file?"
                    delegate:self
                    cancelButtonTitle:@"No"
                    otherButtonTitles:@"Download", nil] autorelease];
            [alert show];
    }
    return YES;
}
- (void)alertView:(UIAlertView *)alertView
    clickedButtonAtIndex:(NSInteger)buttonIndex{
    if(buttonIndex == 1){ //загрузить?
            NSLog(@"Downloading %@ ...", url);
    }
}
@end
```

Глава 8
Табличное представление

Табличное представление является важным и часто используемым объектом графического пользовательского интерфейса в ОС iPhone. Понимание табличных представлений необходимо для написания iPhone-приложений. К счастью, нет ничего проще, чем программировать табличные представления.

Эта глава ознакомит вас с табличными представлениями. В разд. 8.1 будет представлен обзор основных их концепций. В разд. 8.2 мы рассмотрим простейшее приложение с табличным представлением и обязательные методы, которые вы должны реализовать, чтобы реагировать на взаимодействие пользователя с таким представлением.

Из разд. 8.3 вы узнаете, насколько просто добавлять изображения к строкам в таблице. В разд. 8.4 будут представлены понятия секций и приложений с табличным представлением с верхними и нижними колонтитулами. В разд. 8.5 вы ознакомитесь с редактированием табличного представления и изучите приложение, позволяющее удалять ряды.

Из разд. 8.6 вы узнаете, как добавлять новые ряды в табличное представление. В нем будет описана программа, выводящая пользователю представление с данными и добавляющая новые данные в таблицу. В разд. 8.7 мы продолжим обсуждение режима редактирования и рассмотрим приложение для упорядочивания табличных записей, в частности, рядов таблицы.

В разд. 8.8 будет представлен механизм вывода иерархической информации, а также программа, использующая табличное представление для вывода трех уровней иерархии. В разд. 8.9 вы увидите группировку табличных представлений на примере, а разд. 8.10 ознакомит вас с основными понятиями касательно индексированных табличных представлений. Итоги главы будут подведены в разд. 8.11.

8.1. Обзор

Чтобы использовать табличное представление в приложении, вам нужно создать экземпляр класса `UITableView`, настроить его и добавить в качестве дочернего к другому представлению. Класс `UITableView` является подклассом `UIScrollView`, который, в свою очередь, является

подклассом UIView. Табличное представление позволяет создать только один столбец и ноль или более строк. Каждая строка в таблице представляется ячейкой. Ячейка — это экземпляр класса UITableViewCell. Ячейка предоставляет четыре стиля (на выбор). В дополнение у вас есть доступ к ее свойству contentView, позволяющему настроить ячейку по собственному желанию.

Класс UITableView зависит от двух внешних объектов: одного — для предоставления данных для отображения, второго — для контроля внешнего вида таблицы. Объект, предоставляющий данные, должен заимствовать протокол UITableViewDataSource, тогда как объект, отвечающий за визуальные аспекты таблицы, — протокол UITableViewDelegate.

Из следующих разделов вы узнаете, как создавать и настраивать табличное представление, начиная с простой таблицы и постепенно добавляя ей новые свойства.

8.2. Простейшее приложение с табличным представлением

В этом разделе мы рассмотрим простейшее приложение с табличным представлением. Эта программа создает в табличном виде список персонажей из шоу «Симпсоны». Приложение устанавливает значения по умолчанию для табличного представления. Оно не реализует ни одного метода делегата таблицы, так как они — необязательные. Тем не менее программа использует делегат приложения в качестве источника данных и реализует два обязательных метода протокола UITableViewDataSource.

Листинг 8.1 показывает объявление класса делегата приложения TVAppDelegate. Как вы уже знаете, класс заимствует протокол, прописывая его имя после названия суперкласса следующим образом: NSObject <UITableViewDataSource>. В дополнение к экземпляру UITableView myTable мы храним массив строк, представляющий модель данных в экземпляре NSArray theSimpsons.

Листинг 8.1. Объявление делегата приложения (TVAppDelegate.h) для простейшего приложения с табличным представлением

```
#import <UIKit/UIKit.h>
#import <Foundation/Foundation.h>
@interface TVAppDelegate : NSObject<UITableViewDataSource> {
    UIWindow    *window;
    UITableView *myTable;
    NSArray     *theSimpsons;
}
@end
```

Листинг 8.2 демонстрирует реализацию класса TVAppDelegate. Внутри метода applicationDidFinishLaunching: мы производим

всю необходимую инициализацию. После создания главного окна мы генерируем экземпляр табличного представления. Для инициализации табличного представления используется метод `initWithFrame:style:`. Фрейм, используемый для инициализации, — это вся область фрейма приложения, так как мы хотим, чтобы таблица занимала всю доступную программе область. Используемый стиль — `UITableViewStylePlain`. Стиль таблицы следует задать во время фазы инициализации, и позже его нельзя изменить. Если вы пропустите метод инициализации и используете инициализатор класса `UIView initWithFrame:`, то по умолчанию будет применяться стиль `UITableViewStylePlain`. Остальные стили будут описаны в этой главе позже. После инициализации мы наполним массив `theSimpsons` именами, которые будут отображаться в таблице. Далее в качестве источника данных будет установлен экземпляр делегата приложения. Свойством класса `UITableView` для источника данных является `dataSource`, которое объявлено следующим образом:

```
@property(nonatomic, assign) id <UITableViewDataSource> dataSource;
```

Обратите внимание, что это свойство использует директиву `assign` вместо `retain`. Наконец, табличное представление добавляется к главному окну, которое делается ключевым и видимым.

Листинг 8.2. Определение делегата приложения (TVAppDelegate.m) для простейшего приложения с табличным представлением; делегат приложения оперирует табличным представлением и выступает в роли источника данных

```
#import <Foundation/Foundation.h>
#import <UIKit/UIKit.h>
#import "TVAppDelegate.h"
@implementation TVAppDelegate
- (void)applicationDidFinishLaunching:(UIApplication *)application {
    window = [[UIWindow alloc]
            initWithFrame:[[UIScreen  mainScreen] bounds]] ;
    myTable = [[UITableView alloc]
            initWithFrame:[UIScreen mainScreen].applicationFrame
            style:UITableViewStylePlain];
    theSimpsons = [[NSArray arrayWithObjects:
            @"Homer Jay Simpson",
            @"Marjorie \"Marge\" Simpson",
            @"Bartholomew \"Bart\" J. Simpson",
            @"Lisa Marie Simpson",
            @"Margaret \"Maggie\" Simpson",
            @"Abraham J. Simpson",
            @"Santa's Little Helper",
            @"Ned Flanders",
            @"Apu Nahasapeemapetilon",
            @"Clancy Wiggum",
            @"Charles Montgomery Burns",
                    nil] retain];
    myTable.dataSource = self;
    [window addSubview:myTable];
    [window makeKeyAndVisible];
}
```

```
//Обязательные методы источника данных
- (NSInteger)tableView:(UITableView *)tableView
    numberOfRowsInSection:(NSInteger)section{
    return [theSimpsons count];
}
- (UITableViewCell *) tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *) indexPath {
    UITableViewCell *cell =
            [tableView
                    dequeueReusableCellWithIdentifier:@"simpsons"];
    if (cell == nil) {
            cell = [[[UITableViewCell alloc]
                    initWithFrame:CGRectZero
                    reuseIdentifier:@"simpsons"] autorelease];
    }
    // Настройка ячейки
    cell.text = [theSimpsons objectAtIndex:indexPath.row];
    return cell;
}
- (void)dealloc {
    [window release];
    [myTable release];
    [theSimpsons release];
    [super dealloc];
}
@end
```

Два обязательных метода протокола `UITableViewDataSource` следующие.

1. `tableView:numberOfRowsInSection:`. По умолчанию табличное представление будет иметь одну секцию, но вам нужно указать количество столбцов в ней. Данный метод источника данных при вызове запрашивает это количество. Метод объявлен следующим образом:

```
- (NSInteger)tableView:(UITableView *)table
    numberOfRowsInSection:(NSInteger)section;
```

Вам дается два значения — экземпляр табличного представления, позволяющий иметь один и тот же метод источника данных для нескольких экземпляров табличных представлений, и количество секций, которое в данном примере всегда 0 (и игнорируется), так как мы выбираем значение по умолчанию.

2. `tableView:cellForRowAtIndexPath:`. Табличное представление вызывает этот метод, запрашивая источник данных о ячейке таблицы, представляющей заданный столбец. Метод объявлен как

```
- (UITableViewCell *) tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath;
```

В дополнение к экземпляру табличного представления вам предоставляется экземпляр `NSIndexPath`, который используется в Cocoa как класс, представляющий серию индексов. Например, `1.5.8.33` представляет собой индексированный путь. Этот класс расширяется `UITableView` путем объявления категории, показанной в листинге 8.3.

Листинг 8.3. Расширение NSIndexPath для использования в UITableView

```
@interface NSIndexPath (UITableView)
+ (NSIndexPath *)indexPathForRow:(NSUInteger)row
    inSection:(NSUInteger)section;
@property(nonatomic,readonly) NSUInteger section;
@property(nonatomic,readonly) NSUInteger row;
@end
```

Категория добавляет два свойства — `section` и `row`.

Имея экземпляр `NSIndexPath`, вы можете определить конфигурацию ячейки, которую желаете вернуть. Возвращая ячейку, вы можете создать ее с самого начала и вернуть самовысвобождающейся либо вернуть уже созданную кэшированную ячейку. Повторное использование ячеек поощряется. После создания ячейки вы должны использовать назначенный инициализатор `initWithStyle:reuseIdentifier:`, объявленный следующим образом:

```
- (id)initWithStyle:(CGRect)frame
    reuseIdentifier:(NSString *)reuseIdentifier;
```

Значение `frame` может быть `CGRectZero`, так как таблица будет сама оптимально размещать и масштабировать ячейку; `reuseIdentifier` — это строка, используемая для пометки ячейки так, чтобы она легко идентифицировалась для повторного использования. Наш метод создает новую ячейку в следующем выражении:

```
cell = [[[UITableViewCell alloc]
            initWithFrame:CGRectZero
            reuseIdentifier:@"simpsons"] autorelease];
```

Как уже было сказано, следует повторно использовать кэшированную ячейку настолько, насколько это возможно. Чтобы получить повторно используемую ячейку, примените метод экземпляра `UITableView dequeueReusableCellWithIdentifier:`, объявленный как

```
- (UITableViewCell *)dequeueReusableCellWithIdentifier: (NString *)identifier
```

Значение `identifier` — это тот же тег, который использовался при создании ячейки, в нашем случае равный `@"simpsons"`. Если есть доступная ячейка, будет возвращен указатель на нее. Если доступных ячеек нет, вернется значение `nil`.

Получив экземпляр `UITableViewCell`, вам нужно настроить его, используя значения, соответствующие количеству секций и строк. Мы реализуем простейшее табличное представление, поэтому только устанавливаем текстовое свойство ячейки в соответствующее значение из массива `theSimpons` (который представляет модель данных), как показано в следующем выражении:

```
cell.textLabel.text = [theSimpsons objectAtIndex:indexPath.row];
```

Ниже приведен снимок экрана приложения (рис. 8.1).

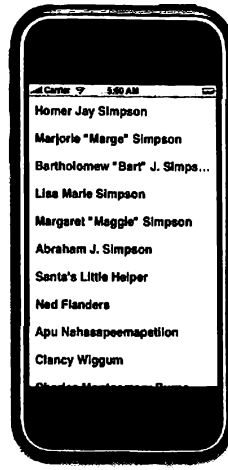

Рис. 8.1. Снимок экрана простейшего приложения с текстовым табличным представлением

8.3. Табличное представление, содержащее изображения и текст

Из предыдущего раздела вы узнали, как создавать табличное представление, отображающее текстовые элементы. Как упоминалось ранее, каждая ячейка может иметь изображение с левой стороны. В листинге 8.4 приведен усовершенствованный метод `tableView:cellForRowAtIndexPath:`, настраивающий ячейки для отображения картинок.

Листинг 8.4. Усовершенствованный метод tableView:cellForRowAtIndexPath:, позволяющий добавлять изображение в левую часть каждой ячейки

```
- (UITableViewCell *)
    tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    UITableViewCell *cell =
            [tableView dequeueReusableCellWithIdentifier:@"simpsons"];
    if (cell == nil) {
            cell = [[[UITableViewCell alloc]
                    initWithFrame:CGRectZero
                    reuseIdentifier:@"simpsons"] autorelease];
    }
    // Настройка ячейки
    cell.text = [theSimpsons objectAtIndex:indexPath.row];
    NSString *imageName =
```

```
            [NSString stringWithFormat:@"%d.png", indexPath.row];
    cell.image = [UIImage imageNamed:imageName];
    return cell;
}
```

Для определения изображения в ячейке установите свойство `image` для экземпляра `UITableViewCell`. Свойство `image` объявлено как

```
@property(nonatomic, retain) UIImage  *image
```

Значение по умолчанию — `nil`, указывающее на отсутствие картинки.

Изображение для каждой строчки загружается из упаковки приложения (гл. 9), используя метод `imageNamed:` класса `UIImage`. Хранящиеся картинки именуются в соответствии с индексом строки. Например, изображение для первой строки будет называться `0.png`. Метод `stringWithFormat:` класса `NSString` применяется для получения имени картинки, используемого в вызове метода `imageNamed:`. Ниже приведен снимок экрана приложения (рис. 8.2).

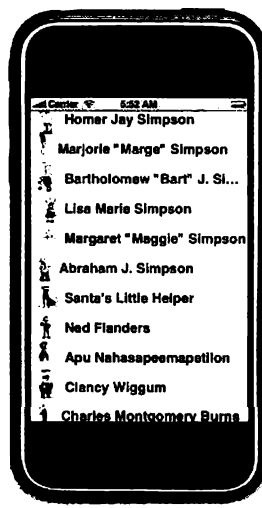

Рис. 8.2. Табличное представление с изображениями и текстом (в связи с авторскими правами реальные изображения персонажей заменены другими картинками)

8.4. Табличное представление с верхним и нижним колонтитулами секции

В предыдущих разделах вы видели табличные представления только с одной секцией. Однако можно получить таблицу с более чем одной

секцией, а также представить эти секции с заголовками верхнего и нижнего колонтитулов.

Модифицируем наш пример так, что таблица будет с двумя секциями. Нам требуются два массива — `theSimpsonsFamily`, содержащий имена семейства Симпсонов, и `theSimponsOthers`, включающий имена остальных персонажей. Чтобы создать две секции, необходимо выполнить следующие действия.

Во-первых, мы изменим метод `numberOfSectionsInTableView:`, возвращающий 2 на запрос количества секций. Во-вторых, нужно изменить `tableView:numberOfRowsInSection:` следующим образом:

```
-(NSInteger)tableView:(UITableView *)tableView
    numberOfRowsInSection:(NSInteger)section {
  if(section == 0) {
    return [theSimpsonsFamily count];
  }
  else {
    return [theSimpsonsOthers count];
  }
}
```

В-третьих, если вы хотите получить верхний колонтитул секции, добавьте следующий метод источника данных:

```
- (NSString *)
    tableView:(UITableView *)tableView
            titleForHeaderInSection:(NSInteger)section{
  if(section == 0) {
    return @"The Simpsons Family";
  }
  else{
    return @"The Others";
  }
}
```

В-четвертых, если вы хотите получить нижний колонтитул секции, добавьте следующий метод источника данных:

```
-(NSString *)
    tableView:(UITableView *)tableView
            titleForFooterInSection:(NSInteger)section {
  if(section == 0) {
    return @"End of Simpsons Family";
  }
  else {
    return @"End of The Others";
  }
}
```

Наконец, измените `tableView:cellForRowAtIndexPath:` так, чтобы он возвращал соответствующую ячейку, следующим образом:

```
- (UITableViewCell *)tableView:(UITableView *)tableView
            cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    UITableViewCell *cell =
```

```
        [tableView dequeueReusableCellWithIdentifier:@"simpsons"];
    if (cell == nil) {
        cell = [[[UITableViewCell alloc]
                 initWithFrame:CGRectZero
                 reuseIdentifier:@"simpsons"] autorelease];
    }
    // Настройка ячейки
    if(indexPath.section == 0){
        cell.text = [theSimpsonsFamily objectAtIndex:indexPath.row];
    }
    else{
        cell.text  = [theSimpsonsOthers objectAtIndex:indexPath.row];
    }
    NSString *imageName = [NSString stringWithFormat:@"%d-%d.png",
        indexPath.row, indexPath.section];
    cell.image = [UIImage imageNamed:imageName];
    return cell;
}
```

Ниже показано табличное представление с секциями и верхними и нижними колонтитулами секций (рис. 8.3). Обратите внимание, что колонтитулы всегда видны при прокрутке содержимого таблицы.

Рис. 8.3. Табличное представление с секциями, верхним и нижним колонтитулами секции

8.5. Табличное представление с возможностью удалять строки

Табличное представление можно перевести в режим редактирования во время выполнения программы. В таком режиме вы можете удалять,

вставлять и упорядочивать строки. В этом разделе мы рассмотрим приложение, позволяющее пользователю удалять строки. Программа использует кнопку, которая при нажатии переводит экземпляр табличного представления в режим редактирования. Затем пользователь нажимает кнопку удаления и подтверждает удаление строки. Источник данных табличного представления получает сообщение с подтверждением удаления. Если источник данных одобряет удаление, то модель данных, представленная изменяемым массивом, обновляется, удалив соответствующий элемент, а экземпляру табличного представления предлагается удалить строку, опционально анимируя процесс удаления. Это основные этапы. Рассмотрим их подробнее.

Листинг 8.5 показывает объявление делегата приложения `TVAppDelegate`. Делегат программы управляет табличным представлением и выступает в роли источника данных. Обратите внимание, что у нас есть изменяемый массив `theSimpsons`, содержимое которого будет реагировать на изменение режима данных. Кнопка `editButton` используется для переключения между редактируемым и нередактируемым режимами.

Листинг 8.5. Объявление (TVAppDelegate.h) делегата для приложения с табличным представлением, обладающим возможностью удалять ряды; делегат приложения управляет табличным представлением и выступает в роли источника данных

```
#import <UIKit/UIKit.h>
#import <Foundation/Foundation.h>
@interface TVAppDelegate : NSObject <UITableViewDataSource> {
    UIWindow        *window;
    UITableView     *myTable;
    NSMutableArray *theSimpsons;
    UIButton        *editButton;
}
@end
```

В листинге 8.6 приведено определение делегата приложения. Сначала мы создаем кнопку редактирования (см. разд. 5.5), которая будет переключать два режима. Логика смены редактируемого и нередактируемого режимов табличного представления находится в методе действия `editAction:`. Чтобы перевести табличное представление в режим редактирования, нужно вызвать метод `setEditing:animated:`, объявленный следующим образом:

```
- (void)setEditing: (BOOL)editing animated:(BOOL)animate
```

Если `editing` равно `YES`, то таблица переходит в режим редактирования. Установите `animate` в `YES` для анимации смены режима. Как только табличное представление получает это сообщение, оно пересылает его каждой видимой строке (ячейке).

В режиме редактирования можно удалить или добавить каждую строку. Если строка находится в режиме редактирования и подготовлена к удалению, она помечается слева значком в виде красного минуса в кружке. Если строка находится в режиме редактирования и подготовлена к вставке,

у нее слева появляется значок в виде зеленого плюса. Возникает вопрос — как табличное представление знает, в какой режим переходить? Ответ прост: опциональный метод делегата `tableView:editingStyleForRowAtIndexPath:` используется для предоставления стиля редактирования для заданной ячейки. Метод объявлен как

```
-(UITableViewCellEditingStyle)
    tableView:(UITableView *)tableView
    editingStyleForRowAtIndexPath:(NSIndexPath *)indexPath
```

Если табличное представление находится в режиме редактирования, у него есть делегат, и если делегат реализует этот метод, то этот метод вызывается для каждой видимой строки, запрашивая у него стиль редактирования для этой строки. Возвращаемое значение может быть `UITableViewCellEditingStyleDelete` либо `UITableViewCellEditingStyleInsert`. Если с представлением не связан делегат (как в нашем примере) или делегат не реализует этот метод, принимается значение `UITableViewCellEditingStyleDelete`.

Листинг 8.6. Определение (TVAppDelegate.h) делегата для приложения с табличным представлением, обладающим возможностью удалять ряды; делегат управляет табличным представлением и выступает в роли источника данных

```
#import <Foundation/Foundation.h>
#import <UIKit/UIKit.h>
#import "TVAppDelegate.h"
@implementation TVAppDelegate
- (void)applicationDidFinishLaunching:(UIApplication *)application {
    window = [[UIWindow alloc]
            initWithFrame:[[UIScreen  mainScreen] bounds]];
    editButton = [[UIButton
            buttonWithType:UIButtonTypeRoundedRect] retain];
    editButton.frame = CGRectMake(105.0, 25.0, 100, 40);
    [editButton setTitle:@"Edit" forState:UIControlStateNormal];
    [editButton addTarget:self action:@selector(editAction:)
            forControlEvents:UIControlEventTouchUpInside];
    [window addSubview:editButton];
    CGRect frame = CGRectMake(0, 70, 320, 420);
    myTable = [[UITableView alloc]
            initWithFrame:frame
            style:UITableViewStylePlain];
    theSimpsons = [[NSMutableArray
            arrayWithObjects:@"Homer Jay Simpson",
            @"Marjorie \"Marge\" Simpson",
            @"Bartholomew \"Bart\" J. Simpson",
            @"Lisa Marie Simpson"
            @"Margaret \"Maggie\" Simpson",
            @"Abraham J. Simpson",
            @"Santa's Little Helper",
            @"Ned Flanders",
            @"Apu Nahasapeemapetilon",
            @"Clancy Wiggum",
            @"Charles Montgomery Burns",
            nil] retain];
    myTable.dataSource = self;
    [window addSubview:myTable];
    [window makeKeyAndVisible];
```

```
}
- (void)editAction:(id)sender{
    if(sender == editButton){
            if([editButton.currentTitle isEqual:@"Edit"] == YES){
                    [editButton setTitle:@"Done" forState:UIControlStateNormal];
                    [myTable setEditing:YES animated:YES];
            } else {
                    [editButton setTitle:@"Edit" forState:UIControlStateNormal];
                    [myTable setEditing:NO animated:YES];
            }
    }
}
- (void)dealloc {
    [window release];
    [myTable release];
    [theSimpsons release];
    [editButton release];
    [super dealloc];
}
//Методы источника данных
- (NSInteger)tableView:(UITableView *)tableView
    numberOfRowsInSection:(NSInteger)section{
    return [theSimpsons count];
}
- (void) tableView:(UITableView *)tableView
    commitEditingStyle:(UITableViewCellEditingStyle) editingStyle
    forRowAtIndexPath:(NSIndexPath *)indexPath{
    if(editingStyle == UITableViewCellEditingStyleDelete){
            [theSimpsons removeObjectAtIndex:indexPath.row];
            [myTable
                    deleteRowsAtIndexPaths:[NSArray arrayWithObject:indexPath]
                    withRowAnimation:UITableViewRowAnimationFade];
    }
}
- (UITableViewCell *)
    tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath{
    UITableViewCell *cell =
            [tableView dequeueReusableCellWithIdentifier:@"simpsons"];
    if (cell == nil) {
            cell = [[[UITableViewCell alloc]
                    initWithFrame:CGRectZero
                    reuseIdentifier:@"simpsons"] autorelease];
    }
    // Настройка ячейки
    cell.text  = [theSimpsons objectAtIndex:indexPath.row];
    return cell;
}
@end
```

В любой момент, когда пользователь нажимает кнопку, вызывается метод `editAction:`. Метод сначала проверяет текущий режим, сверяясь с заголовком кнопки. Если заголовок Edit (Редактировать), то он заменяется на Done (Готово), а табличное представление с анимацией переводится в режим редактирования. В противном случае заголовок кнопки меняется на Edit (Редактировать), а табличное представление с анимацией заканчивает редактирование. Ниже изображено приложение в нередактируемом режиме (рис. 8.4), а также в режиме редактирования (по умолчанию — удаление) (рис. 8.5).

Рис. 8.4. Табличное представление, разрешающее редактирование (редактирование можно начать, нажав кнопку Edit (Редактировать))

Рис. 8.5. Табличное представление в режиме редактирования (режим редактирования по умолчанию – удаление)

Когда пользователь нажимает значок минуса, справа появляется кнопка Delete (Удалить), подтверждающая удаление. Если пользователь нажимает ее, табличное представление посылает сообщение источнику данных `tableView:commitEditingStyle:forRowAtIndexPath:`. Метод объявлен как

```
-(void) tableView:(UITableView *)tableView
    commitEditingStyle:(UITableViewCellEditingStyle)editingStyle
    forRowAtIndexPath:(NSIndexPath *)indexPath
```

Параметр `tableView` — это экземпляр табличного представления, запрашивающий подтверждение редактирования; `editingStyle` — это стиль, в котором находится строка (удаление или добавление); `indexPath` — это объект, содержащий номер секции и строки ячейки, над которой производится действие.

Если строку нужно удалить, приведенный выше метод должен обновить ее модель данных, удалив данные для соответствующей строки и вызвав метод `deleteRowsAtindexPaths:withRowAnimation:`. Метод объявлен как

```
-(void)deleteRowsAtIndexPaths:(NSArray *)indexPaths
    withRowAnimation:(UITableViewRowAnimation)animation
```

Параметр `indexPaths` — это экземпляр `NSArray`, хранящий экземпляры `NSIndexPath` для строк, которые следует удалить. Значение `animation` может быть одним из следующих:

- `UITableViewRowAnimationRight` — указывает, что удаленная строка должна плавно переместиться за правый край экрана;
- `UITableViewRowAnimationLeft` — удаленная строка должна плавно передвинуться за левую границу экрана;
- `UITableViewRowAnimationTop` — удаленная строка должна плавно переместиться за верхний край экрана;
- `UITableViewRowAnimationBottom` — удаленная строка должна плавно передвинуться за нижнюю границу экрана.

Далее изображены табличное представление с кнопкой подтверждения удаления (рис. 8.6), после удаления строки, но все еще находящееся в режиме редактирования (рис. 8.7), и после выхода из режима редактирования после успешного удаления строки `Lisa` (рис. 8.8).

Рис. 8.6. Строка табличного представления с запросом на подтверждение удаления

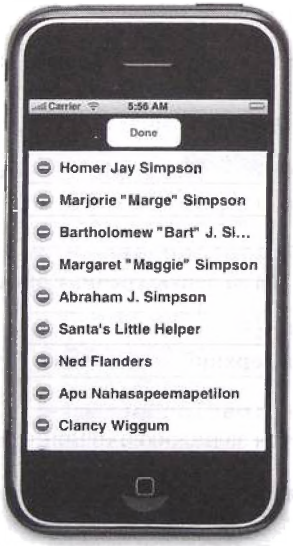

Рис. 8.7. Табличное представление после удаления строки, но все еще находящееся в режиме редактирования

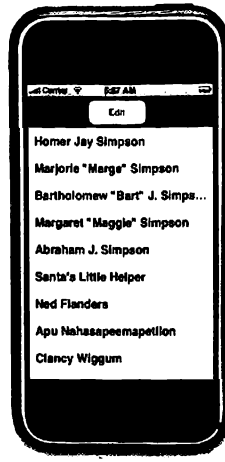

Рис. 8.8. Табличное представление после выхода из режима редактирования и успешного удаления строки

8.6. Табличное представление с возможностью вставки строк

Из предыдущего раздела вы узнали, как настроить табличное представление для перехода в режим редактирования и управления удалением строк. Этот раздел будет посвящен вставке новых строк. Листинг 8.7 показывает объявление делегата демонстрационного приложения.

Делегат приложения создаст новое табличное представление, кнопку редактирования и представление просмотра данных. Он также будет выступать в роли источника данных и делегата табличного представления.

Листинг 8.7. Объявление класса делегата приложения TVAppDelegate в файле TVAppDelegate.h; делегат позволяет вставлять новые записи в таблицу и выступает в роли как источника данных, так и делегата табличного представления

```
#import <UIKit/UIKit.h>
#import <Foundation/Foundation.h>
@interface TVAppDelegate:
    NSObject <UITableViewDelegate, UITableViewDataSource, UITextFieldDelegate> {
    UIWindow        *window;
    UITableView     *myTable;
    NSMutableArray *theSimpsons;
    UIButton        *editButton;
    UIView          *inputACharacterView;
    UITextField     *characterTextField;
}
-(void)insertCharacter;
@end
```

Листинг 8.8 демонстрирует метод `applicationDidFinishLaunching:` для нашего делегата приложения. Сначала метод создает и настраивает кнопку Edit (Редактировать). Затем создается и настраивается табличное представление. Как вы видели в предыдущем разделе, когда пользователь нажимает кнопку редактирования, табличное представление переходит в соответствующий режим. Метод действия кнопки идентичен изученному вами в предыдущем разделе. Ниже изображено стартовое окно приложения (рис. 8.9).

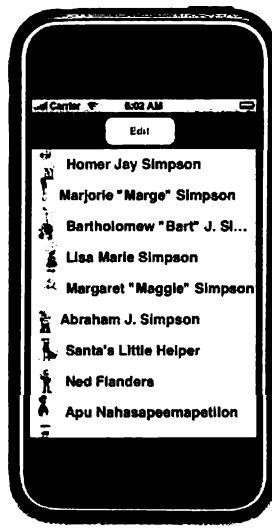

Рис. 8.9. Табличное представление, позволяющее вставлять строки

Листинг 8.8. Метод applicationDidFinishLaunching: делегата приложения, оперирующего табличным представлением с возможностью вставки

```
- (void)applicationDidFinishLaunching:(UIApplication *)application {
    window = [[UIWindow alloc]
            initWithFrame:[[UIScreen  mainScreen] bounds]];
    editButton = [[UIButton buttonWithType:
            UIButtonTypeRoundedRect] retain];
    editButton.frame = CGRectMake(105.0, 25.0, 100, 40);
    [editButton setTitle:@"Edit" forState:UIControlStateNormal];
    [editButton addTarget:self
            action:@selector(editAction:)
            forControlEvents:UIControlEventTouchUpInside];
    [window addSubview: editButton];
    CGRect frame = CGRectMake(0, 70, 320, 420);
    myTable = [[UITableView alloc]
            initWithFrame:frame style:UITableViewStylePlain];
    theSimpsons = [[NSMutableArray arrayWithObjects:
            @"Homer Jay Simpson",
            @"Marjorie \"Marge\" Simpson",
```

```
            @"Bartholomew \"Bart\" J. Simpson",
            @"Lisa Marie Simpson"
            @"Margaret \"Maggie\" Simpson",
            @"Abraham J. Simpson",
            @"Santa's Little Helper",
            @"Ned Flanders",
            @"Apu Nahasapeemapetilon",
            @"Clancy Wiggum",
            @"Charles Montgomery Burns",
            nil] retain];
    myTable.delegate = self;
    myTable.dataSource = self;
    [window addSubview:myTable];
    inputACharacterView  = nil;
    [window makeKeyAndVisible];
}
```

Делегат приложения определяет метод `tableView:editingStyleForRowAtIndexPath:`, необходимый для переопределения стиля редактирования по умолчанию. Метод просто возвращает `UITableViewCellEditingStyleInsert`, как показано ниже:

```
- (UITableViewCellEditingStyle)
    tableView:(UITableView *)tableView
    editingStyleForRowAtIndexPath:(NSIndexPath *)indexPath{
    return UITableViewCellEditingStyleInsert;
}
```

Ниже приведен снимок экрана приложения после входа в стиль редактирования для вставки значений (рис. 8.10).

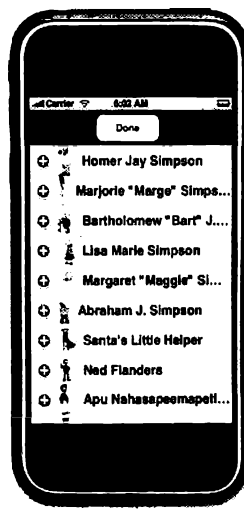

Рис. 8.10. Табличное представление после входа в режим редактирования для вставки значений

Листинг 8.9 показывает метод `tableView:cellForRowAtIndexPath:`. Если картинка не найдена, используется общее изображение, что позволяет добавленной строке иметь свою картинку.

Листинг 8.9. Метод источника данных, производящий ячейки с текстом и изображениями; он принимает заново добавленные записи и снабжает их шаблоном «незнакомого персонажа»

```
- (UITableViewCell *)
    tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath{
    UITableViewCell *cell =
            [tableView dequeueReusableCellWithIdentifier:@"simpsons"];
    if (cell == nil) {
            cell = [[[UITableViewCell alloc]
                    initWithFrame:CGRectZero
                    reuseIdentifier:@"simpsons"] autorelease];
    }
    // Настройка ячейки
    cell.text = [theSimpsons objectAtIndex:indexPath.row];
    NSString *imageName =
            [NSString stringWithFormat:@"%d.png", indexPath.row];
    cell.image = [UIImage imageNamed:imageName];
    if(cell.image == nil)
            cell.image = [UIImage imageNamed:@"unknown-person.gif"];
    return cell;
}
```

Следующий листинг демонстрирует метод `tableView:commitEditingStyle:forRowAtIndexPath:`. Он вызывает другой метод, `insertCharacter`, который на самом деле будет представлять пользователю запись данных:

```
- (void)tableView:(UITableView *)tableView
            commitEditingStyle:(UITableViewCellEditingStyle)
            editingStyle
            forRowAtIndexPath:(NSIndexPath *)indexPath{
    if(editingStyle == UITableViewCellEditingStyleInsert){
     [self insertCharacter];

    }

}
```

Листинг 8.10 показывает метод `insertCharacter`. Метод создает представление и добавляет к нему несколько элементов управления. Кнопка Cancel (Отменить) прибавляется к представлению для отмены ввода данных. В качестве подписи к текстовому полю добавляется также экземпляр `UILabel` со значением `Name:`. Текстовое поле — это поле ввода, куда пользователь вводит новое имя, которое добавится к табличному представлению. Далее показаны представления для отображения данных (рис. 8.11, 8.12).

Рис. 8.11. Представление ввода данных для добавления новой записи в табличное представление перед появлением клавиатуры

Листинг 8.10. Метод insertCharacter, отображающий представление ввода данных

```
- (void)insertCharacter{
    inputACharacterView = [[[UIView alloc]
            initWithFrame:[UIScreen  mainScreen] bounds]] autorelease];
    inputACharacterView.backgroundColor = [UIColor grayColor];
    UIButton *cancelButton =[UIButton
            buttonWithType:UIButtonTypeRoundedRect];
    cancelButton.frame = CGRectMake(105.0, 25.0, 100, 40);
    [cancelButton setTitle:@"Cancel"
            forState:UIControlStateNormal];
    [cancelButton addTarget:self
            action:@selector(cancelAction:)
            forControlEvents:UIControlEventTouchUpInside];
    [inputACharacterView addSubview: cancelButton];
    UILabel *label = [[[UILabel alloc]
            initWithFrame:CGRectMake(0, 100, 70, 30)] autorelease];
    label.text = @"Name:";
    [inputACharacterView addSubview: label];
    characterTextField = [[UITextField alloc]
            initWithFrame:CGRectMake(80, 100, 250, 30)];
    characterTextField.textColor = [UIColor blackColor];
    characterTextField.font = [UIFont systemFontOfSize:17.0];
    characterTextField.placeholder = @"<enter a new character>";
    characterTextField.backgroundColor = [UIColor whiteColor];
    characterTextField.borderStyle = UITextBorderStyleBezel;
```

```
    characterTextField.keyboardType = UIKeyboardTypeDefault;
    characterTextField.returnKeyType = UIReturnKeyDone;
    characterTextField.clearButtonMode = UITextFieldViewModeAlways;
    characterTextField.enablesReturnKeyAutomatically = YES;
    characterTextField.delegate = self;
    [inputACharacterView addSubview: characterTextField];
    [window addSubview:inputACharacterView];
}
```

Рис. 8.12. Представление ввода данных для добавления новой записи в табличное представление после появления клавиатуры

Действием для кнопки отмены является `cancelAction:`, определенный следующим образом:

```
-(void)cancelAction:(id)sender{
    [inputACharacterView removeFromSuperview];
    inputACharacterView = nil;

}
```

Он просто удаляет представление ввода данных `inputACharacterView` из родительского представления `window` и устанавливает `inputACharacterView` в `nil`. Обратите внимание, что метод `removeFromSuperview` высвобождает получателя.

Как вы уже знаете из предыдущих глав, когда пользователь нажимает кнопку **Done** (Готово), вызывается метод делегата текстового поля

`textFieldShouldReturn:`. Внутри этого метода мы добавляем имя, введенное в текстовом поле, к модели данных (изменяемому массиву `theSimpsons`) и перегружаем данные в таблице, посылая сообщение `reloadData` экземпляру табличного представления. После этого мы удаляем представление ввода данных, как уже делали ранее при обработке события отмены ввода данных.

```
- (BOOL)textFieldShouldReturn:(UITextField *)textField{
    [theSimpsons addObject:textField.text];
    [myTable reloadData];
    [inputACharacterView removeFromSuperview];
    inputACharacterView = nil;
    return YES;

}
```

Ниже приведено приложение после добавления новой строки (рис. 8.13).

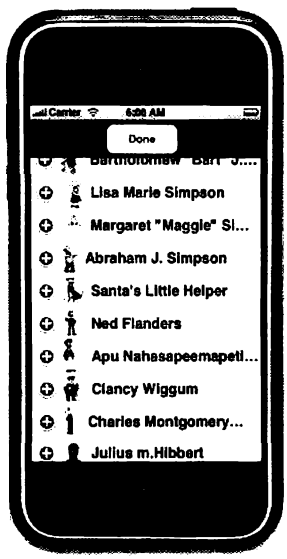

Рис. 8.13. Табличное представление после добавления новой строки внизу таблицы

8.7. Упорядочивание табличных строк

Табличное представление можно сконфигурировать так, что пользователь сможет упорядочивать табличные строки при переходе в редактируемый режим. По умолчанию упорядочивание отключено. Чтобы сделать его доступным,

источник данных должен реализовать метод tableView:moveRowAtIndexPath:toIndexPath:. Как только этот метод определен, с правой стороны каждой строки появляется значок упорядочивания, когда табличное представление находится в режиме редактирования. Чтобы сделать недоступным редактирование определенных строк, источник данных должен реализовать метод tableView:canMoveRowAtIndexPath: и исключить заданные строки.

Рассмотрим подробный пример приложения с табличным представлением, позволяющим упорядочивать строки. Листинг 8.11 показывает делегат приложения, который также выступает в роли источника данных. Обратите внимание, что модель данных theSimpsons является изменяемым массивом, так как мы должны иметь возможность динамически изменять порядок следования строк.

Листинг 8.11. Файл TVAppDelegate.h, объявляющий делегат приложения для упорядочивания табличных строк

```
#import <UIKit/UIKit.h>
#import <Foundation/Foundation.h>
@interface <UITableViewDataSource> {
    TVAppDelegate:NSObject
    UIWindow        *window;
    UITableView     *myTable;
    NSMutableArray  *theSimpsons;
    UIButton        *editButton;
}
@end
```

В листинге 8.12 демонстрируется реализация делегата приложения. Метод applicationDidFinishLaunching: идентичен изученному вами ранее. Он создает табличное представление и кнопку редактирования, а также наполняет значениями модель данных.

Метод tableView:canMoveRowAtIndexPath: определен, чтобы разрешить упорядочивание всех строк, кроме двух — Bart и Santa's Little Helper. Чтобы сделать недоступным упорядочивание этих строк, метод должен возвращать NO.

Если пользователь передвигает строку на новую позицию, вызывается метод tableView:moveRowAtIndexPath:toIndexPath:. Он объявлен как

```
-(void)
    tableView:(UITableView *)tableView
    moveRowAtIndexPath:(NSIndexPath *)fromIndexPath
    toIndexPath:(NSIndexPath *)toIndexPath
```

Здесь fromIndexPath — текущий, а toIndexPath — новый индексированный путь. В нашем случае для перемещения имени из одного места в массиве в другое мы сначала должны захватить объект в текущем местоположении выражением

```
NSString *str = [[theSimpsons objectAtIndex:fromIndexPath.row] retain];
```

Это важно, так как мы собираемся удалить объект из массива, что приведет к его высвобождению. Выражение, удаляющее объект из текущей строки, — следующее:

```
[theSimpsons removeObjectAtIndex:fromIndexPath.row];
```

После удаления объекта мы должны вставить его на новое место следующим образом:

```
[theSimpsons insertObject:str atIndex:toIndexPath.row];
```

Затем мы должны высвободить объект `str`.

Листинг 8.12. Файл TVAppDelegate.m, реализующий делегат приложения для упорядочивания табличных строк

```
#import <Foundation/Foundation.h>
#import <UIKit/UIKit.h>
#import "TVAppDelegate.h"
@implementation TVAppDelegate

- (void)applicationDidFinishLaunching:(UIApplication *)application {
    window = [[UIWindow alloc]
            initWithFrame:[[UIScreen  mainScreen] bounds]];
    editButton =
            [[UIButton buttonWithType:UIButtonTypeRoundedRect] retain];
    editButton.frame = CGRectMake(105.0, 25.0, 100, 40);
    [editButton setTitle:@"Edit" forState:UIControlStateNormal];
    [editButton addTarget:self action:@selector(editAction:)
            forControlEvents:UIControlEventTouchUpInside];
    [window addSubview:editButton];

    CGRect frame = CGRectMake(0, 70, 320, 420);
    myTable = [[UITableView alloc]
            initWithFrame:frame style:UITableViewStylePlain];
    theSimpsons = [[NSMutableArray arrayWithObjects:
            @"Homer Jay Simpson",
            @"Marjorie \"Marge\" Simpson",
            @"Bartholomew \"Bart\" J. Simpson",
            @"Lisa Marie Simpson"
            @"Margaret \"Maggie\" Simpson",
            @"Abraham J. Simpson",
            @"Santa's Little Helper",
            @"Ned Flanders",
            @"Apu Nahasapeemapetilon",
            @"Clancy Wiggum",
            @"Charles Montgomery Burns",
            nil] retain];
    myTable.dataSource = self;
    [window addSubview:myTable];
    [window makeKeyAndVisible];
}
- (void)editAction:(id)sender{
    if(sender == editButton){
            if([editButton.currentTitle isEqual:@"Edit"] == YES){
                    [editButton setTitle:@"Done" forState:UIControlStateNormal];
                    [myTable setEditing:YES animated:YES];
            }
```

```
            else {
                    [editButton setTitle:@"Edit" forState:UIControlStateNormal];
                    [myTable setEditing:NO animated:YES];
            }
    }
}

- (void)dealloc {
    [window release];
    [myTable release];
    [theSimpsons release];
    [editButton release];
    [super dealloc];
}
// Методы источника данных
- (NSInteger) tableView:(UITableView *)tableView
    numberOfRowsInSection:(NSInteger)section{
    return [theSimpsons count];
}
- (BOOL)tableView:(UITableView *)tableView
    canMoveRowAtIndexPath:(NSIndexPath *)indexPath{
    NSString  *string =
            [theSimpsons objectAtIndex:indexPath.row];
    if([string isEqual:@"Bartholomew \"Bart\" J. Simpson"] == YES)
            return NO;
    if([string  isEqual:@"Santa's Little Helper"] == YES)
            return NO;
    return YES;
}

- (void) tableView:(UITableView *)tableView
    moveRowAtIndexPath:(NSIndexPath *)fromIndexPath
    toIndexPath:(NSIndexPath *)toIndexPath{
    NSString *str= [[theSimpsons objectAtIndex:fromIndexPath.row] retain];
    [theSimpsons removeObjectAtIndex:fromIndexPath.row];
    [theSimpsons insertObject:str atIndex:toIndexPath.row];
    [str release];
}

- (UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath{
UITableViewCell *cell =
    [tableView dequeueReusableCellWithIdentifier:@"simpsons"];
if (cell == nil) {
cell = [[[UITableViewCell alloc]
    initWithFrame:CGRectZero
    reuseIdentifier:@"simpsons" ] autorelease];
}
// Настройка ячейки
    cell.text = [theSimpsons objectAtIndex:indexPath.row];
    return cell;
}
@end
```

Метод `tableView:canMOveRowAtIndexPath:` возвращает `NO` для строк `Bart` и `Santa's Little Helper` и `YES` — для остальных строк. Далее изображено табличное представление в момент, когда строка перемещается на новую позицию (рис. 8.14).

Рис. 8.14. Табличное представление с элементами управления упорядочиванием в момент перемещения строки на новое место

Ниже приведено табличное представление после того, как строка передвинулась на новое место (рис. 8.15).

Рис. 8.15. Табличное представление после перемещения строки на новую позицию

8.8. Вывод иерархической информации

Табличные представления идеальны для вывода иерархической информации. Взаимодействуя с иерархической информацией, пользователь начинает сверху и движется по следующим уровням иерархии. Он повторяет данное действие, пока не достигает желаемого уровня. Представим, что пользователь просматривает список имен участников ТВ-шоу (рис. 8.16).

Рис. 8.16. Приложение с табличным представлением, показывающее информацию о ТВ-шоу (верхний уровень)

При нажатии названия шоу ему выводится табличное представление, содержащее имена участников шоу (рис. 8.17).

Рис. 8.17. Второй уровень иерархии табличного представления, который показывает имена главных участников заданного ТВ-шоу («Остаться в живых»); обратите внимание, что кнопка возврата позволяет пользователю вернуться на предыдущий уровень

При выборе участника пользователь видит простое представление с информацией о нем (рис. 8.18).

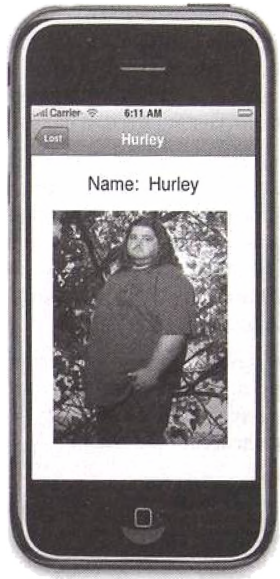

Рис. 8.18. Последний уровень иерархии в приложении: кнопка возврата перемещает пользователя на шоу, в котором участвует этот человек

Для возврата на предыдущие уровни иерархии или даже редактирования данных на текущем уровне можно использовать кнопки навигации.

Существуют два главных класса, которые помогают представить иерархическую информацию пользователю, — контроллер табличного представления и контроллер навигации. Из предыдущих глав вы узнали, как настраивать и использовать контроллер навигации. Контроллер табличного представления `UITableViewController` является подклассом `UIViewController`, который создает и управляет экземпляром `UITableView`. Вы создаете контроллер табличного представления и инициализируете его с помощью метода `initWithStyle:`. Экземпляр табличного представления, созданный контроллером, доступен при использовании свойства контроллера `tableView`. В дополнение к созданию и инициализации табличного представления контроллер выступает в роли как источника данных, так и делегата этого табличного представления.

Необходимо реализовать делегат табличного представления и метод источника данных в вашем подклассе этого контроллера табличного представления.

Создать рабочее приложение, отображающее иерархическую информацию в форме табличных представлений, можно в четыре этапа. Рассмотрим их на конкретном примере.

Шаг 1. Создайте подкласс `UITableViewController` для каждого уровня иерархии. Каждый подкласс этих контроллеров должен переопределять метод `initWithStyle:` для настройки основного заголовка и заголовка кнопки возврата, отображаемых на панели навигации.

Шаг 2. Выберите объект, который создаст контроллер навигации. Создайте и инициализируйте подкласс `UITableViewController`, который будет верхним уровнем иерархии, и поместите его в контроллер навигации как корневой контроллер представления. Объектом, который создаст эти объекты пользовательского интерфейса, обычно является делегат приложения, а вся работа делается в его методе `applicationDidFinishLaunching:`.

Шаг 3. Выберите глобальный объект, который будет доступен каждому табличному контроллеру представления. Внутри этого объекта создайте методы, позволяющие получать/устанавливать модель данных. Обычно таким объектом является делегат приложения.

Шаг 4. Переопределите метод `tableView:didSelectRowAtIndexPath:` внутри каждого контроллера табличного представления. Внутри этого метода вы должны с помощью глобального объекта (обычно делегата приложения) сохранить информацию о выбранной строке, а также создать экземпляр контроллера для следующего уровня и поместить его в контроллер навигации (полученный из глобального объекта, такого как делегат приложения). Объект, который будет отслеживать информацию о выбранной строке на каждом уровне, должен иметь отдельную переменную для каждого уровня, например, `level1IndexPath`, `level2IndexPath` и т. д. Обязательные и необязательные методы для источника данных и делегата табличного представления должны реализовываться внутри каждого подкласса контроллера табличного представления.

Подробный пример. Рассмотрим создание приложения с иерархическим табличным представлением, используя пример с ТВ-шоу (см. рис. 8.16—8.18). Приложение ТВ-шоу имеет три уровня иерархии. Первый уровень отображает таблицу, содержащую название ТВ-шоу, второй — выводит список основных участников, а третий — отображает представление, показывающее имя и изображение конкретного персонажа.

Очевидно, что нам необходимы два подкласса `UITableViewController` для первых двух уровней и один `UIViewController` — для третьего. Первый подкласс табличного представления называется `ShowTableViewController` и управляет табличным представлением, отображающим список ТВ-шоу (то есть первый уровень). Объявление контроллера показано в листинге 8.13.

Листинг 8.13. ShowTableViewController, объявленный в ShowTableViewController.h. Контроллер представляет первый уровень иерархии, отображая список ТВ-шоу

```
#import <UIKit/UIKit.h>
@interface ShowTableViewController : UITableViewController {
}
@end
```

Реализация класса `ShowTableViewController` показана в листинге 8.14. Метод `initWithStyle:` устанавливает заголовок контроллера TV Shows. Этот заголовок далее будет использован контроллером навигации для отображения посередине панели навигации, когда пользователю будет показываться первый уровень иерархии. В этом методе также установится заголовок кнопки возврата. Этим значением будет `Shows`. Когда пользователь выберет (нажмет) заданное шоу, отобразится следующий уровень иерархии, показывая список его участников, а заголовок кнопки возврата будет установлен в значение `Shows`.

Необходимые методы источника данных реализуются в контроллере. У нас есть метод `tableView:numberOfRowsInSection:`, получающий ссылку на делегат приложения и запрашивающий у него количество строк. Позже мы рассмотрим модели данных подробнее. Обязательный метод источника данных `tableView:cellForRowAtIndexPath:` также реализован в контроллере. Для настройки ячейки мы запрашиваем у делегата название шоу, используя метод `showNameAtIndex:`. В дополнение для указания, что ячейка имеет дочерние ячейки, свойство `accessoryType` устанавливается в `UITableViewCellAccessoryDisclosureIndicator`.

Листинг 8.14. Определение ShowTableViewController в файле ShowTableViewController.m; контроллер управляет первым уровнем иерархии в приложении ТВ-шоу

```
#import "ShowTableViewController.h"
#import "ShowCharactersTableViewController.h"
#import "TVAppDelegate.h"

@implementation ShowTableViewController

- (id)initWithStyle:(UITableViewStyle)style {
    if(self = [super initWithStyle:style]) {
            self.title = @"TV Shows";
            self.navigationItem.backBarButtonItem.title = @"Shows";
    }

    return self;
}
- (NSInteger)tableView:(UITableView *)tableView
    numberOfRowsInSection:(NSInteger)section{
    TVAppDelegate *delegate =
            [[UIApplication sharedApplication] delegate];
```

```
    return [delegate numberOfShows];
}
- (UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath {
static NSString *MyIdentifier = @"Show";
UITableViewCell *cell =
    [tableView dequeueReusableCellWithIdentifier:MyIdentifier];
if (cell == nil) {
    cell = [[[UITableViewCell alloc]
            initWithFrame:CGRectZero
            reuseIdentifier:MyIdentifier] autorelease];
}
// Конфигурирование ячейки
    TVAppDelegate *delegate =
            [[UIApplication sharedApplication] delegate];
    cell.text =  [delegate showNameAtIndex:indexPath.row];
    cell.accessoryType = UITableViewCellAccessoryDisclosureIndicator;
    return cell;
}

- (void)tableView:(UITableView *)tableView
    didSelectRowAtIndexPath:(NSIndexPath *)indexPath{
    TVAppDelegate *delegate =
            [[UIApplication sharedApplication] delegate];
    delegate.selectedShow = indexPath;
    ShowCharactersTableViewController *showCharactersController =
            [[[ShowCharactersTableViewController alloc]
                    initWithStyle:UITableViewStylePlain];
    [[delegate navigationController]
            pushViewController:showCharactersController animated:YES];
    [showCharactersController release];
}
@end
```

Метод `tableView:didSelectRowAtIndexPath:` — это место, где мы перемещаем пользователя на следующий уровень иерархии. Сначала мы должны сохранить индексированный путь к выбранной строке в месте, доступном следующему контроллеру представления. Это достигается путем установки свойства делегата приложения `selectedShow` в значение `indexPath`, передаваемое методу. Далее мы создаем экземпляр контроллера следующего уровня и помещаем его на верхний уровень стека контроллера навигации.

Контроллер второго уровня является экземпляром класса `ShowCharactersTableViewController`. Листинг 8.15 показывает объявление контроллера.

Листинг 8.15. ShowCharactersTableViewController, объявленный в файле ShowCharactersTableViewController.h; класс управляет вторым уровнем иерархии в приложении ТВ-шоу

```
#import <UIKit/UIKit.h>
@interface ShowCharactersTableViewController : UITableViewController {
}
@end
```

Реализация контроллера приведена в листинге 8.16. Как и для предыдущего контроллера, мы переопределяем метод `initWithStyle:` для обновления заголовков контроллера и кнопки возврата. У делегата приложения запрашивается название шоу посредством метода `showNameAtIndex:`. Индекс, используемый в данном методе, является глобальным значением `selectedShow.row`, управляемым делегатом приложения и сохраняемым в методе корневого контроллера табличного представления `tableView:didSelectRowAtIndexPath:`.

Листинг 8.16. Определение ShowCharactersTableViewController в файле ShowCharactersTableViewController.m

```
#import "ShowCharactersTableViewController.h"
#import "TVAppDelegate.h"
#import "CharacterViewController.h"

@implementation ShowCharactersTableViewController

-(id)initWithStyle:(UITableViewStyle)style {
    if (self = [super initWithStyle:style]) {
            TVAppDelegate *delegate =
                    [[UIApplication sharedApplication] delegate];
            self.title =
                    [delegate showNameAtIndex:delegate.selectedShow.row];
            self.navigationItem.backBarButtonItem.title =
                    [delegate showNameAtIndex:delegate.selectedShow.row];
    }
    return self;
}
- (NSInteger)tableView:(UITableView *)tableView
    numberOfRowsInSection:(NSInteger)section{
TVAppDelegate *delegate =
            [[UIApplication sharedApplication] delegate];
return [delegate
    numberOfCharactersForShowAtIndex:delegate.selectedShow.row];
}
- (void)tableView:(UITableView *)tableView
    didSelectRowAtIndexPath:(NSIndexPath *)indexPath{
    TVAppDelegate *delegate =
            [[UIApplication sharedApplication] delegate];
    delegate.selectedCharacter = indexPath;
    CharacterViewController *characterController =
            [[CharacterViewController alloc] init];
    [[delegate navigationController]
            pushViewController:characterController animated:YES];
    [characterController release];
}
- (UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath {
static NSString *MyIdentifier = @"Character";
    UITableViewCell *cell =
            [tableView dequeueReusableCellWithIdentifier:MyIdentifier];
if (cell == nil) {
    cell = [[[UITableViewCell alloc]
    initWithFrame:CGRectZero
    reuseIdentifier:MyIdentifier] autorelease];
}
// Конфигурирование ячейки
    TVAppDelegate *delegate =
```

```
            [[UIApplication sharedApplication] delegate];
    cell.text = [delegate
            characterNameForShowIndex:delegate.selectedShow.row
            AtIndex:indexPath.row];
    cell.accessoryType = UITableViewCellAccessoryDisclosureIndicator;
    return cell;
}
@end
```

Метод `tableView:didSelectRowAtIndexPath:` используется для помещения третьего контроллера в контроллер навигации. Этим контроллером представления является узловой контроллер `CharacterViewController`. Перед его помещением в стек мы сохраняем индексированный путь выбранной строки в свойство делегата `selectedCharacter`.

Контроллер `CharacterViewController` объявлен в листинге 8.17 и реализован в листинге 8.18.

Листинг 8.17. Объявление CharacterViewController в файле CharacterViewController.h; этот контроллер управляет узловым представлением в приложении ТВ-шоу

```
#import <UIKit/UIKit.h>

@interface CharacterViewController : UIViewController {
    UILabel  *nameLabel;
    UIView   *theView;
}
@end
```

Метод `init` переопределен, значение заголовка устанавливается равным имени персонажа. Это имя извлекается из делегата приложения посредством метода `characterNameForShowIndex:atIndex:`. Порядковый номер шоу — `selectedShow.row`, а участника — `selectedCharacter.row`.

Код, в котором мы представляем более подробную информацию о пользователе, — это метод `loadView`. Для простоты мы используем только экземпляр `UILabel` для имени и `UIImageView` для изображения персонажа. Эти объекты графического пользовательского интерфейса вам уже знакомы из предыдущих глав.

Листинг 8.18. Реализация CharacterViewController в файле CharacterViewController.m

```
#import "CharacterViewController.h"
#import "TVAppDelegate.h"

@implementation CharacterViewController
- (id)init{
    if (self = [super init]) {
            TVAppDelegate *delegate =
                    [[UIApplication sharedApplication] delegate];
            self.title =
```

```
                    [delegate characterNameForShowIndex:
                    delegate.selectedShow.row
                    AtIndex:delegate.selectedCharacter.row];
    }
    return self;
}
- (void)loadView {
    TVAppDelegate *delegate =
            [[UIApplication sharedApplication] delegate];
    theView  = [[UIView alloc]
            initWithFrame:[UIScreen mainScreen].applicationFrame];
    theView.autoresizingMask =
            UIViewAutoresizingFlexibleHeight |
            UIViewAutoresizingFlexibleWidth;
    theView.backgroundColor = [UIColor  whiteColor];

    CGRect labelFrame =  CGRectMake(80, 10, 190, 50);
    nameLabel = [[UILabel alloc] initWithFrame:labelFrame];
    nameLabel.font = [UIFont systemFontOfSize:25.0];
    nameLabel.textColor = [UIColor blackColor];
    nameLabel.backgroundColor = [UIColor clearColor];
    nameLabel.textAlignment = UITextAlignmentLeft;
    nameLabel.lineBreakMode = UILineBreakModeWordWrap;
    NSString *theName = [delegate
            characterNameForShowIndex:delegate.selectedShow.row
            AtIndex:delegate.selectedCharacter.row];
    nameLabel.text =
            [NSString stringWithFormat:@"%@:  %@", @"Name", theName];
    [theView addSubview: nameLabel];
    UIImageView *imgView = [[UIImageView alloc]
            initWithImage:[UIImage imageNamed:
    [NSString stringWithFormat:@"%@.png", theName]]];
    imgView.frame = CGRectMake(30, 70, 250, 300);
    [theView addSubview:imgView];
    [imgView release];
    self.view = theView;
}
- (void)dealloc {
    [nameLabel release];
    [theView release];
    [super dealloc];
}
@end
```

Листинг 8.19 демонстрирует объявление делегата приложения. Делегат оперирует двумя свойствами для хранения индексов первого и второго уровней, `selectedShow` и `selectedCharacter` соответственно.

Листинг. 8. 19. Объявление делегата приложения. Три контроллера представления получают доступ к модели данных иерархической информации посредством следующих четырех методов делегата приложения:

```
#import <UIKit/UIKit.h>
#import <Foundation/Foundation.h>

@interface TVAppDelegate : NSObject {
    UIWindow        *window;
    UINavigationController  *navigationController;
    NSIndexPath     *selectedShow;
    NSIndexPath     *selectedCharacter;
    NSArray         *theShows;
```

```
}

@property(nonatomic, retain) NSIndexPath *selectedShow;
@property(nonatomic, retain) NSIndexPath *selectedCharacter;
@property(nonatomic, retain)
    UINavigationController *navigationController;

-(NSInteger)numberOfShows;
-(NSString*)showNameAtIndex:(NSInteger) index;
-(NSInteger)numberOfCharactersForShowAtIndex:(NSInteger) index;
-(NSString*)characterNameForShowIndex:(NSInteger) showIndex
    AtIndex:(NSInteger) index;
@end
```

Реализация делегата приложения показана в листинге 8.20. Первое, что нам необходимо сделать в методе `applicationDidFinishLaunching`, — это подготовить модель данных. Модель данных представлена массивом словарей. Каждый словарь представляет собой ТВ-шоу и содержит две записи, первая из которых — это название шоу, а вторая — массив его участников. После инициализации модели данных мы создаем контроллер навигации и помещаем в него контроллер табличного представления первого уровня. Методы, вызываемые контроллерами для получения заданной информации о модели данных, достаточно просты.

Листинг 8.20. Реализация делегата приложения ТВ-шоу

```
#import <Foundation/Foundation.h>
#import <UIKit/UIKit.h>
#import "TVAppDelegate.h"
#import "ShowTableViewController.h"

@implementation TVAppDelegate

@synthesize selectedShow;
@synthesize selectedCharacter;
@synthesize navigationController;

- (void)applicationDidFinishLaunching:(UIApplication *)application {
    [self prepareDataModel];
    window = [[UIWindow alloc]
            initWithFrame:[[UIScreen  mainScreen] bounds]];
    ShowTableViewController *showViewController =
            [[ShowTableViewController alloc]
            initWithStyle:UITableViewStylePlain];
    navigationController = [[UINavigationController alloc]
            initWithRootViewController:showViewController];
    [showViewController release];
    [window addSubview:[navigationController view]];
    [window makeKeyAndVisible];
}
- (void)prepareDataModel {
    NSDictionary  *dic1 =
            [NSDictionary dictionaryWithObjectsAndKeys:
                    @"Seinfeld",
                    @"Name",
                    [NSArray arrayWithObjects:
                    @"Jerry",
                    @"George",
                    @"Elaine",
                    @"Kramer",
```

```
                    @"Newman",
                        @"Frank",
                        @"Susan",
                        @"Peterman",
                        @"Bania", nil], @"Characters", nil];
    NSDictionary  *dic2 =
            [NSDictionary dictionaryWithObjectsAndKeys: @"Lost", @"Name",
                . [NSArray arrayWithObjects:
                        @"Kate",
                        @"Sayid",
                        @"Sun",
                        @"Hurley",
                        @"Boone",
                        @"Claire",
                        @"Jin",
                        @"Locke",
                        @"Charlie",
                        @"Eko",
                        @"Ben", nil], @"Characters", nil];
    theShows = [[NSArray arrayWithObjects:dic1, dic2, nil] retain];
}
-(NSInteger)numberOfShows{
    return [theShows count];
}
-(NSString*)showNameAtIndex:(NSInteger) index{
    return [[theShows objectAtIndex:index] valueForKey:@"Name"];
}
-(NSInteger)numberOfCharactersForShowAtIndex:(NSInteger) index{
    return [[[theShows objectAtIndex:index]
            valueForKey:@"Characters"] count];
}
-(NSString*)characterNameForShowIndex:(NSInteger)showIndex
    AtIndex:(NSInteger) index{
return [[[theShows objectAtIndex:showIndex]
            valueForKey:@"Characters"] objectAtIndex:index];
}
- (void)dealloc {
    [window release];
    [navigationController release];
    [theShows release];
    [super dealloc];
}
@end
```

8.9. Сгруппированные табличные представления

До сих пор мы имели дело с простым стилем табличных представлений. Однако есть и другой стиль, называемый сгруппированным, который можно использовать для настройки табличного представления. Сгруппированное табличное представление в основном применяется в качестве последнего уровня иерархии для отображения информации об элементе, выбранном на предпоследнем уровне.

Сгруппированное табличное представление настраивается тем же способом, что вы видели до сих пор. Все, о чем нужно знать, — что строки каждой секции сгруппированы (вместе). В качестве названия такой группы используется опциональный заголовок, а все остальное происходит по уже известному вам принципу.

Рассмотрим пример — приложение, отображающее пользователю список любимых ТВ-шоу, отсортированных в соответствии с жанром, — комедия, политика и драма. Листинг 8.21 показывает объявление делегата демонстрационной программы. Модель данных представлена в виде трех экземпляров NSArray — comedyShows, politicalShows и dramaShows. Каждый массив будет содержать ТВ-программы для соответствующих секций.

Листинг 8.21. Объявление делегата приложения, демонстрирующего использование сгруппированных табличных представлений

```
#import <UIKit/UIKit.h>
#import <Foundation/Foundation.h>
@interface TVAppDelegate : NSObject <UITableViewDataSource> {
    UIWindow        *window;
    UITableView     *myTable;
    NSArray         *comedyShows, *politicalShows, *dramaShows;
}
@end
```

Реализация делегата приложения приведена в листинге 8.22. В методе applicationDidFinishLaunching: мы создаем экземпляр табличного представления, как вы уже видели ранее. Вместо использования простого стиля мы применим стиль UITableViewStyleGrouped, а затем наполним три массива данными.

Из разд. 8.4 вы узнали, как настраивать секции и их верхние колонтитулы. В реализации настройки различий между простым и сгруппированным стилями нет.

Листинг 8.22. Реализация делегата приложения со сгруппированными табличными представлениями

```
#import <Foundation/Foundation.h>
#import <UIKit/UIKit.h>
#import "TVAppDelegate.h"
@implementation TVAppDelegate
- (void)applicationDidFinishLaunching:(UIApplication *)application {
    window = [[UIWindow alloc]
              initWithFrame:[[UIScreen mainScreen] bounds]];
    CGRect frame = CGRectMake(0, 70, 320, 420);
    myTable = [[UITableView alloc]
              initWithFrame:frame
              style:UITableViewStyleGrouped];
    comedyShows = [[NSArray arrayWithObjects:
              @"Seinfeld",
              @"Everybody Loves Raymond", nil] retain];
    politicalShows = [[NSArray arrayWithObjects:
              @"60 Minutes",
              @"Meet The Press", nil] retain];
    dramaShows  = [[NSArray arrayWithObjects:@"Lost", nil] retain];
    myTable.dataSource = self;
    [window addSubview:myTable];
    [window makeKeyAndVisible];

}
- (void)dealloc {
    [window release];
    [myTable release];
    [comedyShows release];
```

```
    [politicalShows release];
    [dramaShows release];
    [super dealloc];
}
// Методы источника данных
- (NSInteger) numberOfSectionsInTableView:(UITableView *)tableView {
    return 3;
}
- (NSInteger) tableView:(UITableView *)tableView
          numberOfRowsInSection:(NSInteger)section{
    switch (section) {
            case 0:
                    return [comedyShows count];
                    break;
            case 1:
                    return [politicalShows count];
                    break;
            case 2:
                    return [dramaShows count];
                    break;
    }
}

-(UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell =
            [tableView dequeueReusableCellWithIdentifier:@"shows"];
    if (cell == nil) {
            cell = [[[UITableViewCell alloc]
                    initWithFrame:CGRectZero
                    reuseIdentifier:@"shows"] autorelease];
    }
    // Настройка ячейки
    switch (indexPath.section) {
            case 0:
                    cell.text = [comedyShows objectAtIndex:indexPath.row];
                    break;
            case 1:
                    cell.text =
                            [politicalShows objectAtIndex:indexPath.row];
                    break;
            case 2:
                    cell.text = [dramaShows objectAtIndex:indexPath.row];
                    break;
            }
    return cell;
    }
- (NSString *) tableView:(UITableView *)tableView
    titleForHeaderInSection:(NSInteger)section {
    NSString *title = nil;
    switch (section) {
            case 0:
                    title = @"Comedy Shows";
                    break;
            case 1:
                    title = @"Political Shows";
                    break;
            case 2:
                    title = @"Drama Shows";
                    break;
            default:
                    break;
    }
    return title;
}
@end
```

Ниже изображено приложение со сгруппированным табличным представлением (рис. 8.19).

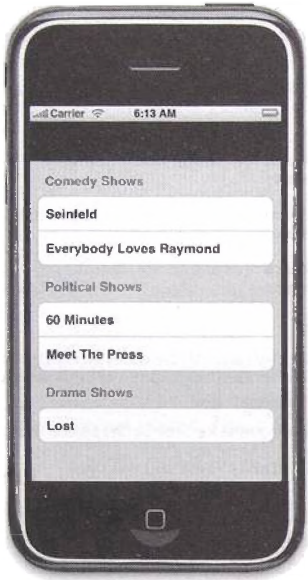

Рис. 8.19. Приложение со сгруппированным табличным представлением

8.10. Индексированные табличные представления

Иногда пользователю нужно представить большие объемы данных. Чтобы сэкономить время поиска определенной строки, вы можете добавить к табличному представлению индекс. Он будет отображаться с правой стороны таблицы. Когда пользователь нажмет заданное значение индекса, таблица прокрутится на эту секцию.

В этом разделе мы рассмотрим приложение, демонстрирующее индексированные представления. Программа представляет пять секций, каждая из которых соответствует политической партии в США. Внутри каждой секции отобразится список кандидатов на пост президента. Каждая секция имеет индекс, представленный первой буквой названия партии. Нажатие буквы заставляет таблицу прокручиваться (если это необходимо), чтобы отобразить нужную партию.

Листинг 8.23 показывает объявление делегата приложения, демонстрирующего индексированные табличные представления. Пять экземпляров `NSArray` используются для представления модели данных.

Листинг 8.23. Объявление делегата приложения, демонстрирующего индексированные табличные представления

```
#import <UIKit/UIKit.h>
#import <Foundation/Foundation.h>
@interface TVAppDelegate:NSObject<UITableViewDataSource> {
    UIWindow     *window;
    UITableView  *myTable;
    NSArray *democratic, *republican, *independent,
            *libertarian, *socialist;
}
@end
```

Листинг 8.24 показывает реализацию делегата приложения с индексированным табличным представлением. Как и в предыдущих примерах, создается и настраивается табличное представление, а модель данных заполняется именами кандидатов.

Метод `tableView:titleForHeaderInSection:` уже встречался вам в предыдущих разделах. Он возвращает заголовки секций табличного представления.

Метод `sectionIndexTitlesForTableView:` вызывается табличным представлением, которое запрашивает у источника данных экземпляры массива `NSString`. Этот массив строк сформирует список индексов справа от табличного представления. Метод объявлен следующим образом:

```
- (NSArray *)sectionIndexTitlesForTableView:(UITableView *)tableView
```

Обратите внимание, что количество элементов в этом массиве необязательно должно быть равным количеству секций в табличном представлении. В нашем примере список индексов содержит `D`, `I`, `L`, `R` и `S`. Каждая буква представляет одну политическую партию. Однако иметь букву индекса для каждой секции не требуется.

Метод `tableView:sectionForSectionIndexTitle:atIndex:` вызывается, запрашивая у источника данных индекс секции, соответствующий заголовку индекса секции и индексу заголовка секции. Объявление метода выглядит следующим образом:

```
- (NSInteger)
    tableView:(UITableView *)tableView
    sectionForSectionIndexTitle:(NSString *)title atIndex:(NSInteger)index
```

Этот метод вызывается, когда пользователь нажимает определенную букву индекса. Индекс буквы индекса и индекс секции — одни и те же, поэтому мы просто возвращаем переданное нам значение индекса. Например, если пользователь нажал букву индекса `S`, метод вызывается с параметром `title`, равным `S`, а `index` — равным 4. Буква `S` соответствует секции `Socialist Party USA`, индекс которой — 4, мы просто возвращаем переданное нам значение `index`. Если мы не захотим выбирать соотношение «один к одному» между буквами индексов и секциями, нам потребуется выполнить дополнительные действия, чтобы вернуть индекс секции.

Листинг 8.24. Реализация делегата приложения, демонстрирующего индексированные табличные представления

```
#import <Foundation/Foundation.h>
#import <UIKit/UIKit.h>
#import "TVAppDelegate.h"
@implementation TVAppDelegate
- (void)applicationDidFinishLaunching:(UIApplication *)application {
    window = [[UIWindow alloc]
            initWithFrame:[[UIScreen  mainScreen] bounds]];
    CGRect frame = CGrectMake(0, 20, 320, 420);
    myTable = [[UITableView alloc]
            initWithFrame:Frame style:UITableViewStylePlain];
            democratic = [[NSArray arrayWithObjects:
            @"Barack Obama",
            @"Joe Biden",
            @"Hillary Clinton",
            @"Christopher Dodd",
            @"John Edwards",
            @"Maurice Robert \"Mike\" Gravel",
            @"Dennis Kucinich", nil] retain];
    republican = [[NSArray arrayWithObjects:
            @"Ron Paul",
            @"John McCain",
            @"Mike Huckabee",
            @"Mitt Romney", nil] retain];
    independent  = [[NSArray arrayWithObjects:
            @"Ralph Nader", nil] retain];
    libertarian = [[NSArray arrayWithObjects:@"Bob Barr", nil] retain];
    socialist = [[NSArray arrayWithObjects:@"Brian Moore", nil] retain];
    myTable.dataSource = self;
    [window addSubview:myTable];
    [window makeKeyAndVisible];
}
- (void)dealloc {
    [window release];
    [myTable release];
    [democratic release];
    [republican release];
    [independent release];
    [libertarian release];
    [socialist release];
    [super dealloc];
}
// Методы источника данных
- (NSInteger)
    numberOfSectionsInTableView:(UITableView *)tableView {
    return 5;
}
- (NSInteger) tableView:(UITableView *)tableView
    numberOfRowsInSection:(NSInteger)section{
      switch (section) {
            case 0:
                  return [democratic count];
            case 1:
                  return [independent count];
            case 2:
                  return [libertarian count];
            case 3:
                  return [republican count];
            case 4:
                  return [socialist count]; }
      }
}
```

```
- (UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath{
            UITableViewCell *cell =
                    [tableView dequeueReusableCellWithIdentifier:@"shows"];
if (cell == nil) {
            cell = [[[UITableViewCell alloc]
                    initWithFrame:CGRectZero
                    reuseIdentifier:@"shows"] autorelease];
    }
    // Настройка ячейки
    switch (indexPath.section) {
            case 0:
                    cell.text = [democratic objectAtIndex:indexPath.row];
                    break;
            case 1:
                    cell.text = [independent objectAtIndex:indexPath.row];
                    break;
            case 2:
                    cell.text = [libertarian objectAtIndex:indexPath.row];
                    break;
            case 3:
                    cell.text = [republican objectAtIndex:indexPath.row];
                    break;
            case 4:
                    cell.text = [socialist objectAtIndex:indexPath.row];
                    break;
    }
    return cell;
}
- (NSString *)tableView:(UITableView *)tableView
titleForHeaderInSection:(NSInteger)section
{
    NSString *title = nil;
    switch (section) {
            case 0:
                    title = @"Democratic";
                    break;
            case 1:
                    title = @"Independent";
                    break;
            case 2:
                    title = @"Libertarian";
                    break;
            case 3:
                    title = @"Republican";
                    break;
            case 4:
                    title = @"Socialist Party USA";
                    break;
    }
    return title;
}
- (NSArray*)sectionIndexTitlesForTableView:(UITableView *)tableView {
    return[NSArray arrayWithObjects:@"D", @"I", @"L", @"R", @"S", nil];
}
- (NSInteger)
tableView:(UITableView *)tableView
sectionForSectionIndexTitle:(NSString *)title
  atIndex:(NSInteger)index {
    return index;
}
@end
```

На рис. 8.20 изображено приложение с индексированным табличным представлением.

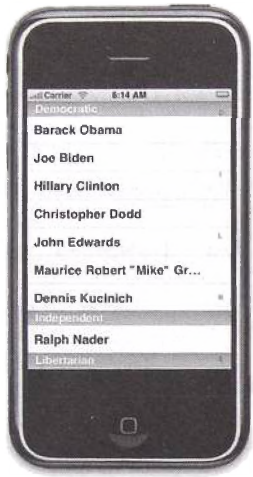

Рис. 8.20. Приложение с индексированным табличным представлением

На рис. 8.21 вы видите табличное представление после нажатия пользователем буквы индекса `S`. Табличное представление прокручивается вниз, делая соответствующую секцию видимой.

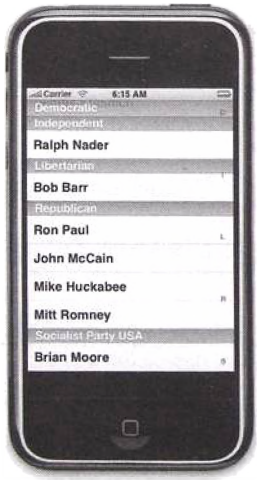

Рис. 8.21. Приложение с индексированным табличным представлением после нажатия индекса

8.11. Резюме

Эта глава ознакомила вас с табличными представлениями. Мы начали с обзора основных идей табличных представлений (разд. 8.1). В разд. 8.2 были описаны простое приложение с табличным представлением и обязательные методы, реализация которых необходима для накопления пользовательских взаимодействий с табличным представлением и реакции на них.

Из разд. 8.3 вы узнали, как добавлять изображения в строки таблицы. В разд. 8.4 были представлены концепция секций и приложение с табличным представлением, содержащее секции и верхние и нижние их колонтитулы.

В разд. 8.5 вы ознакомились с основными возможностями редактирования табличного представления. Вы изучили приложение, позволяющее пользователю удалять строки. В разд. 8.6 рассматривалась вставка новых строк в табличное представление и приложение, отображающее представление ввода данных и добавляющее эти новые данные в строки таблицы. В разд. 8.7 мы продолжили обсуждение режима редактирования и рассмотрели приложение для упорядочивания табличных записей.

В разд. 8.8 вы изучили механизм отображения иерархической информации пользователю и приложение, использующее табличные представления для отображения трех уровней иерархии. В разд. 8.9 на примере были рассмотрены сгруппированные табличные представления, а в разд. 8.10 — основные идеи, касающиеся табличных представлений.

Глава 9
Управление файлами

В этой главе рассматривается управление файлами. Вы научитесь использовать высоко- и низкоуровневые техники записи/чтения данных из файлов. Для выполнения высокоуровневых операций над файлами/директориями вы должны использовать экземпляры класса `NSFileManager`. Для демонстрации низкоуровневого доступа к файлу применяется класс `NSFileHandle`.

В разд. 9.1 рассматривается домашняя директория приложения. Из разд. 9.2 вы узнаете, как получать содержимое данной директории, используя высокоуровневые методы `NSFileManager`. Вы изучите структуру домашней директории и места возможного хранения файлов. Из разд. 9.3 вы узнаете, как создавать и удалять директории. Раздел 9.4 посвящен созданию файлов, а разд. 9.5 — теме атрибутов файлов и директорий (вы узнаете, как получать и устанавливать заданные атрибуты файлов/директорий). В разд. 9.6 мы рассмотрим использование упаковок приложений и низкоуровневого доступа к файлам. Итог главы будет подведен в разд. 9.7.

9.1. Домашняя директория

Приложение и его данные находятся внутри единственной директории, называемой домашней (Home). Ваша программа имеет доступ только к этой директории и ее содержимому. Абсолютный путь к домашней директории отличается от пути в реальном устройстве, однако организация и содержимое идентичны.

Для доступа к домашней директории вы можете использовать функцию `NSHomeDirectory()`, объявленную следующим образом:

```
NSString * NSHomeDirectory (void);
```

Эта функция возвращает объект `NSString`, содержащий абсолютный путь. Вот пример домашней директории на симуляторе:

```
/Users/ali/Library/
Application Support/iPhone Simulator/User/Applications/
F9CC3A4 9-997D-4523-9AFA-B553B5AE41EA
```

На реальном устройстве путь таков:

```
/var/mobile/Applications/F1C43BD0-1AB4-494B-B462-5A7315813D1A
```

В следующем разделе вы изучите структуру домашней директории и узнаете, где можно хранить файлы.

9.2. Поиск в директории

В этом разделе вы научитесь рекурсивному поиску по содержимому данной директории. Листинг 9.1 показывает функцию `main()` для поиска по содержимому домашней директории.

Функция начинает с логирования абсолютного пути к домашней директории. Вывод лога на симуляторе следующий:

```
Absolute path for Home Directory:/Users/ali/Library/Application Support/
iPhone Simulator/User/Applications/F9CC3A49-997D-4523-9AFA-B553B5AE41EA
```

На реальном устройстве он выглядит как

```
Absolute path for Home Directory:/var/mobile/Applications/ F1C43BD0-1AB4-
494B-B462-5A7315813D1A
```

После этого функция получает экземпляр класса `NSFileManager` для файловой системы по умолчанию, используя метод класса `defaultManager`. Используя этот экземпляр, можно производить высокоуровневые вызовы для манипуляции файлами и директориями внутри домашней.

Найти все файлы и директории внутри данной директории несложно. Используйте метод `enumeratorAtPath:`, объявленный следующим образом:

```
- (NSDirectoryEnumerator *) enumeratorAtPath:(NSString *)path
```

Вы передаете путь к директории, поиск по содержимому которой вы хотите осуществить, и получаете экземпляр класса `NSDirectoryEnumerator`. Каждый элемент в энумераторе директории является полным путем к элементу внутри ее. Все пути являются относительными по отношению к данной директории. Вы можете итеративно проходить по этому экземпляру, пропускать поддиректории и даже получать доступ к атрибутам файлов и директорий.

Этот экземпляр является подклассом `NSEnumerator`, поэтому для доступа к объектам используйте метод `nextObject`. Функция `main()` просто получает все объекты и выводит информацию о них в консоль. Ниже приведен лог, созданный на симуляторе (для уменьшения объема текста удалены временные отметки).

```
Found Documents
Found FileMgmt5.app
Found FileMgmt5.app/FileMgmt5
Found FileMgmt5.app/Info.plist
Found FileMgmt5.app/PkgInfo
```

```
Found Library
Found Library/Preferences
Found Library/Preferences/.GlobalPreferences.plist
Found Library/Preferences/com.apple.PeoplePicker.plist
Found tmp
```

Директория Documents (Документы) доступна для хранения данных приложения. Директория tmp используется для временных файлов. Две остальные директории, AppName.app (например, FileMgmt5.app) и Library (Библиотека), не должны быть доступны для запросов файловой системы. Вы можете создавать директории в домашней директории, tmp и Documents (Документы). Можете представить, что будет, если все содержимое домашней директории будет подвергнуто резервному копированию iTunes, исключая директорию tmp.

Листинг 9.1. Функция main(), получающая список содержимого домашней директории

```
#import  <Foundation/Foundation.h>
int main(int argc, char *argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    NSLog(@"Absolute path for Home Directory: %@", NSHomeDirectory());
    NSFileManager  *fileManager = [NSFileManager defaultManager];
    NSDirectoryEnumerator *dirEnumerator =
           [fileManager  enumeratorAtPath:NSHomeDirectory()] ;
    NSString *currPath;
    while (currPath = [dirEnumerator nextObject])
    {
      NSLog(@"Found %@", currPath);
    }
    [pool  release];
    return  0;
}
```

Экземпляр класса `NSDirectoryEnumerator` имеет несколько удобных методов:

- `directoryAttributes` — используйте этот метод для получения словаря атрибутов директории, внутри которой производится поиск (мы поговорим об атрибутах файлов и директорий подробнее в разд. 9.5);
- `fileAttributes` — метод предоставляет словарь атрибутов для текущего объекта поиска и работает для файлов и поддиректорий (подробнее об атрибутах файлов и директорий будет рассказано позже);
- `skipDescendents` — вызвав этот метод, вы можете пропустить директорию во время поиска, если вас не интересует ее содержимое.

9.3. Создание и удаление директории

В этом разделе вы научитесь создавать и удалять поддиректории в домашней директории. Листинг 9.2 демонстрирует функцию `main()`. Что-

бы создать директорию, используйте метод `createDirectoryAtPath:attributes:` экземпляра `NSFileManager`. Этот метод объявлен следующим образом:

```
- (BOOL)createDirectoryAtPath:(NSString *)path
attributes:(NSDictionary *)attributes
```

В качестве входных параметров этот метод принимает путь к создаваемой директории и ее атрибуты (последние будут описаны подробнее в следующем разделе). Чтобы создать директорию с атрибутами по умолчанию, необходимо передать значение `nil` в качестве второго параметра. Если директория была успешно создана, метод возвращает `YES`, в противном случае — `NO`.

Когда директория создана, удаляем ее. Метод удаления файла или директории — `removeItemAtPath:error:`, объявленный следующим образом:

```
- (BOOL)removeItemAtPath:(NSString *)path error:(NSError **)error
```

Методу передается путь к элементу (директории, файлу или ссылке), который нужно удалить, и ссылка на объект `NSError`. Вы можете передать NULL в качестве второго параметра, если не хотите знать, что могло вызвать ошибку (возврат `NO`).

Листинг 9.2. Функция main(), демонстрирующая создание и удаление директорий

```
#import <Foundation/Foundation.h>
int main(int argc, char *argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    NSError *error;
    NSFileManager *fileManager = [NSFileManager defaultManager];
    NSString *newDirPath =
      [NSHomeDirectory() stringByAppendingPathComponent:@"tmp/directory"];
    BOOL success =
      [fileManager createDirectoryAtPath:newDirPath attributes:nil];
    if(success == YES){
      NSLog(@"Directory %@ created successfully!", newDirPath);
      success = [fileManager removeItemAtPath:newDirPath error:&error];
      if (success == YES) {
       NSLog(@"Directory %@ deleted successfully!", newDirPath);
      }
      else{
       NSLog(@"Error deleting directory %@. %@",
            newDirPath[error localizedDescription]);
       return -1;
      }
    }
    else{
      NSLog(@"Error creating directory %@.", newDirPath);
      return -1;
    }
    [pool release];
    return 0;
}
```

Вывод в консоль, сгенерированный на симуляторе, выглядит так:

```
Directory/Users/ali/Library/Application Support/iPhone Simulator/User/
Applications/BCE1C2BE-FAF0-47C2-A689-C20F630604E2/tmp/directory created
successfully!
Directory/Users/ali/Library/Application Support/iPhone Simulator/User/
Applications/BCE1C2BE-FAF0-47C2-A689-C20F630604E2/tmp/directory deleted
successfully!
```

Вывод в консоль, созданный на реальном устройстве, следующий:

```
Directory/var/mobile/Applications/2E723F14-B89B-450B-81BF-6385EFF76D05/tmp/
directory created successfully!
Directory/var/mobile/Applications/2E723F14-B89B-450B-81BF-6385EFF76D05/tmp/
directory deleted successfully!
```

9.4. Создание файлов

В этом разделе мы рассмотрим создание файлов в домашней директории. Чтобы сделать процесс интереснее, загрузим страницу из Интернета, используя протокол HTTP, и сохраним этот HTML-файл в директории tmp. Затем мы используем веб-представление для загрузки HTML-файла из директории tmp и отображения его пользователю. Как вы увидите далее, эти задачи можно без труда выполнить путем использования богатого API.

Листинг 9.3 демонстрирует объявление класса делегата приложения, использованного в нашем примере. Класс подобен изученным вами в главе, посвященной контроллерам представления.

Листинг 9.3. Объявление класса делегата приложения, используемого в примере создания и просмотра локальных файлов

```
#import <UIKit/UIKit.h>

@class MainViewController;
@interface FileAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow  *window;
    MainViewController  *mainCtrl;
}

@property(nonatomic, retain) UIWindow *window;
@end
```

Реализация делегата приложения показана в листинге 9.4. Делегат использует `MainViewController` в качестве дочернего представления главного окна.

Листинг 9.4. Реализация класса делегата приложения, используемого в примере создания и просмотра локальных файлов

```
#import  "FileAppDelegate.h"
#import  "MainViewController.h"
```

```
@implementation FileAppDelegate
@synthesize window;

- (void)applicationDidFinishLaunching:
    (UIApplication *)application {
    window = [[UIWindow alloc] initWithFrame:
            [[UIScreen mainScreen] bounds]];
    mainCtrl = [[MainViewController alloc]
            initWithNibName:nil bundle:nil];
    [window addSubview:mainCtrl.view];
    [window makeKeyAndVisible];
}

- (void)dealloc {
    [window release];
    [mainCtrl release];
    [super dealloc];
}
@end
```

В листинге 9.5 объявлен класс `MainViewController`. Он содержит ссылку на экземпляр `UIWebView`, который будет использоваться для визуализации содержимого локального файла в директории `tmp`. В дополнение он объявляет два метода для создания и визуализации HTML-файла.

Листинг 9.5. Объявление класса MainViewController, используемого в примере создания и просмотра локальных файлов

```
#import <UIKit/UIKit.h>

@interface MainViewController : UIViewController {
    UIWebView *webView;
}
-(void) createAFileInTMP;
-(void) loadWebViewWithFileInTMP;
@end
```

Листинг 9.6 показывает реализацию класса `MainViewController`. Метод `loadView` создает объект веб-представления и дает ему возможность реагировать на масштабирующие жесты. Объект веб-представления создается в качестве представления, управляемого контроллером. Таким образом, оно добавляется в качестве дочернего к главному окну.

Метод `viewDidLoad` вызывается сразу после загрузки представления. Он создает файл посредством вызова метода `createAFileInTMP`, после чего метод загружает веб-представление со скачанным файлом с помощью `loadWebViewWithFileInTMP`.

Листинг 9.6. Реализация класса MainViewController, используемого в примере создания и просмотра локальных файлов

```
#import <Foundation/Foundation.h>
#import <UIKit/UIKit.h>
#import "MainViewController.h"
```

```
@implementation MainViewController
- (void)loadView {
    CGRect rectFrame = [UIScreen mainScreen].applicationFrame;
    webView = [[UIWebView alloc] initWithFrame:rectFrame];
    webView.scalesPageToFit = YES;
    self.view = webView;
}
- (void)viewDidLoad {
  [self createAFileInTMP];
  [self loadWebViewWithFileInTMP];
}
-(void) loadWebViewWithFileInTMP {
    NSFileManager *fileManager = [NSFileManager defaultManager];
    NSData *data;
    NSString *fileName =
            [NSHomeDirectory() stringByAppendingPathComponent:@"tmp/file.html"];
    data = [fileManager contentsAtPath:fileName];
    [webView loadData:data MIMEType:@"text/html"
            textEncodingName:@"UTF-8"
            baseURL:[NSURL URLWithString:@"http://csmonitor.com"]];
}
-(void) createAFileInTMP{
    // Создание файла в директории tmp
    //http://www.csmonitor.com/textedition/index.html
    NSFileManager *fileManager = [NSFileManager defaultManager];
    NSString *fileName =
            [NSHomeDirectory() stringByAppendingPathComponent:@"tmp/file.html"];
    NSURL *theURL = [[NSURL alloc] initWithScheme:@"http"
            host:@"www.csmonitor.com"
            path:@"/textedition/index.html"];
    NSData *data = [[NSData alloc] initWithContentsOfURL:theURL];
    BOOL fileCreationSuccess =
            [fileManager createFileAtPath:fileName
            contents:data attributes:nil];
    if(fileCreationSuccess == NO) {
            NSLog(@"Failed to create the html file");
    }
    [theURL release];
    [data release];
}
- (void)dealloc {
    [webView release];
    [super dealloc];
}
@end
```

Метод `createAFileInTMP` создает объект `NSURL`, указывающий на URL-адрес http://www.csmonitor.com/textedition/index.html. Затем метод создает объект `NSData`, хранящий содержимое файла `index.html`, скачанного с сервера. Чтобы создать файл в локальной файловой системе, мы используем метод `createFileAtPath:contents:attributes:`, объявленный следующим образом:

```
- (BOOL) createFileAtPath:(NSString *)path
    contents:(NSData *)data attributes:(NSDictionary *)attr
```

В качестве первого параметра ему передается путь к файлу, в качестве второго — данные, а третьего — атрибуты. Здесь мы используем атрибуты по умолчанию и передаем `nil`. Используемый путь — абсолютный путь к домашней

директории, в конце которого прибавлено `tmp/file.html`. Если при создании файла возникла проблема, возвращается `NO`, в противном случае — `YES`.

Метод `loadWebViewWithFileInTMP` загружает локальный HTML-файл и отображает его посредством веб-представления. Метод начинает с создания объекта `NSData` и загрузки в него содержимого локального файла, используя метод `contentAtPath:` экземпляра `NSFileManager`. Затем мы загружаем в объект веб-представления содержимое объекта `NSData`.

Ниже приведен пример создания и визуализации файла (рис. 9.1).

Рис. 9.1. Снимок экрана примера создания и визуализации файла

9.5. Считывание и изменение атрибутов

До сих пор мы передавали значение `nil` для словаря атрибутов файлов и директорий. Однако вы можете задать словарь, содержащий атрибуты, отличные от заданных по умолчанию. Более того, вы можете изменять атрибуты объекта файловой системы и после его создания.

Рассмотрим пример считывания/установки атрибуты файла. Листинг 9.7 показывает функцию `main()` программы. Она начинает с создания файла `file.txt` в директории `tmp`, содержащего одну строку текста. Сначала мы получаем из строки объект `NSData`, используя метод `dataUsingEncoding:` экземпляра `NSString` в кодировке `utf-8`. Затем мы создаем файл в файловой системе посредством уже знакомого вам

метода createFileAtPath:contents:attributes:. При создании файла мы используем атрибуты по умолчанию. Создав файл с атрибутами по умолчанию, мы хотели бы считать его атрибуты и посмотреть доступные ключи и их значения. Для считывания атрибутов файла мы используем метод filesAttributesAtPath:traverseLink: экземпляра NSFileManager, который объявлен следующим образом:

```
- (NSDictionary *)fileAttributesAtPath:(NSString *)path traverseLink:(BOOL)flag
```

В качестве первого параметра вы передаете путь к файлу. Если путь указывает на символьный ярлык, можете указать YES для того, чтобы отследить источник ярлыка, либо NO, чтобы вернуть атрибуты самого ярлыка. В случае успеха метод возвращает экземпляр NSDictionary, при неудаче — nil. Переменная attributes используется для сохранения возвращенных значений. Если значение attributes не равно nil, мы выводим в консоль единственный атрибут файла — его размер в байтах. Ключом полученного значения является NSFileSize. Ниже приведены консольный вывод и содержимое объекта attributes на симуляторе сразу после считывания атрибутов файла.

```
2008-08-01 08:12:06.996 FileMgmt4[394:20b] File size is 22
(gdb) po attributes
{
    NSFileCreationDate = 2008-08-01 08:11:49 -0500;
    NSFileExtensionHidden = 0;
    NSFileGroupOwnerAccountID = 20;
    NSFileGroupOwnerAccountName = staff;
    NSFileHFSCreatorCode = 0;
    NSFileHFSTypeCode = 0;
    NSFileModificationDate = 2008-08-01 08:11:49 -0500;
    NSFileOwnerAccountID = 501;
    NSFileOwnerAccountName = ali;
    NSFilePosixPermissions = 420;
    NSFileReferenceCount = 1;
    NSFileSize = 22;
    NSFileSystemFileNumber = 2436813;
    NSFileSystemNumber = 234881026;
    NSFileType = NSFileTypeRegular;
}
```

Чтобы изменить один или более атрибутов файла или директории, вы можете использовать метод setAttributes:ofItemAtPath:error: класса NSFileManager, объявленный следующим образом:

```
-(BOOL)setAttributes:(NSDictionary *)attributes
    ofItemAtPath:(NSString *)path error:(NSError **)error;
```

В качестве первого параметра вы передаете словарь, содержащий один или несколько атрибутов элемента, которые хотите установить, в качестве второго — путь к элементу, а третьего — ссылку на объект NSError.

Ниже приведены доступные ключи атрибутов, относящиеся к файлам и директориям:

- NSFileBusy — используйте этот ключ для указания, занят файл или нет; значение — NSNumber булевого типа;

- NSFileCreationDate — ключ для установки даты создания файла/директории; значение — объект NSDate;
- NSFileExtensionHidden — для указания, является ли расширение файла скрытым; значение — NSNumber булевого типа (ниже приведен пример, показывающий, как устанавливать этот атрибут);
- NSFileGroupOwnerAccountID — ключ для указания группового идентификатора файла; значение задается с помощью объекта NSNumber, содержащего число типа unsigned long;
- NSFileGroupOwnerAccountName — используйте этот ключ для определения имени группы, к которой принадлежит владелец файла; значение — объект типа NSString;
- NSFileHFSCreatorCode — ключ для указания HFS-кода создателя; значение задается с помощью объекта NSNumber, содержащего число типа unsigned long;
- NSFileHFSTypeCode — используйте этот ключ для указания HFS-кода типа; значение определяется посредством объекта NSNumber, содержащего число типа unsigned long;
- NSFileImmutable — ключ для указания, является ли файл изменяемый или нет; значение — NSNumber булевого типа;
- NSFileModificationDate —для определения даты последнего изменения файла; значение этого ключа — объект NSDate;
- NSFileOwnerAccountID — используйте этот ключ для указания идентификатора владельца файла; значение задается с помощью объекта NSNumber, содержащего число типа unsigned long;
- NSFileOwnerAccountName — для указания имени владельца файла; значение для этого ключа — объект типа NSString;
- NSFilePosixPermissions — используйте этот ключ для указания POSIX-разрешений файла; значение задается посредством объекта NSNumber, содержащего число типа unsigned long.

Листинг 9.7. Пример получения/установки атрибутов файла

```
#import <Foundation/Foundation.h>
int main(int argc, char *argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    BOOL success;
    NSFileManager *fileManager = [NSFileManager defaultManager];
    NSString *filePath =
          [NSHomeDirectory() stringByAppendingPathComponent:@"tmp/file.txt"];
    NSData *data = [@"Hello! This is a line."
          dataUsingEncoding:NSUTF8StringEncoding];
    success =
          [fileManager createFileAtPath:filePath
          contents:data attributes:nil];
    if(success == NO) {
      NSLog(@"Error creating file");
```

```
        return -1;
    }
    NSDictionary *attributes =
            [fileManager fileAttributesAtPath:filePath traverseLink:NO];
    if(attributes) {
            NSNumber *fSize = [attributes objectForKey:NSFileSize];
            NSLog(@"File size is %qi", [fSize longLongValue]);
    }
    NSDictionary *newAttributes;
    NSError *error;
    newAttributes =
            [NSDictionary dictionaryWithObject:
            [NSNumber numberWithBool:YES]
            forKey:NSFileExtensionHidden];
    success = [fileManager setAttributes:newAttributes
                    ofItemAtPath:filePath error:&error];
    if(success == NO) {
      NSLog(@"Error setting attributes of file. Error: %@",
            [error localizedDescription]);
      return -1;
    }
    attributes =
            [fileManager fileAttributesAtPath:filePath traverseLink:NO];
    [pool release];
    return 0;
}
```

После изменения `NSFileExtensionHidden` на `YES` объект `attributes` на симуляторе будет представлен следующим образом:

```
(gdb) po attributes {
    NSFileCreationDate = 2008-08-01 08:11:49 -0500;
    NSFileExtensionHidden = 1;
    NSFileGroupOwnerAccountID = 20;
    NSFileGroupOwnerAccountName = staff;
    NSFileHFSCreatorCode = 0;
    NSFileHFSTypeCode = 0;
    NSFileModificationDate = 2008-08-01 08:11:49 -0500;
    NSFileOwnerAccountID = 501;
    NSFileOwnerAccountName = ali;
    NSFilePosixPermissions = 420;
    NSFileReferenceCount = 1;
    NSFileSize = 22;
    NSFileSystemFileNumber = 2436813;
    NSFileSystemNumber = 234881026;
    NSFileType = NSFileTypeRegular;
}
```

Объект `attributes` на реальном устройстве будет выглядеть иначе. Мы отметили несколько изменений, таких как `NSFileGroupOwnerAccountName` и `NSFileOwnerAccountID`. Несмотря на то что атрибут `NSFileExtensionHidden` был успешно изменен системным вызовом, ключ `NSFileExtensionHidden` вообще не появился в объекте `attributes`. Этот пример служит напоминанием, что следует всегда тестировать код на реальном устройстве. Ниже приведены все атрибуты файла, доступные на устройстве:

```
2008-08-01 08:17:39.982 FileMgmt4[164:20b] File size is 22
(gdb) po attributes
{
```

```
    NSFileGroupOwnerAccountID = 501;
    NSFileGroupOwnerAccountName = mobile;
    NSFileModificationDate = 2008-08-01 08:17:35 -0500;
    NSFileOwnerAccountID = 501;
    NSFileOwnerAccountName = mobile;
    NSFilePosixPermissions = 420;
    NSFileReferenceCount = 1;
    NSFileSize = 22;
    NSFileSystemFileNumber = 87161;
    NSFileSystemNumber = 234881026;
    NSFileType = NSFileTypeRegular;
}
```

9.6. Работа с ресурсами и низкоуровневый доступ к файлам

В этом разделе мы рассмотрим две темы.

1. Использование упаковок — как получать доступ к файлам, созданным во время упаковки приложения.

2. Низкоуровневый доступ к файлу — как искать и обновлять файлы.

Наше приложение сохраняет текстовый файл в упаковке программы, как показано на снимке экрана меню Groups and Files (Группы и файлы) оболочки XCode (рис. 9.2). Вы можете сохранять файлы в любом месте, но обычно это делается в директории Resources (Ресурсы).

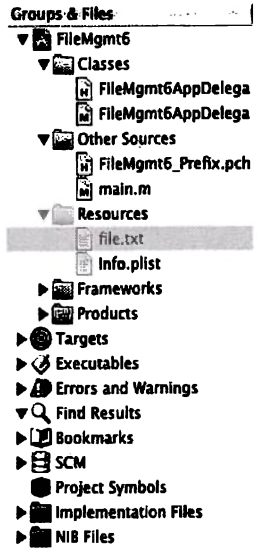

Рис. 9.2. Снимок экрана меню Groups and Files (Группы и файлы) оболочки XCode

Листинг 9.8 показывает функцию `main()`, демонстрирующую загрузку файла из упаковки и его модификацию путем вставки текста. Как вы уже знаете, внутри домашней директории каждого приложения существует поддиректория XXX.app (где XXX — имя приложения). Внутри этой поддиректории сохраняются файлы с данными.

Чтобы облегчить поиск файлов данных внутри упаковки, существует метод экземпляра класса `NSBundle`, который можно использовать для поиска определенного файла с заданным расширением и возврата абсолютного пути к этому ресурсу. Метод объявлен следующим образом:

```
- (NSString *)pathForResource:(NSString *)name ofType:(NSString *)ext;
```

В качестве первого параметра передается путь к файлу ресурса, который вы хотите найти, а в качестве второго — его расширение. В качестве расширения вы можете передать пустую строку или даже `nil`, если имя вашего файла уникально в пределах упаковки. Причина, по которой это срабатывает, кроется в том, что алгоритм поиска возвращает первое появление файла с заданным именем, если параметр `ext` пуст или равен `nil`. Местонахождение файла `file.txt` в упаковке (значение `filePath` в функции `main()`) следующее:

```
/var/mobile/Applications/5ABEB448-7634-4AE8-9833-FC846A81B418/FileMgmt6.app
/file.txt
```

Помните: вы не должны изменять элементы в APP-директории, так как это повлияет на код, поэтому, чтобы изменить файл в упаковке, вам потребуется скопировать его в другую директорию, где и изменять его.

После локализации файла в упаковке и сохранения абсолютного пути к нему в `filePath` мы загружаем его содержимое в объект `NSData` посредством метода `dataWithContentsOfFile:`. Далее создается файл `Documents/fileNew.txt`, содержащий информацию из файла в упаковке.

Исходный файл содержит одну строку — `This is the contents of a file`. Мы бы хотели модифицировать скопированный файл, заменив текст `contents` на `modified`, и выполнить эту задачу посредством низкоуровневых операций, включающих поиск по файлу вместо его загрузки целиком в память, изменение и сохранение файла обратно на диск.

Для низкоуровневых операций с файлами вам требуется получить экземпляр `NSFileHandle`. Этот класс инкапсулирует низкоуровневый механизм доступа к файлам. Природа операций, которые вы хотите произвести с файлом, определяет метод, который вы будете использовать для получения экземпляра `NSFileHandle`. Ниже приведены три доступных метода класса `NSFileHandle`:

- **чтение** — чтобы получить экземпляр только для чтения, используйте метод класса `fileHandleForReadingAtPath:`, объявленный следующим образом:

```
+ (id)fileHandleForReadingAtPath:(NSString *)path
```

- **запись** — для получения экземпляра только для записи примените метод класса `fileHandleForWritingAtPath:`, объявленный как

```
+ (id)fileHandleForWritingAtPath:(NSString *)path
```

- **чтение/запись** — чтобы получить экземпляр для добавления, используйте метод класса `fileHandleForUpdatingAtPath:`, объявленный следующим образом:

```
+ (id)fileHandleForUpdatingAtPath:(NSString *)path
```

Когда вы получаете экземпляр, используя один из этих трех приведенных выше методов, указатель устанавливается на начало файла.

В нашем примере мы открываем файл для добавления. Мы знаем местоположение текста, который требуется вставить, поэтому используем операцию перемещения по экземпляру `NSFileHandle`. Для перемещения по файлу используйте метод `seekToFileOffset:`, объявленный следующим образом:

```
-(void)seekToFileOffset:(unsigned long long)offset
```

Место, куда требуется переместиться, в нашем примере равно 11. После перемещения туда мы записываем в файл текст `modified`, используя метод `writeData:`. Этот метод объявлен следующим образом:

```
-(void)writeData:(NSData *)data
```

По окончании модификации файла мы закрываем объект `NSFileHandle` с помощью метода `closeFile`.

Листинг 9.8. Функция main(), демонстрирующая загрузку файла из упаковки и его модификацию путем записи текста

```
#import <Foundation/Foundation.h>
int main(int argc, char *argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    BOOL success;
    NSFileManager *fileManager = [NSFileManager defaultManager];
    NSString *filePath = [[NSBundle mainBundle] pathForResource:@"file"
          ofType:@"txt"];
    NSData *fileData = [NSData dataWithContentsOfFile:filePath];
    if (fileData) {
      NSString *newFilePath = [NSHomeDirectory()
          stringByAppendingPathComponent:@"Documents/fileNew.txt"];
      success = [fileManager createFileAtPath:newFilePath
          contents:fileData attributes:nil];
    if (success == NO) {
      NSLog(@"Error creating file");
      return -1;
    }
    NSFileHandle *fileHandle =
          [NSFileHandle fileHandleForUpdatingAtPath:newFilePath];
    if (fileHandle) {
          [fileHandle seekToFileOffset:11];
          NSData *appendedData =
```

```
            [@" modified " dataUsingEncoding:NSUTF8StringEncoding];
            [fileHandle writeData:appendedData];
            [fileHandle closeFile];
    }
    else {
            NSLog(@"Error modifying the file");
            return -1;
            }
    }
    else {
            NSLog(@"Could not load file from the app bundle");
            return -1;
    }
    [pool release];
    return 0;
}
```

9.7. Резюме

В этой главе была раскрыта тема управления файлами. Вы научились использовать высоко- и низкоуровневую технику для сохранения/считывания данных из файлов. Для совершения высокоуровневых операций над файлами/директориями использовались экземпляры класса `NSFileManager`. Класс `NSFileHandle` применялся для демонстрации низкоуровневого доступа к файлам.

В разд. 9.1 мы говорили о домашней директории приложения. В разд. 9.2 было показано, как получать содержимое данной директории посредством высокоуровневых методов `NSFileManager`. Вы подробнее изучили структуру домашней директории и места, в которых можно хранить файлы. Из разд. 9.3 вы узнали, как создавать и удалять директории. В разд. 9.4 вы изучали создание файлов, а в разд. 9.5 — атрибуты файлов и директорий. Из последнего вы также узнали, как считывать и устанавливать заданные атрибуты директорий и файлов.

В разд. 9.6 мы рассмотрели использование упаковок приложений и низкоуровневого доступа к файлу.

Глава 10
Работа с базами данных

Эта глава рассказывает об основах механизма баз данных SQLite, который доступен при использовании iPhone SDK. Эта база данных отличается от других известных вам. Базы данных типа Oracle и Sybase – серверные. В них сама база располагается на сервере, который обслуживает запросы клиентов, работающих на других машинах. База данных SQLite – встроенная, в том смысле, что сервер отсутствует, а сам механизм базы данных связан с вашими приложением. База SQLite – бесплатная.

Эта глава подробнее знакомит с основами баз данных (подразумевается, что вы знаете азы языка структурированных запросов (SQL)). Вы должны знать, что база данных состоит из набора *таблиц*, а каждая таблица имеет имя, уникально идентифицирующее ее в базе данных. Каждая таблица состоит из одного или более *полей*, а каждое поле имеет название, уникально идентифицирующее его в таблице. Строка – это вектор значений для каждого поля в данной таблице. Строка часто именуется *записью*.

Данная глава организована следующим образом. В разд. 10.1 мы рассмотрим основные SQL-выражения и их реализацию посредством вызова SQL-функций. В разд. 10.2 вы изучите обработку результирующих множеств, созданных SQL-выражениями. Тема разд. 10.3 – подготовленные выражения. В разд. 10.4 вы ознакомитесь с расширениями SQLite API на примере использования пользовательских функций. В разд. 10.5 и 10.6 будут представлены подробные примеры сохранения и считывания BLOB-полей из базы данных. Итоги главы будут подведены в разд. 10.7.

10.1. Основные операции с базой данных

В этом разделе мы поговорим о некоторых основных SQL-выражениях и их реализации в SQLite. Мы рассмотрим простую программу, которая создает базу данных с одной таблицей. Эта таблица будет хранить записи о биржевых сделках. Каждая запись будет содержать идентификатор сделки (представленный символом сделки), цену покупки, количество купленного товара и дату покупки. Листинг 10.1 представляет функцию `main()`.

Листинг 10.1. Функция main(), демонстрирующая основные SQL-выражения, используя библиотеку вызовов функций SQLite; функция создает базу данных (если та не существует), добавляет новую таблицу и заполняет таблицу записями

```
#import "/usr/include/sqlite3.h"
int main(int argc, char *argv[]) {
    char  *sqlStatement;
    sqlite3  *pDb;
    char  *errorMsg;
    int  returnCode;
```

```
    char  *databaseName;
    databaseName = "financial.db";
    returnCode = sqlite3_open(databaseName, &pDb);
    if(returnCode!=SQLITE_OK) {
      fprintf(stderr, "Error in opening the database. Error: %s",
            sqlite3_errmsg(pDb));
            sqlite3_close(pDb);
            return -1;
    }
    sqlStatement =  "DROP TABLE IF EXISTS  stocks";
    returnCode = sqlite3_exec(pDb, sqlStatement, NULL, NULL, &errorMsg);
    if(returnCode!=SQLITE_OK) {
            fprintf(stderr,
                  "Error in dropping table stocks. Error: %s", errorMsg);
            sqlite3_free(errorMsg);
    }
    sqlStatement = "CREATE TABLE stocks (symbol VARCHAR(5), " "purchasePrice
    FLOAT(10,4), " "unitsPurchased INTEGER, " "purchase_date VARCHAR(10))";
    returnCode = sqlite3_exec(pDb, sqlStatement, NULL, NULL, &errorMsg);
    if(returnCode!=SQLITE_OK) {
            fprintf(stderr,
            "Error in creating the stocks table. Error: %s", errorMsg);
            sqlite3_free(errorMsg);
    }
    insertStockPurchase(pDb, "ALU", 14.23, 100, "03-17-2007");
    insertStockPurchase(pDb, "GOOG", 600.77, 20, "01-09-2007");
    insertStockPurchase(pDb, "NT", 20.23,140, "02-05-2007");
    insertStockPurchase(pDb, "MSFT", 30.23, 5, "01-03-2007");
    sqlite3_close(pDb);
    return 0;
}
```

Перед тем как начать работу с базой данных, откройте ее. Функция SQLite для открытия базы данных — `sqlite3_open()`. Она задается как

```
int sqlite3_open (
    const char *filename, /* Имя файла базы данных (UTF-8) */
    sqlite3 **ppDb /* OUT: Обработчик базы данных */
);
```

В SQLite база данных хранится в файле. Чтобы открыть базу данных, нужно задать имя ее файла в первом параметре `filename`. При успешном открытии функция вернет значение `SQLITE_OK`. Для остальных функций SQLite, работающих с базой данных, требуется дескриптор. Вы передаете ссылку на указатель дескриптора во втором параметре. Если база данных была успешно открыта, дескриптор записывается по этому адресу. Тип дескриптора соединения с базой данных — `sqlite3`. Вы передаете адрес переменной типа `sqlite3*` во втором параметре. Ничего страшного, если базы данных не существует, — она будет создана. Таким образом, эта функция используется как для открытия существующей базы данных, так и создания новой.

Если попытка открыть базу данных была безуспешной, нужно вывести сообщение об ошибке и закрыть базу данных. Функции SQLite `sqlite_errmsg()` передается указатель на дескриптор базы данных, а та возвращает информационную строку, описывающую ошибку. Приведенная выше программа использует эту функцию для отображения сообщения об ошибке при неудачном открытии базы данных. Когда вы закончили работу с базой данных, нужно закрыть ее, для чего используется функция SQLite `sqlite3_close()`. В качестве единственного параметра ей передается указатель на дескриптор открытой базы данных (`sqlite3 *`), полученный в момент открытия.

Успешно открыв базу данных, мы хотели бы произвести несколько операций над таблицами. SQLite предоставляет функцию-помощника, которая выполняет вычисление SQL-выражений за один раз. Это функция `sqlite3_exec()`, которая проста в использовании и хорошо работает со множеством SQL-выражений. Позже вы узнаете, как реализовать эту функцию с использованием других функций SQLite. Функция `sqlite3_exec()` объявлена следующим образом:

```
int sqlite3_exec(
    sqlite3*,   /* Открытая база данных */
    const char *sql,   /* SQL запрос */
    int (*callback)(void*,int,char**,char**),   /*Callback-функция*/
    void *,   /* 1-й аргумент callback-функции */
    char **errmsg   /* Сюда записываются сообщения об ошибке */

);
```

Первый параметр — это указатель на дескриптор базы данных, который мы получили от функции `sqlite3_open()`, а второй — C-строка, представляющая SQL-выражение. Если произошла ошибка, сообщение о ней записывается в область памяти, полученную от функции `sqlite3_malloc()`, а `*errmsg` должно указывать на это сообщение. Callback-функция, если она задана, будет вызываться для каждой строки результата. Мы рассмотрим callback-функции позже, а сейчас обратите внимание, что первый параметр, передаваемый этой функции, будет задаваться в четвертом параметре функции `sqlite3_exec()`. Возвращаемое значение `SQLITE_OK` свидетельствует об успешном исполнении SQL-выражения.

Первое, что мы сделаем в функции `main()`, — удалим таблицу `stocks`, если она существует. SQL-выражение для этого следующее:

```
DROP TABLE IF EXISTS  stocks
```

Это SQL-выражение не возвращает записи, поэтому при вызове `sqlite3_exec()` мы передаем `NULL` как для callback-функции, так и для ее первого параметра. Это SQL-выражение выполняется так:

```
returnCode = sqlite3_exec(pDb, sqlStatement, NULL, NULL, &errorMsg);
```

Удалив таблицу `stocks`, мы можем продолжить и создать новую. SQL-выражение для создания таблицы `stocks` следующее:

```
CREATE TABLE stocks (
    symbol VARCHAR(5), purchasePrice FLOAT(10,4),
    unitsPurchased INTEGER, purchase_date VARCHAR(10) )
```

Вам знакомо это SQL-выражение. Оно означает, что в таблице `stocks` должно быть четыре поля. Первое поле — это набор (максимум пять) символов. Второе поле типа `float` содержит 10 разрядов, четыре из которых — десятичные. Третье поле — типа `integer`, а четвертое, последнее, состоит из набора максимум 10 символов.

SQLite имеет следующие пять классов для хранения данных.

- INTEGER — используется для хранения знаковых целочисленных значений. Количество байтов, реально используемых для хранения, зависит от величины значения и изменяется от одного до восьми.
- REAL — восьмибайтовое IEEE-хранилище с плавающей точкой, представляющее соответствующее число.
- TEXT — область хранения для текста. Текст может быть в одной из следующих кодировок: UTF-8, UTF-16BE или UTF-16-LE.
- BLOB — используется для хранения данных в том виде, в каком они были введены, например, изображения.
- NULL — используется для хранения значения NULL.

После создания таблицы `stocks` мы заносим в нее некоторые значения с помощью функции `insertStockPurchase()`, показанной в листинге 10.2. Например, следующее SQL-выражение добавляет запись о продаже 17.03.2007 100 акций Alcatel-Lucent на сумму $14,23:

```
INSERT INTO stocks VALUES ('ALU', 14.23, 100, '03-17-2007')
```

Мы используем SQLite-функцию `sqlite3_mprintf()` для форматированного вывода строки. Эта функция схожа со стандартной `printf()` библиотеки C за тем исключением, что она записывает результат в область памяти, полученную с помощью функции `sqlite3_malloc()`, так что по завершении вы должны высвободить строку, используя функцию `sqlite3_free()`. В дополнение к известным параметрам форматирования доступны параметры `%q` и `%Q`. Вы должны использовать эти параметры вместо `%s` при работе с текстом. Параметр `%q` работает так же, как и `%s` за тем исключением, что он дублирует каждый символ `'`. Например, строка `She said: 'Hey Ya'all whats up?'` будет отформатирована в строку `She said: "Hey Ya'all whats up?"`. Параметр `%Q` работает так же, как и `%q`, только печатает строку NULL, если значение указателя равно NULL. Он также окружает всю строку парой `'` — при использовании `%Q` строка будет напечатана как `'She said: "Hey Ya'all whats up?"'`.

Листинг 10.2. Функция insertStockPurchase() для добавления записей в таблицу stocks

```
#import "/usr/include/sqlite3.h"
void insertStockPurchase(sqlite3 *pDb, const char*symbol, float price,
    int units, const char* theDate){
    char  *errorMsg;
    int  returnCode;
    char  *st;
    st  = sqlite3_mprintf("INSERT INTO stocks VALUES"
          " ('%q', %f, %d, '%q')", symbol, price, units, theDate);
    returnCode = sqlite3_exec(pDb, st, NULL, NULL, &errorMsg);
    if(returnCode!=SQLITE_OK) {
          fprintf(stderr, "Error in inserting into the stocks table.
                Error: %s", errorMsg);
          sqlite3_free(errorMsg);
    }
    sqlite3_free(st);
}
```

10.2. Обработка результирующих строк

В предыдущем разделе вы увидели, как можно использовать функцию `sqlite3_exec()` для выполнения SQL-выражений, которые не возвращают результата либо вызывающий не заинтересован в его обработке.

Если вам нужно результирующее множество, можете передать указатель на callback-функцию в качестве четвертого параметра функции `sqlite3_exec()`. Эта функция будет вызываться для каждой строки в результирующем множестве.

Callback-функция должна соответствовать следующему образцу:

```
int (*callback)(void*,int,char**,char**)
```

Первый параметр этой функции идентичен четвертому параметру вызываемой функции `sqlite3_exec()`, второй параметр — это количество колонок в текущей строке результата, третий — массив указателей на строки, содержащие значения для каждой колонки текущей строки результирующего множества, а четвертый параметр — это массив указателей на строки, содержащие имена результирующих колонок. Если callback-функция возвращает значение, отличное от нуля, то функция `sqlite_exec()` прекратит выполнение и вернет `SQLITE_ABORT`.

В функции `main()`, показанной в листинге 10.3, вы видите, как можно использовать callback-функцию для обработки результирующего множества. Как вы видели ранее, открывается база данных `financial.db` и выполняется запрос SELECT. Запрос

```
SELECT * from stocks
```

возвращает все записи из таблицы `stocks`. Для выполнения запроса производится следующий вызов SQLite-функции:

```
returnCode = sqlite3_exec(pDb, sqlStatement, processRow, NULL, &errorMsg);
```

Третий параметр не равен `NULL`, как было в предыдущем разделе. Вместо этого мы передаем указатель на функцию `processRow`. Функция `processRow()` показана в листинге 10.4.

Листинг 10.3. Функция main() для извлечения записей с помощью функции sqlite3_exec()

```
#import "/usr/include/sqlite3.h"
int main(int argc, char *argv[]) {
    char    *sqlStatement;
    sqlite3 *pDb;
    char    *errorMsg;
    int     returnCode;
    char    *databaseName;

    databaseName = "financial.db";
```

```
    returnCode = sqlite3_open(databaseName, &pDb);
    if(returnCode!=SQLITE_OK) {
            fprintf(stderr, "Error in opening the database. Error: %s",
                    sqlite3_errmsg(pDb));
            sqlite3_close(pDb);
            return -1;
    }
    sqlStatement =  "SELECT * from stocks";
    returnCode = sqlite3_exec(pDb, sqlStatement,
            processRow, NULL, &errorMsg);
    if(returnCode!=SQLITE_OK) {
            fprintf(stderr, "Error in selecting from stocks table.
            Error: %s", errorMsg);
            sqlite3_free(errorMsg);
    }
    sqlite3_close(pDb);
    return 0;
}
```

Эта функция реализована по образцу callback-функции. Внутри ее есть цикл for, в котором мы отображаем заголовок колонки и ее значение.

Итогом выполнения данной программы будет

```
Record Data:
The value for Column Name symbol is equal to ALU
The value for Column Name purchasePrice is equal to 14.23
The value for Column Name unitsPurchased is equal to 100
The value for Column Name purchase_date is equal to 03-17-2007
Record Data:
The value for Column Name symbol is equal to GOOG
The value for Column Name purchasePrice is equal to 600.77002
The value for Column Name unitsPurchased is equal to 20
The value for Column Name purchase_date is equal to 01-09-2007
Record Data:
The value for Column Name symbol is equal to NT
The value for Column Name purchasePrice is equal to 20.23
The value for Column Name unitsPurchased is equal to 140
The value for Column Name purchase_date is equal to 02-05-2007
Record Data:
The value for Column Name symbol is equal to MSFT
The value for Column Name purchasePrice is equal to 30.23
The value for Column Name unitsPurchased is equal to 5
The value for Column Name purchase_date is equal to 01-03-2007
```

Листинг 10.4. Функция processRow() для обработки результирующих строк

```
#import "/usr/include/sqlite3.h"
static int processRow(void *argument, int argc, char **argv, char **colName){
    printf("Record Data:\n");
    for(int i=0; i<argc; i++){
            printf("The value for Column Name %s is equal to %s\n", colName[i],
                    argv[i] ? argv[i] : "NULL");
    }
    printf("\n");
    return 0;
}
```

10.3. Подготавливаемые выражения

В двух предыдущих разделах для выполнения SQL-выражений мы использовали функцию `sqlite3_exec()`. Она больше всего подходит для SQL-выражений, которые не возвращают данных (таких как `INSERT`, `DROP` и `CREATE`). Для SQL-выражений, возвращающих данные типа `SELECT`, обычно применяются подготавливаемые выражения.

Использование подготавливаемых выражений обычно включает три фазы.

1. Подготовка. На этом этапе вы предоставляете выражение механизму SQLite для компиляции. Механизм компилирует выражение в байт-код и резервирует ресурсы, необходимые для его выполнения.

2. Выполнение. Эта фаза используется для выполнения байт-кода и получения строк из результата выражения. Вы повторяете эту фазу для каждой строки в результирующем множестве.

3. Финализация. По получении всех строк результирующего множества вы финализируете подготавливаемое выражение, так что ресурсы, зарезервированные для него, можно высвободить.

Рассмотрим эти фазы подробнее.

10.3.1. Подготовка

Вы подготавливаете SQL-выражение, используя функцию `sqlite_prepare_v2()`. Функция объявлена следующим образом:

```
int sqlite3_prepare_v2(
    sqlite3 *db,  /* Дескриптор базы данных */
    const char *zSql,  /* SQL выражение в кодировке UTF-8 */
    int nBytes,  /* Длина выражения в байтах */
    sqlite3_stmt **ppStmt,  /* OUT: Дескриптор выражения */
    const char **pzTail  /*OUT: Указатель на неиспользованную
            порцию выражения*/
)
```

Первый параметр `db` — это указатель на дескриптор базы данных, полученный от предыдущего вызова `sqlite3_open()`. SQL-выражение (например, `SELECT`) передается в параметре `zSql`. В третьем параметре вы передаете длину выражения в байтах, а четвертый используется для получения дескриптора выражения. Вы передаете ссылку на переменную типа `sqlite3_stmt*`, и при успешной подготовке SQL-выражения эта переменная будет содержать дескриптор выражения. В случае если `*zSql` указывает на несколько SQL-выражений, после вызова функции `*pzTail` будет указывать на первый байт после первого SQL-выражения `zSql`. Если `*zSql` указывает на единственное SQL-выражение, то лучше передать `NULL` в качестве пятого параметра.

10.3.2. Выполнение

Как только вы скомпилировали SQL-выражение, нужно выполнить его и получить первую строку результата. SQL-выражение выполняется посредством вызова функции `sqlite3_step()`. Объявление функции следующее:

```
int sqlite3_step(sqlite3_stmt*);
```

В качестве единственного параметра функции передается указатель на дескриптор выражения. Если в результирующем множестве есть новая строка, функция возвращает `SQLITE_ROW`. Как только строки заканчиваются, функция возвращает `SQLITE_DONE`.

10.3.3. Финализация

По получении последней строки выражение финализируется посредством вызова функции `sqlite3_finalize()`, объявленной следующим образом:

```
int sqlite3_finalize(sqlite3_stmt *pStmt);
```

В качестве единственного параметра ей передается указатель на дескриптор выражения. Финализация закрывает выражение и высвобождает ресурсы.

Расссмотрим эти этапы на примере. Функция `main()` в листинге 10.5 — это место, где мы открываем базу данных, извлекаем несколько записей из таблицы и выводим одну за другой на печать.

Листинг 10.5. Функция main(), демонстрирующая подготавливаемые выражения

```
#import "/usr/include/sqlite3.h"
int main(int argc, char *argv[]) {
    char    *sqlStatement;
    sqlite3 *database;
    int     returnCode;
    char    *databaseName;
    sqlite3_stmt *statement;
    databaseName = "financial.db";
    returnCode = sqlite3_open(databaseName, &database);
    if(returnCode!=SQLITE_OK) {
            fprintf(stderr, "Error in opening the database. Error:
            %s", sqlite3_errmsg(database));
            sqlite3_close(database);
            return -1;
    }
    sqlStatement = sqlite3_mprintf("SELECT S.symbol, S.unitsPurchased, "
    "S.purchasePrice FROM stocks AS S WHERE " "S.purchasePrice  >= %f", 30.0);
    returnCode = sqlite3_prepare_v2(database,
            sqlStatement, strlen(sqlStatement), &statement, NULL);
    if(returnCode != SQLITE_OK) {
```

```
            fprintf(stderr, "Error in preparation of query. Error: %s",
            sqlite3_errmsg(database));
            sqlite3_close(database);
            return -1;
    }
    returnCode = sqlite3_step(statement);
    while(returnCode == SQLITE_ROW){
            char *symbol;
            int  units;
            double price;
            symbol = sqlite3_column_text(statement, 0);
            units = sqlite3_column_int(statement, 1);
            price = sqlite3_column_double(statement, 2);
            printf("We bought %d from %s at a price equal to %.4f\n",
                    units, symbol, price);
            returnCode = sqlite3_step(statement);
    }
    sqlite3_finalize(statement);
    sqlite3_free(sqlStatement);
    return 0;
}
```

После открытия базы данных мы вызываем функцию `sqlite3_prepare_v2()` для следующего SQL-выражения:

```
SELECT
    S.symbol, S.unitsPurchased, S.purchasePrice
    FROM stocks AS S
    WHERE S.purchasePrice >= 30.0
```

SQL-выражение вернет множество записей из таблицы `stocks`, где `purchasePrice` больше или равно $30. Выражение компилируется следующим образом:

```
returnCode = sqlite3_prepare_v2(database,
    sqlStatement, strlen(sqlStatement), &statement, NULL);
```

Обратите внимание, что мы передаем `NULL` в качестве последнего параметра, так как нам нужно скомпилировать только одно SQL-выражение. Если компиляция выражения прошла удачно, то код возврата будет равен `SQLITE_OK`. Если возникнет ошибка, мы выведем сообщение и покинем функцию `main()`.

После компиляции мы исполняем выражение, чтобы получить первую запись. Функция, используемая для выполнения выражения, — `sqlite3_step()`. Если запись успешно возвращается, код возврата будет `SQLITE_ROW`. Если мы получили код возврата `SQLITE_ROW`, то извлекаем значения для колонок этой строки. Чтобы извлечь значение для колонки, мы используем SQLite-функцию вида `sqlite3_column_XXX()`. Первый параметр этой функции — указатель на SQL-выражение (типа `sqlite3_stmt`), которое было возвращено функцией `sqlite3_prepare_v2()`. Второй параметр — это порядковый номер колонки, где самая левая колонка имеет номер 0. Возвращаемое значение зависит от версии функции.

У нас есть три следующих выражения, соответствующих трем колонкам:

```
symbol  = sqlite3_column_text(statement, 0);
units   = sqlite3_column_int(statement, 1);
price   = sqlite3_column_double(statement, 2);
```

Первый параметр соответствует колонке `S.symbol`. Колонка принадлежит к классу хранилищ `TEXT`. Функция `sqlite3_column_text()` вернет С-строку колонки `symbol`, которая хранится в этой строке. Функции `sqlite3_column_int()` и `sqlite3_column_double()` работают таким же образом за исключением того, что возвращают, соответственно, значения типа `integer` и `double`.

После распечатки значений колонок, составляющих строку, мы переходим к следующей строке результата повторным вызовом функции `sqlite3_step()`. Закончив обработку результата, мы покидаем цикл `while` и финализируем выражение посредством вызова функции `sqlite3_finalize()`. Результат вызова данного запроса с учетом содержимого таблицы `stocks`, наполненной в прошлых разделах, будет выглядеть следующим образом:

```
We bought 20 from GOOG at a price equal to 600.7700
We bought 5 from MSFT at a price equal to 30.2300
```

10.4. Пользовательские функции

Часто мы сталкиваемся с ситуацией, требующей использования функции, которая не была реализована SQL-механизмом. SQLite предоставляет механизм расширения С API и позволяет применение пользовательских функций. Пользователь может определить собственные функции для использования в SQL-выражениях для каждого конкретного соединения с базой данных. Такие функции — временные, в том смысле, что доступны только на время существования соединения с базой данных, а не хранятся в самой базе.

В этом разделе мы рассмотрим применение пользовательских функций посредством добавления функции `Palindrome()` к соединению с базой. Функции `Palindrome(t)` передается текстовый параметр `t`, который проверяется на то, читается ли он справа налево так же, как слева направо. Листинг 10.6 показывает функцию `main()`, демонстрирующую установку пользовательской функции для открытого соединения с базой данных.

Листинг 10.6. Функция main(), демонстрирующая установку пользовательской функции для открытого соединения с базой данных

```
int main(int argc, char *argv[]) {
    char    *sqlStatement;
    sqlite3 *database;
    int     returnCode;
```

```
    char    *databaseName;
    sqlite3_stmt *statement;
    databaseName = "financial.db";
    returnCode = sqlite3_open(databaseName, &database);
    if(returnCode!=SQLITE_OK) {
            fprintf(stderr, "Error in opening the database. Error: %s",
            sqlite3_errmsg(database));
            sqlite3_close(database);
            return -1;
    }
    sqlite3_create_function(database, "Palindrome", 1,
            SQLITE_UTF8, NULL, palindrome, NULL, NULL);
    sqlStatement = sqlite3_mprintf(
            "SELECT S.symbol, S.unitsPurchased, S.purchasePrice "
            "FROM stocks AS S WHERE "
            "Palindrome(S.symbol) = 1 AND S.purchasePrice  >= %f", 30.0);
    returnCode = sqlite3_prepare_v2(database, sqlStatement,
            strlen(sqlStatement), &statement, NULL);
    if(returnCode!=SQLITE_OK) {
            fprintf(stderr, "Error in preparation of query. Error: %s",
            sqlite3_errmsg(database));
            sqlite3_close(database);
            return -1;
    }
    returnCode = sqlite3_step(statement);
    while(returnCode == SQLITE_ROW){
            char *symbol;
            int  units;
            double price;
            symbol = sqlite3_column_text(statement, 0);
            units  = sqlite3_column_int(statement, 1);
            price  = sqlite3_column_double(statement, 2);
            printf("We bought %d from %s at a price equal to %.4f\n",
                    units, symbol, price);
            returnCode = sqlite3_step(statement);
    }
    sqlite3_finalize(statement);
    sqlite3_free(sqlStatement);
    return 0;
}
```

Пользовательская функция для данного соединения устанавливается посредством вызова функции `sqlite3_create_fuction()`. Функция объявлена следующим образом:

```
int sqlite3_create_function(
    sqlite3 *connectionHandle,
    const char *zFunctionName,
    int nArg,
    int eTextRep,
    void*,
    void (*xFunc)(sqlite3_context*,int,sqlite3_value**),
    void (*xStep)(sqlite3_context*,int,sqlite3_value**),
    void (*xFinal)(sqlite3_context*)

)
```

Первый параметр этой функции — дескриптор соединения (базы данных), второй — имя функции, какое оно будет в SQL-выражениях. Имя может отличаться от названия С-функции, на самом деле реализующей

необходимый функционал. Третий параметр используется для указания количества параметров, передаваемых вновь созданной функции, а четвертый используется для указания кодировки параметров. Вы можете устанавливать различные версии одной и той же функции с различными кодировками. Механизм SQLite сможет перенаправить вызовы к соответствующим реализациям функций. Пятый параметр — это случайный указатель. Внутри вашей функции вы можете получать доступ к данному указателю посредством использования `sqlite3_user_data()`. Седьмой параметр — это указатель на C-функцию, реализующую поведение функции, чье логическое имя указано в качестве второго параметра, `zFunctionName`. Позже мы поговорим об этом подробнее. Восьмой и девятый параметры содержат, соответственно, функции выполнения и финализации. Эти две функции используются для выполнения накапливаемых SQL-выражений.

Все пользовательские функции следуют образцу

```
void (sqlite3_context *context, int nargs, sqlite3_value  **values)
```

Функция не возвращает значения, и все три ее параметра — входные. Первый параметр — это контекст SQL-функции. Можете представить его в качестве ID-канала для функции и SQL-механизма, с которым она будет взаимодействовать. Второе передаваемое значение — это количество параметров, используемых при вызове логической функции из SQL-выражения, а третье — массив значений параметров, передаваемых функции.

Тип пользовательских функций — `void`, поэтому результат и ошибки возвращаются средствами механизма SQLite3. Чтобы вернуть вызывающему сообщение об ошибке, используйте функцию `sqlite3_result_error()`. Первый параметр этой функции — контекст (чтобы мехнизм знал, с каким SQL-выражением связана данная ошибка), второй — C-строка, представляющая сообщение об ошибке в текстовом виде, а третий — длина сообщения.

Используемое выражение `SELECT` сходно с тем, которое вы видели в предыдущем разделе за тем исключением, что символ идентификатора транзакции должен быть палиндромом. Выражение `SELECT` выглядит следующим образом:

```
SELECT
    S.symbol, S.unitsPurchased, S.purchasePrice
    FROM stocks AS S
    WHERE Palindrome(S.symbol) = 1 AND S.purchasePrice >= 30.0
```

Чтобы механизм SQLite выполнил данный запрос, для текущего соединения требуется определить функцию `Palindrome()`. Мы определяем функцию следующим выражением:

```
sqlite3_create_function(database, "Palindrome", 1,
    SQLITE_UTF8, NULL, palindrome, NULL, NULL);
```

Листинг 10.7 показывает реализацию функции `palindrome()`.

Листинг 10.7. Пользовательская функция palindrome() и ее реализация

```
#import "/usr/include/sqlite3.h"

int isPalindrome(char *text){
    unsigned char *p1, *p2;
    p1 = text;
    p2 = p1+strlen(text)-1;
    while (*p1==*p2 && (p1<=p2)){
            p1++;
            p2--;
    }
    if(p1>= p2)
            return 1;
    return 0;
}
void palindrome(sqlite3_context *context,
    int nargs, sqlite3_value **values){
    char  *errorMessage;
    if(nargs != 1){
            errorMessage =
                    "Incorrect no of arguments. palindrome(string)";
            sqlite3_result_error(context, errorMessage,
                    strlen(errorMessage));
            return;
    }
    if((sqlite3_value_type(values[0]) != SQLITE_TEXT)){
            errorMessage = "Argument must be of type text.";
            sqlite3_result_error(context, errorMessage,
                    strlen(errorMessage));
            return;
    }
    unsigned char *text;
    text = sqlite3_value_text(values[0]);
    sqlite3_result_int(context, isPalindrome(text));
}
```

Функция `palindrome()` сначала проверяет, равно ли количество параметров 1. Если нет, назад отсылается сообщение об ошибке и происходит возврат из функции. Функция также проверяет тип передаваемого параметра, так как мы ожидаем значения типа TEXT. Функция `sqlite3_value_type()` возвращает тип параметра. Функция объявлена следующим образом:

```
int sqlite3_value_type(sqlite3_value*)
```

Ей передается указатель на значение типа `sqlite3_value`, а возвращается один из пяти следующих типов: SQLITE_INTEGER, SQLITE_FLOAT, SQLITE_BLOB, SQLITE_NULL или SQLITE3_TEXT.

Удостоверившись, что тип параметра — TEXT, мы хотим получить реальное текстовое значение. Для этого используется SQLite-функция `sqlite3_value_text()`. Для других типов есть сходные функции (например, `sqlite3_value_int()`). Как только мы получили переданную строку, мы проверяем, является ли она палиндромом, с помощью функции `isPalindrome()`. Вы должны быть знакомы с этой функцией из начальных курсов компьютерных навыков.

Чтобы отослать результат обратно механизму SQLite, используйте функцию вида `sqlite3_result_XXX()`, которой передается контекст в качестве первого параметра и возвращаемое значение в качестве второго. Например, для возврата целочисленного результата мы используем функцию `sqlite3_result_int()` следующим образом:

```
sqlite3_result_int(context, isPalindrome(text))
```

10.5. Хранение BLOB-значений

В предыдущих разделах мы, в основном, работали с простыми типами данных (строками, целочисленными значениями и числами с плавающей точкой). Кроме данных скалярного и текстового типов, механизм баз данных SQLite поддерживат тип данных BLOB. Класс хранилища BLOB позволяет хранить бинарные данные (например, файлы с изображениями) в неизменном виде. В этом разделе мы рассмотрим механизм сохранения BLOB-значений, а в следующем — получение их из базы.

Чтобы объяснить основные методы вставки BLOB-значений в базу, создадим новую таблицу в базе, хранящую информацию о компаниях, в которые мы собираемся инвестировать. Кроме аббревиатуры компании и ее названия, мы создадим новую колонку типа BLOB, которая будет хранить логотип компании в PNG-формате.

Листинг 10.8 показывает функцию `main()`. Она создает новую таблицу `companies` с помощью следующего SQL-выражения:

```
CREATE TABLE  companies
    (symbol VARCHAR(5)  PRIMARY KEY, name VARCHAR(128), image BLOB)
```

Листинг 10.8. Функция main(), демонстрирующая сохранение BLOB-значений в таблице

```
#import "/usr/include/sqlite3.h"
    int main(int argc, char *argv[]) {
    char    *sqlStatement;
    sqlite3 *pDb;
    char    *errorMsg;
    int     returnCode;
    char    *databaseName;
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    databaseName = "financial.db";
    returnCode = sqlite3_open(databaseName, &pDb);
    if(returnCode!=SQLITE_OK) {
            fprintf(stderr, "Error in opening the database. Error: %s",
                    sqlite3_errmsg(pDb));
            sqlite3_close(pDb);
            return -1;
    }
    sqlStatement =  "DROP TABLE IF EXISTS  companies";
    returnCode = sqlite3_exec(pDb, sqlStatement, NULL, NULL, &errorMsg);
    if(returnCode!=SQLITE_OK) {
            fprintf(stderr, "Error in dropping table companies. Error: %s",
            errorMsg);
```

```
            sqlite3_free(errorMsg);
        }
        sqlStatement = "CREATE TABLE companies "
            "(symbol VARCHAR(5) PRIMARY KEY, " " name VARCHAR(128), image BLOB)";
        returnCode = sqlite3_exec(pDb, sqlStatement, NULL, NULL, &errorMsg);
        if(returnCode!=SQLITE_OK) {
            fprintf(stderr, "Error in creating the companies table. Error: %s",
                errorMsg);
            sqlite3_free(errorMsg);
            return -1;
        }
        insertCompany(pDb, "ALU", "Alcatel-Lucent");
        insertCompany(pDb, "GOOG", "Google");
        insertCompany(pDb, "MSFT", "Microsoft");
        insertCompany(pDb, "NT", "Nortel");
        sqlite3_close(pDb);
        [pool release];
        return 0;
}
```

После создания таблицы companies мы добавим четыре записи, вызвав функцию `insertCompany()`, показанную в листинге 10.9. Функция `insertCompany()` начинается с компиляции следующего выражения INSERT:

```
INSERT INTO companies VALUES (?, ?, ?)
```

Это выражение несколько отличается от использованного нами ранее. Этот тип выражения называется *параметризованным*. Оно использует ?, сигнализирующий, что значение будет указано позднее. Чтобы связать параметр с реальным значением, используйте одну из нескольких функций вида `sqlite3_bind_xxxx()`. Например, чтобы связать целочисленное значение, примените `sqlite3_bind_int()`. Ниже приведены основные функции связывания.

1. Связывание BLOB-значений. Связывающая функция для BLOB-значений объявлена следующим образом:

```
int sqlite3_bind_blob(sqlite3_stmt*, int, const void*, int n, void(*)(void*))
```

Первый параметр этой и следующих функций связывания является указателем на дескриптор выражения, полученного от функции подготовки выражения `sqlite3_prepare_v2()`. Второй параметр — это порядковый номер параметра SQL-выражения, с которым вы хотите связать значение. Обратите внимание, что отсчет порядковых номеров начинается с 1. Третий параметр — это количество байтов в BLOB-значении, а четвертый — указатель на функцию, которая будет вызвана после того, как механизм SQLite закончит выполнение выражения для высвобождения памяти, занятой BLOB-значением. У этого параметра есть два значения:

- `SQLITE_STATIC` — информирует механизм SQLite, что BLOB-значение является статическим и не нуждается в последующем освобождении памяти;
- `SQLITE_TRANSIENT` — сообщает механизму SQLite, что BLOB-значение является временным и нуждается в копировании; механизм

SQLite копирует BLOB-значение перед возвратом из связывающей функции.

2. Связывание текста. Связывающая функция для текста схожа с аналогичной для BLOB-значений:

```
int sqlite3_bind_text(sqlite3_stmt*, int, const char*,
    int n, void(*)(void*))
```

Первые два параметра, как и последний, аналогичны параметрам функции связывания BLOB-значений. Третий параметр — это терминированный нулем текст, который вы хотите связать, а четвертый параметр — длина текста в байтах, исключая терминирующий нуль. Если значение отрицательно, используется количество байтов до первого терминирующего нуля.

3. Связывание целых чисел. Связывающая функция для целых чисел проста:

```
int sqlite3_bind_int(sqlite3_stmt*, int, int)
```

Первые два параметра те же, что и в приведенных выше функциях, а последний — целочисленное значение.

4. Связывание вещественных чисел. Связывающая функция для вещественных чисел также проста и схожа с аналогичной для целых:

```
int sqlite3_bind_double(sqlite3_stmt*, int, double)
```

Первые два параметра те же, что и в описанных выше функциях. Последний параметр — это вещественное значение.

5. Связывание значений NULL. Эта функция проще всех:

```
int sqlite3_bind_null(sqlite3_stmt*, int)
```

Первые два параметра те же, что и в приведенных выше функциях. Значение подразумевается.

Функция `insertCompany()` (листинг 10.9) подразумевает, что для каждой компании доступен PNG-файл. Файлы должны иметь то же имя, что и аббревиатура фирмы. Например, для Alcatel-Lucent логотип хранится в файле `ALU.png`. Чтобы получить размер файла с изображением в байтах, создадим объект `NSData` посредством метода `dataWithContentsOfFile:` класса `NSData`. Этот метод получает содержимое файла и создает на его основе объект `NSData`. Получив байты в объекте Objective-C, мы помещаем их в C-строку, используя следующие два выражения:

```
buffer = malloc([pData length]);
[pData getBytes:buffer];
```

Первое выражение получает буфер длины, равной длине объекта `NSData`. Чтобы получить байты, мы используем метод экземпляра `getBytes:` во втором выражении.

Теперь, когда у нас есть три значения для трех SQL-параметров, используем подходящие функции связывания, чтобы закончить SQL-выражение. Выполнение выражения INSERT аналогично выполнению любого другого подготавливаемого выражения — просто используйте `sqlite3_step()`. Наконец, финализируем выражение и высвободим буфер, так как мы указали SQLITE_STATIC в функции связывания BLOB-значения.

Листинг 10.9. Функция insertCompany() для вставки записи о компании, включающей BLOB-изображение

```
#import "/usr/include/sqlite3.h"
void insertCompany(sqlite3 *pDb, const char* symbol, const char* name){
    int returnCode;
    sqlite3_stmt *pStmt;
    unsigned char *buffer;
    char  *st = "INSERT INTO companies VALUES (?, ?, ?)";
    returnCode = sqlite3_prepare_v2(pDb, st, -1, &pStmt, 0);
    if (returnCode != SQLITE_OK) {
            fprintf(stderr, "Error in inserting into companies table.");
            return;
    }
    NSMutableString *imageFileName =
            [NSMutableString stringWithCString:symbol];
    [imageFileName appendString:@".png"];
    NSData * pData = [NSData dataWithContentsOfFile:imageFileName];
    buffer = malloc([pData length]);
    [pData getBytes:buffer];
    sqlite3_bind_text(pStmt, 1, symbol, -1, SQLITE_STATIC);
    sqlite3_bind_text(pStmt, 2, name, -1, SQLITE_STATIC);
    sqlite3_bind_blob(pStmt, 3, buffer, [pData length], SQLITE_STATIC);
    returnCode = sqlite3_step(pStmt);
    if (returnCode != SQLITE_DONE) {
            fprintf(stderr, "Error in inserting into companies table.");
    }
    returnCode = sqlite3_finalize(pStmt);
     if(returnCode != SQLITE_OK) {
            fprintf(stderr, "Error in inserting into companies table. ");
    }
    free(buffer);
}
```

10.6. Получение BLOB-значений

Из предыдущего раздела вы узнали, как наполнить таблицу записями, содержащими BLOB-колонки. В этом разделе вы научитесь получать эти BLOB-колонки. В примере будет использоваться таблица companies, заполненная выше.

Листинг 10.10 показывает функцию `main()`, используемую для демонстрации получения BLOB-значений. Мы хотели бы получить эти изображения и записать их в файловую систему под другими именами. Функция `main()` открывает базу данных и получает изображения посредством вызова функции `retreiveCompany()`, показанной в листинге 10.11.

Листинг 10.10. Функция main(), демонстрирующая получение BLOB-значений из базы данных

```
int main(int argc, char *argv[]) {
    sqlite3   *pDb;
    int       returnCode;
    char      *databaseName;
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    databaseName = "financial.db";
    returnCode = sqlite3_open(databaseName, &pDb);
    if(returnCode!=SQLITE_OK) {
        fprintf(stderr,"Error in opening the database. Error: %s",
                sqlite3_errmsg(pDb));
        sqlite3_close(pDb);
        return -1;
    }
    retrieveCompany(pDb, "ALU");
    retrieveCompany(pDb, "GOOG");
    retrieveCompany(pDb, "MSFT");
    retrieveCompany(pDb, "NT");
    sqlite3_close(pDb);
    [pool release];
    return 0;
}
```

Мы начинаем с подготовки параметризованного SQL-выражения:

```
SELECT image FROM companies WHERE symbol = ?
```

Затем мы связываем единственный параметр с параметром `symbol` функции. Обратите внимание, что мы могли бы использовать только `sqlite3_mprintf()` без вызова параметризованных запросов. Затем мы выполняем запрос и проверяем результат. Может быть не более одной записи (`symbol` является первичным ключом), поэтому мы получаем BLOB-значение самое большее один раз. Мы используем `NSData` в качестве обертки байтов изображения в следующем выражении:

```
NSData * pData =
            [NSData dataWithBytes:sqlite3_column_blob(pStmt, 0)
                    length:sqlite3_column_bytes(pStmt, 0)];
```

Метод класса `dataWithBytes:length:` объявлен следующим образом:

```
+ (id)dataWithBytes:(const void *)bytes length:(NSUInteger)length
```

Функции в качестве двух параметров передаются байты и длина. Чтобы получить BLOB-значение из результирующей колонки, мы используем функцию `sqlite3_column_blob()`. Ей передаются указатель на дескриптор выражения, который мы получили после вызова функции `sqlite3_prepare_v2()`, и порядковый номер колонки (начиная с 0). Длину BLOB-байтов можно получить с помощью функции `sqlite3_column_bytes()`.

Получив изображение из базы данных и использовав экземпляр `NSData` в качестве обертки, мы можем применить метод `writeToFile:atomically:` экземпляра `NSData` для для записи данных в файл. Метод объявлен следующим образом:

```
- (BOOL)writeToFile:(NSString *)path atomically:(BOOL)useAuxiliaryFile
```

В дополнение к пути файла используется `useAuxiliaryFile` для указания, требуется ли использование временного файла. Если значение YES, данные сначала запишутся во временный файл, а затем этому временному файлу будет дано новое имя. Записав файл, мы финализируем выражение и возвращаемся из функции.

Листинг 10.11. Функция retreiveCompany(), используемая для получения BLOB-изображений из базы данных и записи их в файловую систему

```
#import "/usr/include/sqlite3.h"
void retrieveCompany(sqlite3 *pDb, const char* symbol){
    int     returnCode;
    sqlite3_stmt  *pStmt;
    char  *st = "SELECT image FROM companies WHERE symbol = ?";
    returnCode = sqlite3_prepare_v2(pDb, st, -1, &pStmt, 0);
    if (returnCode!=SQLITE_OK) {
            fprintf(stderr, "Error retrieving image from companies.");
            return;
    }
    sqlite3_bind_text(pStmt, 1, symbol, -1, SQLITE_STATIC);
    returnCode = sqlite3_step(pStmt);
    if(returnCode == SQLITE_ROW){
            NSData * pData =
                    [NSData dataWithBytes:sqlite3_column_blob(pStmt, 0)
                    length:sqlite3_column_bytes(pStmt, 0)];
            NSMutableString *imageFileName =
            [NSMutableString stringWithCString:symbol];
    [imageFileName appendString:@"-2.png"];
    [pData writeToFile:imageFileName  atomically:YES];
    }
    returnCode = sqlite3_finalize(pStmt);
    if(returnCode != SQLITE_OK) {
            fprintf(stderr, "Error inserting into companies.");
    }
}
```

10.7. Резюме

Эта глава осветила основные аспекты использования механизма баз данных SQLite в составе iPhone-приложения.

Основные концепции были представлены с помощью примеров. Мы начали с обсуждения основных SQL-выражений и их реализаций с использованием вызовов функций SQLite. Затем мы обсудили обработку результирующих множеств, генерируемых SQL-выражениями. После этого мы обратились к теме подготовливаемых выражений. Далее мы говорили о расширениях C API SQLite и рассмотрели применение простой пользовательской функции. Наконец, мы разобрали обработку BLOB-значений на примере сохранения и получения файлов с изображениями.

Глава 11
Обработка XML

В этой главе вы научитесь эффективному использованию XML в вашем iPhone-приложении. Глава построена так же, как предыдущие. Она раскрывает основные идеи посредством работающего iPhone-приложения – программы для чтения RSS-ленты.

Структура данной главы следующая. В разд. 11.1 объясняются основные концепции XML и RSS. В разд. 11.2 подробно описан DOM-парсинг. В разд. 11.3 представлена другая техника парсинга XML – SAX. В нем вы научитесь писать SAX iPhone-клиента. В разд. 11.4 мы рассмотрим приложение для чтения RSS-лент, основанное на таблицах. В разд. 11.5 будут подведены итоги и описаны основные шаги, необходимые, чтобы использовать все возможности XML в вашем iPhone-приложении.

11.1. XML и RSS

11.1.1. XML

Расширяемый язык разметки (Extensible Markup Language (XML)) – это спецификация метаязыка для обмена информацией через Интернет. Как метаязык он может использоваться для определения учитывающих специфику приложения языков, которые, в свою очередь, могут применяться для создания XML-документов, следующих семантике этих языков.

Широкие возможности XML обусловлены его расширяемостью, позволяющей определять новые XML-элементы, и текстовой основой, разрешающей открывать и использовать данные приложения на любой операционной системе.

Для использования языка XML необходимо идентифицировать применяемые в нем элементы. XML-элемент включает открывающий тег,

текстовое содержимое и закрывающий тег. Например, элемент person можно представить в XML-документе следующим образом:

```
<person>
содержимое...
</person>
```

Здесь <person> — это открывающий тег, а </person> — закрывающий. Содержимое может состоять из текста и других элементов, например:

```
<person>
    <name>содержимое имени...</name>
    <address>содержимое адреса...</address>
</person>
```

Если текстовое содержимое элемента включает сложные для обработки символы (<, >, & и т. д.), можно использовать их замещения, например, < можно представить в виде <.

XML-документ может иметь только один корневой элемент, и любой элемент может содержать другие элементы, поэтому XML-документ можно представить в виде дерева. Например, следующий XML-документ может быть представлен в виде дерева (рис. 11.1):

```
<?xml_version="1.0"?>
<person>
__<name>
____<first>Homer</first>
____<last>Simpson</last>
__</name>
__<address>
____<street>1094_Evergreen_Terrace</street>
____<city>Springfield</city>
____<state>TA</state>
__</address>
</person>
```

Для работы с XML-документом нужно осуществлять его парсинг (например, составить дерево представлений документа в памяти, как показано на рис. 11.1). Существует несколько методов парсинга. Рассмотрим их вкратце. Парсер libxml2 — это XML-парсер, написанный на C, который доступен и рекомендуется для использования в iPhone OC. Работать с этой библиотекой несложно. Чтобы сконструировать дерево, подобное представленному на рис. 11.1, нужно вызвать несколько функций.

Следует помнить, что в XML не игнорируются пробелы. На рис. 11.1 пробелы представлены как узлы типа TEXT. В libxml2 текстовые узлы являются элементами типа XML_TEXT_NODE, а узлы элементов — типа XML_ELEMENT_NODE.

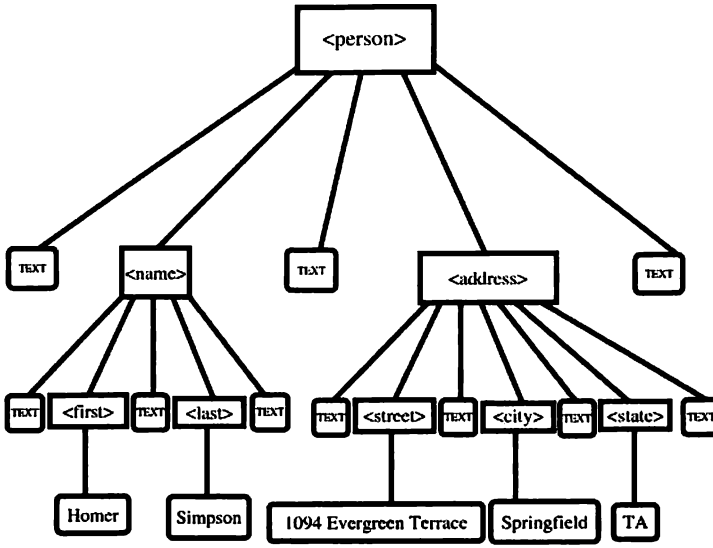

Рис. 11.1. Древовидное представление XML-документа

Теперь, когда вы понимаете, что представляет собой XML, рассмотрим один из способов его применения — RSS.

11.1.2. RSS

Очень простой синдикат (Really Simple Syndicat (RSS)) — это XML-язык, используемый для совместного доступа к веб-содержимому. Как создателю содержимого RSS дает вам возможность информировать читателей о новых публикациях на вашем информационном канале. Как потребителю содержимого RSS позволяет направить вашу интернет-активность на действительно интересующую вас информацию. Например, если вы хотите знакомиться, в основном, с новостями медицины, но вам не хочется тратить время на cnn.com или msnbc.com в поиске медицинских статей. Вам нужно, чтобы cnn.com или msnbc.com сами сообщали, когда на их сайтах появляются новые медицинские статьи. Новостной канал можно настроить на экземпляр XML-файла, основанный на языке RSS и информирующий о новых медицинских статьях на сайтах. Для подписки на XML-файл вы используете программное обеспечение. Пользователь может обновлять копию XML-файла и предоставлять ее вам. Эта схема обеспечивает эффективность и конфиденциальность, так как сайту не нужно знать ваш электронный адрес, чтобы информировать вас о свежих новостях. RSS

может выступать в роли как поставщика, так и потребителя. Производитель поставляет отфильтрованное содержимое, потребляемое клиентом.

Сайты рекламируют существование подобных каналов, размещая специальные значки. Ниже приведены некоторые из них (рис. 11.2). Широкое распространение получил значок универсальной ленты (нижний).

Рис. 11.2. Некоторые RSS-значки; широко распространен значок универсальной ленты (нижний)

Ознакомимся с принципами работы RSS-ленты на примере. Государственная патрульная служба штата Небраска предоставляет RSS-ленту о скрывающихся правонарушителях. RSS-лента имеет URL-адрес. Например, URL-адрес информации о скрывающихся правонарушителях штата Небраска следующий: http://www.nsp.state.ne.us/SOR/Abscondedrss.xml. Листинг 11.1 показывает пример документа этой ленты.

Листинг 11.1. Пример RSS-документа

```
<?xml version="1.0" encoding="UTF-8"?> <rss version="2.0">
    <channel>
      <title>
            Nebraska State Patrol | Absconded Offenders
  </title>
  <link>
    http://www.nsp.state.ne.us/sor/
  </link>
  <description>
The Nebraska State Patrol is currently seeking information on the location of the following
individuals to determine if they are in compliance with the Nebraska Sex Offender Registration
Act. This site is intended to generate information on these individuals and should not be used
solely for the purpose of arrest. Anyone with information please call 402-471-8647.
    </description>
    <image>
      <title>Nebraska State Patrol | SOR</title>
      <url>http://www.nsp.state.ne.us/sor/rsslogo.jpg
      </url>
      <link>http://www.nsp.state.ne.us/sor/</link> </link>
    </image>
```

```
    <item>
      <title>Austen, Kate</title>
      <link>
      http://www.nsp.state.ne.us/sor/200403KA2
      </link>
      <description>
      Absconded - Jefe de una loca mujer
      </description>
    </item>
  </channel>
</rss>
```

Каждый документ RSS-ленты начинается со строки

```
<?xml version="1.0" encoding="UTF-8"?>
```

Она сообщает, что документ является XML-документом. Атрибут `version` является обязательным, а атрибут `encoding` — нет.

Корневым элементом RSS-ленты является `rss`. Этот корневой элемент имеет только один дочерний элемент — `channel`. Элемент `channel` имеет три обязательных дочерних — `title`, `link` и `description`. В дополнение он может содержать несколько необязательных дочерних элементов — `webMaster`, `image` и `copyright`.

Обязательные элементы необходимы, чтобы RSS-лента была валидной. Тем не менее «валидный» не всегда значит «полезный». Чтобы быть полезным, элемент `channel` должен иметь один или более дочерних элементов `child`. Каждая новость в файле RSS-ленты представлена элементом `item`. Элемент `item` содержит три дочерних — `title`, `link` и необязательный `description`. Приложение для чтения представляет заголовок новости, ссылку на нее и опционально ее описание. Заинтересовавшись новостью, вы щелкаете на ссылке (URL-адресе) для перехода на интернет-страницу этой новости.

Теперь, когда вы изучили структуру документа RSS-ленты, используем библиотеку `libxml2` для извлечения информации из RSS-ленты. Сначала рассмотрим приложение для чтения с применением DOM, а затем — другое, использующее SAX.

11.2. Объектная модель документа

Используя объектную модель документа (Document Object Model (DOM)), парсер загружает в память весь XML-документ и представляет его клиенту в виде дерева. Вы можете перемещаться по узлам дерева и извлекать нужную информацию.

Листинг 11.2 показывает код Objective-C, который сначала извлекает по URL-адресу XML-документ и помещает его в строку, с которой может работать библиотека `libxml2`, а затем использует функции `libxml2` для

навигации по обработанному дереву и извлечения соответствующей информации.

Листинг 11.2. DOM XML-парсинг

```
1  #include <libxml/xmlmemory.h>
2  #include <libxml/parser.h>
3
4  -(void)fetchAbsconders{
5  NSAutoreleasePool * pool =
6          [[NSAutoreleasePool alloc] init];
7  NSError *err = nil;
8  NSURL * url = [NSURL URLWithString:feedURL];
9  NSString *URLContents =
10         [NSString stringWithContentsOfURL:url
11         encoding:NSUTF8StringEncoding error:&err];
12 if(!URLContents)
13   return;
14 const char *XMLChars =
15          [URLContents cStringUsingEncoding:NSUTF8StringEncoding];
16
17 if(parser == XML_PARSER_DOM){
18         xmlDocPtr doc =
19                 xmlParseMemory(XMLChars, strlen(XMLChars));
20         xmlNodePtr cur;
21         if (doc == NULL ) {
22                 return;
23         }
24         cur = xmlDocGetRootElement(doc);
25         cur = findNextItem(cur);
26         while (cur){
27                 XOAbsconder *absconder =  getitem(doc, cur);
28                 if(absconder){
29                         [absconders addObject:absconder];
30                 }
31                 cur = findNextItem(cur->next);
32         }
33         xmlFreeDoc(doc);
34 }
```

В строке 8 мы создаем объект NSURL из строкового представления URL-адреса, feedURL, адреса RSS-ленты. Выражения в строках 9—11 используют метод `stringWithContentsOfURL:encoding:error` класса NSString, чтобы создать строку, содержащую данные по URL-адресу. Метод извлекает файл RSS-ленты с сервера и помещает его в URLContents, экземпляр NSString.

В строке 12 мы проверяем, была ли строка успешно создана. Если нет, метод fetchAbsconders возвращается без изменения массива absconders. В окончательном варианте кода мы будем использовать объект error, чтобы информировать клиента об ошибке.

Получив объект NSString с содержимым файла RSS-ленты, нам нужно сконвертировать его в С-строку (char *) — формат, с которым работает libxml2. Это делают выражения в строках 14 и 15. Мы используем

метод cStringUsingEncoding: экземпляра NSString с кодировкой NSUTF8StringEncoding.

Метод fetchAbsconders демонстрирует использование двух схем парсинга XML. Листинг 11.2 показывает первую часть этого метода, осуществляющую DOM-парсинг.

Чтобы работать с XML-документом, используя DOM, сначала нужно загрузить его в память в виде дерева. Для этого применяется функция xmlParseMemory(). В parser.h она объявлена следующим образом:

```
xmlDocPtr  xmlParseMemory (const char * buffer, int size)
```

В качестве входных данных ей передается XML-документ в виде С-строки, а также размер этой строки. Функция возвращает указатель на древовидное представление обработанного документа в виде xmlDocPtr (указатель на xmlDoc).

Структура xmlDoc определена в tree.h. Следующий код демонстрирует ее первые строки:

```
struct _xmlDoc {
    void *_private;  /* данные приложения */
    xmlElementType  type;  /* XML_DOCUMENT_NODE */
    char  *name;  /* имя/имя файла/URI документа */
    struct _xmlNode *children;  /* дерево документа */
    struct _xmlNode *last;  /* ссылка на последний дочерний элемент */
    struct _xmlNode *parent;  /* ссылка на родительский элемент */
...
};
```

У нас есть древовидное представление XML-документа, и мы можем начать его последовательную обработку. Для этого в строке 24 мы получаем корневой элемент, используя функцию xmlDocGetRootElement(). Функция возвращает xmlNodePtr, указывающий на корневой элемент xmlNode.

Каждый узловой элемент представлен структурой xmlNode, определенной в tree.h следующим образом:

```
typedef struct _xmlNode xmlNode;
typedef xmlNode *xmlNodePtr;
struct _xmlNode {
    void *_private; /* данные приложения */
    xmlElementType  type;/* номер типа*/
    const xmlChar  *name; /* название узла или сущности*/
    struct _xmlNode *children; /* ссылка на дочерние узлы*/
    struct _xmlNode *last; /* ссылка на последний дочерний элемент */
    struct _xmlNode *parent; /* ссылка на родительский узел */
    struct _xmlNode *next; /* ссылка на следующий элемент того же уровня */
    struct _xmlNode *prev; /* ссылка на предыдущий элемент того же уровня */
    struct _xmlDoc *doc; /* документ */
    /* Конец общей части */
    xmlNs *ns; /* указатель на ассоциированное пространство имен */
    xmlChar  *content; /* содержимое */
    struct _xmlAttr *properties; /* список свойств*/
```

```
    xmlNs *nsDef; /* определения пространства имен для этого узла */
    void  *psvi; /* для type/PSVI сведений */
    unsigned short  line; /* номер линии */
    unsigned short  extra; /* избыточные данные для  XPath/XSLT */
};
```

Большинство названий этих полей говорят сами за себя. В основном вы будете иметь дело с полями, ссылающимися на другие узлы. Если вы находитесь в конкретном узле, то можете переместиться в родительский, используя поле `parent`. Если вам потребуются дочерние элементы, применяйте `children`. Если нужны одноуровневые элементы (то есть узлы, имеющие тот же родительский, что и текущий узел), используйте поле `next`.

Ниже приведено графическое представление навигационных ссылок, доступных для различных узлов дерева документа (рис. 11.3).

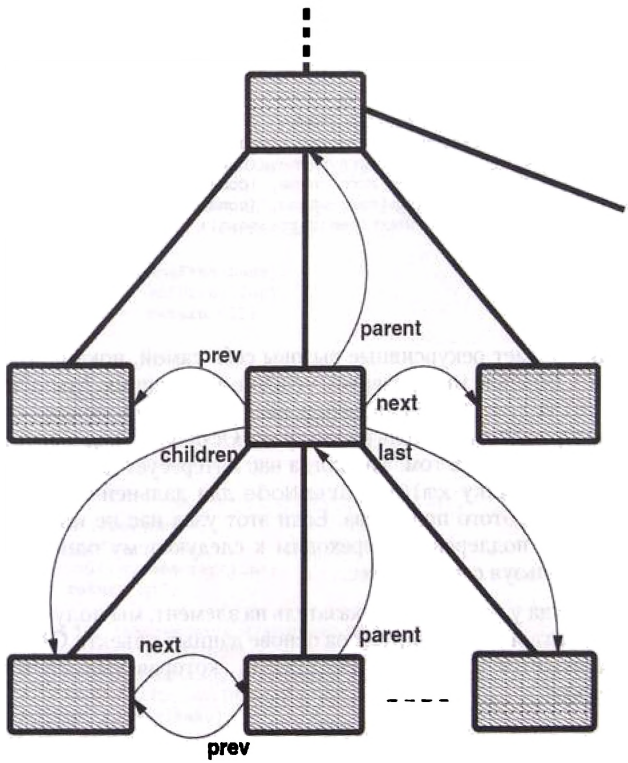

Рис. 11.3. Представление навигационных ссылок, доступных для различных узлов дерева документа

Теперь, когда у нас есть указатель на корневой элемент документа, мы будем искать первый элемент `item` в RSS-ленте. Это показано в выражении в строке 25: `cur = findNextItem(cur)`. Функция `findNextItem()` определена в листинге 11.3.

Листинг 11.3. Поиск элемента item в RSS-ленте

```
xmlNodePtr findNextItem(xmlNodePtr curr){
    if(!curr)
            return curr;
    if ((!xmlStrcmp(curr->name, (const xmlChar *)"item")) &&
            (curr->type == XML_ELEMENT_NODE)) {
            return curr;
    }
    if(curr->type == XML_TEXT_NODE){
            return findNextItem(curr->next);
    }
    if(curr->type == XML_ELEMENT_NODE){
            if ((!xmlStrcmp(curr->name, (const xmlChar *)"channel"))
              || (!xmlStrcmp(curr->name, (const xmlChar *)"rss"))){
              return  findNextItem(curr->xmlChildrenNode);
            }
    }
    if(curr->type == XML_ELEMENT_NODE){
            if((!xmlStrcmp(curr->name, (const xmlChar *)"title"))
              || (!xmlStrcmp(curr->name, (const xmlChar *)"link"))
              || (!xmlStrcmp(curr->name, (const xmlChar *)"description"))
              || (!xmlStrcmp(curr->name, (const xmlChar *)"image"))){
              return  findNextItem(curr->next);
            }
    }
    return NULL;
}
```

Функция делает рекурсивные вызовы себя самой, пока не будет найден тег `item`. Вначале мы проверяем условие завершения. Мы используем функцию `xmlStrcmp()` для проверки, равно ли имя элемента `item`. Если да, то мы возвращаем указатель на этот узел. Остальной код имеет сходную логику. Разница лишь в том, что, когда нас интересует данное поддерево, мы применяем ссылку `xmlChildrenNode` для дальнейшего последовательного анализа этого поддерева. Если этот узел нас не интересует, мы пропускаем все поддерево и переходим к следующему одноуровневому элементу, используя ссылку `next`.

Теперь, когда у нас имеется указатель на элемент, мы получаем три дочерних элемента и конструируем на основе данных объекта Object-C. Эта логика реализуется в функции `getitem()`, которая вызывается следующим образом:

```
XOAbsconder *absconder =  getitem(doc, cur);
```

Функции `getitem()` передаются указатели на документ и узел, а возвращается объект `XOAbsconder` или `nil`. Листинг 11.4 представляет реализацию функции `getItem()`.

Листинг 11.4. Конструирование объекта XOAbsconder из элемента item

```
XOAbsconder* getitem (xmlDocPtr doc, xmlNodePtr curr){
    xmlChar *name, *link, *description;
    curr = curr->xmlChildrenNode;
    if(!curr)
            return nil;
            while (curr && (curr->type == XML_TEXT_NODE))
                    curr = curr->next;
            if(!curr)
                    return nil;
    if ((!xmlStrcmp(curr->name,(const xmlChar *)"title")) &&
            (curr->type == XML_ELEMENT_NODE)) {
            name = xmlNodeListGetString(doc, curr->xmlChildrenNode, 1);
            curr = curr->next;
            while (curr && (curr->type == XML_TEXT_NODE))
                    curr = curr->next;
            if(!curr){
                    xmlFree(name);
                    return nil;
            }
    }
    else
            return nil;
    if ((!xmlStrcmp(curr->name, (const xmlChar *)"link")) &&
                    (curr->type == XML_ELEMENT_NODE)) {
            link = xmlNodeListGetString(doc, curr->xmlChildrenNode, 1);
            curr = curr->next;
            while (curr && (curr->type == XML_TEXT_NODE))
                    curr = curr->next;
            if(!curr){
                    xmlFree(name);
                    xmlFree(link);
                    return nil;
            }
    }
    else
            return nil;
    if ((!xmlStrcmp(curr->name, (const xmlChar *)"description")) &&
            (curr->type == XML_ELEMENT_NODE)) {
            description = xmlNodeListGetString(doc, curr->xmlChildrenNode, 1);
    }
    else{
            xmlFree(name);
            xmlFree(link);
            xmlFree(description);
            return nil;
    }
    XOAbsconder *abscender = [[XOAbsconder alloc]
            initWithName:[NSString stringWithCString:name]
            andURL:[NSString stringWithCString:link]
            andDescription:[NSString stringWithCString:description]];
    [abscender autorelease];
    xmlFree(name);
    xmlFree(link);
    xmlFree(description);
    return abscender;
}
```

Мы последовательно обрабатываем все дочерние элементы узла. Пробелы в XML интерпретируются как валидные дочерние элементы, поэтому в самом начале мы пропускаем их.

```
while (curr && (curr->type == XML_TEXT_NODE))
    curr = curr->next;
```

Пропустив текстовые узлы, мы проверяем наличие трех элементов — `title`, `link` и `decription`. Функция требует их появления именно в этом порядке.

Чтобы получить текстовое значение для каждого из этих трех элементов, можно использовать функцию `xmlNodeListGetString`. Она объявлена в `tree.h` следующим образом:

```
xmlChar *xmlNodeListGetString (xmlDocPtr doc, xmlNodePtr list, int inLine)
```

Эта функция конструирует строку из списка узлов. Если `inLine` равно 1, содержимое сущностей изменяется. Функция возвращает строку, а вызывающий должен высвободить занятую строкой память с помощью функции `xmlFree()`.

Получив текстовые значения трех элементов, мы создаем объект `XOAbsconder`, делаем его самовысвобождающимся, освобождаем память, занятую тремя исходными строками, и возвращаем объект `XOAbsconder`.

Метод `fetchAbsconders` продолжает вызывать функцию `getitem()` и добавлять объекты в массив `absconders` с помощью следующего выражения (см. листинг 11.2):

```
[absconders addObject:absconder];
```

Когда метод `fetchAbsconders` заканчивает работу, массив `absconders` содержит объекты с информацией о скрывающихся правонарушителях. Информация получена из документа RSS-ленты.

11.3. Простой интерфейс для XML

В разд. 11.2 вы увидели, как DOM загружает весь XML-документ в память и позволяет перемещаться по его узлам. В некоторых приложениях размер XML-документа может помешать загрузке целого документа в связи с ограниченной памятью устройства. SAX (Simple API for XML — простой интерфейс для XML) — это другая, отличная от DOM, модель парсинга XML. В SAX вы настраиваете парсер с помощью callback-функций. SAX-парсер будет использовать указатели на функции для вызова ваших функций, информируя о важных событиях. Например, если вас интересует событие `Начало Документа`, вы настраиваете функцию на него и передаете парсеру указатель на нее.

Листинг 11.5 показывает оставшуюся часть метода `fetchAbsconders`, принадлежащую SAX-парсингу.

Листинг 11.5. SAX XML-парсинг (оставшаяся часть метода fetchAbsconders)

```
    else if(parser == XML_PARSER_SAX){
            xmlParserCtxtPtr ctxt = xmlCreateDocParserCtxt(XMLChars);
            int parseResult =
                    xmlSAXUserParseMemory(&rssSAXHandler, self, XMLChars,
                            strlen(XMLChars));
            xmlFreeParserCtxt(ctxt);
            xmlCleanupParser();
    }
    [pool release];
}
```

Чтобы применить SAX в `libxml2`, нужно сначала настроить контекст парсера, используя функцию `xmlCreateDocParserCtxt()`, которой передается единственный параметр — XML-документ в виде C-строки. После этого вы запускаете SAX-парсер посредством вызова `xmlSAXUserParseMemory()`. Функция объявлена в `parser.h` следующим образом:

```
int xmlSAXUserParseMemory (xmlSAXHandlerPtr sax,
    void * user_data, const char * buffer, int size)
```

Эта функция обрабатывает буфер в памяти и при необходимости вызывает зарегистрированные вами функции. Первый параметр функции — указатель на SAX-дескриптор. SAX-дескриптор — это структура, хранящая указатели на ваши callback-функции. Второй параметр — опциональный указатель, зависящий от специфики приложения. Заданное значение будет использоваться в качестве контекста при вызове SAX-парсером ваших callback-функций. Третий и четвертый параметры — это, соответственно, C-строка XML-документа в памяти и ее длина.

В SAX-дескрипторе вы храните указатели на callback-функции. Если вас не интересует данное событие, сохраните `NULL` в данном поле. Ниже приведено определение структуры в `tree.h`.

```
struct _xmlSAXHandler {
    internalSubsetSAXFunc internalSubset
    isStandaloneSAXFunc isStandalone
    hasInternalSubsetSAXFunc  hasInternalSubset
    hasExternalSubsetSAXFunc  hasExternalSubset
    resolveEntitySAXFunc  resolveEntity
    getEntitySAXFunc  getEntity
    entityDeclSAXFunc entityDecl
    notationDeclSAXFunc notationDecl
    attributeDeclSAXFunc  attributeDecl
    elementDeclSAXFunc  elementDecl
    unparsedEntityDeclSAXFunc unparsedEntityDecl
    setDocumentLocatorSAXFunc setDocumentLocator
    startDocumentSAXFunc  startDocument
    endDocumentSAXFunc  endDocument
    startElementSAXFunc startElement
    endElementSAXFunc endElement
    referenceSAXFunc  reference
    charactersSAXFunc characters
    ignorableWhitespaceSAXFunc  ignorableWhitespace
```

```
    processingInstructionSAXFunc  processingInstruction
    commentSAXFunc  comment
    warningSAXFunc  warning
    errorSAXFunc  error
    fatalErrorSAXFunc fatalError: get all the errors
    getParameterEntitySAXFunc getParameterEntity
    cdataBlockSAXFunc  cdataBlock;
    externalSubsetSAXFunc  externalSubset
    unsigned int  initialized
    //Поля-расширения
    void *  _private;
    startElementNsSAX2Func  startElementNs
    endElementNsSAX2Func  endElementNs
    xmlStructuredErrorFunc  serror
}
```

Листинг 11.6 показывает SAX-дескриптор.

Листинг 11.6. SAX-дескриптор

```
static xmlSAXHandler                rssSAXHandler ={
NULL,                               /* internalSubset */
NULL,                               /* isStandalone */
NULL,                               /* hasInternalSubset */
NULL,                               /* hasExternalSubset */
NULL,                               /* resolveEntity */
NULL,                               /* getEntity */
NULL,                               /* entityDecl */
NULL,                               /* notationDecl */
NULL,                               /* attributeDecl */
NULL,                               /* elementDecl */
NULL,                               /* unparsedEntityDecl */
NULL,                               /* setDocumentLocator */
NULL,                               /* startDocument */
NULL,                               /* endDocument */
NULL,                               /* startElement*/
NULL,                               /* endElement */
NULL,                               /* reference */
charactersFoundSAX,                 /* characters */
NULL,                               /* ignorableWhitespace */
NULL,                               /* processingInstruction */
NULL,                               /* comment */
NULL,                               /* warning */
errorEncounteredSAX,                /* error */
fatalErrorEncounteredSAX,           /* fatalError */
NULL,                               /* getParameterEntity */
NULL,                               /* cdataBlock */
NULL,                               /* externalSubset */
XML_SAX2_MAGIC,                     //
NULL,
startElementSAX,                    /* startElementNs */
endElementSAX,                      /* endElementNs */
NULL,                               /* serror */
};
```

В отличие от указателей на функции, поле `initialized` должно быть установлено в значение `XML_SAX2_MAGIC` для указания, что дескриптор используется для SAX2-парсера. Как только вы вызываете

xmlSAXUserParseMemory(), SAX-парсер начинает обработку документа и вызывает зарегистрированные вами callback-функции.

Вас должны интересовать три функции — startElementNsSAX2Func(), endElementNsSAX2Func() и charactersSAXFunc().

Функция startElementNsSAX2Func() вызывается, когда парсер встречает начало нового элемента, и определена в tree.h следующим образом:

```
void  startElementNsSAX2Func (void * ctx,
    const xmlChar * localname,
    const xmlChar * prefix,
    const xmlChar *URI,
    int nb_namespaces,
    const xmlChar ** namespaces,
    int nb_attributes,
    int nb_defaulted,
    const xmlChar ** attributes)
```

где:

- ctx — это пользовательские данные, а также второй параметр при вызове функции xmlSAXUserParseMemory(); в нашем случае это указатель на класс XORSSFeedNebraska;
- localname — локальное имя элемента;
- prefix — префикс пространства имен элемента (если доступен);
- URI — название пространства имен элемента (если доступно);
- nb_namespaces — количество определений пространства имен для данного узла;
- namespaces — указатель на массив пар префикс/URI определений пространств имен;
- nb_attributes — количество атрибутов для данного узла;
- nb_defaulted — количество атрибутов по умолчанию, находящихся в конце массива;
- attributes — указатель на массив значений атрибутов (локальное имя/префикс/URI/значение/конец).

Листинг 11.7. Callback-функция startElementSAX()

```
static void startElementSAX(void *ctx,
            const xmlChar *localname,
            const xmlChar *prefix,
            const xmlChar *URI,
            int nb_namespaces,
```

```
            const xmlChar **namespaces,
            int nb_attributes,
            int nb_defaulted,
            const xmlChar **attributes) {
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
    XORSSFeedNebraska *feedNebraska = (XORSSFeedNebraska*) ctx;
    if (feedNebraska.currentElementContent) {
            [feedNebraska.currentElementContent release];
            feedNebraska.currentElementContent = nil;
    }
    if ((!xmlStrcmp(localname, (const xmlChar *)"item"))) {
            feedNebraska.currAbsconder = [[XOAbsconder alloc] init];
    }
    [pool release];
}
```

Для каждой функции хорошо иметь свой самовысвобождающийся пул. Мы начинаем с приведения `ctx` к указателю на наш класс `XORSFeedNebraska`. Этот класс, как и его родительский, объявлен в листингах 11.8 и 11.9.

Листинг 11.8. Объявление класса XORSSFeedNebraska

```
#import "XORSSFeed.h"
@interface XORSSFeedNebraska : XORSSFeed {
}
@end
```

Листинг 11.9. Объявление класса XORSSFeed

```
@class XOAbsconder;
typedef enum {
    XML_PARSER_DOM, XML_PARSER_SAX
} XMLParser;
@interface XORSSFeed : NSObject {
    NSString  *feedURL;
    NSMutableArray  *absconders;
    XMLParser  parser;
    NSMutableString  *currentElementContent;
    XOAbsconder  *currAbsconder;
}
@property(nonatomic, copy) NSString  *feedURL;
@property(nonatomic, assign) XMLParser  parser;
@property(nonatomic, assign) NSMutableString  *currentElementContent;
@property(nonatomic, assign) XOAbsconder *currAbsconder;
-(id)init;
-(id)initWithURL:(NSString*) feedURL;
-(void)fetchAbsconders;
-(NSUInteger)numberOfAbsconders;
-(XOAbsconder*)absconderAtIndex:(NSUInteger) index;
-(void)addAbsconder:(XOAbsconder*)absconder;
@end
```

Объект `XORSSFeedNebraska` содержит переменную экземпляра типа `NSMutableString`, которая называется `currentElementContent`

и содержит текстовое значение элемента. Она создается в нашей функции `characterFoundSAX()` и используется в `endElementSAX()`. Функция `startElementSAX()` всегда высвобождает переменные, поэтому мы устанавливаем эту переменную экземпляра в `nil` (если она еще не равна `nil`), чтобы быть уверенными, что начинаем, имея пустую строку для сохранения текста. Если имя элемента `item`, мы создаем новый объект класса `XOAbsconder`. Это достаточно простой класс, хранящий три элемента информации о скрывающемся правонарушителе. Листинг 11.10 показывает объявление класса `XOAbsconder`, а листинг 11.11 — его определение.

Листинг 11.10. Объявление класса XOAbsconder

```
#import <UIKit/UIKit.h>
@interface XOAbsconder : NSObject {
    NSString *name;
    NSString *furtherInfoURL;
    NSString *desc;
}
@property(copy) NSString *name;
@property(copy) NSString *furtherInfoURL;
@property(copy) NSString *desc;
-(id)init;
-(id)initWithName:(NSString*)name
    andURL:(NSString*)url
    andDescription:(NSString*)desc;
-(NSString*)description;
@end
```

Листинг 11.11. Определение класса XOAbsconder

```
#import "XOAbsconder.h"

@implementation XOAbsconder
@synthesize name;
@synthesize furtherInfoURL;
@synthesize desc;
-(id)initWithName:(NSString*)name andURL:(NSString*)url
    andDescription:(NSString*)description{
    self = [super init];
    if (self){
            self.name = name;
            self.furtherInfoURL = url;
            self.desc = description;
    }
    return self;
}
-(id)init{
    return [self initWithName:@"" andURL:@"" andDescription:@""];
}
-(NSString*)description{
    return [NSString stringWithString:name];
}
```

```
-(void)dealloc{
    [name release];
    [furtherInfoURL release];
    [desc release];
    [super dealloc];
}
@end
```

Наша функция `endElementNsSAX2Func()` называется `endElementSAX()` и приведена в листинге 11.12.

Листинг 11.12. Определение функции endElementSAX()

```
static void endElementSAX (void *ctx,
    const xmlChar *localname, const xmlChar *prefix, const xmlChar *URI) {
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
    XORSSFeedNebraska  *feedNebraska = (XORSSFeedNebraska*) ctx;
    if ((!xmlStrcmp(localname, (const xmlChar *)"item"))) {
      if(feedNebraska.currAbsconder){
        [feedNebraska addAbsconder:feedNebraska.currAbsconder];
      }
      [feedNebraska.currAbsconder release];
      feedNebraska.currAbsconder = nil;
    }
    else if ((!xmlStrcmp(localname, (const xmlChar *)"title"))) {
      if(feedNebraska.currAbsconder){
       feedNebraska.currAbsconder.name = feedNebraska.currentElementContent;
      }
    }
    else if ((!xmlStrcmp(localname, (const xmlChar *)"link"))) {
      if(feedNebraska.currAbsconder){
       feedNebraska.currAbsconder.furtherInfoURL =
            feedNebraska.currentElementContent;
      }
    }
    else if ((!xmlStrcmp(localname, (const xmlChar *)"description"))) {
      if(feedNebraska.currAbsconder){
       feedNebraska.currAbsconder.desc =
            feedNebraska.currentElementContent;
    }
    }
    if (feedNebraska.currentElementContent) {
      [feedNebraska.currentElementContent release];
      feedNebraska.currentElementContent = nil;
    }
    [pool release];
}
```

Функция вначале проверяет, равно ли имя элемента `item`. Если это так, мы добавляем объект `XOAbsconder`, созданный другими callback-функциями. В противном случае мы проверяем три имени элемента — `title`, `link` и `description`. Для каждого из этих элементов мы устанавливаем соответствующее текстовое значение, считанное функцией `charactersSAXFunc()`. Например, следующее выражение устанавливает переменную экземпляра `desc` в текущее текстовое значение:

```
feedNebraska.currAbsconder.desc = feedNebraska.currentElementContent;
```

Текст элемента хранится в `charactersSAXFunc()`. Функция объявлена в `parser.h` следующим образом:

```
void  charactersSAXFunc (void * ctx, const xmlChar * ch, int len)
```

Функция вызывается парсером, информируя о новых найденных символах. Вместе с контекстом вы получаете строку символов и ее длину. Между началом и концом элемента эту функцию можно вызвать несколько раз. Ваша функция должна знать это и прибавлять новый текст к текущей строке.

Наша функция `charactersSAXFunc()` приведена в листинге 11.13.

Листинг 11.13. Определение функции charactersSAXFunc()

```
static void charactersFoundSAX(void * ctx, const xmlChar * ch, int len){
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
    XORSSFeedNebraska  *feedNebraska =(XORSSFeedNebraska*) ctx;
    CFStringRef str =
            CFStringCreateWithBytes(kCFAllocatorSystemDefault,
                    ch, len, kCFStringEncodingUTF8, false);
    if (!feedNebraska.currentElementContent) {
            feedNebraska.currentElementContent =
                    [[NSMutableString alloc] init];
    }
    [feedNebraska.currentElementContent appendString:(NSString *)str];
    CFRelease(str);
    [pool release];
}
```

Функция начинает с приведения `ctx` к экземпляру `XORSSFeedNebraska`. Используя этот указатель, мы можем вызвать наш класс Objective-C. После этого мы создаем строку из полученных символов, используя функцию `CFStringCreateWithBytes()`, объявленную следующим образом:

```
CFStringRef CFStringCreateWithBytes (
    CFAllocatorRef alloc,
    const UInt8 *bytes,
    CFIndex numBytes,
    CFStringEncoding encoding,
    Boolean isExternalRepresentation
);
```

Первый параметр используется для указания блока распределения памяти (`kCFAllocatorDefault` используется в качестве текущего блока распределения памяти по умолчанию). Второй параметр — это буфер, хранящий символы, третий указывает количество байтов, а четвертый параметр — это кодировка. Для кодировки `UTF8` мы используем `kCFStringEncodingUTF8`. Пятый параметр применяется для указания, находятся ли символы в буфере во внешнем формате представления. Это не так, поэтому указываем `false`.

Имея строковое представление символов, мы проверяем, первый ли раз для данного элемента была вызвана `charactersFoundSAX`. Помните, что парсер может вызывать эту функцию несколько раз, извлекая содержимое элемента. Если это первый вызов, мы выделяем память под изменяемую строку. После этого мы прибавляем строку, которую создали из буфера символов, к нашей изменяемой строке. При вызове функции `endElementSAX()` мы извлекаем данную строку, чтобы создать объект Objective-C `currAbsconder`. Закончив работу со строкой `str`, мы используем функцию `CFRelease()` для ее высвобождения.

Функции обработки ошибок приведены в листингах 11.14 и 11.15. Как и в других функциях обработки событий, то, что вы делаете для обработки ошибок, зависит от вашего приложения. В нашем случае мы высвобождаем создаваемый объект `currAbsconder` и выводим в консоль текстовое описание проблемы.

Листинг 11.14. Определение функции errorEncounteredSAX()

```
static void errorEncounteredSAX (void * ctx, const char * msg, ...){
    XORSSFeedNebraska *feedNebraska = (XORSSFeedNebraska*) ctx;
      if(feedNebraska.currAbsconder){
            [feedNebraska.currAbsconder release];
            feedNebraska.currAbsconder = nil;
      }
            NSLog(@"errorEncountered: %s", msg);
}
```

Листинг 11.15. Определение функции fatalErrorEncounteredSAX()

```
static void fatalErrorEncounteredSAX (void * ctx, const char * msg, ...){
    XORSSFeedNebraska  *feedNebraska = (XORSSFeedNebraska*) ctx;
    if(feedNebraska.currAbsconder){
            [feedNebraska.currAbsconder release];
            feedNebraska.currAbsconder = nil;
    }
            NSLog(@"fatalErrorEncountered: %s", msg);
}
```

11.4. Приложение для чтения RSS

В этом разделе мы рассмотрим рабочее iPhone-приложение, основанное на разработанном ранее коде. Программа будет отображать содержимое RSS-ленты в прокручивающемся табличном представлении.

Листинг 11.16 демонстрирует объявление и определение делегата приложения. Делегат приложения использует экземпляр класса `XORSSFeedNebraska` для извлечения XML-документа и его парсинга для создания элементов.

Метод делегата создает окно и контроллер табличного представления. Затем он запрашивает у экземпляра XORSSFeedNebraska извлечение и парсинг XML-документа посредством вызова метода fetchAbsconders. Объявление и определение контроллера табличного представления приведены в листинге 11.17.

Листинг 11.16. Объявление и определение делегата приложения для чтения RSS

```
#import <UIKit/UIKit.h>
#import "XORSSFeedNebraska.h"
@interface XORSSFeedAppDelegate : NSObject {
    UIWindow *window;
    UINavigationController *navigationController;
    XORSSFeedNebraska *rssFeed;
}
@property (nonatomic, retain) UINavigationController *navigationController;
@end

#import "XORSSFeedAppDelegate.h"
#import "XORSSFeedNebraska.h"
#import "RootViewController.h"
@implementation XORSSFeedAppDelegate
@synthesize navigationController;
-(void)applicationDidFinishLaunching:(UIApplication *)application {
    window =
            [[UIWindow alloc] initWithFrame:
                    [[UIScreen mainScreen] bounds]];
    // Создаем навигацию и контроллеры представления
    RootViewController *rootViewController =
            [[RootViewController alloc] init];
    UINavigationController *aNavigationController =
            [[UINavigationController alloc]
            initWithRootViewController:rootViewController];
    self.navigationController = aNavigationController;
    [aNavigationController release];
    [rootViewController release];
    // Настраиваем и показываем окно
    [window addSubview:[navigationController view]];
    rssFeed = [[XORSSFeedNebraska alloc] init];
    [rssFeed fetchAbsconders];
    // Задаем клавишу окна и делаем ее видимой
    [window makeKeyAndVisible];
}
-(NSString*)xoTitle{
    return @"Nebraska Absconders";
}
-(NSInteger)countOfList {
    return [rssFeed numberOfAbsconders];
}
-(id)objectInListAtIndex:(NSUInteger)theIndex {
    return [rssFeed absconderAtIndex:theIndex];
}
-(void)dealloc {
    [window release];
    [navigationController release];
    [rssFeed release];
```

```
    [super dealloc];
}
@end
```

Как вы уже знаете из предыдущих глав, контроллер табличного представления является источником данных и делегатом табличного представления. Он использует делегат приложения для реакции на запросы о модели данных таблицы (количество строк и т. д.).

Ниже изображено главное окно приложения (рис. 11.4).

Листинг 11.17. Объявление и определение табличного контроллера приложения для чтения RSS

```
#import <UIKit/UIKit.h>
@interface RootViewController : UITableViewController {
}
@end
#import "RootViewController.h"
#import "XOAbsconder.h"
#import "XORSSFeedAppDelegate.h"
@implementation RootViewController
- init {
  if (self = [super init]) {
    XORSSFeedAppDelegate *appDelegate =
      (XORSSFeedAppDelegate*)[[UIApplication sharedApplication] delegate];
    self.title = [appDelegate xoTitle];
    }
    return self;
}
-(NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return 1;
}
-(NSInteger)tableView:(UITableView *)tableView
    numberOfRowsInSection:(NSInteger)section {
    XORSSFeedAppDelegate *appDelegate =
      (XORSSFeedAppDelegate*)[[UIApplication sharedApplication] delegate];
    return [appDelegate countOfList];
}
-(UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    UITableViewCell *cell =
            [tableView dequeueReusableCellWithIdentifier:@"XO"];
    if (cell == nil) {
            cell = [[[UITableViewCell alloc]
            initWithFrame:CGRectZero
                    reuseIdentifier:@"XO"] autorelease];
    }
    XORSSFeedAppDelegate *appDelegate =
      (XORSSFeedAppDelegate*)[[UIApplication sharedApplication] delegate];
    XOAbsconder *absconder =
            [appDelegate objectInListAtIndex:indexPath.row];
    cell.text = [absconder description];
    return cell;
}
@end
```

Рис. 11.4. Приложение для чтения RSS о скрывающихся правонарушителях штата Небраска

11.5. Резюме

Существует два подхода к парсингу XML – DOM и SAX. DOM конструирует древовидное представление XML-документа и позволяет перемещаться по узлам. SAX последовательно обрабатывает документ, в нужные моменты вызывая зарегистрированные вами callback-функции.

DOM следует использовать, если:

- хотите написать необходимый минимум кода;
- обрабатываете большую часть документа;
- хотите записать XML-документ обратно в файл;
- желаете сделать множество модификаций XML-документа;
- XML-документ является центральным для вашего приложения и у вас есть множество методов/объектов, работающих с ним.

Используйте SAX, если:

- XML-документ большой;
- вас интересует только пара узлов в XML-документе;

- приложение последовательно получает доступ к XML-документу.

Для использования DOM-парсинга в вашем приложении выполните следующие действия.

1. Создайте объект `NSURL` из URL-адреса XML-документа.

2. Создайте экземпляр `NSString` для хранения XML-документа посредством вызова метода `stringWithContentsOfURL:encoding:error:` класса `NSString`. Этот метод извлечет XML-документ из Интернета и сохранит его в Cocoa-строке.

3. Конвертируйте Cocoa-строку в C-строку, используя метод `cStringUsingEncoding:` экземпляра `NSString` и кодировку `NSUTF8StringEncoding`.

4. Используйте функцию `xmlParseMemory()` библиотеки `libxml2` для загрузки XML-документа в память и формирования дерева.

5. Получите указатель на корневой элемент и последовательно обойдите дерево в соответствии с требованиями вашего приложения.

6. Для извлечения текстового значения элемента используйте функцию `xmlNodeListGetString()`.

7. В любой момент, когда вы получаете строку от `libxml2`, помните, что необходимо освободить занятую ею память. Для этого используйте функцию `xmlFree()`. Закончив работу, вызовите функцию `xmlFreeDoc()`.

Чтобы использовать в вашем приложении SAX-парсинг, выполните следующие действия.

1. Создайте структуру типа `xmlSAXHandler`. Заполните поля, представляющие интересующие вас события, передавая указатели на ваши функции-обработчики событий. Убедитесь, что поле `initialized` установлено в `XML_SAX2_MAGIC`.

2. Функции для обработки разных событий имеют различные образцы. Например, `charactersFoundSAX()` объявлена следующим образом:

```
charactersFoundSAX(void *ctx, const xmlChar *ch, int len)
```

Найдите соответствующий образец функции в файле `tree.h` библиотеки `libxml2`.

3. Для осмысленного XML-парсинга создайте, по крайней мере, три функции:

- `startElementsSAX2Func` — вызывается, когда парсер находит начальный открывающий тег;
- `charactersSAXFunc` — вызывается (возможно более одного вызова для данного элемента) для извлечения символов внутри элемента;

- `endElementsSAXFunc` — вызывается, когда парсер находит конечный закрывающий тег.

4. Перед началом парсинга документа создайте контекст документа посредством вызова функции `xmlCreateDocParserCtxt()` и передачи С-строки, содержащей весь XML-документ, в качестве единственного аргумента.

5. Настроив функции-обработчики событий и SAX-дескриптор, вызовите функцию `xmlSAXUserParseMemory()` библиотеки `libxml2`, передавая следующие параметры:

- указатель на дескриптор;
- указатель на контекст; контекст может указывать на любой объект (например, Cocoa-объект);
- С-строку, представляющую XML-документ, и ее длину в байтах.

6. Парсер начнет обработку документа и генерацию событий. Если вы зарегистрировали в дескрипторе функцию для данного события, она будет вызвана.

7. После вызова `xmlSAXUserParseMemory()` освободите контекст с помощью вызова `xmlFreeParserCtxt()` и очистите парсер, вызвав `xmlCleanupParser()`.

Задачи

1. Существует несколько веб-сервисов, доступных на http://ws.geonames.org/. Полнотекстовый поиск по Википедии возвращает записи из «Википедии», найденные по запросу. Результат, выполненный данным веб-сервисом, возвращается в виде XML-документа. У этого веб-сервиса есть несколько параметров: в `q` вы указываете запрос, а в `maxRows` — максимальное количество возвращаемых записей. Полное описание веб-сервиса можно найти по адресу

www.geonames.org/export/wikipedia-webservice.html#wikipediaSearch.

Например, URL-запрос

http://ws.geonames.org/wikipediaSearch?q=plano,texas&maxRows=10

возвратит XML-документ с найденными записями. В листинге 11.18 частично представлен XML-документ.

Листинг 11.18. Пример XML-результата веб-сервиса поиска по «Википедии»

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<geonames>
<entry>
<lang>en</lang>
```

```
<title>Plano, Texas</title>
<summary>
Plano is a city in Collin and Denton Counties in
the US state of Texas. Located mainly within
Collin County, it is a wealthy northern suburb
of Dallas. The population was 222,030 at the
2000 census, making it the ninth largest city in
Texas. According to a 2005 census estimate,
Plano had grown to 250,096 making
Plano the sixty-ninth most populous city in the
United States (...)
</summary>
<feature>city</feature>
<countryCode>US</countryCode>
<population>245411</population>
<elevation>0</elevation>
<lat>33.0193</lat>
<lng>-96.7008</lng>
<wikipediaUrl>
http://en.wikipedia.org/wiki/Plano%2C_Texas
</wikipediaUrl>
<thumbnailImg/>
</entry>
.
.
.
</geonames>
```

Напишите iPhone-приложение, представляющее эту информацию в табличном виде.

2. XPath (XML Path Language — язык XML-путей) — это язык для выбора узлов в XML-документе. Более подробно о нем можно прочесть на www.w3.org/TR/xpath и узнать, как использовать библиотеку `libxml2` для XPath-операций.

Глава 12
Информация о местоположении

Эта глава посвящена теме информации о географическом местоположении. В разд. 12.1 мы поговорим о фреймворке Core Location и как с его помощью создавать приложения, позволяющие получить информацию о местоположении устройства. В разд. 12.2 мы рассмотрим простейшее приложение, предоставляющее информацию о местоположении. Из разд. 12.3 вы узнаете о геокодировании и научитесь транслировать почтовые адреса в географические координаты. Из разд. 12.4 вы узнаете, как распознавать движение устройства и отображать эту информацию на картах. В разд. 12.5 мы обсудим, как соотносить почтовые индексы с географической информацией. В этом разделе вы также ознакомитесь с формулой, вычисляющей расстояние между двумя локациями.

12.1. Фреймворк Core Location

Второе поколение iPhone (iPhone 3G) оборудовано микросхемой системы глобального позиционирования (Global Positioning System (GPS)). GPS использует три или четыре спутника для вычисления позиции точки на поверхности Земли методом триангуляции. Точность вычисления с использованием данной технологии составляет 5—40 метров.

Первое поколение iPhone не использует GPS-методов для определения местоположения устройства. Эти методы, такие как идентификация соты, время прибытия (Time of Arrival, TOA) и улучшенное вычисление разницы наблюдаемого времени (Enhanced Observed Time Difference, EOT-D), можно использовать в сочетании с Wi-Fi и Bluetooth, чтобы получить замену отсутствующей микросхемы GPS [3]. Погрешность измерения местоположения будет значительно больше, чем у GPS, и составит 100—500 метров.

Вне зависимости от используемого метода iPhone предоставляет фреймворк Core Location [4] в качестве программного интерфейса, позволяющего применить любой способ определения местоположения. Фреймворк предоставляет классы и протоколы, которые можно использовать как для нахождения текущего местоположения с заданной точностью, так и для планирования будущих обновлений информации о текущем местоположении.

Главный класс фреймворка Core Location — `CLLocationManager`. Это отправная точка, используемая обработчиком для получения текущей

и будущей информации о местоположении. Используйте экземпляр CLLocationManager для планирования будущего обновления текущего местоположения устройства.

Чтобы получить доступ к информации о текущем местоположении, выполните следующие действия.

1. Создайте экземпляр CLLocationManager, если он еще не существует.

2. Настройте экземпляр CLLocationManager. Вам требуется сконфигурировать экземпляр диспетчера с помощью следующих параметров:

- desiredAccuracy — посредством данного свойства вы сообщаете фреймворку требования о точности вычисления местоположения (в метрах). Свойство desiredAccuracy объявлено следующим образом:

```
@property(assign, nonatomic) CLLocationAccuracy desiredAccuracy
```

Разные приложения требуют различной точности. Фреймворк старается определить местоположение с точностью, соответствующей значению данного свойства, но он не может гарантировать это. Есть несколько значений, из которых вы можете выбрать:

— kCLLocationAccuracyBest — задает максимальную точность и является значением по умолчанию;

— kCLLocationAccuracyNearestTenMeters — определяет точность в пределах 10 метров;

— kCLLocationAccuracyHundredMeters — задает точность в пределах 100 метров;

— kCLLocationAccuracyKilometer — определяет точность в пределах 1000 метров;

— kCLLocationAccuracyThreeKilometers — задает точность в пределах трех километров;

- distanceFilter — значение данного свойства определяет частоту получения обновлений информации о местоположении. Вы будете получать обновление, только если устройство переместится на дистанцию, большую или равную указанному значению. Если доступно более точное определение, это значение будет игнорироваться, а вы будете чаще получать обновления. Свойство объявлено следующим образом:

```
@property(assign, nonatomic) CLLocationDistance distanceFilter
```

где CLLocationDistance объявлено как

```
typedef double CLLocationDistance
```

Все значения задаются в метрах. Если вы укажете kCLDistanceFilterNone, то будете получать обновления для всех передвижений устройства;

- delegate — это свойство определяет объект делегата, получающий обновления. Свойство объявлено следующим образом:

```
@property(assign, nonatomic)id<CLLocationManagerDelegate> delegate
```

Делегат реализует протокол CLLocationManagerDelegate. Этот протокол содержит два необязательных метода. Первый — это метод locationManager:didUpdateToLocation:fromLocation:. Он вызывается в любой момент, когда диспетчер локаций хочет предоставить вам обновленную информацию. Метод объявлен следующим образом:

```
-(void)locationManager:(CLLocationManager *)manager
    didUpdateToLocation:(CLLocation *)newLocation
    fromLocation:(CLLocation *)oldLocation
```

В первом параметре вы получаете ссылку на диспетчер локаций. Второй параметр — это экземпляр класса CLLocation, инкапсулирующего местоположение, а третий — еще один, возможно nil, объект CLLocation, содержащий предыдущее местоположение.

Второй метод — locationManager:didFailWithError:. Этот метод делегата вызывается, если диспетчер не смог вычислить текущее местоположение, и объявлен следующим образом:

```
-(void)locationManager:(CLLocationManager *)manager
    didFailWithError:(NSError *)error
```

Вы должны реализовать класс, заимствующий протокол CLLocationManagerDelegate, и связать экземпляр этого класса со свойством делегата экземпляра CLLocationManager.

3. Вызовите startUpdatingLocation. Чтобы получать обновления информации о местоположении, вызовите метод startUpdatingLocation экземпляра CLLocationManager.

4. Вызовите stopUpdatingLocation. Вы должны вызвать stopUpdatingLocation, как только вам больше не понадобится новая информация о текущем местоположении.

Приведенные выше шаги — это основы использования локационных сервисов.

Класс CLLocation. Широта и долгота определяют логическую сеткообразную систему мира. Данные параметры были разработаны и реализованы для определения метоположения на поверхности Земли. Линии широты параллельны и находятся на одинаковом расстоянии друг от друга. Экватор — это 0°, а Северный и Южный полюса — 90; 1° равен приблизительно 69 милям (111 км). Линии долготы проходят от севера к югу. Диапазон долготы — 0—180° (восточной или западной долготы).

Чтобы определить точку на поверхности Земли, вы можете описать ее парой координат (широта и долгота), например 33°1'12", –96°44'19.67". Формат «градус-минута-секунда» можно конвертировать в десятичный формат. Предыдущее положение можно записать в десятичной форме как 33,02, –96,7388.

Местоположение устройства инкапсулируется классом `CLLocation`, который содержит географическую позицию устройства, представленную широтой и долготой. В дополнение он предоставляет высоту устройства над уровнем моря, а также различные значения, описывающие единицы измерения положения. Вы получаете объекты данного типа от диспетчера локаций.

Далее приведены некоторые наиболее важные свойства данного класса:

- `coordinate` — широта и долгота местоположения устройства в градусах. Свойство объявлено следующим образом:

```
@property(readonly, nonatomic) CLLocationCoordinate2D coordinate
```

`CLLocationCoordinate2D` — это структура, объявленная как

```
typedef struct {
    CLLocationDegrees latitude;
    CLLocationDegrees longitude;
} CLLocationCoordinate2D;
```

где `CLLocationDegrees` имеет тип `double`.

- `altitude` — возвращает высоту над уровнем моря в метрах. Положительные значения указывают позицию выше уровня моря, отрицательные — ниже его. Свойство объявлено следующим образом:

```
@property(readonly, nonatomic) CLLocationDistance altitude
```

- `horizontalAccuracy` — если вы представите, что широта и долгота — это координаты центра круга, а `horizontalAccuracy` — его радиус, то устройство может находиться в любой точке данного круга. Свойство объявлено следующим образом:

```
@property(readonly, nonatomic) CLLocationAccuracy horizontalAccuracy
```

Оно имеет тип `CLLocationAccuracy`, который определен как `double`. Отрицательное значение показывает неверное боковое положение.

- `verticalAccuracy` — предоставляет вертикальную точность местоположения. Высота над уровнем моря находится в диапазоне +/– этого значения. Свойство объявлено как

```
@property(readonly, nonatomic) CLLocationAccuracy verticalAccuracy
```

Отрицательные значения говорят о неверном определении высоты над уровнем моря.

- `timestamp` — показывает время последнего определения местоположения. Свойство объявлено следующим образом:

```
@property(readonly, nonatomic) NSDate *timestamp
```

Большинство времени вы получаете объекты `CLLocation` от диспетчера локаций. Чтобы кэшировать объекты данного типа, нужно выделить память под новый объект и инициализировать его. Можете использовать один из двух методов инициализации.

- `initWithLatitude:longitude:`, объявленный следующим образом:

```
-(id)initWithLatitude:(CLLocationDegrees)latitude
    longitude:(CLLocationDegrees)longitude
```

- `initWithCoordinate:altitude:horizontalAccuracy:verticalAccuracy:timestamp:`, объявленный как

```
-(id)initWithCoordinate:(CLLocationCoordinate2D)coordinate
    altitude:(CLLocationDistance)altitude
    horizontalAccuracy:(CLLocationAccuracy)hAccuracy
    verticalAccuracy:(CLLocationAccuracy)vAccuracy
    timestamp:(NSDate *)timestamp;
```

Есть еще один метод, который может быть полезен для нахождения расстояния (в метрах) от одного заданного положения до другого. Это метод `getDistanceFrom:`, который возвращает расстояние между локацией, переданной в первом параметре, и положением, инкапсулированным в приемнике. Метод объявлен следующим образом:

```
-(CLLocationDistance) getDistanceFrom:(const CLLocation *)location
```

Позже из этой главы вы узнаете, как можно реализовать и использовать данный метод в связке с механизмом баз данных.

12.2. Простейшее приложение, предоставляющее информацию о местоположении

Начнем раздел с примера простой программы, определяющей местоположение. Приложение сконфигурирует диспетчер локаций и будет отображать информацию в текстовом представлении. Для простоты реализуем весь функционал программы в одном классе — делегате приложения.

Листинг 12.1 демонстрирует объявление класса делегата приложения. Класс оперирует ссылками на текстовое представление и диспетчер локаций. Переменная экземпляра `noUpdates` используется для подсчета количества обновлений, полученных делегатом приложения. Мы прекратим получать обновления информации о местоположении, когда получим

10 обновлений. Обратите внимание, что мы добавили новую директиву `#import` для фреймворка Core Location.

Листинг 12.1. Объявление класса делегата приложения, используемого в простейшей программе, предоставляющей информацию о местоположении

```
#import <UIKit/UIKit.h>
#import <CoreLocation/CoreLocation.h>

@interface Location1AppDelegate : NSObject
<UIApplicationDelegate, CLLocationManagerDelegate> {
    UIWindow        *window;
    UITextView      *textView ;
    CLLocationManager  *locationMgr;
    NSUInteger      noUpdates;
}
@property (nonatomic, retain) UIWindow *window;
@end
```

Листинг 12.2 показывает реализацию класса делегата приложения. Метод `applicationDidFinishLaunching:` настраивает текстовое представление и добавляет его в качестве дочернего к главному окну. Создается экземпляр диспетчера локаций, и его свойство `delegate` настраивается на экземпляр делегата приложения. Диспетчер локаций запускается, а главное окно делается видимым.

Обновления получаются с помощью метода `locationManager: didUpdateToLocation:fromLocation:` протокола `CLLocationmanagerDelegate`. В нашей реализации данного метода мы просто объединяем текст, находящийся в текстовом представлении, с описанием нового объекта локации. Затем свойство `text` текстового представления устанавливается в данное значение. Когда получены 10 обновлений, диспетчер локаций останавливается посредством вызова метода `stopUpdatingLocation`.

Листинг 12.2. Реализация класса делегата приложения, используемого в простейшей программе, предоставляющей информацию о местоположении

```
#import "Location1AppDelegate.h"
@implementation Location1AppDelegate
@synthesize window;
-(void)locationManager:(CLLocationManager *)manager
    didUpdateToLocation:(CLLocation *)newLocation
           fromLocation:(CLLocation *)oldLocation{
    noUpdates++;
    if(noUpdates >= 10){
           [locationMgr stopUpdatingLocation];
    }
    [self updateLocation:[newLocation description]];
}
-(void) updateLocation:(NSString*) update{
    NSMutableString *newMessage =
           [[NSMutableString alloc] initWithCapacity:100];
    [newMessage appendString:
```

```
            [NSString stringWithFormat:@"Update #:%i\n", noUpdates]];
    [newMessage appendString:update];
    [newMessage appendString:@"\n"];
    [newMessage appendString:[textView text]];
    textView.text = newMessage;
    [newMessage release];
}
-(void)applicationDidFinishLaunching:(UIApplication *)application {
    window = [[UIWindow alloc]
            initWithFrame:[[UIScreen  mainScreen] bounds]];
    CGRect  rectFrame = [UIScreen mainScreen].applicationFrame;
    textView = [[UITextView alloc] initWithFrame:rectFrame];
    textView.editable = NO;
    locationMgr = [[CLLocationManager alloc] init];
    locationMgr.delegate = self;
    noUpdates = 0;
    [locationMgr startUpdatingLocation];
    [window addSubview:textView];
    [window makeKeyAndVisible];
}
-(void)dealloc {
    [textView release];
    [locationMgr release];
    [window release];
    [super dealloc];
}
@end
```

Чтобы успешно скомпилировать данный код, вам нужно добавить ссылку на библиотеку Core Lcoation. В XCode выполните команду меню Project (Проект) ▸ Edit Active Target (Добавить активную цель) (рис. 12.1).

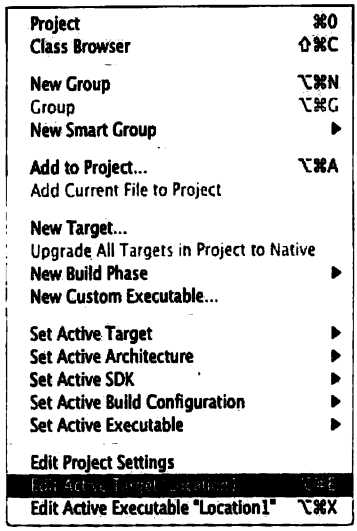

Рис. 12.1. Меню Project (Проект) ▸ Edit Active Target (Добавить активную цель)

Появится окно Target Info (Окно цели) (рис. 12.2).

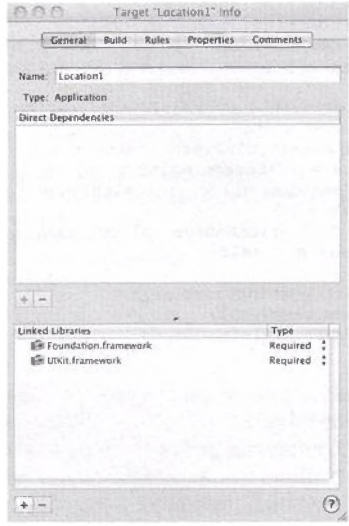

Рис. 12.2. Окно Target Info (Окно цели)

Нажмите кнопку с изображением плюса (внизу слева). Появится список библиотек. Прокрутите его и найдите CoreLocation.framework (рис. 12.3).

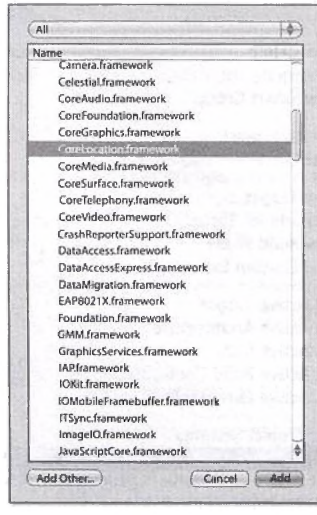

Рис. 12.3. Библиотека CoreLocation.framework

Выберите этот пункт и нажмите **Add** (Добавить). Ниже (рис. 12.4) изображена добавленная библиотека `CoreLocation.framework` в разделе **Linked Libraries** (Связанные библиотеки).

Рис. 12.4. Добавленная библиотека Core Location в разделе Linked Libraries (Связанные библиотеки)

На иллюстрации далее изображен снимок экрана приложения после получения 10 обновлений позиции (рис. 12.5).

Рис. 12.5. Снимок экрана приложения после получения 10 обновлений локаций

12.3. Google Maps API

Google предоставляет HTTP-интерфейс для геокодирования. Геокодирование — это процесс трансляции, при котором адреса вида `3400 W Plano Pkwy, Plano, TX 75075` могут конвертироваться в реальные географические координаты широты и долготы. Для доступа к данному сервису клиент посылает на http://maps.google.com/maps/geo? HTTP-запрос со следующими параметрами:

- `q` — представляет адрес, для которого вы хотите вычислить геоданные;
- `output` — формат результата, который будет отослан вам. Есть несколько таких форматов, как XML и CSV. Значения, разделенные запятой (CSV), наиболее просты для работы;
- `key` — чтобы получить доступ к сервису, вам требуется получить от Google API-ключ. На момент написания книги использование сервиса было бесплатным для общественных сайтов. Тем не менее Google поддерживает защиту сервиса.

Например, вы послали такой HTTP-запрос:

```
http://maps.google.com/maps/geo?q=3400+W+Plano+Pkwy+Plano,
+TX+75075&output=csv&

key=ABQIAAAAERNgBiSqUogvAN307LdVDxSkQMtcTv75TNsQ97
PejimT5pm-BxST0Gma_YCBaUccn3pRis8XjkxM8w
```

Вам возвратится, при условии использования вашего собственного ключа, следующее:

```
200,8,33.010003,-96.757923
```

Если вы используете CSV-формат, то получите от Google четыре значения, разделенных запятой. Первое значение — это статус-код HTTP-протокола; `200` означает OK (для получения более подробной информации см. RFC 2616). Второй параметр — это точность; значение `8` означает `Address Level Accuracy` (полный список значений вы найдете в GGeoAddressAccuracy в документации по Google Maps API). Последние два значения результата, которые интересуют нас больше всего, соответственно, широта и долгота.

Геокодирующее приложение. Рассмотрим приложение, вычисляющее расстояние между двумя адресами. Сначала мы создадим вспомогательный класс `GoogleMapsWrapper`, инкапсулирующий сервис геокодинга. После этого мы покажем делегат приложения, использующий этот вспомогательный класс для отображения расстояния между двумя адресами с помощью текстового представления.

Листинг 12.3 демонстрирует объявление класса GoogleMapsWrapper. Основной метод, объявленный в этом классе, — findGeoInfoForAddress:andReturnLatitude:andLongitude:. В качестве входного параметра этому методу передается адрес, а в качестве выходных параметров возвращаются широта и долгота. Успешный код возврата — 0.

Класс GoogleMapsWrapper также оперирует множеством ключей Google Maps. Эти ключи можно использовать для балансировки нагрузки HTTP-запросов, так как Google ограничивает количество запросов на один ключ. Для сохранения ключа используется метод addKey:, а для извлечения ключа — метод getKey.

Листинг 12.3. Объявление класса GoogleMapsWrapper, используемого в приложении для геокодинга

```
#import <UIKit/UIKit.h>
@interface GoogleMapsWrapper : NSObject {
    NSMutableArray *keys;
}
-(id)init;
-(void)addKey:(NSString*) key;
-(NSString*)getKey;
-(int)findGeoInfoForAddress:(NSString*)address
    andReturnLatitude:(float*) latitude andLongitude:(float*) longitude;
@end
```

Листинг 12.4 показывает реализацию класса GoogleMapsWrapper. Метод findGeoInfoForAddress:andReturnLatitude:andLongitude: сначала создает запрос, который будет использован в HTTP-запросе. После создания объекта NSURL для запроса он соединяется с Google посредством вызова метода initWithContentsOfURL: класса NSData.

Ответ на запрос приходит в формате, разделенном запятыми (CSV), который нужно обработать. Чтобы извлечь четыре значения из запроса, мы используем Cocoa-класс NSScanner. Этот класс применяется для поиска по объекту NSString строк и чисел путем последовательного поиска во всей строке. В дополнение вы можете использовать его для пропуска символов, указанных во множестве. Ответ на запрос сначала конвертируется в объект NSString, contents. После этого создается объект NSScanner посредством вызова метода scannerWithString: класса NSScanner, которому передается данная строка.

Сначала нужно проверить значение статус-кода полученного ответа. Чтобы получить статус-код, мы просим сканер извлечь все символы, начиная с текущей позиции (начала строки) и заканчивая первой запятой. Как только мы получили статус-код в строке statusCode, мы проверяем, равен ли он 200. Если мы получили неравенство, то возвращаем –1, что означает ошибку. Если статус-код равен 200, мы получаем значения широты и долготы, используя метод scanFloat:.

Класс `GoogleMarsWrapper` не реализует балансировки загрузки (например, простой случайный алгоритм). В нашем приложении мы используем только один ключ.

Листинг 12.4. Реализация класса GoogleMapsWrapper, используемого в приложении для геокодинга

```
#import "GoogleMapsWrapper.h"
#define GEO_QUERY     @"http://maps.google.com/maps/geo?q="
#define GEO_CSV_KEY   @"&output=csv&key="
@implementation GoogleMapsWrapper
-(int)findGeoInfoForAddress:(NSString*)address
    andReturnLatitude:(float*) latitude
    andLongitude:(float*) longitude {
    if(!address || !latitude || !longitude){
            return -1;
    }
    NSMutableString *query =
            [[NSMutableString  alloc] initWithString:GEO_QUERY];
    [query appendString:address];
    [query appendString:GEO_CSV_KEY];
    [query appendString:[self getKey]];
    [query replaceOccurrencesOfString:@" "
                    withString:@"%20" options:NSLiteralSearch
                    range:NSMakeRange(0, [query length])];
    NSURL *url= [[NSURL alloc] initWithString:query];
    if (!url){
                    [query release];
                    return -1;
    }
    NSData *data = [[NSData alloc] initWithContentsOfURL:url];
    if(!data){
                    [query release];
                    [url release];
                    *latitude = *longitude = 404;
                    return -1;
    }
    NSString *contents = [[NSString alloc] initWithData:data
            encoding:NSUTF8StringEncoding];
    if(!contents){
            [query release];
            [url release];
            [data release];
            return -1;
    }
    /* Ответ, возвращенный в csv-формате, состоит из четырех значений,
            разделенных запятой:
            HTTP статус-код
            точность (см.константы точности)
            широта
            долгота
            пример: 200,6,42.730070, -73.690570 */
    NSScanner *theScanner;
    NSCharacterSet *comma  =
            [NSCharacterSet characterSetWithCharactersInString:@","];
    NSString  *statusCode;
    theScanner = [NSScanner scannerWithString:contents];
```

```
    if([theScanner scanUpToCharactersFromSet:comma
            intoString:&statusCode]){
            if([statusCode intValue] != 200){
                    *latitude = *longitude = 404;
                    [query release];
                    [url release];
                    [data release];
                    [contents release];
                    return -1;
            }
    }
    if(
            [theScanner scanCharactersFromSet:comma intoString:NULL] &&
            [theScanner scanUpToCharactersFromSet:comma intoString:NULL] &&
            [theScanner scanCharactersFromSet:comma intoString:NULL] &&
            [theScanner scanFloat:latitude] &&
            [theScanner scanCharactersFromSet:comma  intoString:NULL] &&
            [theScanner scanFloat:longitude] ){
            [query release];
            [url release];
            [data release];
            [contents release];
            return 0;
    }
    [query release];
    [url release];
    [data release];
    [contents release];
    return -1;
}
-(NSString*)getKey{
    if([keys count] < 1){
            return @"NULL_KEY";
    }
    return [keys objectAtIndex:0];
}
-(void)addKey:(NSString*) key {
[keys addObject:key];
}
-(id)init{
    self = [super init];
    keys = [[NSMutableArray arrayWithCapacity:1] retain];
    return self;
}
-(void)dealloc{
    [keys release];
    [super dealloc];
}
@end
```

Листинг 12.5 демонстрирует объявление класса делегата приложения. Как уже было сказано, делегат приложения будет создавать главное окно и присоединять к нему текстовое представление. Затем он использует класс `GoogleMapsWrapper` для нахождения расстояния между двумя адресами и вывода информации пользователю в текстовом представлении (рис. 12.6). Класс оперирует ссылками на классы `UITextView` и `GoogleMapsWrapper`.

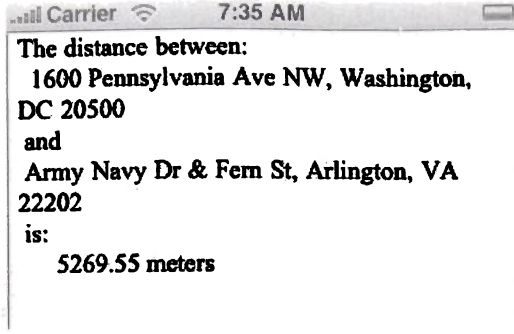

Рис. 12.6. Снимок экрана геокодирующего приложения, показывающего расстояние между двумя почтовыми адресами

Листинг 12.5. Объявление класса делегата приложения, используемого в программе для геокодинга

```
#import <UIKit/UIKit.h>
#import <CoreLocation/CoreLocation.h>
#import "GoogleMapsWrapper.h"
@interface Location2AppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    UITextView *textView;
    GoogleMapsWrapper *gWrapper;
}
@property (nonatomic, retain) UIWindow *window;
@end
```

Листинг 12.6 показывает реализацию класса делегата приложения. Метод `applicationDidFinishLaunching:` настраивает пользовательский интерфейс и вызывает метод `findDistanceFromAddress:toAddress:`, чтобы найти расстояние между Белым домом и Пентагоном. Затем результат форматируется и связывается со свойством `text` объекта текстового представления.

Листинг 12.6. Реализация класса делегата приложения, используемого в программе для геокодинга

```
#import "Location2AppDelegate.h"
#import "GoogleMapsWrapper.h"
#define FROM_ADDRESS @"1600 Pennsylvania Ave NW, Washington, DC 20500"
#define TO_ADDRESS @"Army Navy Dr & Fern St, Arlington, VA 22202"
@implementation Location2AppDelegate
@synthesize window;
-(double) findDistanceFromAddress:(NSString*) from
    toAddress:(NSString *) to{
    float lat, lon;
    CLLocation *fromLocation;
    CLLocation *toLocation;
    if([gWrapper findGeoInfoForAddress:from
            andReturnLatitude:&lat andLongitude:&lon] == 0){
    fromLocation = [[[CLLocation alloc]
            initWithLatitude:lat longitude:lon] autorelease];
    if([gWrapper findGeoInfoForAddress:to andReturnLatitude:&lat
```

```
            andLongitude:&lon] == 0){
            toLocation =[[[CLLocation alloc]
              initWithLatitude:lat longitude:lon] autorelease];
            return [toLocation getDistanceFrom:fromLocation];
            }
            return -1;
    }
    return -1;
}
-(void)applicationDidFinishLaunching:(UIApplication *)application {
    window = [[UIWindow alloc] initWithFrame:[[UIScreen  mainScreen] bounds]];
    CGRect  rectFrame = [UIScreen mainScreen].applicationFrame;
    textView = [[UITextView alloc] initWithFrame:rectFrame];
    textView.editable = NO;
    gWrapper = [[GoogleMapsWrapper alloc] init];
    [gWrapper addKey:@"ABQIAAAAERNgBiSqUogvAN307LdVDx"
            "SkQMtcTv75TNsQ97PejimT5pm-MAxST0Gma_Y"
            "CBaUccn3pRis8XjkxM8w"];
    NSMutableString *outStr = [[NSMutableString alloc]
            initWithFormat:@"The distance between: \n %@ \n and"
            "\n %@ \n is:\n \t \t %.2f meters\n", FROM_ADDRESS, TO_ADDRESS,
            [self findDistanceFromAddress:FROM_ADDRESS
                    toAddress:TO_ADDRESS]];
    textView.text = outStr;
    [window addSubview:textView];
    [window makeKeyAndVisible];
}
-(void)dealloc {
    [gWrapper release];
    [textView release];
    [window release];
    [super dealloc];
}
@end
```

12.4. Отслеживающее приложение с картами местности

Многие iPhone-приложения требуют отображения карты местности. Если вы столкнулись с задачей разработки подобной программы, можете использовать Google Maps API и класс `UIWebView`.

В этом разделе мы разработаем отслеживающее приложение. Сначала программа будет отслеживать и накапливать настраиваемое количество движений устройства. Пользователь может прервать отслеживание или подождать, пока не запишется заданное количество передвижений. В ином случае пользователь сможет пройтись по записям передвижений и визуализировать (на карте) географическое положение и время каждой записи передвижения.

Листинг 12.7 показывает объявление класса делегата приложения. Делегат приложения будет иметь контроллер навигации для пользовательского интерфейса и, таким образом, будет оперировать ссылками на представление и контроллер навигации.

Листинг 12.7. Объявление класса делегата отслеживающего приложения

```
#import <UIKit/UIKit.h>
#import "LocationsViewController.h"
```

```
@interface Location3AppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    LocationsViewController *ctrl;
    UINavigationController *navCtrl;
}
@property (nonatomic, retain) UIWindow *window;
@end
```

Листинг 12.8 демонстрирует реализацию класса делегата приложения. Метод `applicationDidFinishLaunching:` просто создает контроллер представления типа `LocationsViewController` и использует его в качестве корневого для контроллера навигации. Затем представление контроллера навигации добавляется в качестве дочернего к главному окну, после чего последнее делается видимым.

Листинг 12.8. Реализация класса делегата отслеживающего приложения

```
#import "Location3AppDelegate.h"
@implementation Location3AppDelegate
@synthesize window;
-(void)applicationDidFinishLaunching:
(UIApplication *)application { window = [[UIWindow alloc]
    initWithFrame:[[UIScreen mainScreen] bounds]];
    ctrl = [[LocationsViewController alloc]
            initWithNibName:nil bundle:nil];
    navCtrl = [[UINavigationController alloc]
            initWithRootViewController:ctrl];
    [window addSubview:navCtrl.view];
    [window makeKeyAndVisible];
}
-(void)dealloc {
    [ctrl release];
    [navCtrl release];
    [window release];
    [super dealloc];
}
@end
```

В листинге 12.9 объявлен наш контроллер представления. Он заимствует `CLLocationManagerDelegate`, так как будет делегатом созданного им диспетчера локаций. Он объявляет две кнопки на панели навигации для остановки отслеживания передвижений — перехода к следующей и предыдущей записям. Правая кнопка панели навигации будет использоваться как для остановки отслеживания, так и в качестве кнопки Next (Далее). В дополнение контроллер представления оперирует ссылкой на веб-представление для визуализации отслеженных локаций.

Листинг 12.9. Объявление класса контроллера представления LocationsViewController, используемого в отслеживающем приложении

```
#import <UIKit/UIKit.h>
#import <CoreLocation/CoreLocation.h>
@interface LocationsViewController :
    UIViewController <CLLocationManagerDelegate> {
    CLLocationManager  *locationMgr;
    NSUInteger  noUpdates;
```

```
    NSMutableArray  *locations;
    UIWebView  *webView;
    UIBarButtonItem  *rightButton, *leftButton;
    NSUInteger  current;
}
@end
```

Листинг 12.11 показывает реализацию контроллера представления. В методе инициализации `initWithNibName:bundle:` мы создаем две кнопки. Правая кнопка называется **Stop** (Стоп), левая — **Previous** (Предыдущий). Левая кнопка сделана недоступной.

Метод `loadView` создает и настраивает диспетчер локаций. Расстояние, которое требуется обновить, устанавливается равным `MIN_DISTANCE`. В дополнение создается веб-представление. Оно инициализируется с помощью содержимого HTML-файла, который хранится в упаковке приложения. Файл `map3.html` показан в листинге 12.10. Этот файл — один из многих файлов-примеров, которые демонстрируют использование Google Maps API, предоставленных Google. Как вы увидите далее, мы будем использовать JavaScript для динамической модификации внешнего вида карты местности.

Листинг 12.10. HTML-страница, используемая для отображения карты Google для приложения геослежения

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:v="urn:schemas-microsoft-com:vml">
  <head>
    <meta http-equiv="content-type" content="text/html;
          charset=utf-8"/>
    <title>Geo-tracking Example</title>
    <script src="http://maps.google.com/maps?file=api&amp;v=2&amp;key=K"
          type="text/JavaScript">
    </script>
    <script type="text/JavaScript">
    function initialize() {}
    </script>
  </head>
  <body onload="initialize()" onunload="GUnload()">
    <div id="map_canvas" style="width: 500px; height: 500px">
    </div>
  </body>
</html>
```

Получая обновления информации о местоположении, мы сохраняем эти локации в массиве. Получив `NO_OF_LOCATIONS`, мы делаем доступной левую кнопку панели, изменяем ее заголовок на **Next** (Далее) и указываем первую локацию на карте.

Метод `centerMap:` используется для отображения локации на карте. В качестве входных параметров методу передается порядковый номер локации в массиве отслеженных местоположений. Он извлекает информацию о широте и долготе, устанавливает центр карты в этой точке и центрирует карту. В дополнение он открывает информационное окно, отображающее время, когда местоположение было отслежено. Все это делается с помощью JavaScript, как показано ниже. Наконец, мы выполняем код

JavaScript посредством метода веб-представления `stringByEvaluating JavaScriptFromString:`.

```
var map = new GMap2(document.getElementById("map_canvas"));
map.setMapType(G_HYBRID_MAP);
map.setCenter(new GLatLng(37.331689, -122.030731), 18);
map.panTo(map.getCenter());
map.openInfoWindow(map.getCenter(),
        document.createTextNode("Loc: (1/1),
Time: 2008-08-06 19:51:27 -0500"));
```

Ниже изображен снимок экрана отслеживающего приложения во время записи передвижений (рис. 12.7), а также во время просмотра одной из записанных локаций (рис. 12.8).

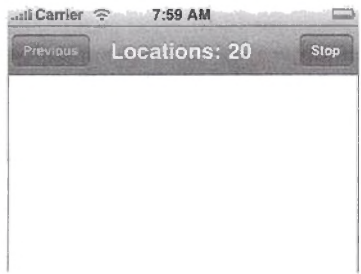

Рис. 12.7. Снимок экрана отслеживающего приложения во время записи передвижений

Рис. 12.8. Снимок экрана отслеживающего приложения во время просмотра локации

Приложение сталкивается с некоторыми этическими проблемами, а может, и вопросами легальности. Если вы захотите запустить данное приложение и спрятать устройство в чьей-нибудь сумке или автомобиле, лишний раз подумайте. Незаконная слежка не только неэтична, но и может привести к серьезным последствиям вплоть до тюрьмы. Можно модифицировать приложение и добавить формирующиеся в реальном времени отчеты о передвижениях, доставляемые заинтересованным лицам. Оставим это вам в качестве упражнения.

Листинг 12.11. Реализация класса контроллера представления LocationsViewController, используемого в отслеживающем приложении

```
#import "LocationsViewController.h"
#define NO_OF_LOCATIONS 100
#define MIN_DISTANCE  100
@implementation LocationsViewController
-(void)locationManager:(CLLocationManager *)manager
    didUpdateToLocation:(CLLocation *)newLocation
           fromLocation:(CLLocation *)oldLocation{
    noUpdates++;
    [locations addObject:newLocation];
    self.title = [NSString
           stringWithFormat:@"Locations: %i", noUpdates];
    if(noUpdates == 1){
           [self centerMap:0];
    }
    if (noUpdates >= NO_OF_LOCATIONS){
           [locationMgr stopUpdatingLocation];
           leftButton.enabled = YES;
           rightButton.title = @"Next";
           current = 0;
           [self centerMap:current];
    }
}
-(void) centerMap:(NSUInteger) index{
    CLLocation *loc = [locations objectAtIndex:index];
    NSString *js = [NSString stringWithFormat:
           @"var map = "
           "new GMap2(document.getElementById(\"map_canvas\"));"
           "map.setMapType(G_HYBRID_MAP);"
           "map.setCenter(new GLatLng(%lf, %lf), 18);"
           "map.panTo(map.getCenter());"
           "map.openInfoWindow(map.getCenter(), "
           "document.createTextNode(\"Loc: (%i/%i), Time: %@\"));",
    [loc coordinate].latitude, [loc coordinate].longitude,
    index+1, [locations count], [loc timestamp]];
    [webView stringByEvaluatingJavaScriptFromString:js];
}
-(id)initWithNibName:(NSString *)nibNameOrNil
    bundle:(NSBundle *)nibBundleOrNil {
    if (self=[super initWithNibName:nibNameOrNil bundle:nibBundleOrNil]){
           rightButton = [[UIBarButtonItem alloc]
             initWithTitle:@"Stop" style:UIBarButtonItemStyleDone
             target:self action:@selector(stopOrNext)];
             self.navigationItem.rightBarButtonItem = rightButton;
           leftButton = [[UIBarButtonItem alloc]
             initWithTitle:@"Previous"
             style:UIBarButtonItemStyleDone target:self
             action:@selector(prev)];
           self.navigationItem.leftBarButtonItem = leftButton;
           leftButton.enabled = NO;
    }
    return self;
```

```
}
-(void)stopOrNext{
    if([rightButton.title isEqualToString:@"Stop"] == YES){
            [locationMgr stopUpdatingLocation];
            leftButton.enabled = YES;
            rightButton.title = @"Next";
            current = 0;
            [self centerMap:current];
    }
    else
    if(current < ([locations count]-1)){
      [self centerMap:++current];
    }
}
-(void)prev{
    if(current > 0 && (current < [locations  count])){
      current = current -1;
      [self centerMap:current];
    }
}
-(void)loadView {
    locations = [[NSMutableArray arrayWithCapacity:10] retain];
    locationMgr = [[CLLocationManager alloc] init];
    locationMgr.distanceFilter = MIN_DISTANCE;
    locationMgr.delegate = self;
    noUpdates = 0;
    CGRect  rectFrame = [UIScreen mainScreen].applicationFrame;
    webView = [[UIWebView alloc] initWithFrame:rectFrame];
    NSString *htmlFilePath =
            [[NSBundle mainBundle]
                    pathForResource:@"map3" ofType:@"html"];
    NSData  *data = [NSData dataWithContentsOfFile:htmlFilePath];
    [webView loadData:data MIMEType:@"text/html"
            textEncodingName:@"utf-8"
            baseURL:[NSURL URLWithString:@"http://maps.google.com/"]];
    [locationMgr startUpdatingLocation];
    self.view = webView;
}
-(void)dealloc {
    [rightButton release];
    [leftButton release];
    [locationMgr release];
    [locations release];
    [super dealloc];
}
@end
```

12.5. Работа с почтовыми индексами

Почтовая служба Соединенных Штатов (United States Postal Service (USPS)) использует кодирующую систему для эффективной доставки почты в пределах США. Подразумевается, что каждый потенциальный получатель почты живет в определенной зоне, представленной почтовым индексом (рис. 12.9). Теоретически почтовые индексы привязаны к географическому местоположению.

Существуют различные доступные базы данных почтовых индексов. Они отличаются точностью и ценой. Можно представить базы данных, соотносящие широту и долготу к данному почтовому индексу, как описывающие центр области, обслуживающей данный почтовый индекс. Суще-

ствует несколько мест, где вы можете приобрести базу данных почтовых индексов Соединенных Штатов. Вы даже можете бесплатно загрузить актуальную базу с сайта [1].

Содержимое файла почтовых индексов США [1] разделено запятыми. Например, последние несколько записей в файле выглядят следующим образом:

```
89508,Reno,NV,39.5296329,-119.8138027,Washoe
91008,Duarte,CA,34.1394513,-117.9772873,Los Angeles
92058,Oceanside,CA,33.1958696,-117.3794834,San Diego
94505,Discovery Bay,CA,37.9085357,-121.6002291,Contra Costa
95811,Sacramento,CA,38.5815719,-121.4943996,Sacramento
```

Чтобы ответить на запросы типа «получить почтовые индексы в радиусе 10 миль от 68508», необходимо выполнить следующие действия.

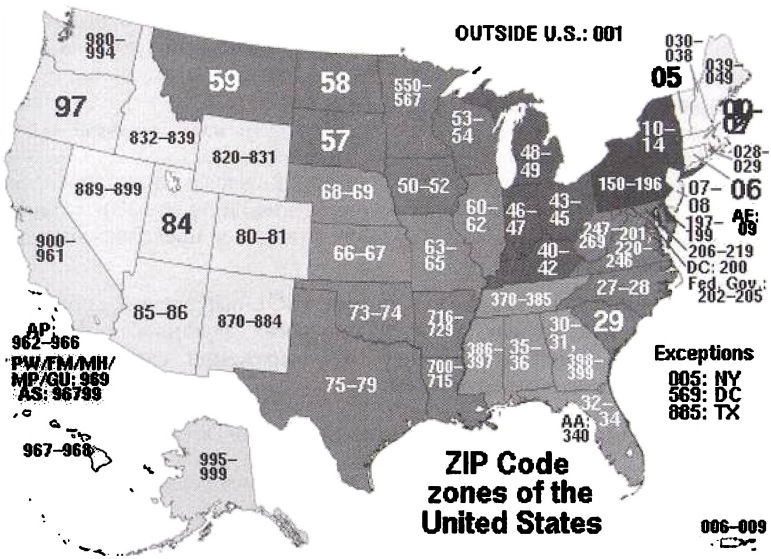

Рис. 12.9. Зоны почтовых индексов Соединенных Штатов [2]

1. Создайте SQLite-таблицу `zipcodes`. Для эффективного поиска рекомендуется поместить данные в базу. Для хранения почтовых индексов можно использовать следующую таблицу:

```
CREATE TABLE zipcodes (
    zipcode int NOT NULL PRIMARY KEY, latitude float(10,8),
    longitude float(10,8), state varchar(2),
    city varchar(128), county varchar(128))
```

Здесь `zipcode` будет первичным ключом, и для каждого почтового индекса у нас будут поля — `latitude`, `longitude`, `state`, `city` и `county`.

2. Заполните таблицу zipcodes географическими данными почтовых индексов, полученных из текстового файла. Данные хранятся в ASCII-файле и разделены точкой. Используйте объект NSScanner для извлечения значений. Полученные значения для каждой строки будут использоваться в качестве входных параметров для SQL-выражения INSERT.

3. Создайте класс Object-C для ответов на запросы. Создав базу данных, нужно разработать новый класс, который будет отвечать на географические запросы. Главный запрос, на который бы мы хотели ответить, — «получить все почтовые индексы, находящиеся в пределах 10 миль от 20007». Этот запрос можно реализовать с помощью следующего метода:

```
-(NSArray*)zipcodesNearLatitude:(float)lat andLongitude:(float) lon
    withinDistance:(float)distance;
```

Рассмотрим возможную реализацию этого метода. Его основное назначение — выполнение и обработка результатов следующего SQL-выражения:

```
SELECT Z.zipcode FROM zipcodes AS Z WHERE
Distance(latitude1, latitude2, Z.latitude, Z.longitude) <= distance
```

Выражение SELECT находит все почтовые индексы, для которых расстояние между почтовым индексом (latitude, longitude) и данной точкой (latitude1, latitude2) располагается в пределах значения distance (в километрах).

Вы научились писать код для этих SQL-выражений, создавать C-функции и использовать их в SQL-запросах. Реализуйте функцию Distance() в приведенном выше SQL-выражении самостоятельно. Листинг 12.12 представляет C-реализацию.

Листинг 12.12. C-реализация пользовательской функции Distance

```
void distance(sqlite3_context *context, int nargs, sqlite3_value  **values){
    char  *errorMessage;
    double  pi = 3.14159265358979323846;
    if(nargs != 4){
            errorMessage="Wrong # of args. Distance(lat1,lon1,lat2,lon2)";
            sqlite3_result_error(context, errorMessage, strlen(errorMessage));
            return;
    }
    if((sqlite3_value_type(values[0]) != SQLITE_FLOAT) ||
            (sqlite3_value_type(values[1]) != SQLITE_FLOAT) ||
            (sqlite3_value_type(values[2]) != SQLITE_FLOAT) ||
            (sqlite3_value_type(values[3]) != SQLITE_FLOAT)){
            errorMessage ="All four arguments must be of type float.";
            sqlite3_result_error(context, errorMessage, strlen(errorMessage));
            return;
    }
    double latitude1, longitude1, latitude2, longitude2;
    latitude1 = sqlite3_value_double(values[0]);
    longitude1 = sqlite3_value_double(values[1]);
    latitude2 = sqlite3_value_double(values[2]);
    longitude2 = sqlite3_value_double(values[3]);
```

```
    double x = sin(latitude1 * pi/180) *
    sin(latitude2 * pi/180) + cos(latitude1 * pi/180) *
    cos(latitude2 * pi/180) *
    cos(abs((longitude2 * pi/180) - (longitude1 * pi/180)));
    x = atan((sqrt(1 - pow(x, 2))) / x);
    x = (1.852 * 60.0 * ((x/pi)*180)) / 1.609344;
    sqlite3_result_double(context, x);
}
```

12.6. Резюме

Эта глава была посвящена теме информации о местоположении. В разд. 12.1 мы говорили о фреймворке Core Location и его использовании в приложениях, предоставляющих информацию о местоположении. В разд. 12.2 мы рассмотрели простую программу, предоставляющую информацию о местоположении. В разд. 12.3 вы получили представление о геокодировании и научились транслировать почтовые адреса в географические координаты. Из разд. 12.4 вы узнали, как отслеживать передвижения устройства и отображать эту информацию на картах местности. В разд. 12.5 были представлены соотношения почтовых индексов и географической информации. В этом разделе вы ознакомились с формулой, по которой вычисляется расстояние между двумя локациями.

Глава 13
Работа с устройствами

В этой главе мы рассмотрим использование некоторых устройств, доступных в iPhone. Из разд. 13.1 вы узнаете, как использовать акселерометр. В разд. 13.2 будет рассказано, как проигрывать небольшие звуковые файлы, а в разд. 13.3 — как воспроизводить видеофайлы. Из разд. 13.4 вы узнаете, как получать информацию об устройстве iPhone/iPod Touch. Использование камеры и фотобиблиотеки будет описано в разд. 13.5. Итоги главы вы найдете в разд. 13.6.

13.1. Работа с акселерометром

iPhone оборудован простым в использовании акселерометром. Акселерометр позволяет определить текущее положение устройства в 3D-пространстве. Вы запрашиваете обновления информации о положении устройства с данной частотой (от 10 до 100 обновлений в секунду) и при каждом обновлении получаете три значения с плавающей точкой. Эти значения представляют ускорение в пространстве по осям x, y и z. Ускорение по каждой оси вычисляется по формуле $g \times s$, где g — ускорение свободного падения (1 g = 9,8 м/с^2).

Положите iPhone перед собой и представьте ось, проходящую через кнопку Home (Возврат) и ушной динамик перпендикулярно полу. Эта ось будет осью y. Положительные значения y показывают, что iPhone ускоряется по данной оси вверх, а отрицательные — вниз по направлению к полу. Ось x проходит справа налево перпендикулярно оси y. Положительные значения показывают, что устройство передвигается вправо, отрицательные — влево. Ось z проходит через устройство. Отрицательные значения показывают, что устройство двигается от вас, а положительные — что оно двигается к вам.

Из-за силы притяжения устройство будет передавать отличные от нуля значения по некоторым или всем осям, даже находясь в неподвижном состоянии. Например, если вы держите устройство перед собой в портретной ориентации (рис. 13.1), оси x и z будут показывать 0 g, а значение по оси y будет -1 g. Проще говоря, к устройству не приложено никаких сил, передвигающих его влево/вправо или вперед/назад, но существует сила в 1 g, передвигающая устройство вниз, — сила притяжения.

Рис. 13.1. Неподвижный iPhone, передающий значения вектора акселерометра (0, −1, 0)

Если вы держите устройство в альбомной ориентации (рис. 13.2), то на ось *x* влияет сила гравитации. Значение компонента *x* вектора, передаваемого акселерометром, будет 1 g.

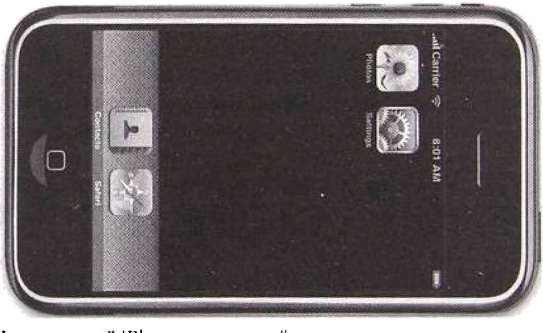

Рис. 13.2. Неподвижный iPhone, передающий значения вектора акселерометра (1, 0, 0)

Если вы держите устройство, как показано на рис. 13.3, значение будет −1 g. Если вы положите iPhone на стол лицевой стороной вверх, значение *z* будет −1 g, а если лицевой стороной вниз – 1 g.

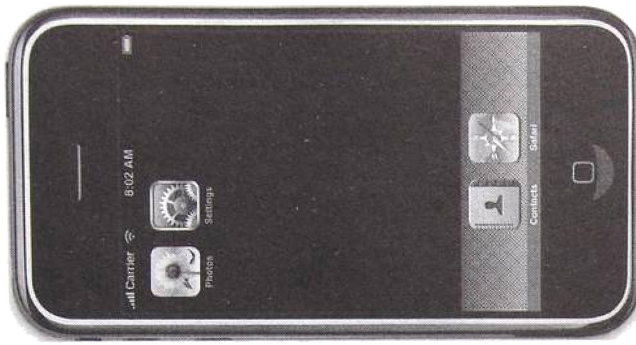

Рис. 13.3. Неподвижный iPhone, передающий значения вектора акселерометра (−1, 0, 0)

Если вы держите iPhone лицевой стороной к себе (см. рис. 13.1) и наклоните его вправо, значения x и y начнут увеличиваться. Если вы наклоните его влево, значение y начнет возрастать, а x — уменьшаться.

Пример. Рассмотрим простое приложение, демонстрирующее использование акселерометра. Пример покажет, как настроить акселерометр и перехватить *встряску*, *прижатие* и *толчок*. В дополнение программа будет сообщать, когда iPhone находится в портретной ориентации с кнопкой Home (Возврат) вверху или внизу перпендикулярно полу.

Для использования акселерометра выполните следующие действия.

1. Получите разделяемый объект акселерометра. В распоряжении приложения есть один акселерометр. Используйте метод `sharedAccelerometer` для получения этого объекта. Метод объявлен следующим образом:

```
+ (UIAccelerometer *) sharedAccelerometer
```

2. Настройте акселерометр. Настройте частоту обновлений, используя свойство `updateInterval`, объявленное следующим образом:

```
@property(nonatomic) NSTimeInterval updateInterval;
```

Здесь `NSTimeInterval` объявлено как `double`. Значение, передаваемое этому свойству, изменяется от `0,1` (частота 10 Hz) до `0,01` (частота 100 Hz).

Необходимо также настроить свойство делегата `delegate`, объявленное как

```
@property(nonatomic, assign) id<UIAccelerometerDelegate> delegate
```

Протокол `UIAccelerometerDelegate` имеет один необязательный метод `accelerometer:didAccelerate:`, объявленный следующим образом:

```
-(void) accelerometer:(UIAccelerometer *)accelerometer
    didAccelerate: (UIAcceleration *)acceleration;
```

Методу передается объект акселерометра и экземпляр UIAcceleration. Объект UIAcceleration содержит значения 3D-вектора (*x*, *y* и *z*) и время (timestamp).

Листинг 13.1 показывает объявление класса делегата приложения с использованием акселерометра. Делегат приложения заимствует протоколы UIApplicationDelegate и UIAccelerometerDelegate. В дополнение он оперирует предыдущим значением акселерометра в переменной экземпляра accelerationValues.

Листинг 13.1. Объявление класса делегата приложения для примера использования акселерометра

```
#import <UIKit/UIKit.h>
@interface AccelAppDelegate :
    NSObject <UIApplicationDelegate, UIAccelerometerDelegate> {
    UIWindow *window;
    UIAccelerationValue accelerationValues[3];
}
@end
```

Листинг 13.2 демонстрирует реализацию класса делегата приложения. Метод applicationDidFinishLaunching: начинается с настройки акселерометра на частоту обновлений 10 Hz и установки поля delegate на экземпляр делегата приложения.

В методе accelerometer:didAccelerate: находится логика распознавания, описанная выше. Для распознавания *встряски* достаточно проследить изменения ускорений хотя бы по двум осям. Мы используем значение изменения, равное 3g, для каждой оси. Например, выражение:

```
BOOL x_big_difference = (fabs(x - accelerationValues[0]) >3);
```

вернет значение YES (1), если разница между предыдущим и текущим значениями ускорений по оси *x* больше 3g.

Чтобы распознать портретную ориентацию iPhone с осью Home (Возврат) в динамик, перпендикулярной полу, когда кнопка Home (Возврат) находится внизу, мы должны убедиться, что значения *x* и *z* равны 0 с некоторой погрешностью, а значение *y* составляет приблизительно –1. Аналогично при распознавании перевернутой ориентации iPhone значение *y* должно быть около 1 g.

Для распознавания *прижатия/толчка* iPhone метод распознает значительное ускорение по оси *z* с небольшими изменениями по *x* и *y*. Если значение *z* изменилось в сторону отрицательного ускорения, мы рассматриваем это как толчок. Если же значение изменилось в положительную сторону, мы интерпретируем это как прижатие.

Листинг 13.2. Реализация класса делегата приложения для примера использования акселерометра

```
#import "AccelAppDelegate.h"
#define BETWEEN(arg, v1, v2) ((arg >= v1) && (arg <= v2 ))
@implementation AccelAppDelegate
-(void)accelerometer:(UIAccelerometer *)accelerometer
    didAccelerate:(UIAcceleration *)acceleration{
    UIAccelerationValue x, y, z;
    x = acceleration.x;
    y = acceleration.y;
    z = acceleration.z;
    NSLog(@"X: %4.2f, Y:%4.2f, Z:%4.2f", x, y, z); // встряска
    BOOL x_big_difference = (fabs(x - accelerationValues[0]) >3);
    BOOL y_big_difference = (fabs(y - accelerationValues[1]) >3);
    BOOL z_big_difference = (fabs(z - accelerationValues[2]) >3);
    int  axes = x_big_difference + y_big_difference + z_big_difference;
    if(axes>= 2){
            NSLog(@"iPhone Shaken!");
    }
    //ориентация
    if(BETWEEN(x, -0.05, 0.05) && BETWEEN(y, -1, -0.95) &&
              BETWEEN(z, -0.05, 0.05)){
            NSLog(@"iPhone perpendicular to ground, Home button down");
    }
    if(BETWEEN(x, -0.05, 0.05) && BETWEEN(y,  0.95, 1) &&
              BETWEEN(z, -0.05, 0.05)){
            NSLog(@"iPhone perpendicular to ground, Home button up");
    }
    // прижатие/толчок
    BOOL x_change = (fabs(x - accelerationValues[0]) < 1);
    BOOL y_change = (fabs(y - accelerationValues[1]) < 1);
    BOOL z_change = (fabs(z - accelerationValues[2]) >= 3);
    if(x_change && y_change && z_change){
            if(z > accelerationValues[2])
                    NSLog(@"hug");
            else
                    NSLog(@"punch");
    }
    accelerationValues[0] = x;
    accelerationValues[1] = y;
    accelerationValues[2] = z;
}
-(void)applicationDidFinishLaunching:(UIApplication *)application {
    CGRect fullScreen = [[UIScreen  mainScreen] bounds];
    window = [[UIWindow alloc] initWithFrame:fullScreen];
    UIAccelerometer  *accelerometer =
            [UIAccelerometer sharedAccelerometer];
    accelerometer.updateInterval = 0.1;
    // 10Hz
    accelerometer.delegate = self;
    [window makeKeyAndVisible];
}
- (void)dealloc {
    [window release];
    [super dealloc];
}
@end
```

13.2. Аудио

В этом разделе мы рассмотрим проигрывание коротких звуковых файлов (продолжительностью менее 30 секунд). Чтобы воспроизвести короткий звуковой файл, необходимо зарегистрировать файл как системный звук и получить его дескриптор. После этого, используя дескриптор, вы можете проиграть этот звук. Если вы закончили работу и не хотите вопроизводить его снова, нужно высвободить этот системный звук.

Чтобы зарегистрировать звуковой файл в качестве системного звука, используйте функцию `AudioServicesCreateSystemSoundID()`, объявленную следующим образом:

```
OSStatus AudioServicesCreateSystemSoundID(
    CFURLRef inFileURL,
    SystemSoundID *outSystemSoundID)
```

Первый параметр — это `CFURLRef` (или его аналог, экземпляр `NSURL`), а второй — ссылка `SystemSoundID`. Эта переменная, 32-битное беззнаковое целое, будет хранить ID системного звука. Возвращаемое значение должно быть 0, что означает успешную регистрацию системного звука.

Чтобы воспроизвести системный звук, используйте функцию `AudioServicesPlaySystemSound()`, объявленную как

```
void AudioServicesPlaySystemSound(SystemSoundID inSystemSoundID)
```

Вы передаете дескриптор системного звука, полученный от предыдущей функции. Чтобы включить вибрацию, можете использовать предопределенный системный идентификатор `kSystemSoundID_Vibrate`.

Для высвобождения системного звука используйте функцию `AudioServicesDisposeSystemSoundID()`, объявленную следующим образом:

```
OSStatus AudioServicesDisposeSystemSoundID(SystemSoundID inSystemSoundID)
```

В качестве параметра передается дескриптор системного звука, полученный от функции регистрации.

Пример. Создадим приложение, проигрывающее звуковой файл каждую минуту. Листинг 13.3 показывает объявление класса делегата приложения. Обратите внимание на директиву `#include` для файла заголовка `<AudioToolbox/AudioToolbox.h>`. Вам также нужно добавить библиотеку `AudioToolbox.framework` к проекту в XCode.

Листинг 13.3. Объявление класса делегата приложения, демонстрирующего проигрывание небольших звуковых файлов

```
#import <UIKit/UIKit.h>
#include <AudioToolbox/AudioToolbox.h>
```

```
@interface AudioAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow  *window;
    SystemSoundID  audioSID;
}
@end
```

Листинг 13.4 демонстрирует реализацию класса делегата приложения. Звуковой файл хранится в упаковке приложения. В методе `applicationDidFinishLaunching:` мы сначала получаем абсолютный путь к файлу `sound.caf`. Затем для этого пути создается объект `NSURL`, используя метод `fileURLWithPath:isDirectory:`. После этого регистрируется системный звук. Типы `CFURL` и `NSURL` взаимозаменяемы, поэтому мы передаем объект `NSURL` вместо ссылки на `CFURL`, `CFURLRef`. Если не возникло ошибок, звук воспроизводится.

Метод `play:` проигрывает звук, после чего настраивает таймер на собственный вызов раз в минуту.

Листинг 13.4. Объявление класса делегата приложения, демонстрирующего проигрывание небольших звуковых файлов

```
#import "AudioAppDelegate.h"
@implementation AudioAppDelegate
-(void)applicationDidFinishLaunching:(UIApplication *)application {
    CGRect screenFrame = [[UIScreen  mainScreen] bounds];
    window = [[UIWindow alloc] initWithFrame:screenFrame];

    NSString *filePath = [[NSBundle mainBundle]
                    pathForResource:@"sound" ofType:@"caf"];
    NSURL *aFileURL = [NSURL fileURLWithPath:filePath isDirectory:NO];
    OSStatus error =
            AudioServicesCreateSystemSoundID(
                    (CFURLRef)aFileURL, &audioSID);
    if(error == 0)
            [self play:nil];
            [window makeKeyAndVisible];
    }

-(void)play:(NSTimer*)theTimer{
    AudioServicesPlaySystemSound(audioSID);
    // настраиваем ежеминутное воспроизведение
    [NSTimer scheduledTimerWithTimeInterval:60.0
            target:self selector:@selector(play:)
            userInfo:nil repeats:NO];
}
-(void)dealloc {
    AudioServicesDisposeSystemSoundID(audioSID);
    [window release];
    [super dealloc];
}
@end
```

13.3. Видео

Чтобы воспроизвести видео в вашем приложении, используйте класс `MPMoviePlayerController`. Создайте и инициализируйте его экземпляр, после чего запросите воспроизвести видео. Этот контроллер проигрывает видео в полноэкранном режиме. Когда воспроизведение закончено, экран приложения снова становится видимым.

Следующий код воспроизводит файл `MyMovie.m4v`, хранящийся в упаковке приложения.

```
NSString *filePath =
    [[NSBundle mainBundle] pathForResource:@"MyMovie"
          ofType:@"m4v"];
    NSURL *fileUrl = [NSURL fileURLWithPath:filePath];
    MPMoviePlayerController *movieController =
          [[MPMoviePlayerController alloc] initWithContentURL:fileUrl];
    movieController.backgroundColor = [UIColor grayColor];
    movieController.movieControlMode = MPMovieControlModeVolumeOnly;
    [movieController play];
```

Данный код сначала находит полный путь к видеофайлу в упаковке и использует его для создания экземпляра `NSURL`. Затем создается экземпляр `MPMoviePlayerController`, инициализирующийся посредством метода `initWithContentURL:`, которому передается экземпляр `NSURL`. Опционально мы можем установить цвет фона и режим управления. Для свойства `movieControlMode` можно определить (или принять значение по умолчанию) `MPMovieControlModeDefault` для появления стандартных элементов управления (кнопок начала и остановки воспроизведения, временной шкалы и т. д.). Чтобы скрыть все элементы управления, используйте `MPMovieControlModeHidden`. Метод `MPMovieControlVolumeOnly` используется для отображения только регулятора громкости.

После фазы инициализации контроллер воспроизводит видео с помощью метода `play`.

13.4. Информация об устройстве

Класс `UIDevice` используется для предоставления информации о вашем iPhone/iPod Touch. Существует единственный экземпляр класса, который можно получить с помощью метода класса `currentDevice`. Рассмотрим фрагменты информации, которые вы можете получить посредством данного экземпляра.

- **Уникальный идентификатор.** Вы можете получить строку, уникально идентифицирующую ваш iPhone, с помощью `uniqueIdentifier`, объявленного следующим образом:

```
@property(nonatomic, readonly, retain) NSString *uniqueIdentifier
```

- **Операционная система.** Получить имя операционной системы можно посредством свойства `systemName`, объявленного как

```
@property(nonatomic, readonly, retain) NSString  *uniqueIdentifier
```

- **Версия операционной системы.** Вы можете получить версию операционной системы с помощью свойства `systemVersion`. Оно объявлено следующим образом:

```
@property(nonatomic, readonly, retain) NSString  *systemVersion
```

- **Модель.** Определить различие между iPhone и iPod Touch можно посредством свойства `model`, объявленного как

```
@property(nonatomic, readonly, retain) NSString *model
```

- **Ориентация.** Ориентацию устройства можно установить с помощью свойства `orientation`. Свойство объявлено следующим образом:

```
@property(nonatomic, readonly) UIDeviceOrientation orientation
```

Возможные значения:

- `UIDeviceOrientationUnknown`;
- `UIDeviceOrientationPortrait`;
- `UIDeviceOrientationPortraitUpsideDown`;
- `UIDeviceOrientationLandscapeLeft`;
- `UIDeviceOrientationLandscapeRight`;
- `UIDeviceOrientationFaceUp`;
- `UIDeviceOrientationFaceDown`.

13.5. Производство и просмотр снимков

В этом разделе вы научитесь использовать камеру для производства снимков. Вы узнаете, что прямой доступ к камере или библиотеке фотографий отсутствует, поэтому требуется использовать предоставленный контроллер, который обрабатывает взаимодействие с пользователем при производстве и редактировании снимков. Контроллер предоставляет конечный вариант изображения. Тот же контроллер можно использовать при просмотре снимков, хранящихся в пользовательской библиотеке. Структура раздела следующая. В подразд. 13.5.1 будут рассмотрены основные шаги, необходимые для обеспечения доступа к камере и фотобиблиотеке, а в подразд. 13.5.2 — подробный пример, демонстрирующий производство и просмотр снимков.

13.5.1. Общий подход

Чтобы получить доступ к камере или просмотреть снимки из пользовательской библиотеки, необходимо использовать предоставляемый вам системой интерфейс. Главный класс, используемый для производства снимков и просмотра уже существующих, — `UIImagePickerController`. Для производства или просмотра снимков выполните следующие действия.

1. Проверьте доступность действия. Хотите ли вы сделать новый снимок или просмотреть уже существующий, необходимо проверить, доступна ли данная функция. Метод класса `UIImagePickerController`, используемый для этих целей, — `isSourceTypeAvailable:`.

2. Создайте экземпляр контроллера. Если заданное действие доступно, необходимо создать экземпляр `UIImagePickerController`, инициализировать его и настроить на нужную функцию. Если тип источника недоступен, создавать контроллер не требуется.

3. Настройте делегат. Метод `UIImagePickerController` будет обеспечивать взаимодействие с пользователем во время производства и просмотра снимков. Нужно настроить делегат на объект и реализовать специальные методы, чтобы получить результат. Делегат реализует протокол `UIImagePickerControllerDelegate`.

4. Отобразите контроллер. Контроллер модально отображается пользователю с помощью dspjdf метода `presentModalViewController: animated:` для существующего контроллера представления, передавая экземпляр `UIImagePickerController` в качестве первого параметра.

5. Обработайте выбор изображения. Когда пользователь выбирает снимок, вызывается метод делегата `imagePickerController:didFinishPickingMediaWithInfo:`. Вам нужно извлечь изображения из словаря и скрыть контроллер, который был модально отображен на экране.

6. Обработайте отмену. Если пользователь отменяет операцию, вызывается метод делегата `imagePickerControllerDidCancel:`. Вам требуется скрыть контроллер, который был модально отображен на экране.

13.5.2. Подробный пример

Рассмотрим приложение, демонстрирующее использование класса `UIImagePickerController`. Программа предоставляет пользователю два действия на выбор — сделать новый снимок или выбрать уже существующий. В зависимости от выбора пользователя приложение настраивает экземпляр класса `UIImagePickerController` и отображает его. Когда пользователь справится с задачей, результат (если он есть) станет фоном основного окна.

Программа реализует два класса — класс делегата приложения и простой контроллер представления. Листинг 13.5 показывает объявление класса делегата приложения `CameraAppDelegate`. Делегат приложения использует экземпляр класса `MainController` для управления пользовательским интерфейсом.

Листинг 13.5. Объявление делегата приложения для работы с камерой

```
#import <UIKit/UIKit. h>
#import  "MainController.h"
@interface  CameraAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow * window;
    MainController * viewController;
}
@property (nonatomic, retain) UIWindow * window;
@end
```

Листинг 13.6 демонстрирует реализацию делегата приложения. Метод `applicationDidFinishLaunching:` создает главное окно и экземпляр контроллера представления `MainController`. Затем он добавляет представление данного контроллера в качестве дочернего к главному окну, после чего делает главное окно ключевым и видимым.

Листинг 13.6. Реализация делегата приложения для работы с камерой

```
#import  "CameraAppDelegate.h"
@implementation  CameraAppDelegate
@synthesize  window;
-(void)applicationDidFinishLaunching:
    (UIApplication *) application {
    window = [[UIWindow  alloc]
           initWithFrame:[[UIScreen mainScreen] bounds]];
    viewController = [MainController  alloc];
    [window  addSubview:viewController.view];
    [window  makeKeyAndVisible];
}
-(void) dealloc {
    [window  release];
    [super  dealloc];
}
@end
```

Листинг 13.7 показывает объявление контроллера представления `MainController`. Он заимствует два протокола — `UIImagePickerControllerDelegate` для обработки взаимодействия с экземпляром `UIImagePickerController` и `UIActionSheetDelegate`, обрабатывающий взаимодействие со списком действий.

Листинг 13.7. Объявление класса MainController приложения для работы с камерой

```
#import <UIKit/UIKit.h>
@interface MainController:
    UIViewController <UIImagePickerControllerDelegate,
    UIActionSheetDelegate> {
}
@end
```

Листинг 13.8 демонстрирует метод loadView класса Main Controller. Этот метод просто создает полноэкранное представление, настраивает его и связывает со свойством view контроллера представления.

Листинг 13.8. Метод loadView класса MainController приложения для работы с камерой

```
-(void)loadView {
    CGRect  rectFrame =
            [UIScreen  mainScreen].applicationFrame;
    UIView * theView =
            [[UIView  alloc] initWithFrame:rectFrame];
    theView.backgroundColor = [UIColor  darkGrayColor];
    theView.autoresizingMask = UIViewAutoresizingFlexibleHeight |
            UIViewAutoresizingFlexibleWidth;
    self.view = theView;
    [theView  release];
}
```

Когда представление завершает загрузку, метод viewDidLoad вызывается автоматически. В этом методе мы создаем список действий, предлагающий пользователю выбрать, делать новый снимок или выбрать уже существующий. Метод устанавливает контроллер представления в качестве делегата и показывает список действий в представлении (листинг 13.9).

Листинг 13.9. Метод viewDidLoad контроллера представления

```
-(void)viewDidLoad {
    UIActionSheet * actionSheet = [[UIActionSheet alloc]
            initWithTitle:@"Choose an option:"
            delegate:self
            cancelButtonTitle:nil
            destructiveButtonTitle:nil
            otherButtonTitles :@"New Picture",
                    @"Select a Picture", nil];
    [actionSheet showInView:self.view];
}
```

Листинг 13.10 показывает метод actionSheet:clickedButtonAt Index:, обрабатывающий нажатия кнопок списка действий. Если была нажата первая кнопка, мы проверяем, доступна ли камера. Для этого используется метод isSourceTypeAvailable: класса UIImagePickerController. Метод объявлен как

```
+ (BOOL)isSourceTypeAvailable:
    (UIImagePickerControllerSourceType)sourceType
```

Единственный параметр типа UIImagePickerControllerSource Type может принимать следующие значения:

- UIImagePickerControllerSourceTypePhotoLibrary — задает в качестве источника фотобиблиотеку;
- UIImagePickerControllerSourceTypeSavedPhotosAlbum — в качестве источника определяет альбом камеры;
- UIImagePickerControllerSourceTypeCamera — задает как источник встроенную камеру.

Метод возвращает YES, если заданный источник доступен, и NO в противном случае. Если камера доступна, метод создает новый экземпляр класса UIImageController и устанавливает свойство источника sourceType в значение UIImagePickerControllerSourceType Camera. Свойство sourceType объявлено следующим образом:

```
@property (nonatomic) UIImagePickerControllerSourceType sourceType
```

Если нажата вторая кнопка, мы повторяем предыдущие шаги с тем исключением, что в качестве источника будем использовать UIImage PickerControllerSourceTypePhotoLibrary. После этого мы устанавливаем делегат контроллера выбора изображения в экземпляр контроллера представления, устанавливаем свойство allowsImageEditing в YES и отображаем контроллер выбора изображения посредством вызова метода контроллера представления presentModalViewController:animated:. Свойство allowsImageEditing используется для определения, разрешено ли пользователю редактировать изображение перед его возвратом.

Листинг 13.10. Метод actionSheet:clickedButtonAtIndex:, обрабатывающий нажатия пользователем списка действий приложения для работы с камерой

```
-(void)actionSheet:(UIActionSheet *) actionSheet
    clickedButtonAtIndex:(NSInteger)buttonIndex
{
    [actionSheet  release];
    UIImagePickerController *picker;
    if (buttonIndex == 0){
            if ((![UIImagePickerController
                    isSourceTypeAvailable:
                            UIImagePickerControllerSourceTypeCamera])
                    ){
                    return;
            }
            picker = [[UIImagePickerController alloc] init];
            picker.sourceType =
                    UIImagePickerControllerSourceTypeCamera;
    }
    else {
            if ((![UIImagePickerController isSourceTypeAvailable:
                    UIImagePickerControllerSourceTypePhotoLibrary])
                    ){
                    return;
            }
            picker = [[UIImagePickerController alloc] init];
            picker.sourceType =
                    UIImagePickerCo ntrollerSourceTypePhotoLibrary;
    }
    picker.delegate = self;
    picker.allowsImageEditing = YES;
    [self presentModalViewController:picker animated : YES];
}
```

Как уже было сказано, контроллер выбора изображений информирует, что пользователь выбрал изображение посредством вызова метода imagePickerController:didFinishPickingImage:editingInfo: своего делегата. Метод объявлен следующим образом:

```
-(void)
    imagePickerController:(UIImagePickerController *)picker
```

```
    didFinishPickingImage:(UIImage *)image
    editingInfo:(NSDictionary *)editingInfo
```

Первый параметр, `picker` — это объект, контролирующий интерфейс выбора изображений. Второй параметр — изображение, выбранное пользователем (возможно, после редактирования), а третий — это словарь, используемый для хранения информации о редактировании (если оно доступно). Для извлечения соответствующей информации из словаря применяются два ключа:

- `UIImagePickerControllerOriginalImage` — для извлечения оригинального изображения (экземпляра `UIImage`) перед редактированием пользователя;
- `UIImagePickerControllerCropRect` — для получения объекта `NSValue`, содержащего значение типа `CGRect` (прямоугольника, по контуру которого было обрезано изображение).

Листинг 13.11 показывает реализацию метода `imagePickerController:didFinishPickingImage:editingInfo:`. Метод просто создает представление с переданным изображением и добавляет это представление в качестве дочернего к представлению главного контроллера. Он также скрывает контроллер представления выбора изображения с помощью вызова метода `dismissModalViewControllerAnimated:`.

Листинг 13.11. Обработка успешного выбора изображения в приложении для работы с камерой

```
- (void) imagePickerController:(UIImagePickerController *)picker
    didFinishPickingImage:(UIImage *)image
    editingInfo:(NSDictionary *)editingInfo
{
    UIImageView *imgView =
        [[UIImageView alloc] initWithImage:image];
    [self.view addSubview:imgView];
    [self dismissModalViewControllerAnimated:YES];
    [picker release];
    [imgView release];
}
```

Листинг 13.12 демонстрирует обработку события, при котором пользователь отменяет выбор изображения. Мы просто скрываем представление выбора изображения и высвобождаем его.

Листинг 13.12. Обработка пользовательской отмены выбора изображения в приложении для работы с камерой

```
-(void)imagePickerControllerDidCancel:
    (UIImagePickerController *)picker
{
    [self dismissModalViewControllerAnimated:YES];
    [picker release];
}
@end
```

Ниже изображен главный экран, который пользователь видит сразу после запуска приложения (рис. 13.4), а также представление, где пользователь делает снимок (рис. 13.5).

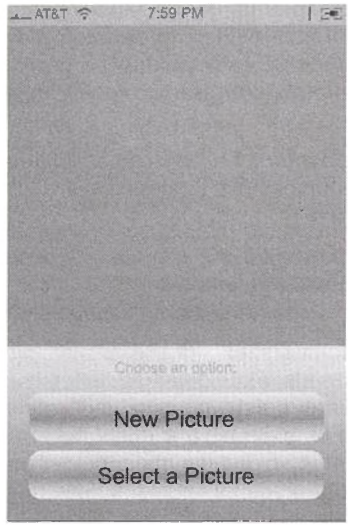

Рис. 13.4. Главный экран, который видит пользователь при запуске приложения с поддержкой камеры

Рис. 13.5. Представление, где пользователь находится в процессе производства снимка

Далее приведено представление, где пользователь только что сделал снимок и представление выбора изображения все еще активно (рис. 13.6). Пользователь может подтвердить, устраивает ли его данный снимок (после чего изображение сохранится, а пользователь увидит главное окно приложения (рис. 13.7)), или нажать Retake (Переснять), чтобы сделать другой снимок.

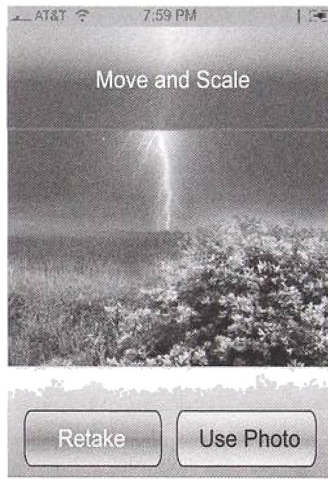

Рис. 13.6. Представление сразу после того, как пользователь сделал снимок (представление выбора изображения все еще активно)

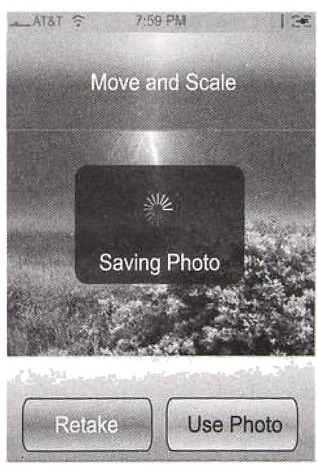

Рис. 13.7. Представление выбора изображения, сохраняющее снимок

Ниже показано представление, когда выбор изображения закончился и управление вернулось к нашему коду (рис. 13.8), а также представление, которое видит пользователь при выборе существующего изображения (рис. 13.9).

Рис. 13.8. Представление, когда выбор изображения закончен и управление возвращено главному окну

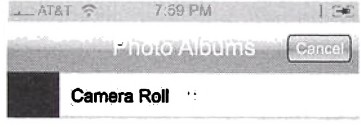

Рис. 13.9. Представление, которое видит пользователь при выборе существующего изображения

13.6. Резюме

В разд. 13.1 вы научились использовать акселерометр. Вы узнали, как получать от него информацию с данной частотой обновления и распознавать ориентацию устройства, а также специфические движения. В разд. 13.2 вы научились проигрывать короткие звуковые файлы и настраивать таймер на выполнение периодических задач. В разд. 13.3 вы научились воспроизводить видеофайлы, из разд. 13.4 узнали, как получать необходимую информацию об устройстве. В разд. 13.5 вы научились использовать камеру для производства снимков. Вы узнали, что прямой доступ к камере или библиотеке фотографий отсутствует, поэтому требуется использовать предоставленный контроллер (`UIImagePickerController`), который обрабатывает взаимодействие с пользователем при производстве и редактировании снимков. Контроллер предоставляет конечный вариант изображения после того, как пользователь закончит работу с ним. Тот же контроллер используется для выбора изображений, хранящихся в библиотеке. Мы рассмотрели основные понятия с помощью приложения, позволяющего пользователю производить снимки либо выбирать из уже существующих.

Приложение А
Сохранение и восстановление состояния программы

В один момент времени может быть запущено только одно приложение, поэтому ваша программа должна давать пользователю ощущение, что она активна все время, даже когда он нажимает кнопку Home (Возврат). Для этого ваше приложение должно сохранять текущее состояние (например, какой уровень иерархии показывается в данный момент, текущий термин поиска и т. д.), когда программа завершается, и восстанавливать это состояние, когда она запускается повторно.

Существует несколько путей, с помощью которых можно оперировать информацией о состоянии, например, использовать базу данных SQLite или обычные файлы. Однако больше всего для данной ситуации подходит список свойств, который является предметом рассмотрения приложения А.

Чтобы использовать список свойств для сохранения/восстановления состояния приложения, нужно следовать приведенным ниже рекомендациям:

- представить состояние в виде словаря или массива;
- элементы состояния могут быть экземплярами типов `NSDictionary`, `NSArray`, `NSData`, `NSDate`, `NSNumber` и `NSString`;
- добавьте элементы состояния в словарь либо массив;
- сериализуйте словарь или массив в объект `NSData`, используя метод класса `NSPropertyListSerialization`;
- объект `NSData` представляет информацию о состоянии либо в форматах XML, либо бинарном; для эффективности используйте бинарный формат;
- чтобы восстановить состояние, загрузите записанный файл в объект `NSData` и используйте метод класса `NSPropertyList Serialization` для получения словаря или массива.

Рассмотрим пример сохранения/восстановления состояния приложения, используя список свойств. Допустим, состояние приложения сохраняется в словарь. Этот словарь содержит пять элементов: два типа `NSString`, один типа `NSNumber`, один типа `NSArray` и один типа `NSDate`.

Листинг А.1 показывает метод делегата приложения, демонстрирующий концепцию списков свойств. Метод создает объект состояния при-

ложения (словарь) и вызывает `saveAppState` для сохранения состояния и `restoreState` для восстановления.

Листинг А.1. Метод делегата, используемый при составлении объекта состояния приложения, его сохранении и последующем восстановлении с помощью списка свойств

```
-(void) applicationDidFinishLaunching:(UIApplication *)application {
    // составляем объект свойств
    state = [[NSMutableDictionary alloc] initWithCapacity:5];
    [state setObject:@"http://www.thegoogle.com" forKey:@"URL"];
    [state setObject:@"smartphones" forKey:@"SEARCH_TERM"];
    [state setObject:[NSNumber numberWithFloat:3.14] forKey:@"PI"];
    [state setObject:
            [NSMutableArray arrayWithObjects:@"Apple iPhone 3G",
             @"Apple iPhone",
             @"HTC Touch Diamond", nil]
             forKey:@"RESULT"];
    [state  setObject:[NSDate date]  forKey:@"DATE"];
    // сохраняем состояние приложения
    [self saveAppState];
    // восстанавливаем состояние приложения
    [self restoreState];
}
```

Листинг А.2 демонстрирует метод `saveAppState`. Последний использует метод `dataFromPropertyList:format:errorDescription:` класса `NSPropertyListSerialization`, чтобы получить объект `NSData` сериализованного словаря. Вы можете использовать два формата — бинарный, задаваемый с помощью `NSPropertyListBinaryFormat_v1_0`, и XML, определяемый посредством `NSPropertyListXMLFormat_v1_0`. Вы также можете передать ссылку на объект `NSString`, который можно использовать для возврата ошибки. При возникновении ошибки вы должны высвободить этот объект. Получив объект `NSData`, можете записать его в локальный файл.

Листинг А.2. Сохранение состояния приложения в список свойств

```
-(void) saveAppState{
    NSString  *theError;
    NSData  *theData = [NSPropertyListSerialization
            dataFromPropertyList:state
            format:NSPropertyListXMLFormat_v1_0
            errorDescription:&theError];
    if(theData){
            NSString *fileName = [NSHomeDirectory()
              stringByAppendingPathComponent:@"Documents/state.plist"];
            [theData writeToFile:fileName atomically:YES];
    }
    else{
            NSLog(@"Error saving app state: %@", theError);
            [theError release];
            //требуется высвободить
    }
}
```

Листинг А.3 показывает содержимое XML-файла `state.plist`, используемого для сохранения состояния приложения. Количество байтов, используемых для сохранения состояния в XML-формате, — 543, тогда как бинарный формат занимает 204 байта.

Листинг А.3. Список свойств в XML-формате

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>DATE</key>
    <date>2008-09-04T17:47:28Z</date>
    <key>PI</key>
    <real>3.1400001049041748</real>
    <key>RESULT</key>
    <array>
            <string>Apple iPhone 3G</string>
            <string>Apple iPhone</string>
            <string>HTC Touch Diamond</string>
    </array>
    <key>SEARCH_TERM</key>
    <string>smartphones</string>
    <key>URL</key>
    <string>http://thegoogle.com</string>
</dict>
</plist>
```

Листинг А.4 демонстрирует метод, используемый для восстановления состояния приложения. Сначала файл считывается в объект NSData с помощью методов, описанных в гл. 9. После этого используется метод класса `propertyListFromData:mutabilityOption:format:errordescription:` для получения объекта словаря. Параметр `mutabilityOption` применяется для указания изменяемости возвращаемых объектов. Если вы укажете `NSPropertyListImmutable`, то словарь, как и все записи в нем, возвратится как неизменяемый. Если вы установите `NSPropertyListmutableContainersAndLeaves`, то словарь и все его записи будут созданы как изменяемые объекты. Параметр `NSPropertyListMutableContainers`, который мы будем использовать, создает изменяемые объекты только для словарей и массивов.

Листинг А.4. Восстановление состояния приложения из списка свойств

```
-(void)restoreState{
    NSString *theError;
    NSPropertyListFormat format;
    NSString *fileName = [NSHomeDirectory()
            stringByAppendingPathComponent:@"Documents/state.plist"];
    NSData *theData = [[NSData alloc]
            initWithContentsOfFile:fileName];
if(theData){
    [state release];
    state = [NSPropertyListSerialization
            propertyListFromData:theData
            mutabilityOption:NSPropertyListMutableContainers
            format:&format
            errorDescription:&theError];
    if (state){
            [state retain];
    }
    else{
            NSLog(@"Error retrieving app state: %@", theError);
            [theError  release]; // требуется высвободить
            }
            [theData release];
    }

}
```

Приложение Б
Запуск внешних программ

Ваше iPhone-приложение может программно вызывать другие iPhone-программы. Более того, вы можете открыть свое iPhone-приложение, чтобы оно могло быть запущено другой iPhone-программой. Для этого вам требуется указать новую URL-схему в упаковке приложения, а система зарегистрирует эту схему, как только программа будет установлена.

Чтобы запустить другое iPhone-приложение, используйте метод `openURL:` экземпляра `UIApplication`, которому передайте URL этого приложения. Следующий фрагмент кода откроет приложение `Maps` и покажет заданный адрес:

```
NSString  *address = @"http://maps.google.com/maps?q=plano,tx";
NSURL *myURL = [NSURL URLWithString:address];
[[UIApplication sharedApplication] openURL:myURL];
```

Чтобы зарегистрировать новую URL-схему, нужно добавить ее в файл `Info.plist`. Ниже изображено, что следует добавить, чтобы зарегистрировать новую URL-схему `lookup` (рис. Б.1.); `URL identifier` может быть любой уникальной строкой.

▼URL types	(1 item)
▼Item 1	(2 items)
URL dentifier	com.mycompany.lookup
▼URL Schemes	(1 item)
Item 1	lookup

Рис. Б.1. Добавление новой URL-схемы в файл Info.plist

Чтобы в действительности запустить приложение, нужно реализовать метод делегата приложения `application:handleOpenURL:`, объявленный следующим образом:

```
-(BOOL)
    application:(UIApplication *)application
    handleOpenURL:(NSURL *)url
```

Вы получаете экземпляр NSURL, инкапсулирующий URL, используемый при вызове. Вы можете использовать множество методов NSURL для производства запросов, параметров и фрагментов. Например, нашу схему lookup можно вызвать с использованием почтового индекса либо названия города. Для поиска по почтовому индексу требуется вызвать приложение посредством `lookup://www?zip#68508`, а для вызова с помощью названия города — `lookup://www?city#Lincoln`.

Листинг Б.1 показывает реализацию обработки URL-схемы lookup. Для выполнения запроса мы используем метод query класса NSURL. Это может быть city либо zip. Для извлечения фрагмента мы используем метод fragment. Приведенная ниже реализация демонстрирует важные части вызова посредством URL. Если вы не можете обработать запрос, то возвращаете NO, иначе вы обслуживаете его и возвращаете YES.

Листинг Б.1. Реализация обработки URL-схемы lookup

```
-(BOOL)application:(UIApplication *)application
handleOpenURL:(NSURL *)url{
    NSString  *query = [url query];
    NSString  *fragment = [url fragment];
    NSMutableString *output;
    if([query isEqualToString:@"zip"]){
           output = [NSMutableString
               stringWithFormat:@"Looking up by zip code %@", fragment];
    }
    else  if([query isEqualToString:@"city"]){
           output = [NSMutableString
               stringWithFormat:@"Looking up by city  %@", fragment];
    }
    else return NO;
    textView.text = output;
    return YES;
}
```

[1] http://geocoder.ibegin.com/downloads.php.

[2] ZIP code: http://en.wikipedia.org/wiki/ZIP code.

[3] B'Far, R, Mobile Computing Principles: Designing and Developing Mobile Applications with UML and XML, Cambridge University Press, 2004.

[5] Beam, M and Davidson, JD, Cocoa in a Nutshell: A Desktop Quick Reference, O'Reilly, 2003.

[6] Brownell, D, SAX2, O'Reilly, 2002.

[7] Davidson, JD, Learning Cocoa with Objective C, 2nd edition, O'Reilly, 2002.

[8] Duncan, A, Objective-C Pocket Reference, 1st edition, O'Reilly, 2002.

[9] Garfinkel, S and Mahoney, MK, Building Cocoa Applications: A Step-by-Step Guide, 1st edition, O'Reilly, 2002.

[10] Hillegass, A, Cocoa® Programming for Mac® OS X, 3rd edition, Addison-Wisley Professional, 2008.

[11] Kochan, S, Programming in Objective-C, Sams, 2003.

[12] Mott, T, Learning Cocoa, O'Reilly, 2001.

[13] Owens, M, The Definitive Guide to SQLite, Apress, Inc., 2006.

[14] Tejkowski, E, Cocoa Programming for Dummies, 1st edition, For Dummies, 2003.

[15] Williams, E, Aviation Formulary V1.43: http://williams.best.vwh.net/avform.htm.

[16] Collections Programming Topics for Cocoa, Apple Reference Library.

[17] Document Object Model (DOM): http://www.w3.org/TR/DOM-Level-2-Core/.

[18] Exception Programming Topics for Cocoa, Apple documentation.

[19] Introduction to the Objective-C 2.0 Programming Language, Apple documentation.

[20] Key-Value Coding Programming Guide, Apple documentation.

[21] libxml2: The XML C parser and toolkit: http://xmlsoft.org/.

[22] Threading Programming Guide, Apple Reference Library.

[23] The XML standard: http://www.w3.org/TR/REC-xml.

Алфавитный указатель

#import, 20
#include, 20

A
alloc, 23
AudioServicesCreateSystemSoundID(), 345
AudioServicesDisposeSystemSoundID(), 345
AudioServicesPlaySystemSound(), 345

C
CGPoint, 90
CGPointMake, 85
CGRect, 84
CGRectMake, 91
CGSize, 91
CGSizeMake, 91
class, 19
CLLocation, 318
    altitude, 320
    coordinate, 320
    horizontalAccuracy, 320
    timestamp, 321
    verticalAccuracy, 320
CLLocationManager, 318
    CLLocationManagerDelegate, 319
    desiredAccuracy, 318
    distanceFilter, 318
    startUpdatingLocation, 319
    stopUpdatingLocation, 319
copyWithZone:, 66

D
dealloc, 26
dynamic, 30

E
exception
    catch, 36
    finally, 36
    throw, 36
    try, 37

G
GPS, 317

I
id, 21
Info.plist, 361

K
KVC, 43

L
libxml2
    charactersSAXFunc(), 309
    xmlChildrenNode, 300
    xmlDocGetRootElement(), 298
    xmlNode, 298
    xmlNodeListGetString, 302
    xmlStrcmp(), 300

M
MPMoviePlayerController, 347
MVC, 143

N
nil, 22
Nill, 21
NSArray, 58
    arrayWithObjects:, 60, 68
    copy, 64
    NSMutableArray, 58
    objectAtIndex:, 63
    sortedArrayUsingSelector:, 72
NSAutoreleasePool, 24
NSBundle, 270

NSCopying, 66
NSData
dataWithContentsOfFile:, 270
writeToFile:atomically:, 290
NSDictionary, 78
allKeys, 79
allValues, 80
dictionaryWithObjectsAndKeys:, 79
isEqualToDictionary:, 80
keysSortedByValueUsingSelector:, 80
objectForKey:, 79
NSError, 41
NSException, 36
exceptionWithName:reason:userInfo:, 39
raise, 37
NSFileHandle, 258
seekToFileOffset:, 271
NSFileManager, 258
defaultManager, 259
enumeratorAtPath:, 259
NSHomeDirectory(), 258
NSIndexPath, 217
row, 218
section, 218
NSInteger, 70
NSInvocationOperation, 50
NSMutableArray, 58
removeObject:, 65
NSMutableDictionary, 78
addEntriesFromDictionary:, 80
removeObjectForKey:, 79
NSMutableSet, 75
addObject, 76
removeObject:, 76
unionSet:, 76
NSNull, 48
NSObject, 22
alloc, 23
conformsToProtocol:, 28
dealloc, 26
init, 23
poseAsClass:, 36
release, 25
NSOperation, 50
NSPropertyListSerialization, 358
NSSelectorFromString(), 22
NSSet, 74
anyObject, 75
containsObject:, 75
intersectsSet:, 75
isSubsetOfSet:, 75
NSMutableSet, 76
NSString, 22
cStringUsingEncoding:, 298
NSMutableString, 22
stringWithContentsOfURL:encoding:error:, 297
NSURL, 362
NULL, 22

R
RSS, 294

S
SEL, 22
self, 23
SQL, 273
SQLite, 273
BLOB, 286
sqlite3_close(), 274
sqlite3_column_XXX(), 281
sqlite3_create_function(), 283
sqlite3_exec(), 279
sqlite3_finalize(), 280, 282
sqlite3_free(), 276
sqlite3_malloc(), 275
sqlite3_prepare_v2(), 279
sqlite3_result_error(), 284
sqlite3_result_XXX(), 286
sqlite3_step(), 281
sqlite3_value, 285
sqlite3_value_type(), 285
SQLITE_DONE, 280
SQLITE_ROW, 280
synthesize, 29

U
UIAccelerometer
delegate, 353
sharedAccelerometer, 342
updateInterval, 344
UIAccelerometerDelegate, 342
Accelerometer:didAccelerate:, 342
UIActionSheet, 192
UIAlertView, 190

UIApplication
openURL:, 361
sendActionsForControlEvents:, 124
UIButton, 135
buttonWithType :,135
UIControl, 122
controlEvents, 124
enabled, 123
highlighted, 123
selected, 123
state, 123
UIDatePicker, 141
UIDevice, 347
model, 348
orientation, 348
systemName, 348
systemVersion, 348
uniqueIdentifier, 348
UIEvent, 98
allTouches, 98
UIImage
imageNamed:, 153
UIImagePickerController, 349
isSourceTypeAvailable:, 349
UIImagePickerControllerDelegate, 349
UINavigationController, 156
UINavigationItem, 166
UIPageControl, 140
UIPickerView, 177
компонент, 177
ряд, 177
UIResponder, 98
touchesBegan:withEvent:, 99
touchesCancelled:withEvent:, 99
touchesEnded:withEvent:, 99
touchesMoved:withEvent:, 99
UIScreen, 91
UISegmentedControl, 137
UISlider, 133
continuous, 133
maximumValue, 133
minimum Value, 133
UISwitch, 134
UITabBarItem, 151
badgeValue, 151
UITableView, 215
dataSource, 216
dequeueReusableCellWithIdentifier:, 217
setEditing:animated:, 223
tableView:canMoveRowAtIndexPath:, 236
UITableViewDataSource, 216
UITableViewStylePlain, 216
UITableViewCell, 215, 219
image, 220
UITableViewDelegate, 215
UITextField, 126
UITextFieldDelegate, 131
UITextInputTraits, 126
UITextView, 186
UITouch, 97
UIView
beginAnimations: 114
commitAnimations, 116
locationInView:, 98
previousLocationInView:, 98
tapCount, 98
UIViewController, 143
dismissModalViewControllerAnimated:, 153
initWithNibName:bundle:, 144
interfaceOrientation, 144
leftBarButtonItem, 168
loadView, 147
modalViewController, 171
navigationItem, 166
parentViewController, 171
presentModalViewController:animated:, 171
rightBarButtonItem, 168
shouldAutorotateToInterfaceOrientation:, 144
tabBarItem, 151
UIWebView, 193
stringByEvaluatingJavaScriptFromString:, 205
UIWebViewDelegate, 208

X
XML, 292
DOM, 296
RSS, 294
SAX, 302

А
Акселерометр, 340
Анимация, 113

Б
Бросить исключение, 37

В
Вибрация, 345
Высвободить, 26

Г
Геокодирование, 317

З
Захват, 24

И
Исключения, 36

К
Камера, 351
Категория, 34
Класс, 20
    методы, 20
    объект, 20
Кодирование «ключ-значение», 42

М
Многопотоковость, 50
Модальный контроллер представления, 170

П
Панель закладок, 149
Позиционирование, 35
Представление, 90
    геометрия, 90
    границы, 94
    дочерние представления, 96
    родительские представления, 96
    фрэйм, 93
    центр, 94
Протокол, 26
    необязательный, 27
    обязательный, 27

Р
Радиоинтерфейс, 149

С
Свойство, 29
    assign, 29
    copy, 29
    dynamic, 29
    getter, 30
    nonatomic, 29
    readonly, 29
    setter, 30
    synthesize, 30
Сообщение, 19
Список свойств, 358
Статический, 19
Супер, 27
Счетчик захватов, 24

Т
Табличное представление, 214

Ц
«Цель-действие», 123

Производственно-практическое издание

Али Махер

ПРОГРАММИРОВАНИЕ ДЛЯ IPHONE

Директор редакции *Л. Бершидский*
Ответственный редактор *В. Обручев*
Художественный редактор *С. Лебедева*

ООО «Издательство «Эксмо»
127299, Москва, ул. Клары Цеткин, д. 18/5. Тел. 411-68-86, 956-39-21.
Home page: **www.eksmo.ru** E-mail: **info@eksmo.ru**

Оптовая торговля книгами «Эксмо»:
ООО «ТД «Эксмо». 142700, Московская обл., Ленинский р-н, г. Видное,
Белокаменное ш., д. 1, многоканальный тел. 411-50-74.
E-mail: **reception@eksmo-sale.ru**

По вопросам приобретения книг «Эксмо» зарубежными оптовыми покупателями обращаться в отдел зарубежных продаж ТД «Эксмо»
E-mail: **international@eksmo-sale.ru**

International Sales: *International wholesale customers should contact Foreign Sales Department of Trading House «Eksmo» for their orders.*
international@eksmo-sale.ru

По вопросам заказа книг корпоративным клиентам, в том числе в специальном оформлении, обращаться по тел. 411-68-59 доб. 2115, 2117, 2118. E-mail: **vipzakaz@eksmo.ru**

Оптовая торговля бумажно-беловыми и канцелярскими товарами для школы и офиса «Канц-Эксмо»:
Компания «Канц-Эксмо»: 142702, Московская обл., Ленинский р-н, г. Видное-2,
Белокаменное ш., д. 1, а/я 5. Тел./факс +7 (495) 745-28-87 (многоканальный).
e-mail: **kanc@eksmo-sale.ru**, сайт: **www.kanc-eksmo.ru**

Полный ассортимент книг издательства «Эксмо» для оптовых покупателей:
В Санкт-Петербурге: ООО СЗКО, пр-т Обуховской Обороны, д. 84Е. Тел. (812) 365-46-03/04.
В Нижнем Новгороде: ООО ТД «Эксмо НН», ул. Маршала Воронова, д. 3. Тел. (8312) 72-36-70.
В Казани: Филиал ООО «РДЦ-Самара», ул. Фрезерная, д. 5. Тел. (843) 570-40-45/46.
В Ростове-на-Дону: ООО «РДЦ-Ростов», пр. Стачки, 243А. Тел. (863) 220-19-34.
В Самаре: ООО «РДЦ-Самара», пр-т Кирова, д. 75/1, литера «Е». Тел. (846) 269-66-70.
В Екатеринбурге: ООО «РДЦ-Екатеринбург», ул. Прибалтийская, д. 24а. Тел. (343) 378-49-45.
В Киеве: ООО «РДЦ Эксмо-Украина», Московский пр-т, д. 9. Тел./факс (044) 495-79-80/81.
Во Львове: ТП ООО «Эксмо-Запад», ул. Бузкова, д. 2. Тел./факс (032) 245-00-19.
В Симферополе: ООО «Эксмо-Крым», ул. Киевская, д. 153. Тел./факс (0652) 22-90-03, 54-32-99.
В Казахстане: ТОО «РДЦ-Алматы», ул. Домбровского, д. 3а. Тел./факс (727) 251-59-90/91.
rdc-almaty@mail.ru

Полный ассортимент продукции издательства «Эксмо»:
В Москве в сети магазинов «Новый книжный»:
Центральный магазин — Москва, Сухаревская пл., 12. Тел. 937-85-81.
Волгоградский пр-т, д. 78, тел. 177-22-11; ул. Братиславская, д. 12. Тел. 346-99-95.
Информация о магазинах «Новый книжный» по тел. 780-58-81.
В Санкт-Петербурге в сети магазинов «Буквоед»:
«Магазин на Невском», д. 13. Тел. (812) 310-22-44.

Подписано в печать 15.07.2010. Формат 70x100$^{1}/_{16}$.
Печать офсетная. Усл. печ. л. 29,81.
Тираж 1500 экз. Заказ № 5837

Отпечатано с готовых файлов заказчика в ОАО «ИПК «Ульяновский Дом печати». 432980, г. Ульяновск, ул. Гончарова, 14

ISBN 978-5-699-40764-4