



Содержание

Предисловие	10
Для кого эта книга	11
Что необходимо для опробования примеров	11
Структура книги	12
Типографские соглашения	13
Использование программного кода примеров	14
Safari® Books Online	15
Как с нами связаться	15
Благодарности	16
Глава 1. Создание веб-приложений для ASP.NET MVC 4 на языке F#	17
Шаблоны проектов F# ASP.NET MVC 4	18
Поиск и установка шаблонов	19
Проект на C#	20
Проект на F#	21
Global.fs	21
HomeController.fs	23
Контроллеры и модели на F#	24
Контроллеры	25
Модели	26
Взаимодействие с базой данных	28
Entity Framework.....	28
Извлечение данных.....	31
Извлечение данных с использованием поставщиков типов	32
Использование преимуществ F#	34
Переход на функциональную парадигму	34
Конвейеры и частичное применение функций.....	36
Создание более функционального контроллера	38

Упрощение за счет сопоставления с образцом	40
Дополнительные темы и понятия.....	44
Улучшение времени отклика с помощью асинхронных операций	44
Кеширование с применением MailboxProcessor	46
Сообщения, как значения типа размеченного объединения	47
Основной агент	48
Использование агента CacheAgent.....	49
Шина сообщений.....	51
SimpleBus.....	52
Публикация сообщений.....	54
Извлечение сообщений.....	56
Стиль продолжений	57
Создание собственных вычислительных выражений.....	58
В заключение	60

Глава 2. Создание веб-служб на языке F#

Установка шаблона проекта WCF	63
Исследование получившегося решения	64
Использование службы.....	67
Погружение в записи	72
Создание службы ASP.NET Web API.....	73
Анализ шаблона.....	74
Взаимодействие с HTTP-службой	78
С использованием объекта HttpClient	79
Поставщик типов JSON	82
Прежде чем покинуть ASP.NET Web API	83
Другие веб-фреймворки	84
Service Stack	84
Nancy.....	87
Frank.....	90
Тестирование своих творений	94
Подготовка	94
Улучшение тестов с применением F#.....	97
FsUnit.....	99
Unquote	101
NaturalSpec.....	102
В заключение	104

Глава 3. К облакам! Использование преимуществ Azure	105
Создание и развертывание приложений F# на платформе Azure	106
Создание рабочей роли на F#	108
Введение в библиотеку Fog	109
Взаимодействие с хранилищами данных Azure.....	110
Большие двоичные объекты.....	110
Таблицы.....	112
Служба хранения очередей.....	114
SQL Azure.....	115
Использование преимуществ Azure Service Bus.....	116
Очереди	116
Темы.....	117
Аутентификация и авторизация	119
Аутентификация и авторизация с применением ACS	120
Аутентификация на основе заявок.....	121
Авторизация на основе заявок.....	122
Создание масштабируемых приложений.....	123
Создание веб-роли.....	124
PlaceOrderCommand	126
Рабочие роли.....	127
Рабочая роль SQL Azure	128
Последние штрихи.....	130
Кеширование.....	131
CDN и автоматическое масштабирование	132
Блистательные примеры на F#	133
{m}brace	134
Cloud Numerics	135
Hadoop MapReduce для .NET	136
В заключение	136
Глава 4. Создание масштабируемых мобильных и веб-приложений	137
Масштабирование с применением веб-сокетов.....	138
Пример использования веб-сокетов на платформе .NET 4.5 и IIS 8	139
Создание сервера веб-сокетов с помощью Fleck.....	144

SignalR	147
Пример создания постоянного соединения	148
Клиент на JavaScript	149
Клиент на F#	150
Пример создания хаба	150
Серверная сторона	151
Клиентская сторона	152
Обретаем мобильность	153
Способ на основе jQuery Mobile	153
Добавляем поддержку Windows Phone	155
Объединение F# и NoSQL.....	158
MongoDB	159
RavenDB	162
CouchDB	163
В заключение	165

Глава 5. Разработка интерфейсов

в функциональном стиле	166
Подготовка почвы.....	167
Знакомство с LiveScript	168
Преимущества.....	168
Применение	169
Пример.....	171
Исследуем Pit.....	173
Преимущества.....	174
Применение	175
Пример.....	176
Погружение в WebSharper	179
Преимущества.....	180
Применение	181
Пример.....	182
В заключение	184

Приложение А. Полезные инструменты и библиотеки

186	186
FAKE (F# Make).....	186
NuGet	186
Основы использования.....	187
Полезные NuGet-пекты.....	188
ExpectThat	192

Приложение В. Полезные веб-сайты	194
fssnip.net	194
tryfsharp.org	194
Visual Studio Gallery.....	195
jQueryMobile.com	195
Приложение С. Клиентские технологии, совместимые с F#	196
CoffeeScript	196
Sass	197
Underscore.js	200
Об авторе	201
Предметный указатель	202



Предисловие

Если проанализировать самые последние веянья в развитии информационных технологий, можно увидеть, что основным направлением является создание облачных, мобильных и веб-решений, масштабируемых в широких пределах и связанных с управлением большими объемами данных. С появлением этих направлений возникла потребность в инструментах, позволяющих специалистам, таким как вы или я, создавать собственные решения в этой области. Что для этого нужно? Какие архитектуры, инструменты, языки и технологии можно использовать для разработки программ, способных выполняться на самых разных устройствах и легко масштабироваться, и при этом обеспечить высокую надежность решений, простоту их сопровождения, тестирования и многократного использования?

Существует множество инструментов, отвечающих нашим потребностям, но для решения наших задач в полном объеме, их возможностей оказывается недостаточно. Чтобы получить максимальную отдачу, необходим язык, специально предназначенный для преодоления сложностей, возникающих в описанных областях разработки. Он должен иметь встроенные средства для преодоления проблем, связанных с конкуренцией, выполнением асинхронных операций и большими объемами данных, и при этом прозрачно интегрироваться с другими языками, технологиями и инструментами, лучше подходящих для решения других задач. К счастью, такой язык существует и называется F#.

В этой книге я покажу, как использовать язык F# для реализации ключевых элементов облачных, мобильных и веб-приложений, и решения упомянутых проблем. Выразительность, широта возможностей, лаконичность и функциональная природа языка F#, в сочетании с уже известными вам технологиями, такими как ASP.NET MVC, ASP.NET Web API, WCF, Windows Azure, HTML5, CSS3, JavaScript, jQuery и jQuery Mobile, позволят вам создавать удивительные приложения, не только соответствующие, но и превосходящие текущие и будущие требования к ним.

Для кого эта книга

Эта книга предназначена для специалистов с опытом работы в .NET, слышавших о преимуществах F#, имеющих хотя бы общее представление о его синтаксисе, и желающих узнать, как объединять F# с другими технологиями для создания облачных, мобильных и веб-приложений. Если вы совершенно не знакомы с F#, я предлагаю заглянуть в другие книги, описывающие основы программирования на этом языке, такие как книга Криса Смита (Chris Smith) «Programming F#, 3.0» (O'Reilly)¹. Если вы не знакомы с другими платформами и фреймворками, упоминаемыми в этой книге, такими как ASP.NET MVC, WCF, ASP.NET Web API, Windows Azure, HTML, CSS и/или jQuery Mobile, их описание можно найти во множестве других книг, где вы сможете почерпнуть всю необходимую информацию.

Что необходимо для опробования примеров

Большая часть примеров для этой книги была создана с помощью Visual Studio 2012. Для опробования примеров я рекомендую использовать версию Visual Studio 2012 Professional или выше; однако, большинство примеров будет также работать в среде F# Tools для Visual Studio Express 2012 for Web, анонсированной 12 сентября 2012 в блоге команды разработчиков F#². Загрузить F# Tools для Visual Studio Express 2012 for Web можно по адресу: <http://www.microsoft.com/web/gallery/install.aspx?appid=FSharpVWD11>. В зависимости от целевой платформы или фреймворка, может потребоваться установить следующие инструменты:

- ❑ ASP.NET MVC 4, можно загрузить по адресу: <http://www.asp.net/mvc/mvc4>.
- ❑ Windows Azure SDK и Developer Tools, можно загрузить по адресу: <http://www.windowsazure.com/en-us/develop/net/>.

Дополнительные библиотеки и инструменты, которые могут потребоваться, упоминаются в соответствующих главах.

¹ Крис Смит, «Программирование на F#», ISBN: 978-5-93286-199-8, Символ-Плюс, 2011. – *Прим. перев.*

² <http://bit.ly/fsharp-blog>.

Структура книги

В этой книге рассказывается обо всем, что необходимо знать, чтобы приступить к разработке облачных, мобильных и веб-приложений на языке F#. Кроме того, здесь описывается множество новейших технологий, платформ и библиотек, таких как Windows Azure, jQuery Mobile, SignalR, CouchDB, RavenDB, MongoDB и других. Ниже подробнее описывается, что вы увидите в каждой главе.

Глава 1, «Создание веб-приложений для ASP.NET MVC 4 на языке F#»

В этой главе рассказывается обо всем, что необходимо знать, чтобы приступить к созданию веб-приложений на языке F# с использованием фреймворка ASP.NET MVC 4 на стороне сервера. Здесь также демонстрируются некоторые дополнительные возможности и особенности языка F#, позволяющие писать более элегантный код.

Глава 2, «Создание веб-служб на языке F#»

Эта глава знакомит с инструментами и понятиями, используемыми при создании веб-служб различных типов, включая службы WCF SOAP и HTTP, и особенностями взаимодействий с некоторыми маленькими веб-фреймворками. Здесь также рассказывается об инструментах и приемах модульного тестирования этих веб-служб.

Глава 3, «К облакам! Использование преимуществ Azure»

Эта глава проведет вас через создание веб-приложений и веб-служб на языке F#, выполняющихся под управлением Windows Azure. Дополнительно в ней даются примеры на F# взаимодействий с различными библиотеками Azure. В конце главы будут представлены некоторые замечательные библиотеки и фреймворки на F# для использования на платформе Azure.

Глава 4, «Создание масштабируемых мобильных и веб-приложений»

В этой главе более подробно освещаются вопросы совместного использования F# с другими технологиями для создания масштабируемых решений, позволяющими повторно использовать мобильные и веб-интерфейсы. Глава включает информацию и примеры использования веб-сокетов, библио-

теки SignalR, различных баз данных NoSQL, и многих других механизмов.

Глава 5, «Разработка интерфейсов в функциональном стиле»

Эта глава знакомит с LiveScript, Pit и WebSharper – инструментами, позволяющими, кроме всего прочего, писать клиентский код в функциональном стиле. Они делают возможным создавать комплексные веб-стеки с использованием концепций функционального программирования. Для каждого инструмента перечисляются его преимущества, начальная информация и примеры использования.

В конце книги вы найдете несколько приложений с информацией, которая может вам пригодиться в освоении приемов разработки ультрасовременных облачных, мобильных и веб-приложений, но не относящаяся к темам, обсуждаемым в основных главах.

Приложение А, «Полезные инструменты и библиотеки»

Здесь перечисляются и коротко описываются некоторые инструменты, которые могут облегчить вам жизнь, как разработчика облачных, мобильных и веб-решений.

Приложение В, «Полезные веб-сайты»

Здесь приводятся ссылки на веб-сайты, предлагающие информацию о языке F#, а также инструменты и библиотеки, упоминаемые в книге.

Приложение С, «Клиентские технологии, совместимые с F#»

Здесь дается краткий обзор некоторых технологий, дополняющих F# при разработке мобильных и веб-приложений.

Типографские соглашения

В этой книге приняты следующие соглашения:

Курсив

Курсив применяется для выделения новых терминов, имен файлов и их расширений.

Моноширинный шрифт

Применяется для представления листингов программного кода, а также в основном тексте для выделения элементов про-

грамм, таких как имена переменных и функций, базы данных, типы данных, переменные окружения, инструкции и ключевые слова.

Моноширинный жирный

Используется для выделения команд или другого текста, которые должны вводиться пользователем.

Моноширинный наклонный

Обозначает текст, который должен замещаться фактическими значениями, вводимыми пользователем или определяемыми из контекста.

Примечание. Так обозначаются советы, предложения и примечания общего характера.

Внимание. Так обозначаются предупреждения и предостережения.

Использование программного кода примеров

Данная книга призвана оказать вам помощь в решении ваших задач. Вы можете свободно использовать примеры программного кода из этой книги в своих приложениях и в документации. Вам не нужно обращаться в издательство за разрешением, если вы не собираетесь воспроизводить существенные части программного кода. Например, если вы разрабатываете программу и используете в ней несколько отрывков программного кода из книги, вам не нужно обращаться за разрешением. Однако в случае продажи или распространения компакт-дисков с примерами из этой книги вам необходимо получить разрешение от издательства O'Reilly. Если вы отвечаете на вопросы, цитируя данную книгу или примеры из нее, получение разрешения не требуется. Но при включении существенных объемов программного кода примеров из этой книги в вашу документацию, вам необходимо будет получить разрешение издательства.

Мы приветствуем, но не требуем добавлять ссылку на первоисточник при цитировании. Под ссылкой на первоисточник мы подразумеваем указание авторов, издательства и ISBN. Например: «Building Web, Cloud, and Mobile Solutions with F# by Daniel Mohl (O'Reilly). Copyright 2013 Daniel Mohl, 978-1-449-33376-8».

За получением разрешения на использование значительных объемов программного кода примеров из этой книги обращайтесь по адресу permissions@oreilly.com.

Safari® Books Online

Safari Books Online (www.safaribooksonline.com) – это виртуальная библиотека, содержащая авторитетную информацию в виде книг и видеоматериалов, созданных ведущими специалистами в области технологий и бизнеса.

Профессионалы в области технологии, разработчики программного обеспечения, веб-дизайнеры, а также бизнесмены и творческие работники используют Safari Books Online как основной источник информации для проведения исследований, решения проблем, обучения и подготовки к сертификационным испытаниям.

Библиотека Safari Books Online предлагает широкий выбор продуктов и тарифов для организаций, правительственных учреждений и физических лиц. Подписчики имеют доступ к поисковой базе данных, содержащей информацию о тысячах книг, видеоматериалов и рукописей от таких издателей, как O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology и десятков других. За подробной информацией о Safari Books Online обращайтесь по адресу: <http://www.safaribooksonline.com/>.

Как с нами связаться

С вопросами и предложениями, касающимися этой книги, обращайтесь в издательство:

O'Reilly Media
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (в Соединенных Штатах Америки или в Канаде)
707-829-0515 (международный)
707-829-0104 (факс)

Список опечаток, файлы с примерами и другую дополнительную информацию вы найдете на сайте книги: <http://oreil.ly/building-web>.

Свои пожелания и вопросы технического характера отправляйте по адресу: bookquestions@oreilly.com.

Дополнительную информацию о книгах, обсуждения, Центр ресурсов издательства O'Reilly вы найдете на сайте: <http://www.oreilly.com>.

Ищите нас на Facebook: <http://facebook.com/oreilly>.

Следуйте за нами на Твиттере: <http://twitter.com/oreillymedia>.

Смотрите нас на YouTube: <http://www.youtube.com/oreillymedia>.

Благодарности

Прежде всего я хотел бы поблагодарить мою супругу Мелиссу (Melissa) и дочь Еву (Eva) за то, что поддерживали меня долгие часы сидения за компьютером, когда я работал над этой книгой.

Я также хочу сказать спасибо всем, кто перечислен ниже, за их труд, поддержку и помощь, и за все их лучшие качества!

- ❑ Дон Сайм (Don Syme) и все остальные члены команды разработчиков F#;
- ❑ Рейчел Румелиотис (Rachel Roumeliotis);
- ❑ Илайджа Манор (Elijah Manor);
- ❑ Стивен Свенсен (Stephen Swensen);
- ❑ Фахад Сухаиб (Fahad Suhaib);
- ❑ Райан Райли (Ryan Riley);
- ❑ Стеффен Форкман (Steffen Forkmann);
- ❑ Антон Таяновский (Anton Tayanovskyy);
- ❑ Адам Гранич (Adam Granicz).



Глава 1. Создание веб-приложений для ASP.NET MVC 4 на языке F#

Любая достаточно развитая технология неотличима от магии.

– Сэр Артур Чарльз Кларк
(Arthur Charles Clarke)

Я всегда испытывал благоговение перед волшебством и с раннего детства обожал наблюдать за фокусниками. Повзрослев, я стал читать все книги подряд, какие только мог найти, описывающие секреты фокусов, изумлявших меня в течение стольких лет. Вскоре я поймал себя на мысли, что изучать секреты фокусов мне нравится больше, чем смотреть их.

Как заметил сэр Артур Чарльз Кларк, технологии часто сравнимы с магией. Возможно поэтому я так полюбил технические науки. Язык F# относится к этой категории даже больше, чем другие языки, которые мне приходилось использовать в моей карьере программиста. Особенности этого языка открывают такие широкие возможности, что их с полным основанием можно назвать волшебством. Иногда бывает трудно определить, как лучше применить это волшебство на практике для создания еще более масштабируемых облачных, мобильных и веб-приложений, работающих еще лучше, еще быстрее. Эта книга покажет вам, как использовать все возможности языка F# для решения повседневных задач разработки.

В этой главе мы начнем свое путешествие с исследования возможности интеграции F# с фреймворком ASP.NET MVC 4. Здесь вы узнаете, как создать проект, как вести разработку на F# с использованием фреймворка ASP.NET MVC и как применять некоторые дополнительные возможности языка F# для улучшения программного кода. Мы также рассмотрим некоторые темы и приемы, не имею-

щие прямого отношения к ASP.NET MVC 4, но часто используемые вместе с этим фреймворком. На протяжении всей главы мы будем снимать покров тайны с особенностей F#, которые на первый взгляд могут показаться магическими.

В последующих главах мы будем знакомиться с другими платформами, технологиями, библиотеками и механизмами, которые можно использовать в программах на языке F# для создания ультрасовременных облачных, мобильных и веб-решений.

Шаблоны проектов F# ASP.NET MVC 4

О выходе предварительной версии ASP.NET MVC 4 для разработчиков было объявлено после конференции Build Conference во второй половине 2011 года. В феврале 2012 было объявлено о выходе бета-версии ASP.NET MVC 4 и в конце мая 2012 последовала предвыпускная (release candidate) версия. Версия 4 принесла множество улучшений и усовершенствований в и без того полнофункциональный фреймворк ASP.NET MVC. Дополнительную информацию о ASP.NET MVC 4 можно найти на веб-сайте проекта <http://www.asp.net/mvc/mvc4>.

Самый эффективный способ интеграции F# с фреймворком ASP.NET MVC 4 – использовать преимущества разделения задач, присущие шаблону проектирования «модель–представление–контроллер» (Model–View–Controller, MVC). Это разграничение можно с успехом использовать для усиления экосистемы C# возможностями языка F#. В случае с фреймворком ASP.NET MVC, это достигается путем создания проекта C# ASP.NET MVC, где будут сосредоточены представления и все программные компоненты, выполняющиеся на стороне клиента, и проекта на F#, для реализации моделей, контроллеров и других компонентов, выполняющихся на стороне сервера. На рис. 1.1 показана реализация типичного шаблона проектирования MVC в ASP.NET MVC с обозначением типов компонентов.

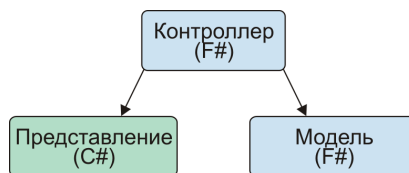


Рис. 1.1. Шаблон проектирования MVC с обозначением типов компонентов

Конечно, приложение с подобной структурой можно создать и вручную, но такой подход быстро становится утомительным. Кроме того, рутинные подготовительные операции являются дополнительным барьером на пути к использованию F# в приложениях на основе ASP.NET MVC. Чтобы помочь устранить эти проблемы, был создан шаблон проекта, доступный в галерее шаблонов проектов Visual Studio Gallery.

Примечание. Для тех, кто по каким-то причинам не может использовать шаблоны проектов ASP.NET MVC 4, в галерее также присутствуют шаблоны ASP.NET MVC 3 и ASP.NET MVC 2. Список большинства доступных шаблонов можно найти по адресу: <http://bit.ly/allfsharpprojecttemplates>.

Поиск и установка шаблонов

Благодаря галерее шаблонов проектов Visual Studio Gallery, поиск и установка шаблонов проектов F# ASP.NET MVC 4 выполняются проще некуда. Просто запустите мастер создания нового проекта любым способом, по вашему выбору, – я предпочитаю комбинацию клавиш **Ctrl+Shift+N** – Выберите пункт **Online (В Интернете)** в левой панели, введите текст «fsharp mvc4» в строке поиска в правом верхнем углу окна, выберите шаблон «F# C# MVC 4» и щелкните на кнопке **OK**. На рис. 1.2 изображено окно мастера создания нового проекта в момент, непосредственно перед щелчком на кнопке **OK**.

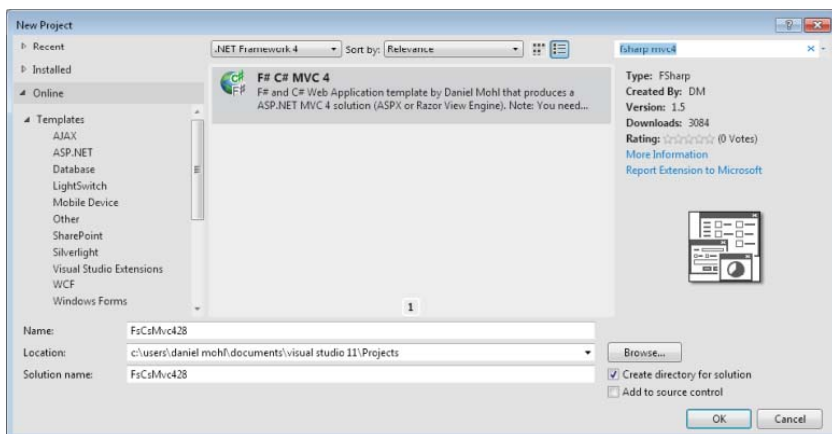


Рис. 1.2. Поиск шаблона проекта в галерее Visual Studio Gallery

Примечание. Описанный порядок действий можно использовать каждый раз при создании нового проекта F# ASP.NET MVC 4, но в действительности достаточно выполнить их один раз. После первичной установки новый шаблон будет доступен в категории «Installed» («Установленные»), в панели слева. Шаблону будет дано имя «F# and C# Web Application (ASP.NET MVC 4)» и вы сможете выбирать его в категории Visual F#→ASP.NET.

После щелчка на кнопке **ОК** появится диалог (изображенный на рис. 1.3), где можно выбрать тип приложения и механизм представлений (раскрывающийся список **View Engine**), а также необходимость включения дополнительного проекта, где будут находиться модульные тесты. После выбора нужных параметров щелкните на кнопке **ОК**. В результате будут созданы все необходимые проекты и установлены пакеты NuGet. В большинстве оставшихся примеров в этой главе будет предполагаться, что в процессе создания приложения был выбран механизм представлений Razor.

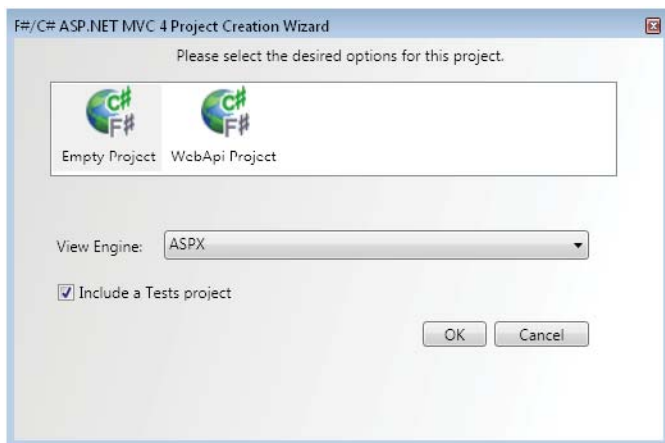


Рис. 1.3. Диалог мастера создания проекта F# ASP.NET MVC

Проект на C#

Если прежде вам приходилось создавать проекты ASP.NET MVC только на C#, приложение C#, созданное выше, покажется вам очень знакомым. В действительности рассматриваемый проект имеет всего три основных отличия:

1. Отсутствует папка *Controllers*.
2. Отсутствует папка *Models*.
3. Файл *Global.asax* не имеет соответствующего ему файла *Global.asax.cs*.

Главная причина этих отличий в том, что перечисленные элементы были перемещены в проект на F#, сгенерированный вместе с данным проектом на C#, но подробнее проект на F# будет рассматриваться в следующем разделе. Файл *Global.asax* не представляет большого интереса. В нем определен лишь один метод для связи с классом на F#. В следующем фрагменте показано содержимое файла *Global.asax*:

```
<%@ Application Inherits="FsWeb.Global" Language="C#" %>
<script Language="C#" RunAt="server">

    // Определение метода Application_Start, вызывающего метод Start
    // класса System.Web.HttpApplication, который наследуется классом Global.
    protected void Application_Start(Object sender, EventArgs e) {
        base.Start();
    }
}
</script>
```

Проект на F#

Если в диалоге мастера создания проекта (рис. 1.3) был выбран шаблон «Empty Project» (пустой проект), получившийся проект на F# будет очень прост. В проект автоматически будут добавлены все необходимые ссылки на сборки MVC и два файла *.fs*: *Global.fs* и *HomeController.fs*. Я уже коротко упоминал файл *Global.fs* и уверен, что вы уже догадались, что содержит файл *HomeController.fs*. Рассмотрим их подробнее в этом разделе.

Global.fs

Как уже упоминалось, файл *Global.fs* содержит большую часть кода, который обычно находится в файле *Global.asax.cs*, но с некоторыми особенностями, характерными для F#. Первое, что можно в нем заметить, – определение типа *Route*. Это *тип записи* на языке F#, предназначенный для создания определений маршрутов. Типы записей по умолчанию являются неизменяемыми. Поэтому они хорошо согласуются с конкурентной природой Веб, не предполагаю-

щей хранения информации о состоянии. Подробнее о типах записей я буду рассказывать далее в этой книге. Тип `Route` объявлен, как показано ниже:

```
type Route = { controller : string
              action : string
              id : UrlParameter }
```

Примечание. Тип `Route` используется только для определения стандартных маршрутов контроллер/действие/ID. Для определения маршрутов других видов необходимо создавать собственные типы.

За объявлением типа `Route` следует определение класса `Global`, который наследует класс `System.Web.HttpApplication`. Код в классе `Global` выглядит почти так же, как в определении аналогичного ему класса на языке `C#`, за исключением вызова метода `MapRoutes` и использования значимых пробелов вместо фигурных скобок для определения области видимости. Главное отличие в вызове метода `MapRoutes` напрямую связано с типом `Route`. Для передачи информации о маршрутах методу `MapRoutes`, благодаря механизму определения типов в языке `F#`, вместо нового анонимного типа создается новая запись типа `Route`. Такой синтаксис создания записей называется *выражение записи* (*record expression*). Ниже приводится определение класса `Global`, где выделен фрагмент, выполняющий создание записи типа `Route`:

```
type Global() =
    inherit System.Web.HttpApplication()

    static member RegisterRoutes(routes:RouteCollection) =
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}")
        routes.MapRoute("Default",
            "{controller}/{action}/{id}",
            { controller = "Home"; action = "Index"
              id = UrlParameter.Optional } )

    member this.Start() =
        AreaRegistration.RegisterAllAreas()
        Global.RegisterRoutes(RouteTable.Routes)
```

HomeController.fs

Файл `HomeController.fs` содержит определение класса `HomeController`. Он наследует класс `Controller` и реализует единственное действие с именем `Index`. Подробнее о контроллерах будет рассказываться ниже в этой главе. Файл `HomeController.fs` содержит следующий код:

```
namespace FsWeb.Controllers

open System.Web
open System.Web.Mvc

[<HandleError>]
type HomeController() =
    inherit Controller()
    member this.Index () =
        this.View() :> ActionResult
```

Кого-то может смутить комбинация символов `:>`, выделенная в предыдущем примере. Эта последовательность обозначает приведение типа вверх (`upcast`) результата вызова `this.View()` к типу `ActionResult`. В данном примере приведение к типу `ActionResult` не является необходимостью, но может потребоваться в некоторых других случаях, поэтому разработчики добавили приведение типа вверх в шаблон с целью демонстрации. Если бы тип возвращаемого значения метода `Index` был определен явно:

```
member this.Index () : ActionResult = ...
```

тогда приведение типа следовало бы записать так:

```
upcast this.View()
```

Так как в данном конкретном случае приведение типа не требуется, этот метод можно упростить, как показано ниже:

```
member this.Index () =
    this.View()
```

Примечание. Проверка возможности приведения типа вверх (`upcast`) производится на этапе компиляции, чтобы гарантировать его допусти-

мость. Но возможность приведения типа вниз (downcast) (например, оператором `:?>`) может быть проверена только на этапе выполнения. Если есть вероятность, что приведение типа вниз может потерпеть неудачу, рекомендуется предварительно выполнять проверку типа с помощью выражения сопоставления (match expression). Выражение приведения типа вниз можно также заключить в инструкцию `try/with` и предусмотреть обработку исключения `InvalidCastException`, но такое решение менее эффективно, чем проверка типа.

Контроллеры и модели на F#

Основная цель этой книги состоит в том, чтобы показать, как лучше использовать F# в обширном стеке технологий, поэтому о контроллерах и моделях будет рассказываться намного больше, чем о представлениях. Язык F# обладает рядом уникальных особенностей, прекрасно подходящих для реализации различных аспектов контроллеров и моделей. С некоторыми из них я познакомлю вас в этом разделе, а в следующих расскажу о более совершенных возможностях.

Чтобы вам было проще, обсуждение контроллеров и моделей будет вестись на примере создания новой страницы в веб-приложении, при этом особое внимание будет уделяться коду, реализующему создание модели и контроллера. Эта страница будет отображать список простое представление списка jQuery Mobile, управляемое и заполняемое новым контроллером и моделью.

Сначала создадим новое представление. Для этого создайте в папке *Views* новую папку *Guitars* и добавьте туда новое представление ASP.NET MVC с именем *Index*. Не забудьте снять флажок **Use a layout or master page:** (Использовать макет или главную страницу) в диалоге мастера создания элемента представления ASP.NET MVC. Теперь можно изменить разметку представления, как показано ниже:

```
@model IEnumerable<FsWeb.Models.Guitar>
<!DOCTYPE html>
<html>
<head>
  <title>@ViewBag.Title</title>
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <link rel="stylesheet"
        href="http://code.jquery.com/mobile/1.0.1/jquery.mobile-1.0.1.min.css" />
</head>

<body>
```

```
<div data-role="page" data-theme="a" id="guitarsPage">
  <div data-role="header">
    <h1>Guitars</h1>
  </div>
  <div data-role="content">
    <ul data-role="listview" data-filter="true" data-inset="true">
      @foreach(var x in Model) {
        <li><a href="#">@x.Name</a></li>
      }
    </ul>
  </div>
  <script src="http://code.jquery.com/jquery-1.6.4.min.js">
  </script>
  <script src="http://code.jquery.com/mobile/1.0.1/jquery.mobile-1.0.1.min.js">
  </script>

  <script>
    $(document).delegate("#guitarsPage", 'pageshow', function (event) {
      $("#div:jqmData(role='content') > ul").listview('refresh');
    });
  </script>
</body>
</html>
```

Примечание. Поскольку представление не является основной целью этого раздела, для простоты я поместил его целиком в один файл с расширением `.cshtml`. Обычно же код на JavaScript принято помещать в отдельный модуль (и, иногда, использовать дополнительные инструменты, такие как библиотека RequireJS, упрощающие загрузку и управление модулями JavaScript). Кроме того, может потребоваться создать отдельную страницу `Layout` для использования во всех страницах мобильного интерфейса. Дополнительно, фреймворк ASP.NET MVC 4 поддерживает ряд соглашений, в соответствии с которыми добавляет к именам представлений «`.Mobile`» или другое окончание, в зависимости от вида устройства. Пoblize познакомиться с рекомендуемыми приемами создания представлений можно по адресу: <http://www.asp.net/mvc/tutorials>.

Контроллеры

Чтобы создать простейший контроллер для нового представления, добавьте в проект на F# новый файл с исходным кодом, дайте ему имя `GuitarsController.fs` и сохраните в нем следующий код:

```
namespace FsWeb.Controllers

open System.Web.Mvc
open FsWeb.Models

[<HandleError>]
type GuitarsController() =
    inherit Controller()
    member this.Index () =
        // Последовательность жестко определена, исключительно в
        // демонстрационных целях.
        // Будет удалена в будущем примере.
        seq { yield Guitar(Name = "Gibson Les Paul")
              yield Guitar(Name = "Martin D-28") }
        |> this.View
```

Выглядит очень похоже на HomeController, за исключением выражения последовательности (sequence expression) и *прямого конвейерного оператора* (pipe-forward). В этом примере выражение последовательности определяет коллекцию экземпляров модели Guitar для передачи представлению. В будущем примере эти «жестко зашитые» данные мы заменим обращением к хранилищу данных.

Второй интересный момент – использование прямого конвейерного оператора. В этом примере прямой конвейерный оператор используется для передачи последовательности экземпляров Guitar в аргументе модели перегруженному методу View, принимающему единственный аргумент obj.

Примечание. Ключевое слово obj – это псевдоним типа object в языке F#. Подробнее о псевдонимах типов будет рассказываться в главе 2.

Модели

Модели могут быть экземплярами записей или классов. Стандартные записи языка F# отлично подходят для представления данных, доступных только для чтения, и обеспечивают простоту моделей. В версии F# 3.0 появился новый атрибут CLIMutable, превращающий записи языка F# в отличный выбор, когда данные также должны быть доступны для чтения/записи. Подробнее об атрибуте CLIMutable будет рассказываться в главе 4. Ниже приводится пример модели Guitar, сконструированной на основе записи:

```
namespace FsWeb.Models
```

```
type Guitar = { Id : Guid; Name : string }
```

Примечание. В версиях F#, ниже F# 3.0, записи также можно было использовать для представления изменяемых данных, хотя и с некоторыми сложностями, обусловленными отсутствием у записей конструкторов без параметров. Более удачным решением этой проблемы (до версии F# 3.0) было использование собственного механизма связывания моделей (model binder).

Второй способ определения моделей на языке F# основан на классах. Пример контроллера в предыдущем разделе предполагает, что используется подход на основе класса. Следующий пример демонстрирует, как определить класс модели `Guitar` (этот класс, как и большая часть примеров в этой книге, был написан с учетом особенностей версии F# 3.0; в версии F# 2.0 синтаксис может несколько отличаться, потому что автоматические свойства (auto-properties) появились только в версии F# 3.0):

```
namespace FsWeb.Models
```

```
type Guitar() =  
    member val Name = "" with get, set
```

В класс модели допускается добавлять любые атрибуты аннотаций данных (Data Annotations). В следующем примере я добавил атрибут `Required` к свойству `Name`:

```
open System.ComponentModel.DataAnnotations
```

```
type Guitar() =  
    [<Required>] member val Name = "" with get, set
```

Примечание. В данном примере атрибуту `Required` не передается значение, но такая возможность может пригодиться во многих случаях, где поддерживается возможность изменения из пользовательского интерфейса.

Ниже приводится модель, которая будет использоваться в примере на основе фреймворка Entity Framework, в следующем разделе:

```
namespace FsWeb.Models

open System
open System.ComponentModel.DataAnnotations

type Guitar() =
    [<Key>] member val Id = Guid.NewGuid() with get, set
    [<Required>] member val Name = "" with get, set
```

Взаимодействие с базой данных

Если запустить веб-приложение прямо сейчас, вы увидите простую страницу, отображающую список названий гитар. Но в этом мало проку, потому что данные жестко определены в исходном коде. К счастью, в F# имеется несколько средств на выбор, позволяющих обращаться к базам данных для сохранения и извлечения данных.

Entity Framework

Фреймворк Entity Framework (EF) – это, пожалуй, один из наиболее распространенных в ASP.NET MVC инструментов взаимодействия с базами данных SQL Server и его распространение продолжается, особенно теперь, когда EF поддерживает поход «сначала код» (code-first). Шаблон F#/C# ASP.NET MVC 4 уже добавил ссылки на сборки, необходимые для работы с EF, поэтому можно сразу приступить к использованию фреймворка и создать класс, наследующий класс DbContext, как показано в следующем примере:

```
namespace FsWeb.Repositories

open System.Data.Entity
open FsWeb.Models

type FsMvcAppEntities() =
    inherit DbContext("FsMvcAppExample")

    do Database.SetInitializer(new CreateDatabaseIfNotExists<FsMvcAppEntities>())

    [<DefaultValue(>>] val mutable guitars : IDbSet<Guitar>
    member x.Guitars with get() = x.guitars and set v = x.guitars <- v
```

Здесь не происходит ничего особенного. Мы просто использовали некоторые стандартные особенности EF API для определения множества IDbSet гитар, и создали свойство Guitars с методами чтения и записи.

Примечание. Поближе познакомиться с EF API можно по адресу: <http://bit.ly/efcodefirstwalkthrough>.

Теперь необходимо добавить класс репозитория, чтобы получить возможность извлекать информацию о гитарах из базы данных.

Примечание. Технически класс репозитория не нужен, но многие считают его удобным, а его применение в приложениях стало стандартной практикой.

Следующий пример содержит определение класса GuitarsRepository:

```
namespace FsWeb.Repositories
{
    type GuitarsRepository() =
        member x.GetAll () =
            use context = new FsMvcAppEntities()
            query { for g in context.Guitars do
                    select g }
        |> Seq.toList
}
```

Примечание. Если вы вынуждены использовать EF в F# 2.0, синтаксис запроса в примере выше не будет работать (поддержка запросов появилась только в версии F# 3.0). Аналогичную возможность в версии F# 2.0 можно получить, если установить пакет NuGet с именем `FSPowerPack.Linq.Community`, открыть `Microsoft.FSharp.Linq.Query` и *заменить код запроса следующим*:

```
query <@ seq { for g in context.Guitars -> g } @> |> Seq.toList
```

Эта реализация из F# PowerPack использует особенность языка F# с названием «цитируемые выражения» (quoted expressions), позволяющую сгенерировать абстрактное синтаксическое дерево (Abstract Syntax Tree, AST) и обработать его. Цитируемые выражения применяются для самых разных нужд, но чаще всего они используются, чтобы сгенерировать код на F# или других языках.

Первое, что делает метод `GetAll`, – создает экземпляр `DbContext`. Обратите внимание, что вместо стандартного ключевого слова `let` здесь

используется `use`. Это гарантирует своевременное удаление объекта по окончании его использования – напоминает обертывание кода инструкцией `using` в языке C#, но не требует обертывать какой-либо дополнительный код.

Затем метод `GetAll` выполняет запрос к базе данных. Для этой цели используется синтаксис *запросов*, появившийся в версии F# 3.0 и упрощающий операции с данными. Несмотря на то, что синтаксис запросов выглядит как новая особенность компилятора, в действительности этот механизм реализован на основе особенности языка F# с названием *вычислительные выражения* (*computation expressions*). Далее в этой главе я покажу, как создавать собственные вычислительные выражения. А в следующем разделе мы детально исследуем вычислительные выражения запросов.

Теперь осталось лишь заменить жестко определенные данные в первоначальной версии действия `Index` в классе `GuitarsController`:

```
[<HandleError>]
type GuitarsController(repository : GuitarsRepository) =
    inherit Controller()
    new() = new GuitarsController(GuitarsRepository())
    member this.Index () =
        repository.GetAll()
    |> this.View
```

Это изменение не только упростило код (особенно действие `Index`), но и добавило сложностей в виде определения нового перегруженного конструктора. Он служит следующим целям.

- Позволяет передавать репозиторий в конструктор, открывая тем самым путь к использованию принципа *инверсии управления* (*Inversion of Control, IoC*) в контейнерах. Ради простоты в предыдущий пример были включены не все изменения, необходимые для оптимального использования принципа *IoC* в контейнерах.
- Делает контроллер более удобным для тестирования. С добавлением перегруженного конструктора появляется возможность передавать репозиторий фиктивного класса, чтобы протестировать действия контроллера без подключения к фактической базе данных.

Поскольку по умолчанию фреймворк ASP.NET MVC требует наличие у контроллера конструктора без параметров, необходимо также добавить следующую строку кода:

```
new() = new GuitarsController(GuitarsRepository())
```

Она объявляет необходимый конструктор, который вызывает основной конструктор с новым объектом `GuitarsRepository`.

В заключение, прежде чем испытать возможность взаимодействий с базой данных, добавьте в файл `web.config`, находящийся в проекте C# Web Application, соответствующую строку подключения, такую как:

```
<add name="FsMvcAppExample"
      connectionString="YOUR CONNECTION STRING"
      providerName="System.Data.SqlClient" />
```

Теперь можно попробовать запустить приложение, чтобы заставить фреймворк EF автоматически создать базу данных и таблицу. Добавьте несколько записей в таблицу `Guitars` и отпразднуйте это! Как должна выглядеть веб-страница после ввода адреса [http://localhost:\[port\]/Guitars](http://localhost:[port]/Guitars) в браузере, показано на рис. 1.4.

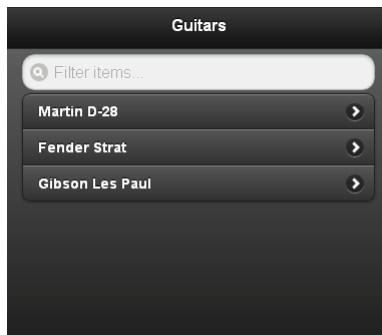


Рис. 1.4. Список гитар, полученный с помощью ASP.NET MVC 4 и jQuery Mobile

Извлечение данных

Новый синтаксис запросов, который был показан в предыдущем разделе, выглядит и действует подобно LINQ в C#/VB.NET. Ниже представлено несколько коротких примеров его использования:

Извлечение информации о гитаре по названию.

```
member x.GetByName name =
    use context = new FsMvcAppEntities()
    query { for g in context.Guitars do
            where (g.Name = name) }
    |> Seq.toList
```

Сортировка списка гитар по названию.

```
member x.GetAllAlphabetic () =
    use context = new FsMvcAppEntities()
    query { for g in context.Guitars do
            sortBy g.Name }
    |> Seq.toList
```

Извлечение первых X записей.

```
member x.GetTop rowCount =
    use context = new FsMvcAppEntities()
    query { for g in context.Guitars do
            take rowCount }
    |> Seq.toList
```

Дополнительные примеры можно найти на странице <http://fsharp3sample.codeplex.com/>.

Примечание. Во многих примерах выше можно использовать F# PowerPack Linq, хотя синтаксис этого инструмента не отличается той ясностью.

Извлечение данных с использованием поставщиков типов

В версии F# 3.0 появилась еще одна новая особенность – *поставщики типов* (type providers), еще больше упрощающая взаимодействия с базами данных. Чтобы воспользоваться поставщиком типов для доступа к базе данных, сначала необходимо добавить ссылку на сборку `FSharp.Data.TypeProviders`. После этого появится возможность использовать стандартные поставщики типов для работы с базами данных, такие как `SqlConnection`. Этот поставщик извлекает схему базы данных и генерирует соответствующие типы. Например:

```
open Microsoft.FSharp.Data.TypeProviders

type DbConnection =
    SqlConnection<ConnectionStringName="FsMvcAppExample",
        ConfigFile="web.config">

type GuitarsRepository2() =
    member x.GetAll () =
        use context = DbConnection.GetDataContext()
        query { for g in context.Guitars do
            select (Guitar(Id = g.Id, Name = g.Name)) }
    |> Seq.toList
```

Примечание. Чтобы научить механизм IntelliSense распознавать свойства и методы `context` внутри проекта F# WebApp, необходимо приложить дополнительные усилия. Для этого просто создайте файл `web.config` в проекте F# WebApp и добавьте соответствующие элементы в строку подключения кс базе данных.

Поставщик типов позволяет использовать тот же синтаксис запросов, что был показан в разделе «Entity Framework» (выше). Возможно, не все поймут, чем это решение отличается от предыдущего, на основе фреймворка Entity Framework. Самое важное достижение состоит в том, что тип `FsMvcAppEntities` стал ненужным и от него можно полностью отказаться. Кроме того, класс модели `Guitar` стал проще, благодаря устранению атрибута [`<Key>`].

Мы не будем погружаться в тонкости работы поставщиков типов, так как это выходит за рамки книги, но в общих чертах их работу можно описать так:

1. Добрая фея вдует в ваш компьютер волшебную пыль.
2. И теперь вы можете использовать вычислительные выражения запросов для взаимодействия с базами данных.

Примечание. Желаящие поближе познакомиться с внутренним устройством поставщиков типов, могут обратиться к документации на сайте MSDN (<http://msdn.microsoft.com/ru-ru/library/55361034.aspx>), описывающей создание поставщиков типов. Кроме того, я написал пример реализации собственного поставщика типов, который можно найти в моем блоге (<http://bit.ly/dmohltyperproviderexample>). Имейте в виду, что этот пример написан с использованием предварительной версии F# 3.0 для разработчиков. Синтаксис F# мог претерпеть некоторые изменения с тех пор.
