Исчерпывающее руководство по созданию простого программного кода для решения сложных задач

Программирование на F#

Символ®
O'REILLY®

Крис Смит

Programming F#

Chris Smith

O'REILLY®

Программирование на F#

Крис Смит

Символ®

Санкт-Петербург — Москва
2011

Крис Смит

Программирование на F#

Перевод А. Киселева

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Выпускающий редактор	*П. Щеголев*
Редактор	*Ю. Бочина*
Научный редактор	*С. Тепляков*
Корректор	*О. Макарова*
Верстка	*К. Чубаров*

Смит К.

Программирование на F#. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 448 с., ил.

ISBN 978-5-93286-199-8

F# – это мультипарадигмальный язык программирования, который не только помогает повысить производительность труда за счет использования функционального стиля разработки, но и позволяет применять при создании приложений уже имеющиеся навыки объектно-ориентированного и императивного программирования. Книга «Программирование на F#» поможет открыть множество преимуществ этого языка, включая возможность доступа ко всем замечательным инструментам и библиотекам платформы .NET.

Это исчерпывающее руководство, написанное Крисом Смитом, одним из основных разработчиков F# компании Microsoft, знакомит с синтаксисом языка, реализацией асинхронных и параллельных вычислений, с расширенными концепциями языка F#, такими как цитируемые и вычислительные выражения.

От читателя не требуется знание конкретных технологий, хотя общий опыт программирования, безусловно, желателен. Единственное требование – это желание воспользоваться преимуществами функционального программирования при разработке своих проектов, будь то реализация численных алгоритмов, анализ данных или сценарии для личного использования. В этом случае издание послужит хорошей отправной точкой на пути изучения фундаментальных и расширенных концепций языка F#.

ISBN 978-5-93286-199-8
ISBN 978-0-596-15364-9 (англ)

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 324-5353, www.symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Подписано в печать 12.01.2011. Формат 70×100 1/16. Печать офсетная.
Объем 28 печ. л. Тираж 1500 экз. Заказ №
Отпечатано с готовых диапозитивов в ГУП «Типография «Наука»
199034, Санкт-Петербург, 9 линия, 12.

Оглавление

Вступительное слово 13
Предисловие 15

Часть I. Мультипарадигмальное программирование 23

Глава 1. Введение в F# 25
Знакомство с F# 25
Visual Studio 2010 26
Ваша вторая программа на F# 27
Значения 28
Пробельные символы имеют значение 29
.NET Interop 30
Комментарии 31
F# Interactive 31
Управление файлами с исходными кодами F# 34
Глава 2. Основы 36
Элементарные типы 36
Элементарные числовые типы 37
Арифметика 39
Функции преобразования 40
Тип bigint 41
Битовые операции 42
Символы 42
Строки 43
Булевы значения 45
Сравнивание и равенство 46
Функции 46
Вывод типов 47
Обобщенные функции 49
Область видимости 50
Управление потоком выполнения 52
Основные типы 55
Тип unit 55
Кортежи 56

Списки ... 58
Агрегатные операторы ... 63
Тип option ... 67
Функция printfn ... 68
Строение программы на языке F# ... 71
Модули ... 71
Пространства имен ... 72
Запуск программы ... 73
Глава 3. Функциональное программирование ... 75
Программирование с помощью функций ... 76
Неизменяемость ... 77
Функции как значения ... 78
Рекурсивные функции ... 82
Символьные операторы ... 85
Композиция функций ... 86
Сопоставление с образцом ... 93
Отсутствие совпадений ... 94
Именованные образцы ... 95
Сопоставление с литералами ... 96
Ограничение when ... 97
Группировка образцов ... 98
Сопоставление структур данных ... 99
За пределами выражений сопоставления ... 100
Альтернативный синтаксис лямбда-выражений ... 101
Размеченные объединения ... 102
Использование размеченных объединений для создания древовидных структур ... 104
Сопоставление с образцом ... 105
Методы и свойства ... 107
Записи ... 108
Клонирование записей ... 109
Сопоставление с образцом ... 109
Вывод типов ... 110
Методы и свойства ... 111
Отложенные вычисления ... 111
Типы Lazy ... 112
Последовательности ... 112
Выражения последовательности ... 114
Функции модуля Seq ... 115
Агрегатные операторы ... 116

Глава 4. Императивное программирование 118
Понятие памяти в .NET 119
Значимые типы и ссылочные типы 120
Значения по умолчанию 120
Совмещение имен ссылочных типов 123
Изменение значений 123
Ссылочные ячейки 125
Изменяемые записи 126
Массивы 127
Обращение к элементам массива 128
Срезы массивов 130
Создание массивов 131
Сопоставление с образцом 132
Эквивалентность массивов 132
Функции модуля Array 133
Многомерные массивы 135
Типы изменяемых коллекций 136
List<'T> 136
Dictionary<'K,'V> 138
HashSet<'T> 140
Циклы 141
Циклы while 141
Циклы for 142
Исключения 144
Обработка исключений 145
Повторная генерация исключений 148
Определение новых типов исключений 148
Глава 5. Объектно-ориентированное программирование 151
Программирование с применением объектов 151
Преимущества ООП 152
Недостатки ООП 152
System.Object 153
Общие методы 153
Эквивалентность объектов 155
Сгенерированная эквивалентность 157
Классы 159
Явное создание 160
Неявное создание классов 162
Обобщенные классы 163
Собственный идентификатор 165

Методы и свойства ... 165
Свойства ... 166
Установка значений свойств в конструкторе ... 167
Методы ... 167
Статические методы, свойства и поля ... 169
Перегрузка методов ... 171
Модификаторы доступа ... 172
Наследование ... 175
Переопределение методов ... 177
Категории классов ... 178
Приведение типов ... 181
Глава 6. Программирование на платформе .NET ... 185
Платформа .NET ... 185
CLI ... 185
Сборка мусора ... 186
Интерфейсы ... 189
Использование интерфейсов ... 190
Определение интерфейсов ... 191
Объектные выражения ... 193
Реализация интерфейсов с помощью объектных выражений ... 193
Создание производных классов с помощью объектных выражений ... 195
Методы расширения ... 196
Расширение модулей ... 197
Перечисления ... 198
Создание перечислений ... 198
Преобразование ... 199
Когда использовать перечисления, а когда размеченные объединения ... 200
Структуры ... 201
Создание структур ... 202
Ограничения ... 204
Когда использовать структуры, а когда записи ... 204
Глава 7. Прикладное функциональное программирование ... 206
Единицы измерения ... 207
Определение единиц измерения ... 209
Преобразование единиц измерения ... 210
Обобщенные единицы измерения ... 211
Активные шаблоны ... 212
Одновариантные активные шаблоны ... 214
Частичные активные шаблоны ... 215

Параметризованные активные шаблоны ... 216
Многовариантные активные шаблоны ... 218
Использование активных шаблонов ... 219
Использование модулей ... 223
Преобразование модулей в классы ... 223
Намеренное сокрытие ... 227
Управление порядком использования модулей ... 228
Работа со списками ... 230
Операции над списками ... 230
Использование списков ... 232
Хвостовая рекурсия ... 233
Стек ... 235
Введение в хвостовую рекурсию ... 236
Шаблоны хвостовой рекурсии ... 239
Программирование с применением функций ... 243
Карринг ... 243
Избавление от избыточного кода ... 244
Замыкания ... 245
Функциональные шаблоны проектирования ... 247
Мемоизация ... 248
Функции как изменяемые значения ... 250
Отложенные вычисления ... 252

Глава 8. Прикладное объектно-ориентированное программирование ... 254
Операторы ... 254
Перегрузка операторов ... 254
Индексаторы ... 256
Добавление срезов ... 259
Ограничения обобщенных типов ... 261
Делегаты и события ... 264
Определение делегатов ... 266
Объединение делегатов ... 267
События ... 268
Создание событий ... 268
Класс Event<_,_> ... 271
Модуль Observable ... 272
Создание событий .NET ... 276

Часть II. Программирование на языке F# ... 279

Глава 9. Сценарии ... 281
Файлы сценариев на языке F# ... 282
Директивы ... 283

Общие директивы ... 283
Директивы сценариев ... 284
Рецепты по созданию сценариев ... 286
Выделение цветом ... 286
Воспроизведение звука ... 287
Обход дерева каталогов ... 288
Простой запуск процессов ... 289
Автоматизация операций в Microsoft Office ... 290
Глава 10. Вычислительные выражения ... 293
На пути к вычислительным выражениям ... 293
Построители вычислительных выражений ... 297
Собственные построители вычислительных выражений ... 301
Асинхронные вычислительные выражения ... 301
Вычислительное выражение округления ... 302
Вычислительное выражение, сохраняющее состояние ... 303
Глава 11. Асинхронное и параллельное программирование ... 310
Работа с потоками ... 311
Запуск потоков ... 312
Пул потоков .NET ... 314
Разделяемые данные ... 315
Асинхронное программирование ... 319
Асинхронные вычислительные выражения ... 322
Библиотека Async ... 323
Асинхронные операции ... 329
Создание собственных асинхронных примитивов ... 330
Ограничения ... 331
Параллельное программирование ... 332
Parallel.For ... 333
Модуль Array.Parallel ... 334
Библиотека PFX ... 334
Примитивы ... 336
Параллельные структуры данных ... 341
Глава 12. Рефлексия ... 345
Атрибуты ... 345
Применение атрибутов ... 347
Определение новых атрибутов ... 348
Рефлексия типов ... 349
Получение информации о типе ... 350
Рефлексия типов языка F# ... 354
Динамическое создание экземпляров ... 356
Создание экземпляров ... 357
Создание экземпляров типов языка F# ... 357

Динамический вызов ... 358
Оператор «знак вопроса» ... 359
Использование рефлексии ... 361
Декларативное программирование ... 361
Расширяемая архитектура ... 365
Глава 13. Цитирование ... 370
Основы цитирования ... 371
Декомпозиция цитируемых выражений ... 372
Цитирование тела метода ... 375
Декомпозиция произвольного кода ... 377
Практическое применение: перенос вычислений на другие платформы ... 379
Создание цитируемых выражений ... 381
Подстановка выражений ... 382
Выполнение цитируемых выражений ... 382
Практическое применение: создание производных ... 384
Приложение А. Обзор библиотек .NET ... 388
Визуализация ... 388
Windows Forms ... 389
Windows Presentation Foundation ... 391
Обработка данных ... 398
Регулярные выражения ... 399
Работа с XML ... 406
Сохранение данных ... 410
Файлы ... 410
Сериализация данных ... 411
Десериализация ... 414
Стандартная библиотека языка F# ... 415
FSharp.Core.dll ... 415
F# PowerPack ... 418
Приложение В. Взаимодействие программ на F# с кодом на других языках .NET ... 421
Взаимодействие с другими языками .NET ... 422
Использование кода F# в программах на языке C# ... 422
Использование кода C# в программах на языке F# ... 427
Взаимодействие с неуправляемым кодом ... 431
Platform Invoke ... 431
COM ... 433
Алфавитный указатель ... 437

Вступительное слово

Эта книга отмечает переломный момент в истории развития языка F# – переход от состояния исследовательского проекта подразделения Microsoft Research в Кембридже, уходящего корнями в такие языки программирования, как OCaml и Haskell, до стабильного, эффективного и удивительно производительного инструмента создания емких и лаконичных программ для платформы .NET. С выходом Visual Studio 2010 новое поколение программистов получило в свое распоряжение язык, который позволит им приближать будущее, разрабатывая программное обеспечение, будь то просто красивый программный код, революционные фреймворки, высокопроизводительные веб-сайты, новые методологии программирования, улучшенные алгоритмы математических вычислений, надежные механизмы тестирования, интеллектуальная параллелизация, улучшенное моделирование или любые другие проявления «грамотного программирования» на F#. Как создателю F# мне очень приятно, что именно Крис Смит (Chris Smith), один из основных разработчиков F#, представляет этот язык программирования широкой аудитории.

Язык F# представляет собой сочетание простоты и элегантности типизированного функционального программирования с широкими возможностями платформы .NET. Хотя для многих программистов типизированное функциональное программирование является относительно новой парадигмой и требует дополнительного изучения, оно существенно упрощает разработку программ. Программы на языке F# должны строиться из основных композиционных элементов, а возможность вывода типов делает их короче и понятнее. В этой книге Крис сначала представляет основные парадигмы языка F#: функциональное программирование, императивное программирование и объектно-ориентированное программирование, а затем демонстрирует, как они могут использоваться при создании программ. Особое внимание он уделяет простоте и ясности описания фундаментальных элементов языка, попутно подчеркивая, насколько приятным может быть процесс программирования вообще и программирования на языке F# в частности.

При грамотном подходе язык F# существенно упрощает разработку программ. Например, единицы измерения в языке F#, описываемые в этой книге и являющиеся важной вехой в истории развития языков программирования, ликвидируют сложности, связанные с использованием вещественных чисел в прикладных областях. Кроме того, разви-

тие языка F# в значительной степени определила ориентация на принципы асинхронной и параллельной обработки данных. В этой книге Крис представляет основы языка F# и платформы .NET, позволяющие вам освоиться в этом мире. Кроме того, язык F# обладает мощной поддержкой объектно-ориентированного программирования, и эта тема, так близкая сердцу Криса, также освещается в книге.

Прежде всего, F# – это практический язык программирования, и Крис ручается, что читатель получит здесь всю информацию, необходимую для успешного применения текущего поколения инструментов F# для создания программ. Крис описывает версию 1.0 языка F#, входящую в состав Visual Studio 2010, – первую официальную версию. Коллектив разработчиков F# и я чрезвычайно признательны Крису за его вклад в развитие языка и его инструментов. Я надеюсь, что вы получите такое же удовольствие, программируя на языке F#, как и мы с Крисом, разрабатывая F# в составе команды.

Дон Сайм (Don Syme),
ведущий разработчик и создатель языка F#,
Microsoft Research

Предисловие

В Visual Studio появилась поддержка нового языка программирования – F#. Он помогает писать более выразительные, простые в поддержке программы и дает возможность использовать весь богатейший потенциал платформы .NET. Знание языка F# не только позволит вам повысить свою производительность, но и обеспечит вам определенные преимущества перед другими программистами. Овладев приемами функционального программирования, представленными в F#, вы сможете применять их в программах, написанных на других языках, а также по-новому взглянуть на программирование в целом.

Введение в F#

Что же представляет собой язык F#? В двух словах, F# – это *мультипарадигмальный* язык программирования, основанный на .NET, то есть язык программирования, исходно поддерживающий различные *стили* программирования. Я избавлю вас от знакомства с полной историей развития этого языка и коротко отмечу лишь наиболее важные моменты:

- F# поддерживает функциональное программирование, то есть такой стиль программирования, когда описывается, *что* должна делать программа, а не *как* она должна работать.
- F# поддерживает объектно-ориентированное программирование. Язык F# позволяет заключать программный код в классы и объекты, что дает возможность упростить его.
- F# поддерживает императивное программирование. На языке F# можно реализовать изменение содержимого областей памяти, читать и записывать файлы, обмениваться данными по сети и так далее.
- F# является статически типизированным языком. Вследствие этого информация о типах доступна уже на этапе компиляции, что обеспечивает большую надежность программного кода – F# не позволит заткнуть круглое отверстие квадратной пробкой.
- F# – это язык для платформы .NET. Он работает на основе общей языковой инфраструктуры (Common Language Infrastructure, CLI), и поэтому при использовании этого языка доступны механизм автоматической сборки мусора (управления памятью) и мощные библиотеки классов. Кроме того, F# поддерживает все концепции, харак-

терные для .NET, такие как делегаты, перечисления, структуры, P/Invoke[1] и так далее.

Даже без понимания этих специальных терминов становится ясно, что F# – это мощный язык с широкими возможностями. Но не волнуйтесь, мы со всеми ними познакомимся шаг за шагом.

Кому адресована эта книга

Эта книга не является учебником по программированию и предполагает наличие у читателя знаний основных понятий, таких как циклы, функции и рекурсия. Однако она не требует наличия опыта функционального программирования или знакомства с платформой .NET.

Если вы имеете опыт использования C# или VB.NET, вы будете чувствовать себя совершенно комфортно. Несмотря на то, что в языке F# применяются иные подходы к программированию, тем не менее все имеющиеся у вас знания .NET вы сможете применять в программах на F#.

Если вы знакомы с такими языками программирования, как OCaml или Haskell, синтаксис F# покажется вам очень знакомым. Язык F# обладает большинством особенностей, присущих этим языкам, и добавляет множество новых возможностей, обеспечивающих интеграцию с платформой .NET.

Что необходимо для работы с этой книгой

Поддержка языка F# является составной частью Visual Studio 2010. В эту поддержку входят компилятор F# и система управления проектами, а также такие особенности, как подсветка синтаксиса и механизм автодополнения IntelliSense. При этом для создания программ на F# не обязательно иметь полную версию Visual Studio 2010. Поддержку F# можно обеспечить поверх Visual Studio 2008 Shell – бесплатной и ограниченной версии Visual Studio. Загрузить Visual Studio 2008 Shell (Integrated Mode) можно на странице:

http://www.microsoft.com/downloads/details.spx?FamilyID=40646580-97FA-4698-B65F-620D4B4B1ED7

После установки Visual Studio 2008 Shell вы можете установить последнюю версию F# Community Technology Preview, загрузив ее с сайта Microsoft F# Developer Center (*http://fsharp.net*).

Кроме того, имеется возможность создавать и распространять программы на языке F#, используя открытую (*http://open-source.org*) платформу Mono (*http://www.mono-project.com/*).

---

[1] Механизм вызова неуправляемого программного кода. – *Прим. перев.*

Структура книги

Книга делится на две части. В первой части основное внимание уделяется раскрытию мультипарадигмальной природы языка F#. Первые главы посвящены отдельным парадигмам программирования на F#, а последующие помогут вам понять основные особенности языка. К концу первой части вы освоите язык F# и стиль программирования на нем.

Во второй части представлено несколько дополнительных концепций, но основное внимание уделяется применению F# в специализированных областях. К концу второй части вы будете знать, как использовать F# для программирования параллельных вычислений и создания предметно-ориентированных языков программирования.

Часть I «Мультипарадигмальное программирование»

Глава 1 «Введение в F#» представляет язык F# и интегрированную среду разработки Visual Studio 2010. Я рекомендую прочитать эту главу даже тем, кто уже знаком с Visual Studio, потому что использование языка F# привносит свои особенности в создание и управление проектами.

Глава 2 «Основы» знакомит с основными типами и понятиями, которые являются базой для всех остальных глав.

Глава 3 «Функциональное программирование» знакомит с функциональным программированием и особенностями создания программного кода F# с использованием этого стиля.

Глава 4 «Императивное программирование» описывает, как изменять значения и состояние программы при использовании императивного стиля.

Глава 5 «Объектно-ориентированное программирование» описывает особенности объектно-ориентированного программирования, начиная с определения простых типов и заканчивая обсуждением таких понятий, как наследование и полиморфизм.

Глава 6 «Программирование для .NET» рассматривает некоторые концепции, не зависящие от используемого стиля программирования и обусловленные особенностями платформы .NET Framework и общей языковой инфраструктуры (Common Language Infrastructure, CLI).

Глава 7 «Прикладное функциональное программирование» охватывает дополнительные темы, относящиеся к функциональному программированию, такие как концевая рекурсия[1] (tail recursion) и функциональные шаблоны проектирования.

[1] Иногда ее называют «хвостовая рекурсия». – *Прим. перев.*

Глава 8 «Прикладное объектно-ориентированное программирование» описывает приемы создания и использования богатой системы типов. Особое внимание в этой главе уделяется использованию функциональных аспектов языка F# применительно к объектно-ориентированному программному коду.

Часть II «Программирование на F#»

Глава 9 «Сценарии» рассматривает F# как язык сценариев и описывает особенности создания сценариев на языке F#.

Глава 10 «Вычисляемые выражения» представляет дополнительную особенность языка F#, которая позволяет устранить избыточность программного кода и расширить ядро языка F# новыми возможностями.

Глава 11 «Асинхронное и параллельное программирование» рассказывает о том, как в программах на языке F# можно использовать преимущества многоядерных процессоров, а также о средствах параллельного программирования, имеющихся в языке F# и в библиотеках .NET.

Глава 12 «Рефлексия» рассматривает библиотеку, имеющуюся в .NET и позволяющую выполнять рефлексию, и порядок ее использования для создания декларативных программ.

Глава 13 «Цитирование» представляет цитируемые выражения в языке F# и рассказывает, как они могут использоваться в метапрограммировании и для выполнения программного кода F# на других вычислительных платформах.

Приложения

В этой книге также имеется пара приложений, в которых подробно рассматриваются некоторые дополнительные понятия, которые могут вам пригодиться.

В приложении A кратко рассматриваются существующие технологии, доступные на платформе .NET, и способы их использования в программах на языке F#.

В приложении B описываются способы организации взаимодействия программ на языке F# с существующими библиотеками и неуправляемым программным кодом с применением механизмов P/Invoke и COM-interop.

Типографские соглашения

В книге приняты следующие соглашения:

Курсив

Применяется для выделения терминов, когда они упоминаются впервые.

`Моноширинный шрифт`

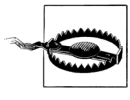
Применяется для представления программного кода и ключевых слов языка F#.

`Моноширинный жирный`

Используется для выделения фрагментов программного кода.

Особое внимание следует уделять фрагментам текста, выделенным, как показано ниже.

Примечания, выделенные таким способом, содержат дополнительную информацию для вдумчивых читателей.



Предупреждения, выделенные таким способом, призваны помочь избежать распространенных ошибок.

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)

Мы тщательно проверили всю информацию, которая приводится в этой книге, тем не менее вы можете столкнуться с некоторыми изменениями в языке F#, появившимися уже после выхода книги (или даже с ошибками в тексте!). Книга имеет собственную веб-страницу, где приводится список ошибок и рассказывается о наших планах по выходу следующих изданий. Адрес веб-страницы:

http://oreilly.com/catalog/9780596153649

Вы можете также отправлять свои сообщения по электронной почте. Чтобы отправить вопрос в список рассылки или запросить каталог, отправляйте сообщения по адресу:

info@oreilly.com

Свои отзывы о книге отправляйте по адресу:

bookquestions@oreilly.com

Дополнительную информацию об этой и других книгах, а также технические статьи о языке F# и его обсуждение вы найдете на веб-сайте издательства O'Reilly:

http://www.oreilly.com

или на сайте O'Reilly .NET DevCenter:

http://www.oreillynet.com/dotnet

или на портале Microsoft Developer Network:

http://www.msdn.com/fsharp

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 значительные по объему 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: «Programming F#, by Chris Smith. Copyright 2010 Chris Smith, 978-0-596-15364-9».

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 – это виртуальная библиотека, которая позволяет легко и быстро находить ответы на вопросы среди более чем 7500 технических и справочных изданий и видеороликов.

Подписавшись на услугу, вы сможете загружать любые страницы из книг и просматривать любые видеоролики из нашей библиотеки. Читать книги на своих мобильных устройствах и сотовых телефонах. Получать доступ к новинкам еще до того, как они выйдут из печати.

Читать рукописи, находящиеся в работе, и посылать свои отзывы авторам. Копировать и вставлять отрывки программного кода, определять свои предпочтения, загружать отдельные главы, устанавливать закладки на ключевые разделы, оставлять примечания, печатать страницы и пользоваться массой других преимуществ, позволяющих экономить ваше время.

Благодаря усилиям O'Reilly Media данная книга также доступна через услугу Safari Books Online. Чтобы получить полный доступ к электронной версии этой книги, а также книг с похожими темами издательства O'Reilly и других издательств, подпишитесь бесплатно по адресу *http://my.safaribooksonline.com*.

Благодарности

Я получил все лавры за создание этой книги, однако основной объем работ по созданию языка F# был выполнен коллективом проекта F# в корпорации Microsoft в Редмонде и в подразделении Microsoft Research в Кембридже (Англия). Я получил огромное удовольствие, работая с удивительно талантливыми людьми, без которых язык F# так и остался бы простой пометкой в списке дел, приклеенном к монитору Дона Сайма (Don Syme).

Ниже я перечислил сотрудников Microsoft, участвовавших в создании этого языка:

Анар Алимов (Anar Alimov)
Эндрю Кеннеди (Andrew Kennedy)
Баоф Фенг (Baofa Feng)
Брайан Макнамара (Brian McNamara)
Дэниел Квирк (Daniel Quirk)
Дмитрий Ломов (Dmitry Lomov)
Доминик Куни (Dominic Cooney)
Дон Сайм (Don Syme)
Гордон Хогенсон (Gordon Hogenson)
Грег Неверов (Greg Neverov)
Джеймс Маргетсон (James Margetson)
Джо Пэймер (Joe Pamer)
Йомо Фишер (Jomo Fisher)
Кайл Росс (Kyle Ross)
Лоран ле Блюн (Laurant Le Blun)
Люк Хобан (Luke Hoban)
Матео Тавеггиа (Matteo Taveggia)
Ральф Хербрич (Ralf Herbrich)
Сантош Захария (Santosh Zachariah)
Шон Ванг (Sean Wang)
Тимоти Ын (Timothy Ng)
Томаш Петричек (Tomas Petricek)

Кроме того, эта книга не появилась бы на свет без помощи замечательных редакторов издательства O'Reilly, членов сообщества F# и многих других: Джона Осборна (John Osborn), Лаурель Рума (Laurel Ruma), Брайана Макдональда (Brian MacDonald), Мэтта Эллиса (Matt Ellis), Андре ван Мейлбрука (André van Meulebrouck), Стефена Тоуба (Stephen Toub), Джареда Парсонса (Jared Parsons), «почетного члена» Дастина Кэмпбелла (Dustin «honorary member» Campbell), Майкла де ла Маза (Michael de la Maza), Алекса Пика (Alex Peake), Райана Каванафа (Ryan

Cavanaugh), Брайана Пика (Brian Peek), Мэтью Подвысоцки (Matthew Podwysocki), Ричарда Минерича (Richard Minerich), Кейт Мур (Kate Moore) и наконец, Стюарта Боуэрса (Stuart Bowers) – вы настоящие мастера 9-го дана в F#.

Об авторе

Крис Смит работает в Microsoft в группе разработки языка F#. Его должность инженера-программиста в отделе тестирования позволила ему в совершенстве овладеть языком F#. Крис имеет степень магистра информатики, полученную в Вашингтонском университете, и испытывает настоящую страсть к любым вкусным напиткам, в бокалы с которыми добавляется зонтик. Вы можете встретиться с ним в Интернете, в его персональном блоге «Chris Smith's Completely Unique View» (совершенно уникальный взгляд Криса Смита) по адресу: *http://blogs.msdn.com/chrsmith/*.

I

Мультипарадигмальное программирование

1

Введение в F#

F# – это мощный язык программирования, поддерживающий несколько парадигм программирования. Эта глава представляет собой краткое введение в основы F# – компилятор, инструменты и место языка в Visual Studio 2010.

В этой главе вы напишете пару простых программ на языке F#, а затем я познакомлю вас с ключевыми особенностями Visual Studio, имеющими отношение к разработке приложений на этом языке. Я не буду здесь подробно описывать Visual Studio, поэтому для расширения своих представлений об этой интегрированной среде разработки рекомендую вам заняться самостоятельными исследованиями или обратиться к электронной документации по адресу *http://msdn.microsoft.com/en-us/vstudio/default.aspx*[1].

Даже если вы уже знакомы с Visual Studio, вам все равно стоит прочитать эту главу. Создание и отладка проектов на языке F# выполняется точно так же, как и проектов на языке C# или VB.NET, однако проекты на F#, состоящие из нескольких файлов, имеют некоторые уникальные особенности. Кроме того, при работе с этим языком поддерживается такая возможность, как *F# Interactive*, которая позволит вам резко поднять свою производительность. Не пропустите!

Знакомство с F#

Во многих книгах по программированию принято писать программы «Hello, World», и я не буду отказываться от этой традиции. Откройте текстовый редактор Блокнот (Notepad) или любой другой текстовый ре-

[1] На русском языке: *http://msdn.microsoft.com/ru-ru/vstudio/default.aspx* – *Прим. перев.*

дактор по своему выбору и создайте новый файл с именем *HelloWorld.fs* со следующим содержимым:

```
// HelloWorld.fs
printfn "Hello, World"
```

Отлично! Вы только что написали свою первую программу на языке F#. Чтобы скомпилировать эту программу, необходимо воспользоваться компилятором F#, `fsc.exe`, находящимся в папке *Program Files\Microsoft F#\v4.0*. (Или, если вы пользуетесь Mono, в каталоге, выбранном при установке F#.) В следующем фрагменте показано использование компилятора F# из командной строки для сборки и запуска приложения:

```
C:\Program Files\Microsoft F#\v4.0>fsc HelloWorld.fs
Microsoft F# Compiler, (c) Microsoft Corporation, All Rights Reserved
F# Version 1.9.8.0, compiling for .NET Framework Version v4.0.21017

C:\Program Files\Microsoft F#\v4.0>HelloWorld.exe
Hello, World!
```

Visual Studio 2010

Инструментальные средства наполняют жизненной силой любой язык программирования, и F# не является исключением. Хотя у вас все получится, даже если вы будете писать программы на F# в своем любимом текстовом редакторе и вызывать компилятор исключительно из командной строки, тем не менее использование инструментальных средств позволит вам добиться более высокой производительности. Подобно языкам C# и VB.NET, F# имеет полноценную поддержку в Visual Studio. Для языка F# в среде Visual Studio поддерживаются все известные возможности, такие как отладчик, IntelliSense (механизм автодополнения), шаблоны проектов и другие.

Чтобы создать новый проект на языке F#, откройте среду разработки Visual Studio и выберите пункт меню File → New Project (Файл → Создать → Проект)[1], чтобы открыть диалог New Project (Создать проект), как показано на рис. 1.1. В левой панели выберите пункт Visual F#, а в правой панели выберите пункт F# Application (Приложение F#) и щелкните на кнопке OK.

После щелчка на кнопке OK в диалоге New Project (Создать проект) вы увидите пустое окно редактора – чистый холст, готовый к созданию вашего шедевра на F#.

Вернемся еще раз к нашей программе «Hello, World». Введите следующий фрагмент в окне редактора программного кода F#:

```
printfn "Hello, World"
```

[1] Здесь и далее перевод интерфейсных элементов будет даваться в соответствии с русифицированной версией Visual Studio 2010. – *Прим. перев.*

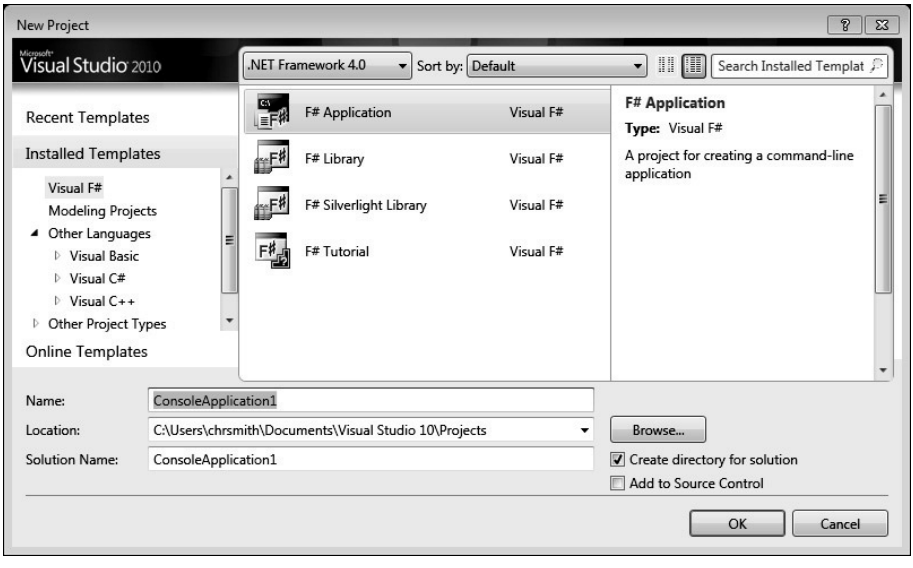


Рис. 1.1. *Выберите пункт F# Application, чтобы создать новый проект F#*

Теперь нажмите комбинацию клавиш Control + F5, чтобы запустить приложение. Когда приложение запустится, на экране появится окно консоли, в котором будет выведен вполне ожидаемый текст, как показано на рис. 1.2.

Рис. 1.2. *Программа «Hello, World» на языке F#*

Ваша вторая программа на F#

Кому-то может показаться странной программа, в которой отсутствует явный метод Main. Почему такое возможно, вы узнаете в следующей главе, а пока давайте создадим чуть более осмысленную, хотя такую же простую программу, чтобы почувствовать особенности синтаксиса языка F#.

В примере 1.1 приводится исходный текст программы, которая принимает два параметра командной строки и выводит их в консоли. Кроме того, она выводит текущее время.

Пример 1.1. Mega Hello World

```
(*
Mega Hello World:
Принимает два параметра командной строки и выводит их
вместе со значением текущего времени в окно консоли.
*)

open System

[<EntryPoint>]
let main (args : string[]) =

    if args.Length <> 2 then
        failwith "Error: Expected arguments <greeting> and <thing>"

    let greeting, thing = args.[0], args.[1]
    let timeOfDay = DateTime.Now.ToString("hh:mm tt")

    printfn "%s, %s at %s" greeting thing timeOfDay

    // Код завершения программы
    0
```

Теперь, когда у вас имеется реальный код на языке F#, вам, вероятно, будет интересно узнать, что в нем происходит. Рассмотрим эту программу строчка за строчкой, чтобы понять, как она работает.

Значения

В примере 1.1 содержатся три значения с именами `greeting`, `thing` и `timeOfDay`:

```
let greeting, thing = args.[0], args.[1]
let timeOfDay = DateTime.Now.ToString("hh:mm tt")
```

Обратите внимание на ключевое слово `let`, которое связывает *имя* со *значением*. Важно отметить, что, в отличие от большинства других языков программирования, значения в F# являются *неизменяемыми* по умолчанию, то есть их нельзя изменить после того, как они будут инициализированы. Почему значения являются неизменяемыми, мы узнаем в главе 3, а пока достаточно будет отметить, что это одна из особенностей функционального программирования.

Кроме того, язык F# отличает символы верхнего и нижнего регистров, поэтому два значения, имена которых отличаются только регистром символов, считаются разными:

```
let number = 1
let Number = 2
let NUMBER = 3
```

Имя значения может содержать буквы, цифры, символы подчеркивания _ и апострофы ‘ в любых комбинациях. Единственное ограничение: имя должно начинаться с буквы или с символа подчеркивания.

Допускается заключать имена значений в пары обратных апострофов. В этом случае имена могут содержать любые символы, за исключением символов табуляции и перевода строки. Благодаря этому вы можете ссылаться на значения и функции в библиотеках, написанных на других языках .NET, имена которых могут конфликтовать с ключевыми словами языка F#:

```
let ``this.Isn't %A% good value Name$!@#`` = 5
```

Пробельные символы имеют значение

В других языках программирования, таких как C#, для обозначения окончания инструкций и выделения блоков программного кода используются точка с запятой и фигурные скобки. При этом программисты для повышения удобочитаемости обычно оформляют программный код, используя отступы, поэтому эти дополнительные символы часто лишь загромождают программный код.

В языке F# пробельные символы – пробелы и символы перевода строки – имеют важное значение. Компилятор F# позволяет использовать пробельные символы для разграничения блоков кода. Например, любой код, имеющий отступ больше, чем ключевое слово `if`, считается телом инструкции `if`. Поскольку символы табуляции могут соответствовать различному числу пробелов, они запрещены к использованию в коде F#.

Вы можете настроить редактор Visual Studio так, что он будет автоматически преобразовывать символы табуляции в пробелы, для чего следует изменить соответствующий параметр в диалоге Tools → Options → Text Editor → F# (Сервис → Параметры → Текстовый редактор → F#).

Возвращаясь к примеру 1.1, обратите внимание, что тело метода `main` имеет отступ в четыре пробела и отступ в теле инструкции `if` увеличен еще на четыре пробела:

```
let main (args : string[]) =

    if args.Length <> 2 then
        failwith “Error: Expected arguments <greeting> and <thing>”

    let greeting, thing = args.[0], args.[1]
```

```
    let timeOfDay = DateTime.Now.ToString("hh:mm tt")

    printfn "%s, %s at %s" greeting thing timeOfDay

    // Код завершения программы
    0
```

Если бы тело `failwith` инструкции `if` не имело дополнительного отступа в четыре пробела и располагалось бы на том же расстоянии от начала строки, что и ключевое слово `if`, компилятор F# выдал бы предупреждение. Проблема состоит в том, что в этом случае компилятор не смог бы распознать `failwith` как тело инструкции `if`:

```
[<EntryPoint>]
let main (args : string[]) =

    if args.Length <> 2 then
    failwith "Error: Expected arguments <greeting> and <thing>"
```

```
Warning FS0058: possible incorrect indentation: this token is offside of
context started at position (25:5). Try indenting this token further or
using standard formatting conventions
```

(Предупреждение: возможно, неправильные отступы: этот маркер находится вне контекста, начиная с позиции (25:5). Попробуйте увеличить отступ маркера или использовать стандартные соглашения о форматировании.)

Основное правило заключается в следующем: все, что относится к методу или к инструкции, должно иметь дополнительный отступ относительно ключевого слова, с которого начинается метод или инструкция. Так, в примере 1.1 весь код в теле метода `main` имеет отступы относительно первого ключевого слова `let`, а код в инструкции `if` имеет отступы относительно ключевого слова `if`. По мере обретения навыков работы с программным кодом F# вы быстро убедитесь, что отсутствие точек с запятой и фигурных скобок существенно упрощает написание кода и его чтение.

.NET Interop

В примере 1.1 также демонстрируется, как программы на языке F# взаимодействуют с существующими библиотеками .NET:

```
open System

// ...

let timeOfDay = DateTime.Now.ToString("hh:mm tt")
```

Платформа .NET Framework содержит множество библиотек, упрощающие создание различных приложений, начиная от приложений с графическим интерфейсом и работы с базами данных и заканчивая веб-при-

ложениями. Программы на языке F# могут без всяких ограничений использовать любые библиотеки .NET и обращаться к ним непосредственно. Свойство `DateTime.Now`, используемое в примере 1.1, находится в пространстве имен `System`, в сборке `mscorlib.dll`. Код на языке F#, в свою очередь, также может использоваться в программах, написанных на других языках, поддерживаемых платформой .NET.

За дополнительной информацией о библиотеках .NET вы можете обратиться к приложению А, где приводится краткий обзор доступных библиотек.

Комментарии

Подобно любым языкам программирования, F# позволяет добавлять комментарии в программный код. Однострочные комментарии начинаются с двух символов слеша // – все, что следует за ними до конца строки, будет игнорироваться компилятором:

```
// Код завершения программы
```

Для добавления более объемных описаний, занимающих несколько строк, можно использовать многострочные комментарии, которые заключаются в пары символов (* и *):

```
(*
Mega Hello World:
Принимает два параметра командной строки и выводит их
вместе со значением текущего времени в окно консоли.
*)
```

При создании приложений на языке F# в среде Visual Studio можно использовать еще одну разновидность комментариев: *комментарии XML-документирования*. Если строка комментария начинается с трех символов слеша /// и находится непосредственно перед идентификатором, Visual Studio будет отображать текст комментария при наведении указателя мыши на этот идентификатор.

На рис. 1.3 приводится пример XML-документирования и соответствующая ему всплывающая подсказка.

F# Interactive

К настоящему моменту мы уже написали некоторый код на языке F# и выполнили его. Далее в книге вы встретите еще множество примеров. При опробовании примеров из книги вы *могли бы* каждый раз создавать новые проекты, однако в среде Visual Studio имеется очень удобный инструмент, который называется F# Interactive, или FSI. Используя окно FSI, вы поймете, что этот инструмент не только упрощает работу с примерами из этой книги, но и помогает разрабатывать приложения.

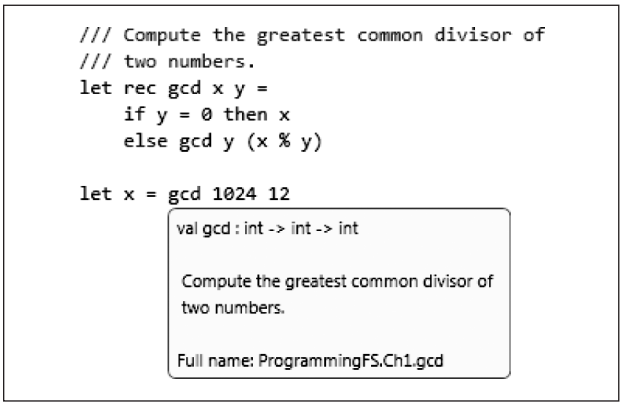


Рис. 1.3. *Комментарии документирования XML*

F# Interactive (Интерактивный сеанс F#) относится к классу инструментов, известных под названием *REPL* (Read-Evaluate-Print Loop – цикл чтения-вычисления-вывода). Этот инструмент принимает код F#, компилирует его, выполняет и выводит результаты. Это позволяет легко и быстро экспериментировать с программным кодом F#, не создавая новые проекты или полноценные приложения только ради того, чтобы увидеть результаты работы фрагмента из пяти строк.

При использовании языков C# и VB.NET вы должны скомпилировать и затем запустить свое приложение, чтобы увидеть результаты, что существенно усложняет процесс работы с небольшими фрагментами программного кода. Конечно, во время отладки в среде Visual Studio можно использовать Immediate Window (Окно интерпретации), но оно может использоваться только для вычисления выражений и не позволяет писать программный код, такой как определение новых функций или классов.

В Visual Studio с типичными настройками окно F# Interactive можно открыть, нажав комбинацию клавиш Control+Alt+F. Как только окно FSI будет открыто, в него можно вводить код F#, пока вы не введете последовательность `;;` и символ перевода строки. Введенный код будет скомпилирован и выполнен, как показано на рис. 1.4.

После выполнения каждого фрагмента в окне FSI все имена, созданные в нем, будут выводиться в виде `val <имя>`. Если для выражения не было указано имя, ему автоматически будет присвоено имя `it`. Вслед за именем идентификатора будет выводиться символ двоеточия, *тип* результата и фактическое значение. Например, на рис. 1.4 видно, что было создано значение типа `int` с именем `x` и со значением, равным `42`.

Если вы не используете Visual Studio, можно воспользоваться консольной версией F# Interactive – утилитой `fsi.exe`, которая находится в том же каталоге, что и `fsc.exe`.

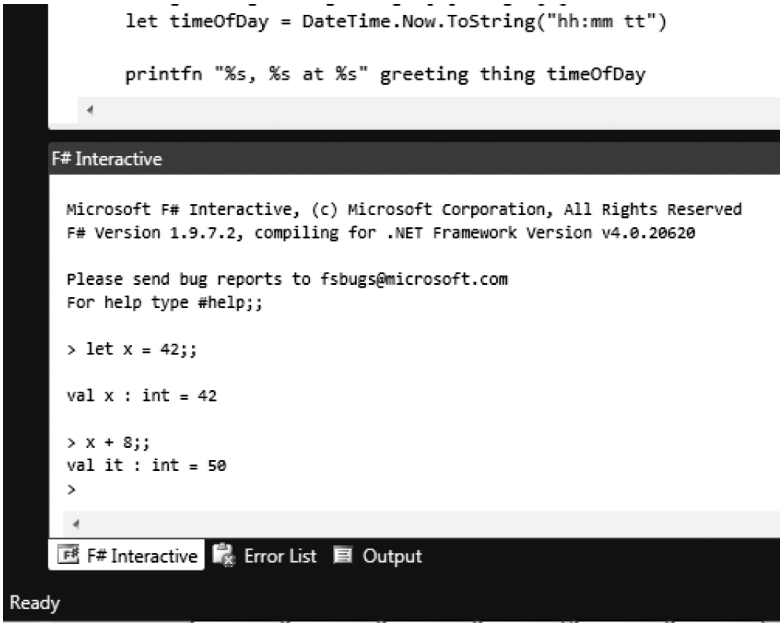


Рис. 1.4. *Окно F# Interactive*

Попробуйте выполнить следующие фрагменты в окне FSI. Обратите внимание, что каждый фрагмент завершается символами ;;:

```
> 2 + 2;;
val it : int = 4
> // Создаем два значения
let x = 1
let y = 2.3;;

val x : int = 1
val y : float = 2.3

> float x + y;;
val it : float = 3.3
> let cube x = x * x * x;;

val cube : int -> int

> cube 4;;
val it : int = 64
```

FSI существенно упрощает тестирование и отладку приложений благодаря тому, что в Visual Studio вы можете переслать F# код из своего текущего проекта в окно FSI, выделив его и нажав комбинацию клавиш Alt+Enter.

Если в окне редактора Visual Studio выделить весь код примера 1.1 и нажать Alt+Enter, в окне FSI вы увидите следующее:

```
>

val main : string []-> int
```

Это позволяет писать код в редакторе Visual Studio, пользуясь такими возможностями, как подсветка синтаксиса и IntelliSense, а тестировать его – с помощью окна FSI. Вы могли бы проверить работу метода `main`, скопировав его в окно `FSI` и просто вызвав:

```
> main [| "Hello"; "World" |];;
Hello, World at 10:52 AM
val it : int = 0
```

Большинство примеров в этой книге было скопировано непосредственно из окна FSI. Я также рекомендую вам использовать интерактивный сеанс при изучении синтаксиса языка F#.

Управление файлами с исходными кодами F#

Когда вы только начинаете изучать программирование на языке F#, большинство ваших программ будут существовать только в окне FSI или, может быть, в виде одного файла. Однако со временем проекты F# будут быстро расти в размерах, и программный код будет распределен по нескольким файлам и, в конце концов, не по одному проекту.

Управление проектами, содержащими несколько файлов на языке F#, имеет свои особенности. В языке F# имеет большое значение порядок, в котором выполняется компиляция файлов.

Вы можете использовать только те функции и классы, которые были определены выше в этом же файле или в другом файле, скомпилированном перед тем файлом, в котором используется эта функция или класс. Если изменить порядок компиляции исходных файлов, программа может перестать собираться!

Причина, по которой порядок компиляции имеет большое значение, связана с *выводом типов* (type inference) – темой, которая будет рассматриваться в следующей главе.

Исходные файлы F# компилируются в том же порядке, в каком они отображаются в окне Solution Explorer (Обозреватель решений), сверху вниз. Всякий раз, когда создается новый файл, он добавляется в конец списка, однако если необходимо изменить порядок компиляции файлов, можно щелкнуть правой кнопкой мыши на требуемом файле и в контекстном меню выбрать пункт Move Up (Вверх) или Move Down (Вниз), как показано на

рис. 1.5. Кроме того, для переупорядочения файлов можно использовать горячие комбинации клавиш Alt+Стрелка вверх и Alt+Стрелка вниз.

Теперь, когда вы получили представление о том, как компилировать приложения на языке F#, далее в книге мы сосредоточимся исключительно на изучении синтаксиса и семантики этого языка программирования.

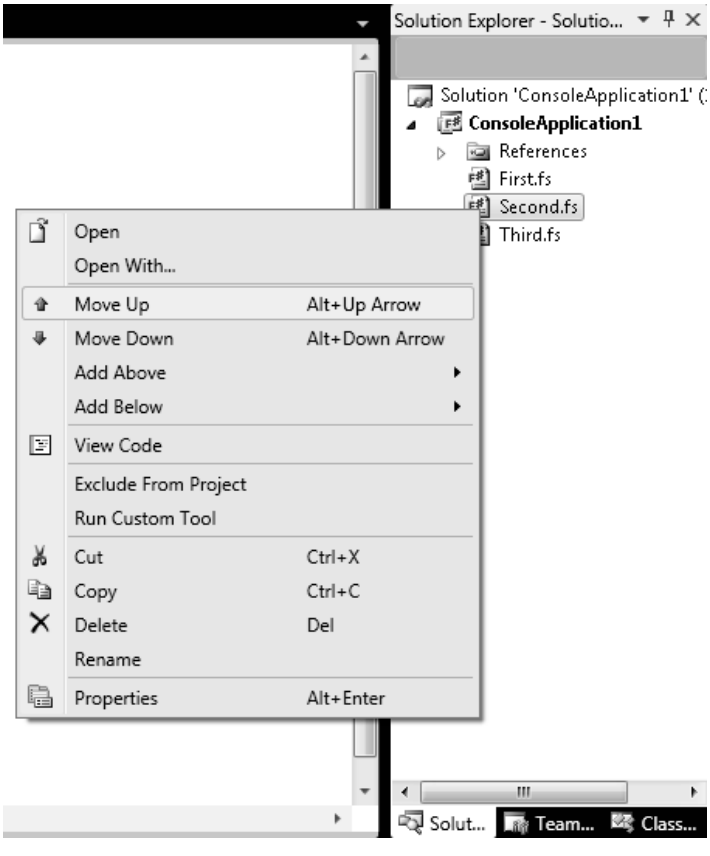


Рис. 1.5. *Переупорядочение файлов в проекте F#*

2

Основы

В главе 1 вы написали свою первую программу на языке F#. Я подробно рассмотрел ее, чтобы вы могли получить представление о том, что она делает, но большая часть кода в ней пока еще остается для вас загадкой. В этой главе я познакомлю вас с фундаментальными понятиями, необходимыми для полного понимания уже написанного кода, но, что еще более важно, я представлю еще несколько примеров, с помощью которых вы сможете усвоить основы F#, прежде чем мы перейдем к изучению более сложных возможностей языка.

В первом разделе этой главы рассматриваются элементарные типы, такие как `int` и `string`, которые являются строительными блоками любых программ на языке F#. Затем я расскажу о функциях, с помощью которых вы сможете манипулировать данными.

В четвертом разделе детально рассматриваются такие фундаментальные типы, как `list`, `option` и `unit`. Понимание принципов использования этих типов позволит вам перейти к изучению объектно-ориентированного и функционального стилей программирования на F#, представленных в последующих главах.

К концу этой главы вы научитесь писать на этом языке простые программы, выполняющие обработку данных. В последующих главах вы узнаете, как расширять возможности программного кода и увеличивать его выразительность, а пока начнем с начала.

Элементарные типы

Тип – это некоторая абстракция, которая в первую очередь имеет отношение к обеспечению целостности. Типы являются своего рода гарантией выполнения необходимых преобразований. Некоторые типы являются достаточно простыми, например тип целых чисел, тогда как другие являются более абстрактными, например функции. Язык F#

относится к языкам со статической типизацией – в том смысле, что проверка типов выполняется на этапе компиляции. Например, если функция принимает целое число в виде параметра, вы получите ошибку на этапе компиляции, если попытаетесь передать ей строку.

Подобно C# и VB.NET, язык F# поддерживает полный комплект *элементарных типов .NET* (которые одинаковы в большинстве языков программирования). Они встроены в F# и отделены от *пользовательских типов*, которые вы определяете сами.

Чтобы создать значение, просто воспользуйтесь операцией *связывания* `let`.

Операция связывания имеет и более широкие возможности, однако мы отложим их рассмотрение до главы 3. А пока достаточно будет знать, что операцию связывания (с использованием ключевого слова `let`) можно использовать для создания нового идентификатора. Например, ниже определяется новое значение `x` в окне сеансе FSI:

```
> let x = 1;;

val x : int = 1
```

Элементарные числовые типы

Элементарные числовые типы делятся на две группы: используемые для представления целых чисел и вещественных чисел. Целочисленные типы могут иметь разные размеры, поэтому значения некоторых типов занимают меньше памяти и могут представлять меньший диапазон чисел. Кроме того, целые числа могут быть знаковыми или беззнаковыми, что определяет, могут ли они иметь отрицательные значения или нет.

Вещественные типы также могут иметь различные размеры – в обмен на увеличение объема занимаемой памяти они могут обеспечивать более высокую точность представления значений.

Для определения новых числовых значений используется оператор связывания `let`, за которым следует целочисленный или вещественный литерал с необязательным суффиксом. Суффикс определяет тип целого или вещественного числа. Полный перечень элементарных числовых типов и их суффиксов приводится в табл. 2.1.

```
> let answerToEverything = 42UL;;

val answerToEverything : uint64 = 42UL

> let pi = 3.1415926M;;

val pi : decimal = 3.1415926M

> let avogadro = 6.022e23;;

val avogadro : float = 6.022e+23
```

Таблица 2.1. Элементарные числовые типы в языке F#

Тип	Суффикс	Тип .NET	Диапазон
byte	uy	System.Byte	от 0 до 255
sbyte	y	System.SByte	от –128 до 127
int16	s	System.Int16	от –32768 до 32767
uint16	us	System.UInt16	от 0 до 65535
int, int32		System.Int32	от -2^{31} до $2^{31}-1$
uint32	u	System.UInt32	от 0 до $2^{32}-1$
int64	L	System.Int64	от -2^{63} до $2^{63}-1$
uint64	UL	System.UInt64	от 0 до $2^{64}-1$
float		System.Double	Вещественное двойной точности согласно стандарту **IEEE64. Представляет значения, состоящие примерно из 15 значащих цифр.**
float32	f	System.Single	Вещественное одинарной точности согласно стандарту **IEEE32. Представляет значения, состоящие примерно из 7 значащих цифр.**
decimal	M	System.Decimal	Вещественный тип фиксированной точности хранит точно 28 цифр после запятой.

Кроме того, в языке F# допускается указывать значения в шестнадцатеричной (по основанию 16), восьмеричной (по основанию 8) и в двоичной (по основанию 2) системах счисления, используя префиксы `0x`, `0o` и `0b` соответственно:

```
> let hex = 0xFCAF;;

val hex : int = 64687

> let oct = 0o7771L;;

val oct : int64 = 4089L

> let bin = 0b00101010y;;

val bin : sbyte = 42y

> (hex, oct, bin);;

val it : int * int64 * sbyte = (64687, 4089L, 42y)
```

Если вы знакомы со стандартами IEEE32 и IEEE64, для представления вещественных чисел вы можете также использовать шестнадца-

теричную, восьмеричную и двоичную формы записи. Компилятор F# автоматически преобразует двоичные значения в вещественные числа. При использовании других систем счисления для представления вещественных чисел используйте суффикс `LF` для преобразования в тип `float` и `lf` – для преобразования в тип `float32`:

```
> 0x401E000000000000LF;;
val it : float = 7.5

> 0x00000000lf;;
val it : float32 = 0.0f
```

Арифметика

К значениям элементарных числовых типов можно применять стандартные арифметические операторы. В табл. 2.2 перечислены все поддерживаемые операторы. Как и в других языках среды CLR (Common Language Runtime – общеязыковая среда исполнения), при делении целых чисел результат округляется до ближайшего меньшего целого, а остаток отбрасывается.

Таблица 2.2. Арифметические операторы

Оператор	Описание	Пример	Результат
`+`	Сложение	`1 + 2`	`3`
`-`	Вычитание	`1 - 2`	`-1`
`*`	Умножение	`2 * 3`	`6`
`/`	Деление	`8L / 3L`	`2L`
`**`	Возведение в степень[a]	`2.0 ** 8.0`	`256.0`
`%`	Деление по модулю (остаток от деления)	`7 % 3`	`1`

[a] Оператор `**` возведения в степень может применяться только к значениям типов `float` и `float32`. Чтобы возвести в степень целочисленное значение, его необходимо либо преобразовать в вещественное число, либо использовать функцию `pown`.

По умолчанию арифметические операторы не выполняют проверку на переполнение, поэтому если, например, при выполнении операции сложения целых чисел произойдет переполнение, в результате получится отрицательное значение. (Аналогично при выполнении операции вычитания, если результат окажется слишком маленьким, чтобы поместиться в целое число, будет возвращено положительное значение.)

```
> 32767s + 1s;;
val it : int16 = -32768s
```

```
> -32768s + -1s;;
val it : int16 = 32767s
```

Если переполнение целых чисел может вызывать проблемы, вам следует подумать о возможности использования более емких типов или применять арифметические операции с проверкой, как описывается в главе 7.

В языке F# поддерживаются все стандартные математические функции, которые можно было бы ожидать; полный их перечень приводится в табл. 2.3.

Таблица 2.3. Стандартные математические функции

Функция	Описание	Пример	Результат
abs	Абсолютное значение числа	abs -1.0	1.0
ceil	Округление вверх до ближайшего целого	ceil 9.1	10
exp	Возведение e в степень	exp 1.0	2.718
floor	Округление вниз до ближайшего целого	floor 9.9	9.0
sign	Знак числа	sign -5	-1
log	Натуральный логарифм	log 2.71828	1.0
log10	Логарифм по основанию 10	log10 1000.0	3
sqrt	Корень квадратный	sqrt 4.0	2.0
cos	Косинус	cos 0.0	1.0
sin	Синус	sin 0.0	0.0
tan	Тангенс	tan 1.0	1.557
pown	Возведение целого числа в степень	pown 2L 10	1024L

Функции преобразования

Один из принципов, которым следует язык F#, заключается в отсутствии каких-либо неявных преобразований. Это означает, что компилятор не будет выполнять неявные автоматические преобразования элементарных типов, такие как преобразование значения типа int16 в значение типа int64. Это устраняет трудноуловимые ошибки, вызванные неявными преобразованиями. Для преобразования значений элементарных типов следует явно использовать функции преобразования, перечисленные в табл. 2.4. Все стандартные функции преобразования принимают значения любых элементарных типов, включая строки и символы.

Таблица 2.4. Функции преобразования элементарных числовых типов

Функция	Описание	Пример	Результат
sbyte	Преобразует значение в тип sbyte	sbyte -5	-5y
byte	Преобразует значение в тип byte	byte "42"	42uy
int16	Преобразует значение в тип int16	int16 'a'	97s
uint16	Преобразует значение в тип uint16	uint16 5	5us
int32, int	Преобразует значение в тип int	int 2.5	2
uint32	Преобразует значение в тип uint32	uint32 0xFF	255
int64	Преобразует значение в тип int64	int64 -8	-8L
uint64	Преобразует значение в тип uint64	uint64 "0xFF"	255UL
float	Преобразует значение в тип float	float 3.1415M	3.1415
float32	Преобразует значение в тип float32	float32 8y	8.0f
decimal	Преобразует значение в тип decimal	decimal 1.23	1.23M

Если эти функции принимают строку, то она разбирается с помощью семейства методов System.Convert, которые в случае недопустимых значений генерируют исключение System.FormatException.

Тип bigint

Если вам приходится иметь дело со значениями, превышающими 2^{64}, можно воспользоваться типом данных bigint, имеющимся в языке F# и позволяющим представлять целые значения произвольной величины. Хотя тип bigint – это всего лишь псевдоним для типа System.Numerics.BigInteger, стоит отметить, что ни в C#, ни в VB.NET нет специального синтаксиса для поддержки целых чисел произвольной величины.

Литералы типа bigint должны оканчиваться символом I. В примере 2.1 определяются значения типа bigint.

Пример 2.1. Тип bigint используется для представления целочисленных значений произвольной величины

```
> open System.Numerics

// Единицы измерения объема данных
let megabyte = 1024I * 1024I
let gigabyte = megabyte * 1024I
let terabyte = gigabyte * 1024I
let petabyte = terabyte * 1024I
let exabyte = petabyte * 1024I
let zettabyte = exabyte * 1024I;;
```

```
val megabyte : BigInteger = 1048576
val gigabyte : BigInteger = 1073741824
val terabyte : BigInteger = 1099511627776
val petabyte : BigInteger = 1125899906842624
val exabyte : BigInteger = 1152921504606846976
val zettabyte : BigInteger = 118059162071741130342 4
```

Несмотря на то, что операции со значениями типа `bigint` чрезвычайно оптимизированы, тем не менее они выполняются существенно медленнее, чем операции над значениями элементарных целочисленных типов.

Битовые операции

Элементарные целочисленные типы поддерживают битовые операции, позволяющие манипулировать отдельными битами. Битовые операторы обычно используются при чтении и записи двоичных данных в файлы. Перечень битовых операций приводится в табл. 2.5.

Таблица 2.5. Битовые операторы

Оператор	Описание	Пример	Результат
`&&&`	Битовое «И»	`0b1111 &&& 0b0011`	`0b0011`
`|||`	Битовое «ИЛИ»	`0xFF00 ||| 0x00FF`	`0xFFFF`
`^^^`	Битовое «Исключающее ИЛИ»	`0b0011 ^^^ 0b0101`	`0b0110`
`<<<`	Сдвиг влево	`0b0001 <<< 3`	`0b1000`
`>>>`	Сдвиг вправо	`0b1000 >>> 3`	`0b0001`

Символы

Платформа .NET основана на использовании Юникода, поэтому символы в ней представляются 2-байтными символами в кодировке UTF-16. Чтобы определить литерал символа, можно просто указать символ Юникода в одиночных кавычках. Символы могут также задаваться в виде шестнадцатеричных кодов символов.

В следующем фрагменте определяется список гласных символов и выводится результат задания символа в виде шестнадцатеричного кода:

```
> let vowels = ['a'; 'e'; 'i'; 'o'; 'u'];;

val vowels : char list = ['a'; 'e'; 'i'; 'o'; 'u']

> printfn "Hex u0061 = '%c'" '\u0061';;
Hex u0061 = 'a'
val it : unit = ()
```