Введение

Если мы скажем, что язык C# и связанная с ним среда .NET Framework является одной из самых важных технологий для разработчиков за много лет, это не будет преувеличением. .NET спроектирована как новая среда, в рамках которой можно разработать практически любое приложение для Windows, в то время как C# — новый язык программирования, предназначенный специально для работы с .NET. С помощью C# вы можете, например, создавать динамическую Web-страницу, Web-службу XML, компонент распределенного приложения, компонент доступа к базе данных, классическое Windows-приложение рабочего стола или даже интеллектуальное клиентское приложение, имеющее возможность онлайновой и автономной работы. Настоящая книга посвящена .NET Framework 2.0 и новому ее выпуску — .NET Framework 3.0. Если вы имеете опыт кодирования в версиях 1.0, 1.1 или 2.0, то, возможно, встретите разделы, в которых не найдете для себя ничего нового. Мы старались специально отмечать те элементы, которые впервые появились в .NET Framework 3.0.

Пусть вас не смущает марка ".NET". Часть "NET" в имени подчеркивает уверенность Microsoft в том, что распределенные приложения, в которых обработка распределена между клиентом и сервером — это движение вперед, но C# — не только язык для написания Internet-приложений или любых сетевых приложений. Он предоставляет средства для кодирования программного обеспечения практически любого типа или компонентов, которые может понадобиться написать для платформы Windows. Между тем, C# и .NET призваны перевернуть способ написания программ, и сделать программирование для Windows намного легче, чем оно когда-либо было.

Это довольно самонадеянное утверждение, и оно требует доказательств. В конце концов, мы все знаем, насколько быстро меняются компьютерные технологии. Каждый год Microsoft выпускает новое программное обеспечение, инструменты программирования, версии Windows, при этом утверждая, что они в высшей степени хороши для разработчиков. Так в чем же отличие .NET и C#?

Значение .NET и C#

Чтобы понять истинное значение .NET, полезно вспомнить природу многих технологий Windows, которые появились за последние десять лет или около того. Хотя внешне они могут показаться совсем непохожими, но все операционные системы, начиная с Windows 3.1 (представленной в 1992 году), и до Windows Server 2003, имеют в своем ядре один и тот же программный интерфейс Windows API. С каждой новой версией Windows к этому API добавляется огромное число новых функций, но все это — процесс эволюции и расширения API, а не замена его.

То же самое можно сказать о многих новых технологиях и средах (каркасах), которые мы использовали для разработки программного обеспечения Windows. Например, о модели *COM* (Component Object Model — объектная модель компонентов), изначально возникшей как *OLE* (Object Linking and Embedding — связывание и внедрение объектов). Во время ее появления она была просто средством, с помощью которого можно было связывать разные типы документов Office, так что, например, вы могли

поместить небольшую таблицу Excel в документ Word. Затем она эволюционировала до COM, *DCOM* (Distributed COM — распределенная COM) и в конечном итоге появилась COM+ — развитая технология, формирующая базу для взаимодействия почти всех компонентов, также реализующая транзакции, службы сообщений и пулы объектов.

Microsoft выбрала этот эволюционный подход к программному обеспечению по очевидной причине — заботясь об обратной совместимости. За многие годы независимыми разработчиками для Windows был разработан гигантский объем программного обеспечения, и Windows не достигла бы такого успеха, если бы всякий раз, когда Microsoft представляла новую технологию, существующая кодовая база переставала бы работать!

В то время как обратная совместимость всегда была важнейшим свойством технологий Windows и одной из сильных сторон платформы Windows, она также имела большой недостаток. Всякий раз, когда технология развивалась и добавлялись новые средства, это приводило к усложнению по сравнению с тем, что было раньше.

Стало ясно, что нужно что-то менять. Microsoft не могла до бесконечности расширять существующие средства разработки и языки, постоянно все более усложняя их, чтобы удовлетворить противоречивые требования поддержки нового оборудования и обратной совместимости с тем, что было во времена, когда Windows впервые стала популярной — в начале 90-х прошлого века. Наступил момент, когда нужно было начать с чистого листа, имея простой, но достаточный набор языков, сред и средств разработки, которые позволили бы разработчиками легко писать полезное и работоспособное программное обеспечение.

Таким "началом с нуля" стали C# и .NET. Грубо говоря, .NET — это каркас, API-интерфейс для программирования на платформе Windows. Вместе с .NET Framework, C# представляет собой язык, спроектированный "с нуля", предназначенный для работы с .NET и впитавший в себя все достижения прогресса сред разработки и понимание принципов объектно-ориентированного программирования, которое выкристаллизовалось за последние 20 лет.

Прежде чем продолжить, следует пояснить, что обратная совместимость вовсе не была утеряна. Существующие программы продолжают работать, и каркас .NET спроектирован так, что способен работать с этим существующим программным обеспечением. В настоящее время коммуникации между программными компонентами Windows почти целиком строятся с использованием COM. Принимая это во внимание, .NET предоставляет оболочки вокруг существующих компонентов COM, так что компоненты .NET могут свободно общаться с ними.

Правда, вы не обязаны изучать C#, чтобы писать код для .NET. Microsoft предлагает расширенный C++, еще один новый язык, называемый J#, а также вносит существенные изменения в Visual Basic, превратив его в более мощный язык — Visual Basic .NET, чтобы позволить на любом из этих языков разрабатывать приложения, ориентированные на среду .NET. Однако все эти языки скорее представляют собой результат эволюции в течение ряда лет, и не разработаны изначально с учетом требований современных технологий.

Эта книга научит вас программировать на C#, одновременно закладывая необходимый фундамент знаний о работе архитектуры .NET. Мы не только раскроем основы языка C#, но также представим примеры приложений, использующих широкий диапазон взаимосвязанных технологий, включая доступ к базам данных, динамические Web-страницы, расширенную графику и доступ к каталогам. Единственное требование — чтобы вы были знакомы хотя бы с одним из других высокоуровневых языков программирования, используемых в Windows — C++, Visual Basic или J++.

Преимущества .NET

Мы уже говорили в общих чертах о том, насколько замечателен .NET, но пока не сказали ничего о том, как он облегчает жизнь вам как разработчику. В этом разделе мы кратко опишем некоторые усовершенствованные средства .NET.

- **Объектно-ориентированное программирование**. И среда .NET Framework, и C# изначально полностью базировались на объектно-ориентированных принципах.
- **Хороший дизайн**. Библиотека базовых классов, которая спроектирована "с нуля", исключительно интуитивно понятным образом.
- **Независимость от языка**. Благодаря .NET, код всех языков, то есть Visual Basic .NET, C#, J# и управляемого C++, компилируется в общий язык промежуточного уровня — *Intermediate Language*. Это значит, что все эти языки обладают возможностями взаимодействия, как никогда ранее.
- **Лучшая поддержка динамических Web-страниц**. Хотя ASP предлагал высокую степень гибкости, он также был и неэффективен из-за своих интерпретируемых сценарных языков, а недостаток объектно-ориентированного дизайна часто приводил к запутанному коду ASP. .NET предлагает интегрированную поддержку Web-страниц с применением новой технологии — ASP.NET. В ASP.NET код ваших страниц компилируется и может быть написан на языке высокого уровня, поддерживающего .NET — таком как C#, J# или Visual Basic 2005.
- **Эффективный доступ к данным**. Набор компонентов .NET, известный под общим названием ADO.NET, предоставляет эффективный доступ к реляционным базам данных и широкому разнообразию других источников данных. Также имеются компоненты, предоставляющие доступ к файловой системе и каталогам. В частности, в .NET встроена поддержка XML, что позволяет манипулировать данными, которые могут быть импортированы их других, не-Windows платформ, и экспортированы на них.
- **Разделение кода**. Среда .NET полностью изменила способ разделения кода между приложениями, введя концепцию *сборки* (assembly), которая заменила традиционные библиотеки DLL. Сборки имеют форматные средства для указания версий, и одновременно в системе могут существовать разные версии одних и тех же сборок.
- **Повышенная безопасность**. Каждая сборка также может содержать встроенную информацию безопасности, которая в точности описывает, кому и каким категориям пользователей или процессов какие методы каких классов разрешено вызывать. Это обеспечивает очень высокую степень контроля за тем, как могут использоваться сборки, которые вы поставляете.
- **Инсталляция с нулевым воздействием**. Существует два типа сборок: разделяемые и приватные. Разделяемые сборки — это обычные библиотеки, доступные всему программному обеспечению, в то время как приватные сборки предназначены для использования совершенно определенными программами. Приватные сборки полностью самодостаточны, поэтому процесс инсталляции прост. Нет никаких элементов реестра, нужные файлы просто помещаются в соответствующую папку файловой системы.

- **Поддержка Web-служб**. .NET предлагает полностью интегрированную поддержку разработки Web-служб — все так же просто, как и создание приложений любого другого типа.
- **Visual Studio 2005**. .NET поставляется со средой разработки Visual Studio 2005, которая одинаково хорошо справляется с языками C++, C#, J# и Visual Basic 2005, а также с ASP.NET. Visual Studio 2005 интегрирует в себе все лучшие средства соответствующих специфичных языковых сред Visual Studio .NET 2002/2003 и Visual Studio 6.
- **C#**. C# представляет собой новый объектно-ориентированный язык, предназначенный для применения с .NET.

Более подробно преимущества .NET рассматриваются в главе 1.

Что нового в .NET Framework 2.0

Первая версия .NET Framework (1.0) была реализована в 2002 году с огромным энтузиазмом. Следующая версия — .NET Framework 2.0 — была представлена в 2005 году и рассматривается как главный выпуск этой среды.

В каждом выпуске Microsoft всегда старается обеспечить сохранение работоспособности ранее написанного кода. До сих пор им удавалось успешно достигать этой цели.

Создайте установочный сервер, чтобы полностью протестировать обновление ваших приложений для .NET Framework 2.0 вместо простого обновления работающих приложений.

Ниже мы детализируем некоторые изменения, внесенные в .NET Framework, а также дополнения Visual Studio 2005 — среды разработки для .NET Framework 2.0.

Интеграция с SQL Server

После долгого перерыва наконец-то вышла последняя версия SQL Server. Эта версия, SQL Server 2005, во многих отношениях особенная. Для разработчика .NET важнее всего то, что .NET Framework 2.0, Visual Studio 2005 и SQL Server 2005 теперь тесно между собой связаны — в том смысле, что реализованы в сочетании. Это достаточно важно, поскольку известно, что большинство приложений строятся с применением этих всех трех частей, и что они должны обновляться совместно, чтобы работать друг с другом в прозрачной согласованной манере.

Благодаря тому факту, что теперь сам SQL Server 2005 служит хостом для CLR (Common Language Runtime — общеязыковая исполняющая среда), можно избежать построения аспектов ваших приложений, связанных с базами данных, с применением языка T-SQL. Вместо этого можно строить такие элементы, как хранимые процедуры, триггеры и даже типы данных, на любом совместимом с .NET языке, например, на C#.

SQL Server Express — это версия SQL Server 2005 года, которая заменила MSDE. Эта версия лишена тех строгих ограничений, которые были присущи MSDE.

Поддержка 64-разрядных вычислений

Большая часть программирования в наши дни выполняется на 32-разрядных машинах. Это было огромным шагом вперед в разработке приложений, когда компьютеры из 16-разрядных превратились в 32-разрядные. Но все больше и больше предприятий переходят на самые современные 64-разрядные серверы от таких компаний, как Intel

(с процессорами Itanium) и AMD (с процессорами x64), и .NET Framework 2.0 теперь включает 64-разрядную поддержку такой миграции.

Специалисты в Microsoft интенсивно трудились над тем, чтобы все, что вы строите в 32-разрядном мире .NET, работало в мире 64-разрядном. Это значит, что на все, что вы делаете с SQL Server 2005 или ASP.NET, переход на 64-разрядные машины никак не повлияет. Было внесено множество изменений в CLR, чтобы заставить работать 64-разрядную версию .NET. Изменения коснулись таких вещей, как сборка мусора (чтобы обрабатывать большие объемы данных), процесс компиляции JIT, обработка исключений и многое другое.

Переход на 64-разрядную архитектуру предлагает некоторые значительные преимущества. Наиболее важно (и это — самая очевидная причина), что 64-разрядные серверы поддерживают большее адресное пространство. Переход на 64 разряда также позволяет сделать примитивные типы большими, чем раньше. Например, целое значение 2^{32} дает 4 294 967 296, в то время как 2^{64} — уже 18 446 744 073 709 551 616. Это очень удобно для тех приложений, которым нужно вычислять вещи вроде государственного долга США или другие большие числа.

Такие компании, как Microsoft и IBM, подталкивают своих клиентов обратить внимание на 64-разрядные системы. Одними из главных областей применения являются средства баз данных и средства виртуального хранения — именно здесь переход на 64-разрядную архитектуру наиболее оправдан.

Visual Studio 2005 можно инсталлировать и запустить на 64-разрядном компьютере. IDE-среда Visual Studio 2005 включает в себя и 32-разрядный, и 64-разрядный компиляторы. Одно предупреждение: 64-разрядный .NET Framework предназначен только для Windows Server 2003 SP1 или выше либо для других 64-разрядных операционных систем Microsoft, которые могут появиться в будущем.

Когда вы строите свое приложение на Visual Studio 2005, то можете изменять свойства проекта так, чтобы оно было скомпилировано специально для 64-разрядных компьютеров. Чтобы найти эти настройки, следует обратиться открыть окно свойств приложения и перейти в нем на вкладку Build (Построение). На вкладке Build щелкните на кнопке Advanced (Дополнительно), после чего появится диалоговое окно Advanced Compiler Setting (Дополнительные настройки компилятора). В нижней части этого окна можно изменить тип целевого процессора. Здесь вы можете указать, чтобы ваше приложение было построено либо для 64-разрядного компьютера Intel, либо 64-разрядного компьютера AMD.

Обобщения

Чтобы сделать коллекции более мощным средством и также повысить их эффективность и удобство, в .NET Framework 2.0 были представлены обобщения (generics). Это нововведение означает, что такие языки, как C# и Visual Basic 2005, теперь могут строить приложения, использующие обобщенные типы. Идея обобщений не нова. Они выглядят подобно шаблонам C++, хотя несколько отличаются от них. Обобщения можно найти и в других языках, например, в Java. Их появление в языках .NET Framework 2.0 выглядит как ценный подарок пользователям.

Обобщения позволяют создавать обобщенные коллекции, которые, тем не менее, строго типизированы — что оставляет меньше шансов допустить ошибку (потому что они выявляются во время компиляции), повышает производительность и предоставляет в ваше распоряжение средство IntelliSense (автоматическое завершение конструкций кода), когда вы работаете с коллекциями.

Чтобы применить обобщения в коде, необходимо обратиться к пространству имен `System.Collections.Generic`. Это откроет доступ к обобщенным версиям классов `Stack`, `Dictionary`, `List` и `Queue`.

Рассмотрим пример, демонстрирующий использование обобщенной версии класса `Stack`:

```
void Page_Load(object sender, EventArgs e)
{
  System.Collections.Generic.Stack<string> myStack =
    New System.Collections.Generic.Stack<string>();
  myStack.Push("St. Louis Rams");
  myStack.Push("Indianapolis Colts");
  myStack.Push("Minnesota Vikings");
  Array myArray;
  myArray = myStack.ToArray();
  foreach(string item in myArray)
  {
    Label1.Text += item + "<br />";
  }
}
```

В этом примере класс `Stack` явно приведен, чтобы содержать в себе элементы типа `string`. Здесь вы специфицируете тип коллекции с применением угловых скобок. То есть, в этом примере класс `Stack` приводится к типу `string` с использованием синтаксиса `Stack<string>`. Если вы захотите привести его к чему-то другому, отличному от класса `Stack` элементов `string` (например, элементов `int`), то должны будете специфицировать `Stack<int>`.

Поскольку коллекция элементов в классе `Stack` приведена к специфическому типу непосредственно при создании класса `Stack`, нет необходимости приводить его элементы к типу `object` и позднее (в цикле `foreach`) — обратно к типу `string`. Этот процесс называется упаковкой и обходится недешево. Поскольку в этом коде типы специфицированы заранее, возрастает производительность работы с коллекцией.

В дополнение к сказанному о различных типах коллекций, вы также можете использовать обобщения с классами, делегатами, методами и тому подобным. Механизм обобщений детально раскрыт в главе 9.

Анонимные методы

Анонимные методы позволяют поместить программный код внутрь делегата, так что его можно выполнить позднее вместо создания совершенно нового метода. Например, если вы не используете анонимные методы, то должны применять делегат в манере, подобной следующему:

```
public partial class Default_aspx
{
  void Page_Load(object sender, EventArgs e)
  {
    this.Button1.Click += ButtonWork;
  }
  void ButtonWork(object sender, EventArgs e)
  {
    Label1.Text = "Вы щелкнули на кнопке!";
  }
}
```

Однако с помощью анонимных методов можно поместить необходимое действие непосредственно в делегат, как показано в следующем примере:

```
public partial class Default_aspx
{
  void Page_Load(object sender, EventArgs e)
  {
    this.Button1.Click += delegate(object myDelSender, EventArgs myDelEventArgs)
    {
      Label1.Text = "Вы щелкнули на кнопке!";
    };
  }
}
```

Используя анонимные методы, нет необходимости создавать отдельный метод. Вместо этого необходимый код помещается непосредственно после объявления делегата. Операторы и шаги, которые должны быть выполнены делегатом, помещаются между фигурными скобками и завершаются точкой с запятой.

Типы, допускающие `null`

Благодаря появлению обобщений в основе .NET Framework 2.0 стало возможно создавать типы, допускающие `null`-значения — используя `System.Nullable<T>`. Это идеальное решение для ситуаций, когда, например, нужно создавать наборы элементов типа `int`, в которых могут содержаться значения `null`. Раньше всегда было трудно создать `int` со значением `null` изначально или же присваивать `null` переменным `int`.

Чтобы создать допускающий `null` тип `int`, вы должны использовать следующий синтаксис:

```
System.Nullable<int> x = new System.Nullable<int>;
```

Появился также новый модификатор типа, который также объявлять тип как допускающий значения `null`. Это показано в следующем примере:

```
int? salary = 800000
```

Такая способность объявлять типы, допускающие `null`, не является свойством исключительно C#, поскольку встроена в .NET Framework и стала возможной благодаря появлению средства обобщений в .NET. По этой причине вы также найдете типы, допускающие `null`, и в Visual Basic 2005.

Итераторы

Итераторы позволяют использовать циклы `foreach` с вашими собственными пользовательскими типами. Чтобы достичь этого, необходимо, чтобы ваш класс реализовывал интерфейс `IEnumerable`, как показано ниже:

```
using System;
using Systm.Collections;
public class myList
{
  internal object[] elements;
  internal int count;
  public IEnumerator GetEnumerator()
  {
    yield return "St. Louis Rams";
    yield return "Indianapolis Colts";
    yield return "Minnesota Vikiings";
  }
}
```

Чтобы использовать интерфейс IEnumerable, необходимо сослаться на пространство имен System.Collections. Имея все это, вы можете выполнять итерации по своему пользовательскому классу, как показано ниже:

```
void Page_Load(object sender, EventArgs e)
{
  myList IteratorList = new myList();
  foreach(string item in IteratorList)
  {
       Response.Write(item.ToString() + "<br />");
  }
}
```

Частичные классы

Частичные (partial) классы — новое средство .NET Framework 2.0, и C# опять использует преимущества этого нововведения. Частичные классы позволяют вам разделить один класс на множество файлов, которые позднее, при компиляции, комбинируются в единый класс.

Чтобы объявить частичный класс, нужно просто использовать ключевое слово partial в любых классах, которые должны объединяться с другими, формируя единый класс. Ключевое слово partial предшествует имени класса, который комбинируется с оригинальным классом. Например, у вас может быть простой класс Calculator, объявленный следующим образом:

```
public class Calculator
{
  public int Add(int a, int b)
  {
     return a + b;
  }
}
```

С этого момента вы можете создать второй класс, который присоединится к первому, как показано ниже:

```
public partial class Calculator
{
  public int Subtract(int a, int b)
  {
     return a - b;
  }
}
```

При компиляции эти классы будут собраны вместе в единый класс Calculator, как если бы они изначально составляли единое целое.

Введение в .NET Framework 3.0

Последней версией каркаса .NET является .NET Framework 3.0. Эта версия предлагает несколько исключительно новых возможностей, среди которых построение приложений нового типа с использованием Windows Presentation Foundation (WPF), а также приложений и библиотек на основе Windows Communication Foundation (WCF), Windows Workflow Foundation (WF) и Windows CardSpace.

Упомянутые дополнения — это именно то, что и образует .NET Framework 3.0. В настоящей книге им посвящено множество глав.

Для чего подходит C#

В определенном смысле C# по отношению к языкам программирования представляет собой то же самое, что .NET по отношению к среде Windows. Точно так же, как Microsoft добавляла все больше и больше средств к Windows и Windows API в течение последнего десятилетия, так и Visual Basic 2005 и C++ подвергались постоянному расширению. Хотя Visual Basic и C++ в результате этого стали весьма мощными языками, оба они страдают от проблем, связанных с "тяжелым наследием" их эволюции.

В случае Visual Basic 6 и ранних версий сильная сторона языка определялась тем фактом, что его было легко понять, и он значительно облегчал решение многих задач программирования, скрывая от разработчиков детали Windows API и инфраструктуру компонентов COM. Отрицательная же сторона состояла в том, что Visual Basic никогда не был по-настоящему объектно-ориентированным, так что крупные приложения быстро теряли внутреннюю организацию, и их становилось трудно сопровождать. К тому же, поскольку синтаксис Visual Basic был унаследован от ранних версий BASIC (который, в свою очередь, был задуман скорее как простой для понимания начинающими программистами язык, нежели для написания крупных коммерческих приложений), на самом деле он не был приспособлен для построения хорошо структурированных объектно-ориентированных программ.

С другой стороны, корни C++ относятся к определению языка ANSI C++. Он не полностью совместим со стандартом ANSI (хотя и близок к нему) по той простой причине, что Microsoft впервые разработала свой компилятор C++ до того, как определение ANSI стало официальным. К сожалению, это привело к двум проблемам. Во-первых, корни ANSI C++ лежат в технологиях двадцатилетней давности, поэтому ему не хватает поддержки современных концепций (таких как строки Unicode и генерация XML-документации), и он содержит некоторые архаичные синтаксические структуры, предназначенные для компиляторов прошлых лет (такие как отделение объявления от определения функций-членов). Во-вторых, разработчики Microsoft одновременно пытаются развивать C++, чтобы он стал языком, предназначенным для выполнения высокопроизводительных задач под Windows, а потому вынуждены были добавить к нему огромное количество специфичных для Microsoft ключевых слов и разнообразных библиотек. В результате язык C++ для Windows стал чрезвычайно "запутанным". Попробуйте спросить разработчиков C++, сколько определений строковых типов они знают: `char*`, `LPTSTR`, `string`, `CString` (версия MFC), `CString` (версия WTL), `wchar_t*`, `OLECHAR*` и так далее и тому подобное...

Теперь появилась .NET — полностью новая среда, которая повлекла за собой появление новых расширений обоих языков. Microsoft добавила еще больше специфичных ключевых слов с C++, и провела полную реконструкцию Visual Basic, создав Visual Basic .NET и Visual Basic 2005 — язык, который унаследовал некоторый базовый синтаксис VB, но дизайн которого полностью отличается от того, к чему мы привыкли; он практически стал совершенно новым языком.

В упомянутом контексте Microsoft решила предложить разработчикам альтернативу — язык, ориентированный специально на .NET, к тому же спроектированный с чистого листа. В результате появился Visual C# 2005. Официально Microsoft описывает C# как "простой, современный, объектно-ориентированный и безопасный к типам язык программирования, унаследованный от C и C++". Большинство независимых обозревателей, вероятно, изменили бы это на "унаследованный от C, C++ и Java". Такое описание является технически точным, но мало что говорит о красоте и элегантности языка. Синтаксически C# очень похож на C++ и Java, в том смысле, что многие ключевые слова — те же, кроме того, C# также разделяет с языками C++ и Java ту же блочную

структуру с фигурными скобками для выделения блоков кода и точками с запятой для завершения операторов. Первое впечатление от фрагмента кода C# состоит в том, что он выглядит подобно C++ или Java. Но, несмотря на это внешнее сходство, C# изучить намного легче, чем C++, и по сложности освоения он примерно равен Java. Его дизайн в большей степени соответствует современным инструментам разработки, чем у обоих его предшественников, и он предлагает простоту в использовании, как у Visual Basic, вместе с высокой производительностью и низкоуровневым доступом к памяти, характерные для C++, когда это необходимо. Ниже перечислены возможности C#, которые следует отметить особо.

- Полная поддержка классов и объектно-ориентированного программирования, включая наследование реализации и интерфейсов, виртуальные функции и перегрузку операций.
- Согласованный и четко определенный набор базовых типов.
- Встроенная поддержка автоматической генерации XML-документации.
- Автоматическая очистка динамически распределяемой памяти.
- Средство маркировки классов и методов пользовательскими атрибутами. Это может быть полезно для документирования и может иметь некоторый эффект при компиляции (например, помеченные методы могут компилироваться только для отладочных сборок).
- Полная поддержка библиотеки базовых классов .NET наряду с легким доступом к Windows API (если вы действительно в этом нуждаетесь, что случается нечасто).
- Указатели и прямой доступ в память при необходимости доступны, но язык спроектирован так, что в большинстве случаев без них можно обойтись.
- Поддержка свойств и событий в стиле Visual Basic.
- Простым изменением опций компилятора можно собирать либо исполняемые программы, либо библиотеки компонентов .NET, которые могут быть вызваны из стороннего кода — так же, как это делается с элементами управления Active X (COM-компонентами).
- Возможность использования для написания динамических Web-страниц ASP.NET и Web-служб XML.

Бо́льшая часть перечисленного также касается Visual Basic 2005 и управляемого C++. Тот факт, что C# изначально спроектирован для работы с .NET, однако, означает, что он поддерживает средства .NET в более полной мере, и предлагает в этом контексте более подходящий синтаксис, чем прочие языки. Хотя язык C# сам по себе очень похож на Java, есть несколько существенных преимуществ. В частности, язык Java не предназначен для работы в среде .NET.

Прежде чем завершить тему, следует отметить и ряд ограничений C#. Одна область, для которой данный язык не предназначен — это критичный по времени или исключительно высокопроизводительный код — когда приходится заботиться о том, чтобы цикл занимал 1000 или 1050 тактов процессора, когда необходимо очищать ресурсы в течение миллисекунд после того, как отпадает потребность в них. Вероятно, C++ продолжает оставаться самым непревзойденным из высокоуровневых языков в этой области. C# недостает некоторых ключевых средств для построения наиболее высокопроизводительных приложений, включая возможность специфицировать встроенные функции и деструкторы, которые гарантированно запускаются в определенной точке кода. Однако пропорциональное отношение таких приложений к их общему числу чрезвычайно низко.

Что необходимо для написания и выполнения кода C#

.NET Framework работает под управлением Windows 98, 2000, XP и 2003. Чтобы писать код, используя .NET, необходимо инсталлировать .NET SDK, если только вы не используете Windows Server 2003, который поставляется с предустановленной средой .NET Framework 1.0 и 1.1. Если вы собираетесь работать с примерами, представленными в этой книге, то потребуется инсталлировать .NET Framework 2.0, даже если вы работаете в среде Windows Server 2003. По умолчанию .NET Framework 2.0 не включена в этот сервер. .NET Framework 3.0 для своей работы требует одну из следующих операционных систем: Windows XP SP2, Windows Server 2003 SP1 или Windows Vista.

К тому же, если только вы не собираетесь писать код C#, используя текстовый редактор или некоторую среду разработки от независимых поставщиков, то почти наверняка захотите установить Visual Studio 2005. Полный SDK не требуется, чтобы выполнять управляемый код, но исполняющая среда .NET необходима. Возможно, вам понадобится поставлять ее вместе с вашим кодом для тех клиентов, у которых она еще не установлена.

Как организована эта книга

В главе 1 этой книги мы представим обзор общей архитектуры .NET, чтобы подготовить почву, которая нужна для разработки управляемого кода. Далее книга организована в виде набора разделов, в которых описывается как сам язык C#, так и его применение во многих областях.

Часть I. Язык C#

Эта часть закладывает хороший фундамент знаний о самом языке C#. Она не предполагает знания какого-либо конкретного языка, хотя предполагается, что вы — опытный программист. Мы начнем с обзора базового синтаксиса C# и его типов данных, а затем обсудим объектно-ориентированные возможности C#, после чего можно будет перейти к более сложным темам программирования на C#.

Часть II. Visual Studio

Эта часть посвящена главной IDE-среде, применяемой разработчиками на C# во всем мире — Visual Studio 2005. В двух главах этой части будут показаны лучшие способы использования этого инструмента для построения приложений на базе .NET Framework 2.0 и 3.0, а также рассматриваются вопросы развертывания проектов.

Часть III. Библиотеки базовых классов

В этой части рассматриваются принципы программирования в среде .NET. В частности, описываются концепции безопасности, многопоточность, локализация, транзакции, построение Windows-служб и генерация собственных библиотек в виде сборок.

Часть IV. Данные

Здесь рассматривается доступ к базам данных средствами ADO.NET и взаимодействие с каталогами и файлами. Также подробно раскрывается поддержка .NET технологии XML на стороне операционной системы Windows, а также средства .NET, встроенные в SQL Server 2005.

Часть V. Презентации

Эта часть сосредоточена на построении классических приложений Windows, которые в .NET называются Windows Forms. Windows Forms представляют собой версии приложений "толстого" клиента, а применение .NET для их построения — простой и быстрый способ решения этой задачи. В дополнение к рассмотрению Windows Forms рассматривается GDI+ — технология, используемая для построения приложений, работающих с расширенной графикой. В этой части также раскрываются компоненты, работающие на Web-сайтах, доставляющие Web-страницы. Сюда входит огромный объем новых средств, предлагаемых ASP.NET 2.0. Кроме того, в этой же части рассматриваются вопросы построения приложений на базе Windows Presentation Foundation.

Часть VI. Коммуникации

Эта часть полностью посвящена коммуникациям. Здесь описываются Web-службы, применяемые для коммуникаций, независящих от платформы, .NET Remoting — для коммуникаций между клиентами и серверами .NET, Enterprise Services (службы уровня предприятия) — для служб, работающих в фоновом режиме, и коммуникации DCOM. На примере системы асинхронных сообщений показаны коммуникации в отключенном режиме. В этой части также рассматриваются вопросы использования Windows Communication Foundation и Windows Workflow Foundation.

Часть VII. Дополнительные сведения

В этой части рассматриваются вопросы построения приложений, которые используют новые возможности Windows Vista. Кроме того, здесь также дается описание формирующейся технологии LINQ и того, как она может применяться в приложениях на C#.

Часть VIII. Приложения

В состав этой части входит несколько приложений, детализирующих принципы объектно-ориентированного программирования, а также информация, специфичная для языка C#.

Соглашения

В этой книге используется множество стилей текста, которые призваны помочь отличать разнородную информацию. Ниже представлены примеры используемых стилей и объяснение их назначения.

- *Важные слова* выделены курсивом.
- Клавиши, которые вы нажимаете на клавиатуре, представлены как <Ctrl>, <Enter> и так далее.

Код появляется в книге в разных видах. Если слово, о котором говорится в тексте, является частью исходного кода, например, в обсуждении конструкции `if...else`, оно представлено моноширинным шрифтом.

Блоки кода представлены моноширинным шрифтом (иногда с полужирным начертанием, если нужно обратить на код особое внимание):

```
public static void Main()
{
  AFunc(1,2,"abc");
}
```

Советы, подсказки и базовая информация выделены курсивом.

Важные части информации появляются в рамках.

Синтаксис применения методов, свойств и тому подобного представлен в следующем формате:

```
Regsvcs BookDistributor.dll [COM+AppName] [TypeLibrary.tbl]
```

Здесь курсивом выделены объектные ссылки, переменные или значения параметров, которые должны быть заменены; квадратные скобки указывают на необязательные параметры.

Исходный код

Работая с примерами этой книги, вы можете либо вводить весь их код вручную, либо использовать файлы исходного кода, находящиеся на прилагаемом к книге компакт-диске.

От издательства

Вы, читатель этой книги, и есть главный ее критик и комментатор. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересно услышать и любые другие замечания, которые вам хотелось бы высказать в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам бумажное или электронное письмо, либо просто посетить наш Web-сервер и оставить свои замечания там. Одним словом, любым удобным для вас способом дайте нам знать, нравится или нет вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

Посылая письмо или сообщение, не забудьте указать название книги и ее авторов, а также ваш обратный адрес. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию последующих книг. Наши координаты:

E-mail: info@dialektika.com

WWW: http://www.dialektika.com

Информация для писем из:

России: 115419, Москва, а/я 783

Украины: 03150, Киев, а/я 152