Часть I

Язык C#

В части I описаны элементы языка C#: ключевые слова, синтаксис и операторы. Кроме того, здесь рассмотрены основные инструменты программирования C# (например, способы организации ввода-вывода и средства получения информации о типе), которые тесно связаны с языком C#.

Глава 1

Создание языка C#

Язык C# — это очередная ступень бесконечной эволюции языков программирования. Его создание вызвано процессом усовершенствования и адаптации, который определял разработку компьютерных языков в течение последних лет. Подобно всем успешным языкам, которые увидели свет раньше, C# опирается на прошлые достижения постоянно развивающегося искусства программирования.

В языке C# (созданном компанией Microsoft для поддержки среды .NET Framework) проверенные временем средства усовершенствованы с помощью самых современных технологий. C# предоставляет очень удобный и эффективный способ написания программ для современной среды вычислительной обработки данных, которая включает операционную систему Windows, Internet, компоненты и пр. В процессе становления язык C# переопределил весь "ландшафт" программирования.

Назначение этой главы — рассмотреть C# в исторической среде, исследовать мотивы его создания и конструктивные особенности, а также степень влияния на него других языков программирования. Описана связь C# со средой .NET Framework.

Генеалогическое дерево C#

Компьютерные языки существуют не в вакууме. Они связаны друг с другом, причем на каждый новый язык в той или иной форме влияют его предшественники. В процессе такого "перекрестного опыления" средства из одного языка адаптируются другим, удачная новинка интегрируется в существующий контекст, а отжившая конструкция отбрасывается за ненадобностью. Примерно так и происходит эволюция компьютерных языков и развитие искусства программирования. Не избежал подобной участи и C#.

Языку C# "досталось" богатое наследство. Он — прямой потомок двух самых успешных языков программирования (C и C++) и тесно связан с не менее успешным языком Java. Понимание природы этих взаимосвязей крайне важно для понимания C#. Поэтому знакомство с C# мы начнем с рассмотрения исторической среды этих трех языков.

Язык C, или начало современной эпохи программирования

Начало современной эпохи программирования отмечено созданием языка C. Он был разработан Дэнисом Ритчи (Dennis Ritchie) в 1970-х годах для компьютера PDP-11 компании DEC (Digital Equipment Corporation), в котором использовалась операционная система UNIX. Несмотря на то что некоторые известные языки программирования, в особенности Pascal, достигли к тому времени значительного развития и признания, именно язык C определил направление сегодняшнего программирования.

Язык C вырос из кризиса программного обеспечения 1960-х годов и революционного перехода к *структурному программированию*. До структурного программирования многие программисты испытывали трудности при написании больших программ, поскольку обозначилась тенденция вырождения программной логики и появления так называемого "спагетти-кода" (spaghetti code) с большим размером процедур и интенсивным использованием оператора перехода `goto`. Такие программы были весьма трудны для изучения и модификаций. В структурных языках программирования эта проблема решалась посредством добавления точно определенных управляющих конструкций, вызова подпрограмм с локальными переменными и других усовершенствований. Структурные языки позволили писать довольно большие программы в приемлемые сроки.

Хотя в то время уже существовали другие структурные языки, C был первым языком, в котором удачно сочетались мощь, элегантность, гибкость и выразительность. Его лаконичный и к тому же простой в применении синтаксис в совокупности с философией, подразумевающей возложение ответственности на программиста (а не на язык), быстро завоевал множество сторонников. С точки зрения сегодняшнего дня, этот язык, возможно, несколько трудноват для понимания, но программистам того времени он показался порывом свежего ветра, которого они так долго ждали. В результате C стал самым популярным структурным языком программирования 1980-х годов.

Но многоуважаемый язык C имел ограничения. Одним из его недостатков была невозможность справиться с большими программами. Если проект достигал определенного размера, то дальнейшая его поддержка и развитие были связаны с определенными трудностями. Местоположение этой "точки насыщения" зависело от конкретной программы, программиста и используемых им средств, но вероятность ее достижения очень возрастала, когда количество строк в программе приближалось к 5 000.

Создание ООП и C++

К концу 1970-х размер проектов стал приближаться к критическому, при превышении которого методика структурного программирования и язык C "опускали руки". Поэтому стали появляться новые подходы к программированию, позволяющие решить эту проблему. Один из них получил название *объектно-ориентированного программирования* (ООП). Используя ООП, программист мог справляться с программами гораздо большего размера, чем прежде. Но вся беда состояла в том, что C, самый популярный на то время язык, не поддерживал ООП. Желание работать с объектно-ориентированной версией языка C в конце концов и привело к созданию C++.

Язык C++ был разработан Бьярни Страуструпом (Bjarne Stroustrup) в компании Bell Laboratories (Муррей Хил, Нью-Джерси), и годом создания считается 1979-й. Первоначально создатель нового языка назвал его "C с классами", но в 1983 году это имя было изменено на C++. C++ полностью включает элементы языка C. Таким образом, C можно считать фундаментом, на котором построен C++. Большинство дополнений, которые Страуструп внес в C, были предназначены для поддержки объектно-ориентированного программирования. По сути, C++ — это объектно-ориентированная версия языка C. Возводя "здание" C++ на фундаменте C, Страуструп обеспечил плавный переход многих программистов на "рельсы" ООП. Вместо необходимости изучать совершенно новый язык, C-программисту достаточно было освоить лишь новые средства, позволяющие использовать преимущества объектно-ориентированной методики.

На протяжении 1980-х годов C++ интенсивно развивался и к началу 1990-х уже был готов для широкого использования. Рост его популярности носил взрывоподобный характер, и к концу этого десятилетия он стал самым широко используемым языком программирования. В наши дни язык C++ по-прежнему имеет неоспоримое превосходство при разработке высокопроизводительных программ системного уровня.

Важно понимать, что создание C++ не было попыткой изобрести совершенно новый язык программирования. Это было своего рода усовершенствование и без того очень успешного языка. Такой подход к разработке языков (взять за основу существующий язык и поднять его на новую ступень развития) дал начало тенденции, которая продолжает жить и сегодня.

Internet и появление языка Java

Следующей ступенью на лестнице прогресса языков программирования стал язык Java, который первоначально назывался Oak (в переводе с англ. "дуб"). Работа над его созданием началась в 1991 году в компании Sun Microsystems. Основной движущей силой разработки Java был Джеймс Гослинг (James Gosling). В его рабочую группу входили Патрик Нотон (Patrick Naughton), Крис Уортс (Chris Warth), Эд Фрэнк (Ed Frank) и Майк Шеридан (Mike Sheridan).

Java — это структурный объектно-ориентированный язык программирования, синтаксис и основополагающие принципы которого "родом" из C++. Своими новаторскими аспектами Java обязан не столько прогрессу в искусстве программирования (хотя и это имело место), сколько изменениям в компьютерной среде. Еще до наступления эры Internet большинство программ писалось, компилировалось и предназначалось для выполнения с использованием определенного процессора и под управлением конкретной операционной системы. Несмотря на то что программисты всегда старались делать свои программы так, чтобы их можно было применять неоднократно, возможность легко переносить программу из одной среды в другую не была еще достигнута, к тому же проблема переносимости постоянно отодвигалась, решались же более насущные проблемы. Однако с появлением всемирной сети Internet, в которой оказались связанными различные типы процессоров и операционных систем, старая проблема переносимости заявила о себе уже в полный голос. Для ее решения понадобился новый язык программирования, и им стал Java.

Интересно отметить, что, хотя единственным наиболее важным аспектом Java (и причиной быстрого признания) является возможность создавать на нем межплатформенный (совместимый с несколькими операционными средами) переносимый программный код, исходным импульсом для возникновения Java стала не сеть Internet, а настоятельная потребность в не зависящем от платформы языке, который можно было бы использовать в процессе создания программного обеспечения для встроенных контроллеров. В 1993 году стало очевидным, что проблемы межплатформенной переносимости, четко проявившиеся при создании кода для встроенных контроллеров, также оказались весьма актуальными при попытке написать код для Internet. Ведь Internet — это безбрежная компьютерная среда, в которой "обитает" множество компьютеров различных типов. И оказалось, что одни и те же методы решения проблемы переносимости в малых масштабах можно успешно применить и к гораздо бо́льшим, т.е. в Internet.

В Java переносимость достигается посредством преобразования исходного кода программы в промежуточный код, именуемый *байт-кодом* (bytecode), т.е. машинно-независимый код, генерируемый Java-компилятором. Байт-код выполняется виртуальной машиной Java (Java Virtual Machine — JVM) — специальной операционной системой. Следовательно, Java-программа могла бы работать в любой среде, где доступна JVM. А поскольку JVM относительно проста для реализации, она быстро стала доступной для большого количества сред.

Использование Java-программами байт-кода радикально отличало их от C- и C++-программ, которые почти всегда компилировались для получения исполняемого машинного кода. Машинный код связан с конкретным процессором и операционной системой. Поэтому, если C/C++-программу нужно выполнить в другой системе, ее необходимо перекомпилировать в машинный код, соответствующий этой среде. Следовательно, чтобы создать C/C++-программу, предназначенную для выполнения в различных средах, необходимо иметь несколько различных исполняемых (машинных) версий этой программы. Это было непрактично и дорого. И наоборот, использование для выполнения Java-программ промежуточного языка было элегантным и рентабельным решением. Именно это решение было адаптировано для языка C#.

Как уже упоминалось, Java — потомок C и C++. Его синтаксис основан на синтаксисе C, а объектная модель — продукт эволюции объектной модели C++. Хотя Java-код несовместим с C или C++ ни снизу вверх, ни сверху вниз, его синтаксис так похож на синтаксис языка C, что толпы C/C++-программистов могли с минимальными усилиями переходить к программированию на Java. Более того, поскольку язык Java строился на существующей парадигме (и усовершенствовал ее), Джеймсу Гослингу ничто не мешало сосредоточить внимание на новых возможностях этого языка. Подобно тому как Страуструпу не нужно было "изобретать колесо" при создании C++, так и Гослингу при разработке Java не было необходимости создавать совершенно новый язык программирования. Более того, создание Java показало, что языки C и C++ — прекрасный "субстрат" для "выращивания" новых компьютерных языков.

Создание C#

Разработчики Java успешно решили многие проблемы, связанные с переносимостью в среде Internet, но далеко не все. Одна из них — *межъязыковая возможность взаимодействия* (cross-language interoperability) программных и аппаратных изделий разных поставщиков, или *многоязыковое программирование* (mixed-language programming). В случае решения этой проблемы программы, написанные на разных языках, могли бы успешно работать одна с другой. Такое взаимодействие необходимо для создания больших систем с распределенным программным обеспечением (ПО), а также для программирования компонентов ПО, поскольку самым ценным является компонент, который можно использовать в широком диапазоне компьютерных языков и операционных сред.

Кроме того, в Java не достигнута полная интеграция с платформой Windows. Хотя Java-программы могут выполняться в среде Windows (при условии установки виртуальной машины Java), Java и Windows не являются прочно связанными средами. А поскольку Windows — это наиболее широко используемая операционная система в мире, то отсутствие прямой поддержки Windows — серьезный недостаток Java.

Чтобы удовлетворить эти потребности, Microsoft разработала язык C#. C# был создан в конце 1990-х годов и стал частью общей .NET-стратегии Microsoft. Впервые он увидел свет в качестве α-версии в середине 2000 года. Главным архитектором C# был Андерс Хейлсберг (Anders Hejlsberg) — один из ведущих специалистов в области языков программирования, получивший признание во всем мире. Достаточно сказать, что в 1980-х он был автором весьма успешного продукта Turbo Pascal, изящная реализация которого установила стандарт для всех будущих компиляторов.

C# непосредственно связан с C, C++ и Java. И это не случайно. Эти три языка — самые популярные и самые любимые языки программирования в мире. Более того, почти все профессиональные программисты сегодня знают C и C++, и большинство знает Java. Поскольку C# построен на прочном, понятном фундаменте, то переход от этих "фундаментальных" языков к "надстройке" происходит без особых усилий со стороны программистов. Так как Андерс Хейлсберг не собирался изобретать свое "колесо", он сосредоточился на введении усовершенствований и новшеств.

Генеалогическое дерево C# показано на рис. 1.1. "Дедушкой" C# является язык C. От C язык C# унаследовал синтаксис, многие ключевые слова и операторы. Кроме того, C# построен на улучшенной объектной модели, определенной в C++. Если вы знаете C или C++, то с C# вы сразу станете друзьями.

C# и Java связаны между собой несколько сложнее. Как упоминалось выше, Java также является потомком C и C++. У него тоже общий с ними синтаксис и сходная объектная модель. Подобно Java C# предназначен для создания переносимого кода. Однако C# — не потомок Java. Скорее C# и Java можно считать двоюродными братьями, имеющими общих предков, но получившими от родителей разные наборы

“генов”. Если вы знаете язык Java, то вам будут знакомы многие понятия C#. И наоборот, если в будущем вам придется изучать Java, то, познакомившись с C#, вам не придется осваивать многие средства Java.

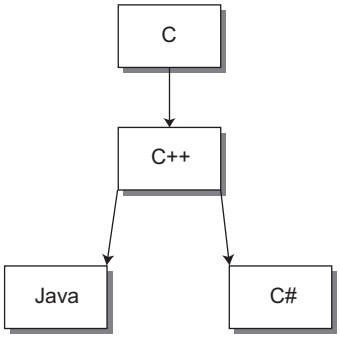

Рис. 1.1. Генеалогическое дерево C#

C# содержит множество новых средств, которые описаны в этой книге. Самые важные из них связаны со встроенной поддержкой программных компонентов. Именно наличие встроенных средств написания программных компонентов и позволило C# называться *компонентно-ориентированным языком*. Например, C# включает средства, которые напрямую поддерживают составные части компонентов: свойства, методы и события. Все же самым важным качеством компонентно-ориентированного языка является его способность работать в среде многоязыкового программирования.

Связь C# с оболочкой .NET Framework

Несмотря на то что C# — самодостаточный компьютерный язык, у него особые взаимоотношения со средой .NET Framework. И на это есть две причины. Во-первых, C# изначально разработан компанией Microsoft для создания кода, выполняющегося в среде .NET Framework. Во-вторых, в этой среде определены библиотеки, используемые языком C#. И хотя можно отделить C# от .NET Framework, эти две среды тесно связаны, поэтому очень важно иметь общее представление о .NET Framework и понимать, почему эта среда столь важна для C#.

О среде .NET Framework

Оболочка .NET Framework определяет среду для разработки и выполнения сильно распределенных приложений, основанных на использовании компонентных объектов. Она позволяет “мирно сосуществовать” различным языкам программирования и обеспечивает безопасность, переносимость программ и общую модель программирования для платформы Windows. Важно при этом понимать, что .NET Framework по своему существу не ограничена применением в Windows, т.е. программы, написанные для нее, можно затем переносить в среды, отличные от Windows.

Связь среды .NET Framework с C# обусловлена наличием двух очень важных средств. Одно из них, *Common Language Runtime* (CLR), представляет собой систему, которая управляет выполнением пользовательских программ. CLR — это составная часть .NET Framework, которая делает программы переносимыми, поддерживает многоязыковое программирование и обеспечивает безопасность.

Второе средство, *библиотека классов* .NET-оболочки, предоставляет программам доступ к среде выполнения. Например, если вам нужно выполнить операцию ввода-вывода, скажем, отобразить что-либо на экране, то для этого необходимо использовать .NET-библиотеку классов. Если вы — новичок в программировании, термин *класс* вам может быть незнаком. Ниже вы найдете подробное объяснение этого понятия, а пока ограничимся кратким его определением: класс — это объектно-ориентированная конструкция, с помощью которой организуются программы. Если программа ограничивается использованием средств, определенных .NET-библиотекой классов, она может выполняться везде (т.е. в любой среде), где поддерживается .NET-система. Поскольку C# автоматически использует .NET-библиотеку классов, C#-программы автоматически переносимы во все .NET-среды.

Функционирование системы CLR

Система CLR управляет выполнением .NET-кода. Вот как это происходит. В результате компиляции C#-программы получается не исполняемый код, а файл, который содержит специальный псевдокод, именуемый *промежуточным языком Microsoft* (*Microsoft Intermediate Language* — MSIL). MSIL определяет набор переносимых инструкций, которые не зависят от типа процессора. По сути, MSIL определяет переносимость ассемблера. И хотя концептуально MSIL подобен байт-коду Java, это не одно и то же.

Цель CLR-системы — при выполнении программы перевести ее промежуточный код в исполняемый. Таким образом, программа, подвергнутая MSIL-компиляции, может быть выполнена в любой среде, для которой реализована CLR-система. В этом частично и состоит способность среды .NET Framework добиваться переносимости программ.

Код, написанный на промежуточном языке Microsoft, переводится в исполняемый с помощью *JIT-компилятора*. "JIT" — сокр. от выражения "*j*ust-*i*n-*t*ime", означающего выполнение точно к нужному моменту (так обозначается стратегия принятия решений в самый последний подходящий для этого момент в целях обеспечения их максимальной точности). Этот процесс работает следующим образом. При выполнении .NET-программы CLR-система активизирует JIT-компилятор, который преобразует MSIL-код в ее "родной" код на требуемой основе, поскольку необходимо сохранить каждую часть программы. Таким образом, C#-программа в действительности выполняется в виде "родного" кода, несмотря на то, что первоначально она была скомпилирована в MSIL-код. Это значит, что программа будет выполнена практически так же быстро, как если бы она с самого начала была скомпилирована с получением "родного" кода, но с "добавлением" преимуществ переносимости от преобразования в MSIL-код.

В результате компиляции C#-программы помимо MSIL-кода образуются и *метаданные* (metadata). Они описывают данные, используемые программой, и позволяют коду взаимодействовать с другим кодом. Метаданные содержатся в том же файле, где хранится MSIL-код.

Сравнение управляемого кода с неуправляемым

В общем случае при написании C#-программы создается код, называемый *управляемым* (managed code). Управляемый код выполняется под управлением CLR-системы. У такого выполнения в результате есть как определенные ограничения, так и немалые достоинства. К числу ограничений относится необходимость иметь, во-

первых, специальный компилятор, который должен создавать MSIL-файл, предназначенный для работы под управлением CLR-системы, и, во-вторых, этот компилятор должен использовать библиотеки среды .NET Framework. Достоинства же управляемого кода — современные методы управления памятью, возможность использовать различные языки, улучшенная безопасность, поддержка управления версиями и четкая организация взаимодействия программных компонентов.

Что же понимается под неуправляемым кодом? Все Windows-программы до создания среды .NET Framework использовали неуправляемый код, который не выполняется CLR-системой. Управляемый и неуправляемый код могут работать вместе, поэтому факт создания C#-компилятором управляемого кода отнюдь не ограничивает его возможность выполняться совместно с ранее созданными программами.

Спецификация универсального языка

Несмотря на то что управляемый код обладает достоинствами, предоставляемыми CLR-системой, но если он используется другими программами, написанными на иных языках, то для достижения максимального удобства и простоты использования он должен соответствовать *спецификации универсального языка* (Common Language Specification — CLS). Эта спецификация описывает набор свойств, которыми одновременно должны обладать различные языки. Соответствие CLS-спецификации особенно важно при создании программных компонентов, которые предназначены для использования программами, написанными на других языках. CLS-спецификация включает подмножество *системы поддержки общих типов* (Common Type System — CTS). CTS-система определяет правила в отношении типов данных. Безусловно, C# поддерживает как CLS-, так и CTS-спецификации.