Лекции

Оглавление

Часть 1. Ядро языка

Лекция 1. Язык программирования и среда разработки. Цели курса

Основной целью этого курса является изучение основ объектного стиля разработки программных проектов. Для программиста, владеющего этими основами, не столь важно, на каком конкретном языке программирования или в какой среде ему необходимо разработать тот или иной программный проект, – на любом языке он будет создавать программный продукт требуемого качества. Тем не менее, у каждого программиста есть свои предпочтения, свой любимый язык и среда разработки.

В этой книге в качестве языка программирования выбран язык C# и его версия 3.0, в качестве среды разработки программных проектов – Visual Studio 2008, Professional Edition и Framework .Net в версии 3.5.

Язык C#

Язык C# является наиболее известной новинкой в области языков программирования. По сути это язык программирования, созданный уже в 21-м веке. Явившись на свет в недрах Microsoft, он с первых своих шагов получил мощную поддержку. Язык признан международным сообществом. В июне 2006 года Европейской ассоциацией по стандартизации принята уже четвертая версия стандарта этого языка: Standard ECMA-334 C# Language Specifications, 4-th edition – http://www.ecma-international.org/publications/standards/Ecma-334.htm

Международной ассоциацией по стандартизации эта версия языка узаконена как стандарт ISO/IEC – 23270. Заметим, что первая версия стандарта языка была принята еще в 2001 году. Компиляторы Microsoft строятся в соответствии с международными стандартами языка.

Язык C# является молодым языком и продолжает интенсивно развиваться. Каждая новая версия языка включает принципиально новые свойства. Не стала исключением и версия 3.0, рассматриваемая в данном учебном курсе.

Руководителем группы, создающей язык C#, является сотрудник Microsoft Андреас Хейлсберг. Он был известен в мире программистов задолго до того, как пришел в Microsoft. Хейлсберг входил в число ведущих разработчиков одной из самых популярных сред разработки – Delphi. В Microsoft он участвовал в создании версии языка Java – J++, так что опыта в написании языков и сред программирования ему не занимать. Как отмечал сам Андреас Хейлсберг, C# создавался как язык компонентного программирования, и в этом одно из главных достоинств языка, дающее возможность повторного использования созданных компонентов. Создаваемые компилятором компоненты являются само-документируемыми, помимо программного кода содержат метаинформацию, описывающую компоненты, и поэтому могут выполняться на различных платформах.

Из других важных факторов отметим следующие:

- C# создавался и развивается параллельно с каркасом Framework .Net и в полной мере учитывает все его возможности;
- C# является полностью объектно-ориентированным языком;
- C# является мощным объектным языком с возможностями наследования и универсализации;
- C# является наследником языка C++. Общий синтаксис, общие операторы языка облегчают переход от языка C++ к C#;
- сохранив основные черты своего родителя, язык стал проще и надежнее;
- благодаря каркасу Framework .Net, ставшему надстройкой над операционной системой, программисты C# получают преимущества работы с виртуальной машиной;
- Framework .Net поддерживает разнообразие типов приложений на C#;
- реализация, сочетающая построение надежного и эффективного кода, является немаловажным фактором, способствующим успеху C#.

В каком направлении развивается язык C#? Назовем новинки, появившиеся в версии 3.0.

На первое место я бы поставил возможности создания качественно новых типов проектов на C#. Конечно, новые типы проектов нельзя отнести к новинкам языка C#. Эти возможности предоставляет каркас Framework .Net 3.5 и Visual Studio 2008. Но поскольку язык, среда разработки и каркас среды тесно связаны, то с точки зрения программистов, работающих на C#, возможности построения программных проектов на C# существенно расширились.

Введение в язык инструмента, получившего название LINQ (Language Integrated Query). Сегодня ни один серьезный проект на C# не обходится без обмена данными с внешними источниками данных – базами данных, Интернет и прочими хранилищами. В таких ситуациях приходилось использовать специальные объекты (ADO .Net или их более ранние версии). При работе с ними нужно было применять SQL – специальный язык запросов. Благодаря LINQ язык запросов становится частью языка программирования C#. Тем самым реализована давняя мечта программистов – работать с данными, находящимися в различных внешних источниках, используя средства, принадлежащие языку программирования, не привлекая дополнительные инструментальные средства и языки.

Введение в язык инструментария, характерного для функционального стиля программирования, – лямбда-выражений, анонимных типов и функций. Андреас Хейлсберг полагает, что смесь императивного и функционального стилей программирования упрощает задачи разработчиков, поскольку функциональный стиль позволяет разработчику сказать, что нужно делать, не уточняя, как это должно делаться.

Новые возможности появились при реализации параллелизма в программных проектах.

Будущее C#

Следующая версия языка C# 4.0 должна появиться параллельно с выходом новой версии Visual Studio 2010. Продолжается работа над версией C# 5.0. Можно отметить три основные тенденции в развитии языка – декларативность, динамичность и параллельность. Разработчики пытаются придать языку C# свойства, рас-

ширяющие традиционные возможности процедурных языков. Явно заметен тренд к функциональным языкам с их декларативным стилем. Такие свойства появились уже в C# 3.0, в следующих версиях они только расширяются.

В новой версии Visual Studio 2010 должны появиться новые динамические языки программирования: «железный змей» – Iron Python и Iron Ruby. Эти языки проще устроены, во многом из-за того, что не являются строго типизированными и потому не позволяют проводить контроль типов еще на этапе компиляции. В C# 4.0 введена возможность задания динамических переменных, аналогично тому, как это делается в динамических языках.

Параллельные вычисления в ближайшие 5-10 лет станут реальностью повседневной работы программиста. В этом направлении развивается техника. Языки программирования должны поддерживать эту тенденцию.

Компилятор как сервис, программирование на лету, – такие возможности должны появиться в C# 5.0. Можно не сомневаться, что C#-программистам в ближайшие годы скучать не придется.

Visual Studio 2008

Как уже отмечалось, принципиальной новинкой этой версии является возможность построения новых типов программных проектов, что обеспечивается новой версией каркаса Framework .Net 3.5. Если не считать этой важной особенности, то идейно Visual Studio 2008 подобна предыдущим версиям Visual Studio 2005 и Visual Studio 2003.

Рассмотрим основные особенности среды разработки Visual Studio.

Открытость

Среда разработки программных проектов является открытой языковой средой. Это означает, что наряду с языками программирования, включенными в среду фирмой Microsoft – **Visual C++ .Net** (с управляемыми расширениями), **Visual C# .Net**, **Visual Basic .Net**, – в среду могут добавляться любые языки программирования, компиляторы которых создаются другими фирмами.

Таких расширений среды **Visual Studio** сделано уже достаточно много, практически они существуют для всех известных языков – **Fortran** и **Cobol**, **RPG** и **Component Pascal**, **Eiffel, Oberon** и **Smalltalk**.

Новостью является то, что Microsoft не включила в Visual Studio 2008 поддержку языка Java. Допустимые в предыдущих версиях проекты на языке J++ в Visual Studio 2008 в настоящее время создавать нельзя, ранее созданные проекты в студии не открываются.

Открытость среды не означает полной свободы. Все разработчики компиляторов при включении нового языка в среду разработки должны следовать определенным ограничениям. Главное ограничение, которое можно считать и главным достоинством, состоит в том, что все языки, включаемые в среду разработки Visual Studio .Net, должны использовать единый каркас – Framework .Net. Благодаря этому достигаются многие желательные свойства: легкость использования компонентов, разработанных на различных языках; возможность разработки нескольких частей одного приложения на разных языках; возможность бесшовной отладки

такого приложения; возможность написать класс на одном языке, а его потомков – на других языках. Единый каркас приводит к сближению языков программирования, позволяя вместе с тем сохранять их индивидуальность и имеющиеся у них достоинства. Преодоление языкового барьера – одна из важнейших задач современного мира. Visual Studio .Net, благодаря единому каркасу, в определенной мере решает эту задачу в мире программистов.

Framework .Net – единый каркас среды разработки приложений

В каркасе Framework .Net можно выделить два основных компонента:

- статический – FCL (Framework Class Library) – библиотеку классов каркаса;
- динамический – CLR (Common Language Runtime) – общеязыковую исполнительную среду.

Библиотека классов FCL – статический компонент каркаса

Понятие каркаса приложений – Framework Applications – появилось достаточно давно, оно широко использовалось еще в четвертой версии Visual Studio. Библиотека классов MFC (Microsoft Foundation Classes) играла роль каркаса приложений Visual C++.

Несмотря на то, что каркас был представлен только статическим компонентом, уже тогда была очевидна его роль в построении приложений. Уже в то время важнейшее значение в библиотеке классов MFC имели классы, задающие архитектуру строящихся приложений. Когда разработчик выбирал один из возможных типов приложения, например, архитектуру Document-View, то в его приложение автоматически встраивались класс Document, задающий структуру документа, и класс View, задающий его визуальное представление. Класс Form и классы, задающие элементы управления, обеспечивали единый интерфейс приложений. Выбирая тип приложения, разработчик изначально получал нужную ему функциональность, поддерживаемую классами каркаса. Библиотека классов поддерживала и традиционные для программистов классы, задающие расширенную систему типов данных, в частности, динамические типы данных – списки, деревья, коллекции, шаблоны.

За прошедшие годы роль каркаса в построении приложений существенно возросла – прежде всего, за счет появления его динамического компонента, о котором чуть позже поговорим подробнее. Что же касается статического компонента – библиотеки классов, то здесь появился ряд важных нововведений.

Единство каркаса

Каркас стал единым для всех языков среды разработки. Поэтому на каком бы языке программирования не велась разработка, она работает с классами одной и той же библиотеки. Многие классы библиотеки, составляющие общее ядро, используются всеми языками. Отсюда единство интерфейса приложения, на каком бы языке оно не разрабатывалось, единство работы с коллекциями и другими контейнерами данных, единство связывания с различными хранилищами данных и прочая универсальность.

Встроенные примитивные типы

Важной частью библиотеки FCL стали классы, задающие примитивные типы – те типы, которые считаются встроенными в язык программирования. Типы каркаса покрывают основное множество встроенных типов, встречающихся в языках программирования. Типы языка программирования проецируются на соответствующие типы каркаса. Тип, называемый в языке Visual Basic – Integer, а в языках C++ и C# – int, проецируется на один и тот же тип каркаса – System.Int32. В языке программирования, наряду с «родными» для языка названиями типов, разрешается пользоваться именами типов, принятыми в каркасе. Поэтому, по сути, все языки среды разработки могут пользоваться единой системой встроенных типов, что, конечно, способствует облегчению взаимодействия компонентов, написанных на разных языках.

Структурные типы

Частью библиотеки стали не только простые встроенные типы, но и структурные типы, задающие организацию данных – строки, массивы; динамические типы данных – стеки, очереди, списки, деревья. Это также способствует унификации и реальному сближению языков программирования.

Архитектура приложений

Существенно расширился набор возможных архитектурных типов построения приложений. Помимо традиционных Windows- и консольных приложений, появилась возможность построения Web-приложений. Большое внимание уделяется возможности создания повторно используемых компонентов – разрешается строить библиотеки классов, библиотеки элементов управления и библиотеки Web-элементов управления. Популярным архитектурным типом являются Web-службы, ставшие сегодня благодаря открытому стандарту одним из основных видов повторно используемых компонентов.

Модульность

Число классов библиотеки FCL велико (несколько тысяч), поэтому понадобился способ их структуризации. Логически классы с близкой функциональностью объединяются в группы, называемые пространством имен (Namespace). Основным пространством имен библиотеки FCL является пространство System, содержащее как классы, так и другие вложенные пространства имен. Так, уже упоминавшийся примитивный тип Int32 непосредственно вложен в пространство имен System и его полное имя, включающее имя пространства, – System.Int32.

В пространство System вложен целый ряд других пространств имен. Например, в пространстве System.Collections находятся классы и интерфейсы, поддерживающие работу с коллекциями объектов – списками, очередями, словарями. В пространство System.Collections, в свою очередь, вложено пространство имен Specialized, содержащее классы со специализацией, например, коллекции, элементами которых являются только строки. Пространство System.Windows.Forms

содержит классы, используемые при создании Windows-приложений. Класс Form из этого пространства задает форму – окно, заполняемое элементами управления, графикой, обеспечивающее интерактивное взаимодействие с пользователем.

По ходу курса мы будем знакомиться со многими классами библиотеки FCL.

Общеязыковая исполнительная среда CLR – динамический компонент каркаса

Важным шагом в развитии каркаса Framework .Net стало введение динамического компонента каркаса – **исполнительной среды CLR**. С появлением CLR процесс выполнения приложений стал принципиально другим.

Двухэтапная компиляция. Управляемый модуль и управляемый код

Компиляторы языков программирования, включенные в Visual Studio .Net, создают код на промежуточном языке **IL (Intermediate Language) –** ассемблерном языке**.** В результате компиляции проекта, содержащего несколько файлов, создается так называемый **управляемый модуль** – переносимый исполняемый файл (Portable Executable или PE-файл). Этот файл содержит код на IL и метаданные – всю информацию, необходимую для CLR, чтобы под ее управлением PE-файл мог быть исполнен. Метаданные доступны и конечным пользователям. Классы, входящие в пространство имен Reflection, позволяют извлекать метаинформацию о классах, используемых в проекте. Этот процесс называется отражением. Об атрибутах классов, отображаемых в метаданные PE-файла, мы еще будем говорить неоднократно. В зависимости от выбранного типа проекта, PE-файл может иметь разные уточнения – exe, dll, mod или mdl.

Заметьте, PE-файл, имеющий уточнение exe, хотя и является exe-файлом, но это не обычный исполняемый Windows файл. При его запуске он распознается как PE-файл и передается CLR для обработки. Исполнительная среда начинает работать с кодом, в котором специфика исходного языка программирования исчезла. Код на IL начинает выполняться под управлением CLR (по этой причине **код** называется **управляемым**). Исполнительную среду следует рассматривать как виртуальную IL-машину. Эта машина транслирует «на лету» требуемые для исполнения участки кода в команды реального процессора, который в действительности и выполняет код.

Виртуальная машина

Отделение каркаса от студии явилось естественным шагом. Каркас Framework .Net перестал быть частью студии и стал надстройкой над операционной системой. Теперь компиляция и создание PE модулей на IL отделено от выполнения, и эти процессы могут быть реализованы на разных платформах.

В состав CLR входят трансляторы JIT (Just In Time Compiler), которые и выполняют трансляцию IL в командный код той машины, где установлена и функционирует исполнительная среда CLR. Конечно, в первую очередь Microsoft реализовала CLR и FCL для различных версий Windows, включая Windows 98/Me/

NT 4/2000, 32 и 64-разрядные версии Windows XP, Windows Vista и семейство .Net Server. Облегченная версия Framework .Net разработана для операционных систем Windows CE и Palm.

Framework .Net развивается параллельно с развитием языков программирования, среды разработки программных проектов и операционных языков. Версия языка C# 2.0 использовала версию Framework .Net 2.0. Операционная система Windows Vista включила в качестве надстройки Framework .Net 3.0. Язык C# 3.0 и Visual Studio 2008 работают с версией Framework .Net 3.5.

Framework .Net является свободно распространяемой виртуальной машиной. Это существенно расширяет сферу его применения. Производители различных компиляторов и сред разработки программных продуктов предпочитают теперь также транслировать свой код в IL, создавая модули в соответствии со спецификациями CLR. Это обеспечивает возможность выполнения их кода на разных платформах.

Компилятор JIT, входящий в состав CLR, компилирует IL код с учетом особенностей текущей платформы. Благодаря этому создаются высокопроизводительные приложения. Следует отметить, что CLR, работая с IL кодом, выполняет достаточно эффективную оптимизацию и, что не менее важно, защиту кода. Зачастую нецелесообразно выполнять оптимизацию на уровне создания IL кода, она иногда может не улучшить, а ухудшить ситуацию, не давая CLR провести оптимизацию на нижнем уровне, где можно учесть особенности процессора.

Дизассемблер и ассемблер

Для проекта, построенного на C#, иногда полезно провести анализ построенного PE-файла, его IL кода и связанных с ним метаданных. В состав Framework SDK входит **дизассемблер** – ildasm, выполняющий дизассемблирование PE-файла и показывающий в наглядной форме метаданные и IL код с комментариями. Мы иногда будем пользоваться результатами дизассемблирования. У меня на компьютере кнопка, вызывающая дизассемблер, находится на рабочем столе. Вот путь к папке, в которой обычно находится дизассемблер:

C:\Program Files\Microsoft Visual Studio .Net\FrameworkSDK\Bin\ildasm.exe

Профессионалы, предпочитающие работать на низком уровне, могут программировать на языке ассемблера IL. В этом случае в их распоряжении будет вся мощь библиотеки FCL и все возможности CLR. У меня на компьютере путь к папке, где находится **ассемблер**, следующий:

C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\ilasm.exe

В этой книге к ассемблеру мы обращаться не будем, упоминаю о нем для полноты картины.

Метаданные

Переносимый исполняемый PE-файл является самодокументируемым файлом и, как уже говорилось, содержит код и **метаданные**, описывающие код. Файл начинается с манифеста и включает в себя описание всех классов, хранимых в PE-файле, их свойств, методов, всех аргументов этих методов – всю необходимую CLR информацию. Поэтому помимо PE-файла не требуется никаких дополни-

тельных файлов, записей в реестр, вся нужная информация извлекается из самого файла. Среди классов библиотеки FCL имеется класс **Reflection**, методы которого позволяют извлекать необходимую информацию. Введение метаданных – не только важная техническая часть CLR, но также часть новой идеологии разработки программных продуктов. Мы увидим, что и на уровне языка C# самодокументированию уделяется большое внимание.

Мы увидим также, что при проектировании класса программист может создавать его атрибуты, добавляемые к метаданным PE-файла. Клиенты класса могут, используя класс **Reflection**, получать эту дополнительную информацию и на ее основании принимать соответствующие решения.

Сборщик мусора – Garbage Collector и управление памятью

Еще одной важной особенностью построения CLR является то, что исполнительная среда берет на себя часть функций, традиционно входящих в ведение разработчиков трансляторов, и облегчает тем самым их работу. Один из таких наиболее значимых компонентов CLR – **сборщик мусора (Garbage Collector)**. Под сборкой мусора понимается освобождение памяти, занятой объектами, которые стали бесполезными и не используются в дальнейшей работе приложения. В ряде языков программирования (классическим примером является язык C/C++) память освобождает сам программист, в явной форме отдавая команды, как на создание, так и на удаление объекта. В этом есть своя логика – «я тебя породил, я тебя и убью». Однако можно и нужно освободить человека от этой работы. Неизбежные ошибки программиста при работе с памятью тяжелы по последствиям, и их крайне тяжело обнаружить. Как правило, объект удаляется в одном модуле, а необходимость в нем обнаруживается в другом далеком модуле. Обоснование того, что программист не должен заниматься удалением объектов, а сборка мусора должна стать частью исполнительной среды, дано достаточно давно. Наиболее полно оно обосновано в работах Бертрана Мейера и в его книге «Object-Oriented Construction Software», первое издание которой появилось еще в 1988 году.

В CLR эта идея реализована в полной мере. Задача сборки мусора снята не только с программистов, но и с разработчиков трансляторов; она решается в нужное время и в нужном месте – исполнительной средой, ответственной за выполнение вычислений. Здесь же решаются и многие другие вопросы, связанные с использованием памяти, в частности, проверяется и не допускается использование «чужой» памяти, не допускаются и другие нарушения. Данные, удовлетворяющие требованиям CLR и допускающие сборку мусора, называются **управляемыми данными**.

Но как же, спросите вы, быть с языком C++ и другими языками, где есть нетипизированные указатели, адресная арифметика, возможности удаления объектов программистом? Ответ следующий – CLR позволяет работать как с управляемыми, так и с **неуправляемыми данными**. Однако использование неуправляемых данных регламентируется и не поощряется. Так, в C# модуль, использующий неуправляемые данные (указатели, адресную арифметику), должен быть помечен как небезопасный (unsafe), и эти данные должны быть четко зафиксированы. Об этом мы еще будем говорить при рассмотрении языка C# в последующих лекциях.

Исполнительная среда, не ограничивая возможности языка и программистов, вводит определенную дисциплину в применении потенциально опасных средств языков программирования.

Исключительные ситуации

Что происходит, когда при вызове некоторой функции (процедуры) обнаруживается, что она не может нормальным образом выполнить свою работу? Возможны разные варианты обработки такой ситуации. Функция может возвращать код ошибки или специальное значение типа HResult, может **выбрасывать исключение**, тип которого характеризует возникшую ошибку. В CLR принято во всех таких ситуациях выбрасывать исключение. Косвенно это влияет и на язык программирования. Выбрасывание исключений наилучшим образом согласуется с исполнительной средой. В языке C# выбрасывание исключений, их дальнейший перехват и обработка – основной рекомендуемый способ обработки **исключительных ситуаций**.

События

У CLR есть свое видение того, что представляет собой тип. Есть формальное описание **общей системы типов CTS** – Common Type System. В соответствии с этим описанием каждый тип, помимо полей, методов и свойств, может содержать и **события**. При возникновении событий в процессе работы с тем или иным объектом данного типа посылаются сообщения, которые могут получать другие объекты. Механизм обмена сообщениями основан на **делегатах** – функциональном типе. Надо ли говорить, что в язык C# встроен механизм событий, полностью согласованный с возможностями CLR. Мы подробно изучим все эти механизмы, рассматривая их на уровне языка.

Исполнительная среда CLR обладает мощными динамическими механизмами – сборки мусора, динамического связывания, обработки исключительных ситуаций и событий. Все эти механизмы и их реализация в CLR написаны на основании практики существующих языков программирования. Но уже созданная исполнительная среда в свою очередь влияет на языки, ориентированные на использование CLR. Поскольку язык C# создавался одновременно с созданием CLR, то, естественно, он стал языком, наиболее согласованным с исполнительной средой, и средства языка напрямую отображаются в средства исполнительной среды.

Общие спецификации и совместимые модули

Уже говорилось, что каркас Framework .Net облегчает межъязыковое взаимодействие. Для того чтобы классы, разработанные на разных языках, мирно уживались в рамках одного приложения, для их бесшовной отладки и возможности построения разноязычных потомков, они должны удовлетворять некоторым ограничениям. Эти ограничения задаются **набором общеязыковых спецификаций – CLS** (Common Language Specification). Класс, удовлетворяющий спецификациям CLS, называется **CLS-совместимым**. Он доступен для использования в других языках, классы которых могут быть клиентами или наследниками совместимого класса.

Спецификации CLS точно определяют, каким набором встроенных типов можно пользоваться в **совместимых модулях**. Понятно, что эти типы должны быть общедоступными для всех языков, использующих Framework .Net. В совместимых модулях должны использоваться управляемые данные и выполняться некоторые другие ограничения. Заметьте, ограничения касаются только интерфейсной части класса, его открытых свойств и методов. Закрытая часть класса может и не удовлетворять CLS. Классы, от которых не требуется совместимость, могут использовать специфические особенности языка программирования.

Framework .Net 3.5

Рассмотрим новинки, появившиеся в последней версии Framework .Net 3.5. Прежде всего заметим, что практически все новинки языка C# 3.0 поддержаны нововведениями в Framework .Net 3.5.

LINQ и деревья выражений

Уже говорилось, что в C# 3.0 встроен язык запросов к данным, что существенно облегчает работу с данными, поступающими из внешних источников. Этот языковый механизм поддерживается классами библиотеки FCL Framework .Net 3.5. Пространство System.Linq содержит классы, задающие типы, интерфейсы, стандартные операторы запроса. Пространства System.Data.Linq, System.Data.Linq.Mapping поддерживают работу с реляционными базами данных. Классы пространства System.XML.Linq поддерживают запросы к XML- данным. Новые классы DataRowComparer, DataRowExtensions, DataTableExtensions позволяют локально хранить данные, заменяя объекты DataSet ADO .Net. Классы из пространства System.Linq.Expressions позволяют работать с деревьями выражений, используемых в запросах.

Подробнее эти классы будут рассмотрены в соответствующей главе, посвященной работе с данными и инструментарием Linq.

Windows Presentation Foundation

В Visual Studio 2008 появились новые типы проектов, основанные на возможностях, предоставляемых технологией WPF (Windows Presentation Foundation). Эта технология позволяет строить новое поколение систем презентации – с новыми графическими возможностями, связыванием данных и прочими элементами, придающими приложению принципиально новые свойства. Предполагается, что этот тип приложений постепенно будет вытеснять традиционные Windows-приложения, основанные на понятии окна.

Windows Communication Foundation (WCF) и Windows Workflow Foundation (WF)

Технологии WCF и WF позволяют строить специализированные приложения и службы (Services), позволяющие приложениям обмениваться данными, используя асинхронный ввод-вывод.

ASP .NET

Новые возможности Framework .Net 3.5 облегчают разработку Веб- приложений, в частности, построение сайтов с AJAX (Asynchronous Javascript and XML) – свойствами. Приложения с такими свойствами становятся более быстрыми и удобными, позволяя при взаимодействии с сервером не перезагружать всю страницу полностью.

Другие новинки

Трудно, да и не имеет особого смысла перечислять все нововведения, появившиеся в Framework .Net 3.5. При обсуждении новых возможностей построения приложений на языке C#, несомненно, речь будет идти и о том, как эти возможности поддерживаются в CLR и FCL.

Управляемый и неуправляемый код

Как уже отмечалось, результатом проекта, написанного на C# и скомпилированного в Visual Studio 2008, является сборка (assembly), которая содержит IL-код проекта и манифест, полностью описывающий сборку. Сборка может быть создана на одном компьютере, на одной платформе, а выполняться на другом компьютере с другим типом процессора, с другой операционной системой. Для выполнения сборки необходимо и достаточно установки на целевом компьютере соответствующей версии Framework .Net, представляющего надстройку над операционной системой.

Когда мы говорим о сборках, язык программирования, на котором создавался исходный код, уже не имеет значения, его особенности никак не отражаются в сборке. Сборки, созданные на VB или C++ с управляемыми расширениями, неотличимы от сборок, которые созданы на C# или других языках, включенных в состав Visual Studio 2008 и использующих каркас Framework .Net при компиляции управляемого кода.

С другой стороны, понятно, что в состав Visual Studio 2008 могут включаться языки, не применяющие Framework .Net, не создающие сборки с управляемым кодом, а использующие собственные библиотеки и собственные каркасы приложений (Framework Applications). В частности, на языке C++ в рамках Visual Studio 2008 можно писать проекты, работающие с библиотеками MFC и ATL, ориентированные исключительно на C++ и создающие в результате компиляции проекта обычные exe-файлы.

Сегодня на всех компьютерах, работающих под управлением любой из версий Windows, установлена соответствующая версия Framework .Net, так что на таких компьютерах могут выполняться и сборки, и обычные exe-файлы. Поскольку Framework .Net, так же как и C#, стандартизован и является свободно распространяемым программным продуктом, его можно встретить и на тех компьютерах, где нет Windows.

На рис. 1.1 показана схема функционирования компьютера, позволяющего выполнять как сборки – управляемый код, так и обычные exe-файлы – неуправляемый код.

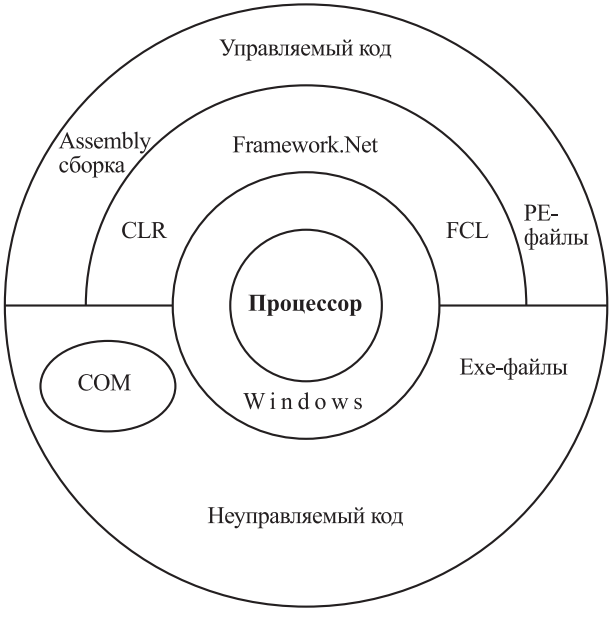

Рис. 1.1. Управляемый и неуправляемый код

Заметьте: два мира программ, выполняемые по-разному, могут взаимодействовать друг с другом – из управляемого кода возможен вызов программ с неуправляемым кодом и наоборот. В проектах, написанных на C#, можно управлять офисными приложениями – документами Word и Excel. Офисные документы – это COM-объекты, принадлежащие миру неуправляемого кода, а проекты C# – это сборки, жители страны с управляемым кодом.

Проекты C# в Visual Studio 2008

При запуске Visual Studio 2008, которая, надеюсь, уже установлена на Вашем компьютере, открывается стартовая страница. В окне «Recent Projects» стартовой страницы есть две скромные, непрезентабельного вида ссылки – «Open Project...» и «Create Project...».

Они задают две основные функции, которые может выполнять разработчик в Visual Studio 2008, – он может открыть существующий проект и работать с ним или создать и работать с новым проектом. В большинстве случаев после открытия стартовой страницы щелчком по одной из ссылок мы переходим к созданию или открытию проекта. Вид стартовой страницы показан на рис. 1.2.

Стартовая страница, помимо перехода к выполнению основных задач разработчика, предоставляет важные дополнительные возможности. Во-первых, здесь расположен список текущих проектов, позволяющий сразу же перейти к работе с нужным проектом из этого списка.

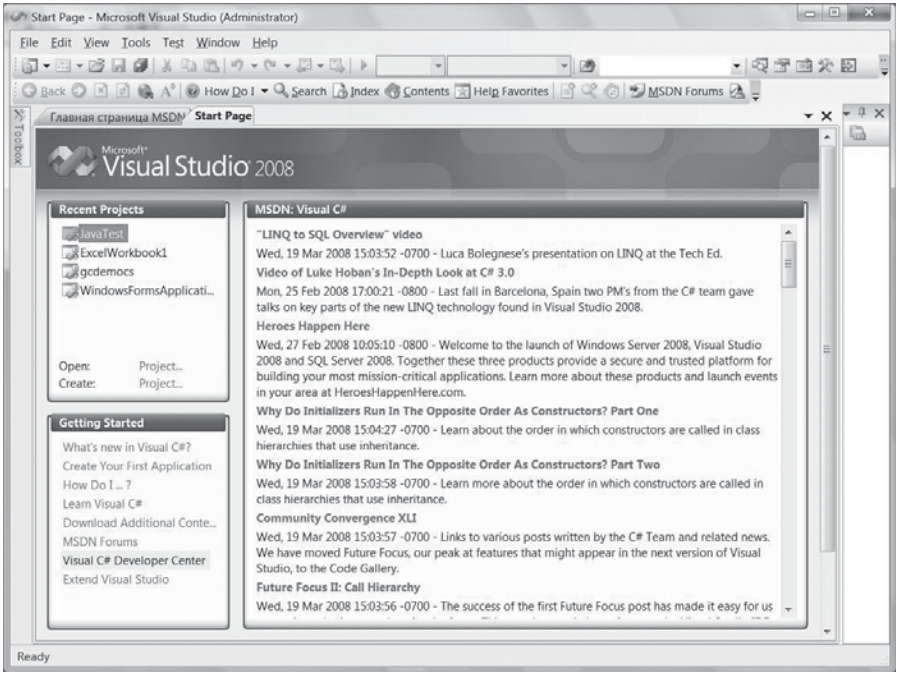

Рис. 1.2. Вид стартовой страницы

Для компьютера, подключенного к интернет, стартовая страница автоматически связывается с сайтом, содержащим текущую информацию по C# и Visual Studio 2008, – по умолчанию показываются новости с сайта msdn. Выбрав соответствующий пункт из раздела Getting Started (Давайте Начнем), можно получить информацию из центра разработчиков C#, можно подключиться к одному из трех форумов по языку C#, можно получить нужную справку в режиме «on line».

На стартовой странице, помимо вкладки «StartPage», расположена вкладка «Главная страница MSDN», позволяющая перейти к соответствующему сайту. На рис. 1.3 показана страница, открытая при выборе этой вкладки.

Коль скоро речь зашла о получении текущей информации и справок по языку C#, упомяну несколько полезных ресурсов:

http://forums.msdn.microsoft.com/en-us/forums/ – англоязычный сайт предоставляет доступ к различным форумам, в том числе форумам по языку C#;

http://msdn.microsoft.com/ru-ru/default.aspx – русскоязычный сайт msdn;

http://msdn.microsoft.com/en-us/vcsharp/default.aspx – англоязычный сайт по языку C# на msdn;

http://csharpfriends.com/ – англоязычный сайт, где можно найти нужную информацию, задать вопросы и получить ответы от сообщества разработчиков.