

Краткое содержание

Предисловие	22
Введение	24

ЧАСТЬ I. ОСНОВЫ CLR

Глава 1. Модель выполнения кода в среде CLR	28
Глава 2. Компоновка, упаковка, развертывание и администрирование приложений и типов	58
Глава 3. Совместно используемые сборки и сборки со строгим именем	94

ЧАСТЬ II. ПРОЕКТИРОВАНИЕ ТИПОВ

Глава 4. Основы типов	122
Глава 5. Примитивные, ссылочные и значимые типы	142
Глава 6. Основные сведения о членах и типах	186
Глава 7. Константы и поля	210
Глава 8. Методы	215
Глава 9. Параметры	245
Глава 10. Свойства	263
Глава 11. События	286
Глава 12. Обобщения	302
Глава 13. Интерфейсы	333

ЧАСТЬ III. ОСНОВНЫЕ ТИПЫ ДАННЫХ

Глава 14. Символы, строки и обработка текста	356
Глава 15. Перечислимые типы и битовые флаги.	403
Глава 16. Массивы	416
Глава 17. Делегаты	434
Глава 18. Настраиваемые атрибуты.	464
Глава 19. Null-совместимые значимые типы	485

ЧАСТЬ IV. КЛЮЧЕВЫЕ МЕХАНИЗМЫ

Глава 20. Исключения и управление состоянием	496
Глава 21. Автоматическое управление памятью (уборка мусора)	554
Глава 22. Хостинг CLR и домены приложений	606
Глава 23. Загрузка сборок и отражение	636
Глава 24. Сериализация	666
Глава 25. Взаимодействие с компонентами WinRT	698

ЧАСТЬ V. МНОГОПОТОЧНОСТЬ

Глава 26. Потоки исполнения	724
Глава 27. Асинхронные вычислительные операции.	747
Глава 28. Асинхронные операции ввода-вывода	787
Глава 29. Примитивные конструкции синхронизации потоков	820
Глава 30. Гибридные конструкции синхронизации потоков ..	854
Словарь соответствия русскоязычных и англоязычных терминов	893

Содержание

Предисловие	22
Введение	24
От издателя перевода	26

ЧАСТЬ I. ОСНОВЫ CLR

Глава 1. Модель выполнения кода в среде CLR	28
Компиляция исходного кода в управляемые модули	28
Объединение управляемых модулей в сборку	32
Загрузка CLR	34
Исполнение кода сборки	37
IL-код и верификация	44
Небезопасный код	45
IL и защита интеллектуальной собственности	46
NGen.exe	47
Библиотека FCL	47
CTS	49
CLS	52
Взаимодействие с неуправляемым кодом	57

Глава 2. Компоновка, упаковка, развертывание и администрирование приложений и типов	58
Задачи развертывания в .NET Framework	58
Компоновка типов в модуль	60
Файл параметров	61
Несколько слов о метаданных	64

Объединение модулей для создания сборки	71
Добавление сборок в проект в среде Visual Studio	78
Использование утилиты Assembly Linker	79
Включение в сборку файлов ресурсов	81
Ресурсы со сведениями о версии сборки	82
Номера версии	86
Региональные стандарты	87
Развертывание простых приложений (закрытое развертывание сборок)	88
Простое средство администрирования (конфигурационный файл) ..	90
Алгоритм поиска файлов сборки	92

Глава 3. Совместно используемые сборки и сборки

со строгим именем	94
Два вида сборок — два вида развертывания	95
Назначение сборке строгого имени	96
Глобальный кэш сборки	102
Построение сборки, ссылающейся на сборку со строгим именем ..	104
Устойчивость сборок со строгими именами к несанкционированной модификации	106
Отложенное подписание	107
Закрытое развертывание сборок со строгими именами	110
Как исполняющая среда разрешает ссылки на типы	111
Дополнительные административные средства (конфигурационные файлы)	115
Управление версиями при помощи политики издателя	117

ЧАСТЬ II. ПРОЕКТИРОВАНИЕ ТИПОВ

Глава 4. Основы типов	122
Все типы — производные от System.Object	122
Приведение типов	124
Приведение типов в C# с помощью операторов is и as	126
Пространства имен и сборки	128
Связь между сборками и пространством имен	132
Как разные компоненты взаимодействуют во время выполнения ..	133

Глава 5. Примитивные, ссылочные и значимые типы	142
Примитивные типы в языках программирования	142
Проверяемые и непроверяемые операции для примитивных типов	146
Ссылочные и значимые типы	150
Как CLR управляет размещением полей для типа	155
Упаковка и распаковка значимых типов	156
Изменение полей в упакованных значимых типах посредством интерфейсов (и почему этого лучше не делать)	169
Равенство и тождество объектов	172
Хеш-коды объектов	175
Примитивный тип данных <code>dynamic</code>	177
Глава 6. Основные сведения о членах и типах	186
Члены типа	186
Видимость типа	189
Дружественные сборки	189
Доступ к членам типов	191
Статические классы	193
Частичные классы, структуры и интерфейсы	194
Компоненты, полиморфизм и версии	196
Вызов виртуальных методов, свойств и событий в CLR	198
Разумное использование видимости типов и модификаторов доступа к членам	202
Работа с виртуальными методами при управлении версиями типов	205
Глава 7. Константы и поля	210
Константы	210
Поля	212
Глава 8. Методы	215
Конструкторы экземпляров и классы (ссылочные типы)	215
Конструкторы экземпляров и структуры (значимые типы)	219
Конструкторы типов	222
Методы перегруженных операторов	226
Операторы и взаимодействие языков программирования	229

Особое мнение автора о правилах Microsoft, связанных с именами методов операторов	229
Методы операторов преобразования	230
Методы расширения	234
Правила и рекомендации	237
Расширение разных типов методами расширения	238
Атрибут расширения	240
Частичные методы	241
Правила и рекомендации	244
Глава 9. Параметры	245
Необязательные и именованные параметры	245
Правила использования параметров	246
Атрибут DefaultParameterValue и необязательные атрибуты	248
Неявно типизированные локальные переменные	248
Передача параметров в метод по ссылке	251
Передача переменного количества аргументов	257
Типы параметров и возвращаемых значений	259
Константность	261
Глава 10. Свойства	263
Свойства без параметров	263
Автоматически реализуемые свойства	267
Осторожный подход к определению свойств	268
Свойства и отладчик Visual Studio	270
Инициализаторы объектов и коллекций	271
Анонимные типы	273
Тип System.Tuple	276
Свойства с параметрами	279
Выбор главного свойства с параметрами	283
Производительность при вызове методов доступа	284
Доступность методов доступа свойств	285
Обобщенные методы доступа свойств	285
Глава 11. События	286
Разработка типа, поддерживающего событие	287

Этап 1. Определение типа для хранения всей дополнительной информации, передаваемой получателям уведомления о событии	288
Этап 2. Определение члена-события.	289
Этап 3. Определение метода, ответственного за уведомление зарегистрированных объектов о событии.. . . .	290
Этап 4. Определение метода, преобразующего входную информацию в желаемое событие	292
Реализация событий компилятором	293
Создание типа, отслеживающего событие	295
Явное управление регистрацией событий	298

Глава 12. Обобщения 302

Обобщения в библиотеке FCL	307
Инфраструктура обобщений	308
Открытые и закрытые типы.	309
Обобщенные типы и наследование.	311
Идентификация обобщенных типов	313
Разрастание кода	314
Обобщенные интерфейсы	315
Обобщенные делегаты.	316
Контравариантные и ковариантные аргументы-типы в делегатах и интерфейсах.	317
Обобщенные методы.	319
Обобщенные методы и выводение типов	320
Обобщения и другие члены	322
Верификация и ограничения	322
Основные ограничения	325
Дополнительные ограничения	327
Ограничения конструктора	328
Другие проблемы верификации	329

Глава 13. Интерфейсы 333

Наследование в классах и интерфейсах	333
Определение интерфейсов	334
Наследование интерфейсов.	335

Подробнее о вызовах интерфейсных методов	338
Явные и неявные реализации интерфейсных методов (что происходит за кулисами)	339
Обобщенные интерфейсы	341
Обобщения и ограничения интерфейса	344
Реализация нескольких интерфейсов с одинаковыми сигнатурами и именами методов	345
Совершенствование безопасности типов за счет явной реализации интерфейсных методов	346
Опасности явной реализации интерфейсных методов	348
Дилемма разработчика: базовый класс или интерфейс?	351

ЧАСТЬ III. ОСНОВНЫЕ ТИПЫ ДАННЫХ

Глава 14. Символы, строки и обработка текста	356
Символы	356
Тип <code>System.String</code>	359
Создание строк	359
Неизменяемые строки	362
Сравнение строк	362
Интернирование строк	369
Создание пулов строк	372
Работа с символами и текстовыми элементами в строке	372
Прочие операции со строками	375
Эффективное создание строк	375
Создание объекта <code>StringBuilder</code>	376
Члены типа <code>StringBuilder</code>	377
Получение строкового представления объекта	379
Форматы и региональные стандарты	380
Форматирование нескольких объектов в одну строку	384
Создание собственного средства форматирования	386
Получение объекта посредством разбора строки	389
Кодировки: преобразования между символами и байтами	391
Кодирование и декодирование потоков символов и байтов	397
Кодирование и декодирование строк в кодировке <code>Base-64</code>	398
Защищенные строки	399

Глава 15. Перечислимые типы и битовые флаги	403
Перечислимые типы	403
Битовые флаги	409
Добавление методов к перечислимым типам	413
Глава 16. Массивы	416
Инициализация элементов массива	418
Приведение типов в массивах	421
Базовый класс System.Array	423
Реализация интерфейсов IEnumerable, ICollection и IList	424
Передача и возврат массивов	425
Массивы с ненулевой нижней границей	426
Внутренняя реализация массивов	427
Небезопасный доступ к массивам и массивы фиксированного размера	432
Глава 17. Делегаты	434
Знакомство с делегатами	434
Обратный вызов статических методов	437
Обратный вызов экземплярных методов	438
Тонкости использования делегатов	439
Обратный вызов нескольких методов (цепочки делегатов)	443
Поддержка цепочек делегатов в C#	448
Дополнительные средства управления цепочками делегатов	448
Обобщенные делегаты	451
Упрощенный синтаксис работы с делегатами	452
Упрощение 1: не создаем объект делегата	452
Упрощение 2: не определяем метод обратного вызова	453
Упрощение 3: не создаем обертку для локальных переменных для передачи их методу обратного вызова	457
Делегаты и отражение	460
Глава 18. Настраиваемые атрибуты	464
Сфера применения настраиваемых атрибутов	464
Определение класса атрибутов	468

Конструктор атрибута и типы данных полей и свойств	471
Выявление настраиваемых атрибутов	473
Сравнение экземпляров атрибута	477
Выявление настраиваемых атрибутов без создания объектов, производных от Attribute	480
Условные атрибуты.	484

Глава 19. Null-совместимые значимые типы..... 485

Поддержка в C# null-совместимых значимых типов	487
Оператор объединения null-совместимых значений	490
Поддержка в CLR null-совместимых значимых типов	491
Упаковка null-совместимых значимых типов.	491
Распаковка null-совместимых значимых типов	492
Вызов метода GetType через null-совместимый значимый тип.	492
Вызов интерфейсных методов через null-совместимый значимый тип	493

ЧАСТЬ IV. КЛЮЧЕВЫЕ МЕХАНИЗМЫ

Глава 20. Исключения и управление состоянием..... 496

Определение «исключения»	496
Механика обработки исключений	498
Блок try	499
Блок catch	499
Блок finally	501
CLS-совместимые и CLS-несовместимые исключения	503
Класс System.Exception	505
Классы исключений, определенные в FCL	509
Генерирование исключений	511
Создание классов исключений	513
Продуктивность вместо надежности	515
Приемы работы с исключениями	524
Активно используйте блоки finally	525
Не надо перехватывать все исключения	526
Корректное восстановление после исключения	528

Отмена незавершенных операций при невозстановимых исключениях	529
Соккрытие деталей реализации для сохранения контракта	530
Необработанные исключения	533
Отладка исключений	537
Скорость обработки исключений	540
Области ограниченного выполнения	543
Контракты кода	546

Глава 21. Автоматическое управление памятью (уборка мусора)

Управляемая куча	554
Выделение ресурсов из управляемой кучи	555
Алгоритм уборки мусора	557
Уборка мусора и отладка	560
Поколения	562
Запуск уборки мусора	568
Большие объекты	569
Режимы уборки мусора	570
Программное управление уборщиком мусора	573
Мониторинг использования памяти приложением	574
Освобождение ресурсов при помощи механизма финализации	576
Типы, использующие системные ресурсы	583
Интересные аспекты зависимостей	588
Другие возможности уборщика мусора для работы с системными ресурсами	590
Внутренняя реализация финализации	594
Мониторинг и контроль времени жизни объектов	597

Глава 22. Хостинг CLR и домены приложений

Хостинг CLR	606
Домены приложений	609
Доступ к объектам из других доменов	612
Выгрузка доменов	624
Мониторинг доменов	626

Уведомление о первом управляемом исключении домена	627
Использование хостами доменов приложений.	628
Исполняемые приложения	628
Полнофункциональные интернет-приложения Silverlight	629
Microsoft ASP.NET и веб-службы XML	629
Microsoft SQL Server	630
Будущее и мечты	630
Нетривиальное управление хостингом	631
Применение управляемого кода	631
Разработка надежных хост-приложений	631
Возвращение потока в хост	633
Глава 23. Загрузка сборок и отражение	636
Загрузка сборок	637
Использование отражения для создания динамически расширяемых приложений	641
Производительность отражения	642
Нахождение типов, определенных в сборке	644
Объект Type	644
Создание иерархии типов, производных от Exception	646
Создание экземпляра типа	648
Создание приложений с поддержкой подключаемых компонентов ..	650
Нахождение членов типа путем отражения	653
Нахождение членов типа	654
Обращение к членам типов	658
Использование дескрипторов привязки для снижения потребления памяти процессом	663
Глава 24. Сериализация	666
Практический пример сериализации/десериализации	667
Сериализуемые типы	672
Управление сериализацией и десериализацией	673
Сериализация экземпляров типа	677
Управление сериализованными и десериализованными данными ..	679
Определение типа, реализующего интерфейс ISerializable, не реализуемый базовым классом	684

Контексты потока ввода-вывода	686
Сериализация в другой тип и десериализация в другой объект	688
Суррогаты сериализации	691
Цепочка селекторов суррогатов	694
Переопределение сборки и/или типа при десериализации объекта ..	695

Глава 25. Взаимодействие с компонентами WinRT..... 698

Проекция уровня CLR и правила системы типов компонентов WinRT.	700
Основные концепции системы типов WinRT.....	700
Проекция уровня .NET Framework.	705
Асинхронные вызовы WinRT API из кода .NET	705
Взаимодействия между потоками WinRT и потоками .NET	710
Передача блоков данных между CLR и WinRT	712
Определение компонентов WinRT в коде C#.....	715

ЧАСТЬ V. МНОГОПОТОЧНОСТЬ

Глава 26. Потоки исполнения..... 724

Для чего Windows поддерживает потоки?	724
Ресурсоемкость потоков.	725
Так дальше не пойдет!	729
Тенденции развития процессоров	732
CLR- и Windows-потоки	733
Потоки для асинхронных вычислительных операций	734
Причины использования потоков	736
Планирование и приоритеты потоков.	739
Фоновые и активные потоки.	744
Что дальше?	746

Глава 27. Асинхронные вычислительные операции..... 747

Пул потоков в CLR	747
Простые вычислительные операции	748
Контексты исполнения.	750
Скоординированная отмена.	752

Задания	757
Завершение задания и получение результата	758
Отмена задания	760
Автоматический запуск задания по завершении предыдущего	762
Дочерние задания	764
Структура задания	765
Фабрики заданий	767
Планировщики заданий	769
Методы For, ForEach и Invoke класса Parallel	771
Встроенный язык параллельных запросов	775
Периодические вычислительные операции	779
Разновидности таймеров	782
Как пул управляет потоками	783
Ограничение количества потоков в пуле	783
Управление рабочими потоками	784

Глава 28. Асинхронные операции ввода-вывода 787

Операции ввода-вывода в Windows	787
Асинхронные функции C#	792
Преобразование асинхронной функции в конечный автомат	795
Расширяемость асинхронных функций	799
Асинхронные функции и обработчики событий	803
Асинхронные функции в FCL	804
Асинхронные функции и исключения	806
Другие возможности асинхронных функций	807
Потоковые модели приложений	810
Асинхронная реализация сервера	813
Отмена операций ввода-вывода	814
Некоторые операции ввода-вывода должны выполняться синхронно	815
Проблемы FileStream	816
Приоритеты запросов ввода-вывода	817

Глава 29. Примитивные конструкции синхронизации потоков	820
Библиотеки классов и безопасность потоков	822
Примитивные конструкции пользовательского режима и режима ядра	824
Конструкции пользовательского режима	825
Volatile-конструкции	826
Interlocked-конструкции	832
Реализация простой циклической блокировки	837
Универсальный Interlocked-паттерн	841
Конструкции режима ядра	843
События	847
Семафоры	850
Мьютексы	851
Глава 30. Гибридные конструкции синхронизации потоков ..	854
Простая гибридная блокировка	854
Защитное владение потоком и рекурсия	857
Гибридные конструкции в FCL	859
Классы ManualResetEventSlim и SemaphoreSlim	859
Класс Monitor и блоки синхронизации	860
Класс ReaderWriterLockSlim	866
Класс OneManyLock	868
Класс CountdownEvent	871
Класс Barrier	872
Выводы по гибридным конструкциям	873
Блокировка с двойной проверкой	875
Паттерн условной переменной	880
Асинхронная синхронизация	882
Классы коллекций для параллельного доступа	888
Словарь соответствия русскоязычных и англоязычных терминов	893

Посвящение

Кристин. Словами не выразить то, что я думаю о нашей совместной жизни. Я с нежностью отношусь к нашей семье и ко всем нашим семейным радостям. Каждый день моей жизни наполнен любовью к тебе.

Адэн (9 лет) и Грант (5 лет). Вы оба вдохновляли меня, учили меня играть и весело проводить время. Наблюдения за тем, как вы росли, доставляли мне радость. Мне повезло стать частью вашей жизни. Я люблю и ценю вас больше всех на свете.

Предисловие

Вот мы и снова встретились. Кто бы мог подумать? Ага, знаю — я *должна* была это предвидеть!

Жизнь в браке — один сплошной «*День сурка*». Если вы не видели этот фильм, посмотрите; вы внезапно поймете, почему одни и те же ошибки приходится повторять снова и снова. Когда Джефф сказал мне, что он не будет писать следующую книгу, я сразу поняла, что это пустое обещание. Джефф *не может* не написать следующую книгу. Только сегодня мы с ним обсуждали еще одну книгу, которую он тоже совершенно не собирается писать (только почему-то уже написал целую главу). Это у него в крови. Породистая лошадь рождается для скачек, а Джефф рожден для того, чтобы писать книги.

Джефф предсказуем, как смена времен года. Он не может держаться подальше от ноликов и единичек на своем жестком диске. И когда нормальные люди мирно спят в своих постелях, внутренний будильник Джеффа начинает звонить где-то в 3 часа ночи (когда наш 5-летний сын залезает в нашу постель — еще одно явление, с которым, похоже, ничего не поделаешь). Какая-то таинственная сила направляет Джеффа в кабинет и заставляет его мозг решать всевозможные маленькие и большие проблемы. Другим остается лишь перевернуться на другой бок и снова заснуть — зная, что Джефф где-то рядом решает эти проблемы за нас — словно некий кибер-супергерой, спасающий программные потоки от преждевременной гибели.

Однако Джеффу недостаточно копить всю эту информацию в своем личном уголке Вселенной. Его начинают терзать угрызения совести — он должен поделиться своими мыслями с другими; записать их на бумаге. Они словно радиоволны, которые передаются в пространстве, чтобы их кто-то где-то принял. И все это делается для вас, дорогой читатель; перед вами свидетельство его страсти к технологиям Microsoft.

В этой книге добавилось авторской мудрости. Валяясь на солнышке, Джефф становится старше, и с течением лет он начинает оглядываться назад. Размышляя о вещах с более зрелых позиций, он переписал главу, посвященную отражению (reflection). Возможно, вы оцените его поэтический подход к теме. В этой главе рассказано о том, как заставить программный код задавать вопросы о другом коде, а также приводятся более глубокие рассуждения о том, почему отражение работает именно так, а не иначе. Наденьте халат, сядьте в кожаное кресло и предайтесь высокому размышлению о своем коде и смысле жизни.

Также в книге добавился материал о `async/await`. Очевидно, это развитие темы `AsyncEnumerator`, о которой мой любимый говорил какое-то время назад. Тогда мне порой казалось, что он ни о чем другом говорить уже не способен! Правда, после всех разговоров я так толком и не запомнила, что это такое. Джефф работал с группой

в «большом М», доводя до ума механизм `async/await`, и теперь этот материал вошел в книгу, чтобы порадовать читателя.

Еще одно крупное дополнение к книге вызывает у меня больше всего эмоций. Надеюсь, все читатели ознакомятся с главой о WinRT и возьмут этот материал на вооружение. WinRT — такой термин для технарей, которое каким-то образом означает: «Сделайте Мне Крутое Приложение Для Моего Планшета — ПРЯМО СЕЙЧАС!» Да, все верно; это новая исполнительная среда Windows для сенсорных экранов. Мои дети обожают кидаться птичками в свиней. Мне нравятся приложения с цветочками, и определенно планшеты полезны для образовательных целей. Дайте волю фантазии! И пожалуйста, сделайте так, чтобы эта глава принесла мне какую-нибудь пользу. Иначе мое терпение с Джеффом и его вечной работой над книгами иссякнет, и я запру его в комнате с вязальными спицами и без электричества. Выбирайте, программисты: либо пишем классные приложения с WinRT, либо новых книг от Джеффа уже не будет!

Короче говоря, Джефф (при вашем неустанном содействии) выдал очередной шедевр, и наша семья может вернуться к своему нормальному состоянию. Хотя что считать нормой? Возможно, состояние работы над очередной книгой уже стало нормальным.

*В терпеливом ожидании следующей книги,
Кристин Трейс (жена Джеффа)
октябрь 2012 г.*



На помощь! Спасите Джеффа от вязания!

Введение

В октябре 1999 года люди из Microsoft впервые продемонстрировали мне платформу .NET Framework, общезыковую среду выполнения (Common Language Runtime, CLR) и язык программирования C#. Все это произвело на меня сильное впечатление: я сразу понял, что мой подход к написанию программного обеспечения претерпит существенные изменения. Меня попросили проконсультировать команду разработчиков, и я немедленно согласился. Поначалу я думал, что платформа .NET Framework представляет собой абстрактную прослойку между Win32 API (Application Program Interface, интерфейс прикладного программирования) и моделью COM (Component Object Model, объектная модель компонентов). Но чем больше я изучал платформу .NET Framework, тем яснее видел, насколько я ее недооценивал. В некотором смысле это операционная система. У нее собственные диспетчер виртуальной памяти, система безопасности, файловый загрузчик, механизм обработки ошибок, модель изоляции приложений (домены приложений), модель многопоточности и многое другое. В этой книге все эти (и многие другие) темы освещаются таким образом, чтобы вы могли эффективно проектировать и реализовывать программное обеспечение на этой платформе.

Я написал текст этой книги в октябре 2012 года, и уже прошло 13 лет с тех пор, как я работаю с платформой .NET Framework и языком программирования C#. За эти 13 лет я создал разнообразные приложения и как консультант компании Microsoft внес значительный вклад в разработку платформы .NET Framework. В качестве сотрудника своей компании Wintellect (<http://Wintellect.com>) я работал со многими заказчиками, помогая им проектировать программное обеспечение, отлаживать и оптимизировать программы, решать их проблемы, связанные с .NET Framework. Весь этот опыт помог мне реально узнать, какие именно проблемы возникают у людей при работе с платформой .NET Framework и как эти проблемы решать. Я попытался сопроводить этими знаниями все темы, представленные в этой книге.

Для кого эта книга

Цель этой книги — объяснить, как разрабатывать приложения и многократно используемые классы для .NET Framework. В частности, это означает, что я собираюсь рассказать, как работает среда CLR и какие возможности она предоставляет разработчику. В этой книге также описываются различные классы стандартной библиотеки классов (Framework Class Library, FCL). Ни в одной книге невозможно описать FCL полностью — она содержит тысячи типов, и их число продолжает расти ударными темпами. Так что я остановлюсь на основных типах, с которыми

должен быть знаком каждый разработчик. И хотя в этой книге не рассматриваются отдельно подсистема графического интерфейса пользователя Windows Forms, система для построения клиентских приложений Windows Presentation Foundation (WPF), Microsoft Silverlight, веб-службы XML, веб-формы, Microsoft ASP.NET MVC, Windows Store Apps и т. д., технологии, описанные в ней, применимы ко *всем* этим видам приложений.

В книге используются системы Microsoft Visual Studio 2012, .NET Framework 4.5 и компилятор C# версии 5.0. Компания Microsoft старается по возможности обеспечивать обратную совместимость при выпуске новой версии этих программных продуктов, поэтому многое из того, что обсуждается в данной книге, применимо и к ранним версиям. Все примеры в книге написаны на языке C#, но так как в среде CLR можно использовать различные языки программирования, книга пригодится также и тем, кто пишет на других языках программирования.

ПРИМЕЧАНИЕ

Примеры кода, приведенные в книге, можно загрузить с сайта компании Wintellect (<http://Wintellect.com/Books>).

Я и мои редакторы усердно потрудились, чтобы дать вам наиболее точную, свежую, исчерпывающую, удобную, понятную и безошибочную информацию. Однако даже такая фантастическая команда не может предусмотреть все. Если вы обнаружите в этой книге ошибки (особенно ошибки в программном коде) или же у вас появится конструктивное предложение, то я буду вам очень признателен за сообщение об этом по моему адресу *JeffreyR@Wintellect.com*.

Благодарности

Я бы не написал эту книгу без помощи и технической поддержки многих людей. В частности, я хотел бы поблагодарить мою семью. Очень сложно измерить время и усилия, которые требуются для написания хорошей книги, но я знаю точно, что без поддержки моей жены Кристин и двух сыновей, Адэна и Гранта, я бы не смог написать эту книгу. Очень часто наши планы совместного времяпровождения срывались из-за жесткого графика работы. Теперь, по завершении работы над книгой, я наконец предвкушаю радость общения с семьей.

В написании книги мне помогли многие замечательные люди. Участники группы .NET Framework (многих из которых я считаю своими друзьями) занимались рецензированием материалов книги и беседовали со мной, давая информацию для размышлений. Кристоф Насарп (Christophe Nasarre), с которым я работал уже в нескольких проектах, выполнил титанический труд по проверке моей работы и позаботился о точности и корректности формулировок. Он внес очень большой вклад в повышение качества книги. Как всегда, было очень приятно работать с ре-

дакторской командой Microsoft Press. Отдельную благодарность я хочу выразить Бену Райану (Ben Ryan), Дэвону Масгрейву (Devon Musgrave) и Кэрол Диллингем (Carol Dillingham). Спасибо также Сюзи Карр (Susie Carr) и Кандейс Синклер (Candace Sinclair) за редактуру и помощь при выпуске книги.

Поддержка

Мы приложили максимум усилий, чтобы эта книга и сопроводительные материалы содержали как можно более точную информацию. Все исправления и изменения публикуются в нашем разделе Microsoft Press на сайте [oreilly.com](http://go.microsoft.com/fwlink/?Linkid=266601) по адресу <http://go.microsoft.com/fwlink/?Linkid=266601>.

При обнаружении ошибки, отсутствующей в списке, вы сможете сообщить о ней на той же странице.

Если вам понадобится дополнительная поддержка, обращайтесь в службу поддержки Microsoft Press Book Support по адресу mspinput@microsoft.com.

Пожалуйста, обратите внимание, что по указанным адресам техническая поддержка программных продуктов Microsoft не предоставляется.

Мы хотим знать ваше мнение

Мы приветствуем любые отзывы на эту книгу. Пожалуйста, сообщите свое мнение о книге в опросе, который находится по адресу <http://microsoft.com/learning/booksurvey>.

Опрос займет совсем немного времени, а мы внимательно прислушиваемся ко всем замечаниям и предложениям. Заранее спасибо за ваше участие!

Будьте в курсе

Общение продолжается! Мы в Твиттере: <http://twitter.com/MicrosoftPress>.

От издателя перевода

Для удобства читателей мы добавили в книгу краткий словарь соответствия русскоязычных и англоязычных терминов. Ваши замечания, предложения и вопросы отправляйте по адресу электронной почты comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

Подробную информацию о наших книгах вы найдете на веб-сайте издательства <http://www.piter.com>.

ЧАСТЬ I

Основы CLR

Глава 1. Модель выполнения кода в среде CLR	28
Глава 2. Компоновка, упаковка, развертывание и администрирование приложений и типов.	58
Глава 3. Совместно используемые сборки и сборки со строгим именем	94

Глава 1. Модель выполнения кода в среде CLR

В Microsoft .NET Framework появилось много новых концепций, технологий и терминов. Цель этой главы — дать обзор архитектуры .NET Framework, познакомить с новыми технологиями этой платформы и определить термины, с которыми вы столкнетесь при работе с ней. Также в этой главе изложен процесс построения приложения или набора распространяемых компонентов (файлов), которые содержат типы (классы, структуры и т. п.), и затем объяснено, как выполняется приложение.

Компиляция исходного кода в управляемые модули

Итак, вы решили использовать .NET Framework как платформу разработки. Отлично! Ваш первый шаг — определить вид создаваемого приложения или компонента. Предположим, что этот вопрос уже решен, все спроектировано, спецификации написаны и все готово для начала разработки.

Теперь надо выбрать язык программирования. И это непростая задача — ведь у разных языков имеются разные возможности. Например, с одной стороны, «неуправляемый код» C/C++ дает доступ к системе на низком уровне. Вы вправе распоряжаться памятью по своему усмотрению, при необходимости создавать программные потоки и т. д. С другой стороны, Microsoft Visual Basic 6.0 позволяет очень быстро строить пользовательские интерфейсы и легко управлять СОМ-объектами и базами данных.

Название среды — *общезыковая среда выполнения* (Common Language Runtime, CLR) — говорит само за себя: это среда выполнения, которая подходит для разных языков программирования. Основные возможности CLR (управление памятью, загрузка сборок, безопасность, обработка исключений, синхронизация) доступны в любых языках программирования, использующих эту среду. Например, при обработке ошибок среда выполнения опирается на исключения, а значит, во всех языках программирования, использующих эту среду выполнения, сообщения об ошибках передаются при помощи механизма исключений. Или, например, среда выполнения позволяет создавать программные потоки, а значит, во всех языках программирования, использующих эту среду, тоже могут создаваться потоки.

Фактически во время выполнения программы в среде CLR неизвестно, на каком языке программирования разработчик написал исходный код. А это значит, что можно выбрать любой язык программирования, который позволяет проще всего решить конкретную задачу. Разрабатывать программное обеспечение можно на любом языке программирования, если только используемый компилятор этого языка поддерживает CLR.

Так в чем же тогда преимущество одного языка программирования перед другим? Я рассматриваю компиляторы как средства контроля синтаксиса и анализа «правильности кода». Компиляторы проверяют исходный код, убеждаются, что все написанное имеет некий смысл, и затем генерируют код, описывающий решение данной задачи. Разные языки программирования позволяют разрабатывать программное обеспечение, используя различный синтаксис. Не стоит недооценивать значение выбора синтаксиса языка программирования. Например, для математических или финансовых приложений выражение мысли программиста на языке APL может сохранить много дней работы по сравнению с применением в данной ситуации языка Perl.

Компания Microsoft разработала компиляторы для следующих языков программирования, используемых на этой платформе: C++/CLI, C# (произносится «си шарп»), Visual Basic, F# (произносится «эф шарп»), Iron Python, Iron Ruby и ассемблер Intermediate Language (IL). Кроме Microsoft, еще несколько компаний и университетов создали компиляторы, предназначенные для среды выполнения CLR. Мне известны компиляторы для Ada, APL, Caml, COBOL, Eiffel, Forth, Fortran, Haskell, Lexico, LISP, LOGO, Lua, Mercury, ML, Mondrian, Oberon, Pascal, Perl, Php, Prolog, RPG, Scheme, Smalltalk и Tcl/Tk.

Рисунок 1.1 иллюстрирует процесс компиляции файлов с исходным кодом. Как видно из рисунка, исходный код программы может быть написан на любом языке, поддерживающем среду выполнения CLR. Затем соответствующий компилятор проверяет синтаксис и анализирует исходный код программы. Вне зависимости от типа используемого компилятора результатом компиляции будет являться *управляемый модуль* (managed module) — стандартный переносимый исполняемый (portable executable, PE) файл 32-разрядной (PE32) или 64-разрядной Windows (PE32+), который требует для своего выполнения CLR. Кстати, управляемые сборки всегда используют преимущества функции безопасности «предотвращения выполнения данных» (DEP, Data Execution Prevention) и технологию ASLR (Address Space Layout Optimization), применение этих технологий повышает информационную безопасность всей системы.

Компиляторы машинного кода производят код, ориентированный на конкретную процессорную архитектуру, например x86, x64 или ARM. В отличие от этого, все CLR-совместимые компиляторы генерируют IL-код. (Подробнее об IL-коде рассказано далее в этой главе.) IL-код иногда называют *управляемым* (managed code), потому что CLR управляет его выполнением.

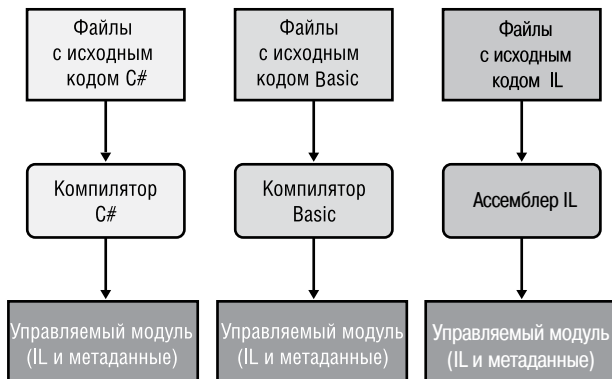


Рис. 1.1. Компиляция исходного кода в управляемые модули

В табл. 1.1 описаны составные части управляемого модуля.

Таблица 1.1. Части управляемого модуля

Часть	Описание
Заголовок PE32 или PE32+	Стандартный заголовок PE-файла Windows, аналогичный заголовку Common Object File Format (COFF). Файл с заголовком в формате PE32 может выполняться в 32- и 64-разрядной версиях Windows, а с заголовком PE32+ — только в 64-разрядной. Заголовок обозначает тип файла: GUI, CUI или DLL, он также имеет временную метку, показывающую, когда файл был собран. Для модулей, содержащих только IL-код, основной объем информации в заголовке PE32(+) игнорируется. В модулях, содержащих машинный код, этот заголовок содержит сведения о машинном коде
Заголовок CLR	Содержит информацию (интерпретируемую CLR и утилитами), которая превращает этот модуль в управляемый. Заголовок включает нужную версию CLR, некоторые флаги, метку метаданных MethodDef точки входа в управляемый модуль (метод Main), а также месторасположение/размер метаданных модуля, ресурсов, строгого имени, некоторых флагов и пр.
Метаданные	Каждый управляемый модуль содержит таблицы метаданных. Есть два основных вида таблиц — это таблицы, описывающие типы данных и их члены, определенные в исходном коде, и таблицы, описывающие типы данных и их члены, на которые имеются ссылки в исходном коде
Код Intermediate Language (IL)	Код, создаваемый компилятором при компиляции исходного кода. Впоследствии CLR компилирует IL в машинные команды

Каждый компилятор, предназначенный для CLR, помимо генерирования IL-кода, должен также создавать полные *метаданные* (metadata) для каждого управляемого модуля. Проще говоря, метаданные — это набор таблиц данных, описывающих то, что определено в модуле, например типы и их члены. В метаданных также есть таблицы, указывающие, на что ссылается управляемый модуль, например на импортируемые типы и их члены. Метаданные расширяют возможности таких старых технологий, как библиотеки типов COM и файлы IDL (Interface Definition Language, язык описания интерфейсов). Важно отметить, что метаданные CLR содержат куда более полную информацию. И, в отличие от библиотек типов и IDL-файлов, они всегда связаны с файлом, содержащим IL-код. Фактически метаданные всегда встроены в тот же EXE- или DLL-файл, что и код, так что их нельзя разделить. А поскольку компилятор генерирует метаданные и код одновременно и привязывает их к конечному управляемому модулю, возможность рассинхронизации метаданных и описываемого ими IL-кода исключена.

Метаданные находят много применений. Перечислим лишь некоторые из них.

- ❑ Метаданные устраняют необходимость в заголовочных и библиотечных файлах при компиляции, так как все сведения об упоминаемых типах/членах содержатся в файле с реализующим их IL-кодом. Компиляторы могут читать метаданные прямо из управляемых модулей.
- ❑ Среда Microsoft Visual Studio использует метаданные для облегчения написания кода. Ее функция IntelliSense анализирует метаданные и сообщает, какие методы, свойства, события и поля предпочтительны в данном случае и какие именно параметры требуются конкретным методам.
- ❑ В процессе верификации кода CLR использует метаданные, чтобы убедиться, что код совершает только «безопасные по отношению к типам» операции. (Проверка кода обсуждается далее.)
- ❑ Метаданные позволяют сериализовать поля объекта, а затем передать эти данные по сети на удаленный компьютер и там провести процесс десериализации, восстановив объект и его состояние на удаленном компьютере.
- ❑ Метаданные позволяют сборщику мусора отслеживать жизненный цикл объектов. При помощи метаданных сборщик мусора может определить тип объектов и узнать, какие именно поля в них ссылаются на другие объекты.

В главе 2 метаданные описаны более подробно.

Языки программирования C#, Visual Basic, F# и IL-ассемблер всегда создают модули, содержащие управляемый код (IL) и управляемые данные (данные, поддерживающие сборку мусора). Для выполнения любого управляемого модуля на машине конечного пользователя должна быть установлена среда CLR (в составе .NET Framework) — так же, как для выполнения приложений MFC или Visual Basic 6.0 должны быть установлены библиотека классов Microsoft Foundation Class (MFC) или DLL-библиотеки Visual Basic.

По умолчанию компилятор Microsoft C++ создает EXE- и DLL-файлы, которые содержат неуправляемый код и неуправляемые данные. Для их выполнения CLR не требуется. Однако если вызвать компилятор C++ с параметром /CLR в командной строке, он создаст управляемые модули (и конечно, для работы этих модулей должна быть установлена среда CLR). Компилятор C++ стоит особняком среди всех упомянутых компиляторов производства Microsoft — только он позволяет разработчикам писать как управляемый, так и неуправляемый код и встраивать его в единый модуль. Это также единственный компилятор Microsoft, позволяющий программистам определять в исходном коде как управляемые, так и неуправляемые типы данных. Компилятор Microsoft предоставляет разработчику непревзойденную гибкость, позволяя использовать существующий неуправляемый код на C/C++ из управляемого кода и постепенно, по мере необходимости, переходить на управляемые типы.

Объединение управляемых модулей в сборку

На самом деле среда CLR работает не с модулями, а со сборками. *Сборка* (assembly) — это абстрактное понятие, понять смысл которого на первых порах бывает нелегко. Во-первых, сборка обеспечивает логическую группировку одного или нескольких управляемых модулей или файлов ресурсов. Во-вторых, это наименьшая единица многократного использования, безопасности и управления версиями. Сборка может состоять из одного или нескольких файлов — все зависит от выбранных средств и компиляторов. В контексте среды CLR сборкой называется то, что мы обычно называем *компонентом*.

О сборках довольно подробно рассказано в главе 2, а здесь достаточно подчеркнуть, что это концептуальное понятие обозначает способ объединения группы файлов в единую сущность.

Рисунок 1.2 поможет понять суть сборки. На этом рисунке изображены некоторые управляемые модули и файлы ресурсов (или данных), с которыми работает некоторая программа. Эта программа создает единственный файл PE32(+), который обеспечивает логическую группировку файлов. При этом в файл PE32(+) включается блок данных, называемый *манифестом* (manifest). Манифест представляет собой обычный набор таблиц метаданных. Эти таблицы описывают файлы, которые входят в сборку, общедоступные экспортируемые типы, реализованные в файлах сборки, а также относящиеся к сборке файлы ресурсов или данных.

По умолчанию компиляторы сами выполняют работу по преобразованию созданного управляемого модуля в сборку, то есть компилятор C# создает управляемый модуль с манифестом, указывающим, что сборка состоит только из одного файла. Таким образом, в проектах, где есть только один управляемый модуль и нет файлов

ресурсов (или файлов данных), сборка и является управляемым модулем, поэтому выполнять дополнительные действия по компоновке приложения не нужно. В случае если необходимо сгруппировать несколько файлов в сборку, потребуются дополнительные инструменты (например, компоновщик сборок `AL.exe`) со своими параметрами командной строки. О них подробно рассказано в главе 2.

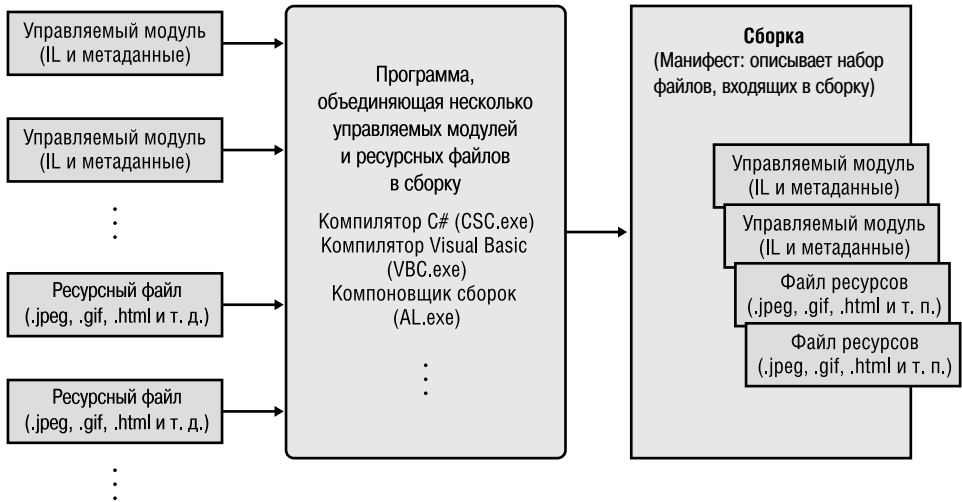


Рис. 1.2. Объединение управляемых модулей в сборку

Сборка позволяет разделить логическое и физическое представления компонента, поддерживающего многократное использование, безопасность и управление версиями. Разбиение программного кода и ресурсов на разные файлы полностью определяется желаниями разработчика. Например, редко используемые типы и ресурсы можно вынести в отдельные файлы сборки. Отдельные файлы могут загружаться по запросу из Интернета по мере необходимости в процессе выполнения программы. Если некоторые файлы не потребуются, то они не будут загружаться, что сохранит место на жестком диске и сократит время установки программы. Сборки позволяют разбить на части процесс развертывания файлов, при этом все файлы будут рассматриваться как единый набор.

Модули сборки также содержат сведения о других сборках, на которые они ссылаются (в том числе номера их версий). Эти данные делают сборку *самоописываемой* (self-describing). Другими словами, среда CLR может определить все прямые зависимости данной сборки, необходимые для ее выполнения. Не нужно размещать никакой дополнительной информации ни в системном реестре, ни в доменной службе AD DS (Active Directory Domain Services). Вследствие этого развертывать сборки гораздо проще, чем неуправляемые компоненты.

Загрузка CLR

Каждая создаваемая сборка представляет собой либо исполняемое приложение, либо библиотеку DLL, содержащую набор типов для использования в исполняемом приложении. Разумеется, среда CLR отвечает за управление исполнением кода. Это значит, что на компьютере, выполняющем данное приложение, должна быть установлена платформа .NET Framework. В компании Microsoft был создан дистрибутивный пакет .NET Framework для свободного распространения, который вы можете бесплатно поставлять своим клиентам. Некоторые версии операционной системы семейства Windows поставляются с уже установленной платформой .NET Framework.

Для того чтобы понять, установлена ли платформа .NET Framework на компьютере, попробуйте найти файл `MSCorEE.dll` в каталоге `%SystemRoot%\system32`. Если он есть, то платформа .NET Framework установлена. Однако на одном компьютере может быть установлено одновременно несколько версий .NET Framework. Чтобы определить, какие именно версии установлены, проверьте содержимое следующих подкаталогов:

```
%SystemRoot%\Microsoft.NET\Framework  
%SystemRoot%\Microsoft.NET\Framework64
```

Компания Microsoft включила в .NET Framework SDK утилиту командной строки `CLRVer.exe`, которая выводит список всех версий CLR, установленных на машине, а также сообщает, какая именно версия среды CLR используется текущими процессами. Для этого нужно указать параметр `-all` или идентификатор интересующего процесса.

Прежде чем переходить к загрузке среды CLR, поговорим поподробнее об особенностях 32- и 64-разрядных версий операционной системы Windows. Если сборка содержит только управляемый код с контролем типов, она должна одинаково хорошо работать на обеих версиях системы. Дополнительной модификации исходного кода не требуется. Более того, созданный компилятором готовый EXE- или DLL-файл будет правильно выполняться в Windows версий x86 и x64, а библиотеки классов и приложения Windows Store будут работать на машинах с Windows RT (использующих процессор ARM). Другими словами, один и тот же файл будет работать на любом компьютере с установленной платформой .NET Framework.

В исключительно редких случаях разработчикам приходится писать код, совместимый только с какой-то конкретной версией Windows. Обычно это требуется при работе с небезопасным кодом (`unsafe code`) или для взаимодействия с неуправляемым кодом, ориентированным на конкретную процессорную архитектуру. Для таких случаев у компилятора C# предусмотрен параметр командной строки `/platform`. Этот параметр позволяет указать конкретную версию целевой платформы, на которой планируется работа данной сборки: архитектуру x86, использующую только 32-разрядную систему Windows, архитектуру x64, использующую только 64-разрядную операционную систему Windows, или архитектуру ARM, на которой работает только