3

Краткий обзор языка C#

Эта глава предлагает молниеносное турне на тему возможностей языка C#. Да, мы уже слышим ваши возражения: “Но ведь просто невозможно сжать всю информацию о C# в одну главу!” Мы и не планируем это. Мы собираемся предложить здесь только краткий обзор языка и ознакомить вас с его возможностями. Сами эти возможности будут описаны более подробно в следующих главах.

Язык C#

Вы уже знаете, что название C# произносится “си-диез” и что язык C# является объектно-ориентированным, обеспечивающим типовую безопасность (т.е. вы не можете приписать значение типу, к которому это значение не относится; подробности этого понятия будут обсуждаться позже) и что он подобен C или C++.

Основы языка C#

Начнем наше обсуждение C# с рассмотрения универсальной программы “Hello, World!”.

```
using System;

class Hello
{
    static void Main()
    {
        Console.WriteLine("Hello, World!");
    }
}
```

Этот программный код представляет собой исходный код программы, сохраняемый в текстовом файле с расширением `.cs` (например, в файле `helloworld.cs`). Программа на языке C# может состоять из одного или нескольких файлов с исходным кодом. Файлы с исходным кодом превращаются в программы с помощью компилятора. Мы здесь будем использовать компилятор командной строки, а не более сложные инструменты, например, такие как компилятор Visual Studio .NET.

Чтобы скомпилировать наш файл, мы используем следующую команду:

```
csc helloworld.cs
```

Обратите внимание на то, что для компиляции требуется, чтобы в системе было установлено программное обеспечение .NET Framework. Оно также требуется для того, чтобы запускать полученные в результате компиляции исполняемые модули программ.

Здесь `csc` обозначает имя компилятора C#, поставляемого в рамках .NET Framework (на самом деле именем компилятора является `csc.exe`, но расширение здесь указывать не обязательно), а `helloworld.cs` соответствует файлу с исходным кодом C#, который передается компилятору как аргумент для компиляции. В результате компиляции файла `helloworld.cs` получается исполняемый файл, получающий по умолчанию имя `helloworld.exe`.

При запуске этого исполняемого файла мы получим следующий вывод:

```
Hello, World!
```

Анализ исходного кода

Давайте рассмотрим исходный программный код.

```
using System;
```

Здесь директива `using` ссылается на пространство имен `System`. Это пространство имен предлагается библиотекой классов CLI (Common Language Infrastructure — общеязыковая инфраструктура). Иногда аббревиатуру CLI используют и в качестве названия для .NET Framework. Это пространство имен содержит класс `Console`, который мы с вами будем использовать в строках исходного кода некоторых примеров. Использование директивы `using` позволяет нам использовать имена типов без уточнения в имени типа принадлежности данного типа к данному пространству имен. Что это значит? Ну, это значит, что мы получаем возможность уменьшить объем создаваемого программного кода, поскольку мы можем использовать

```
Console.WriteLine
```

вместо

```
System.Console.WriteLine
```

Конечно, здесь наша экономия составила всего семь символов, но для больших программ соответствующие цифры могут оказаться более впечатляющими.

Обратите внимание на то, что метод с именем `Main` входит в состав класса, названного здесь `Hello`.

Модификатор `static` здесь используется для того, чтобы соответствующий метод был методом класса, а не экземпляра класса (не волнуйтесь, если вы пока что не знаете, что это означает — роль модификаторов `static` будет обсуждаться подробнее в одной из следующих глав).

Метод `Main` является местом начала выполнения приложения. Для этого используется термин *точка входа* приложения.

Вывод “Hello, World!” обрабатывается библиотекой классов, которая автоматически выполняет всю работу, необходимую для отображения заданного текста на экране.

Типы

Тип — это элемент классификации различных значений и выражений в данном языке программирования. Компьютер хранит все данные в виде нулей и единиц, но сами данные должны иметь свой контекст или свое значение. Чтобы сохранить такое значение, используются типы. В C# поддерживаются два базовых типа:

- типы, характеризуемые значением;
- ссылочные типы (т.е. типы, характеризуемые ссылкой).

Эти базовые типы кратко обсуждаются в следующих разделах и более широко в следующих главах книги. На данный момент вам достаточно знать о том, что такие типы есть и они поддерживаются в C#.

Типы, характеризуемые значением

К типам, характеризуемым значением, относятся следующие типы:

- типы перечня,
- типы структуры,
- простые типы (например, такие как `char`, `float` и `int`).

Переменная типа, характеризуемого значением, содержит данные, отличные от данных ссылочных переменных (вы вскоре в этом убедитесь). Точно так же любая переменная типа, характеризуемого значением, будет иметь свою собственную копию данных, не влияющих на данные других копий.

Ссылочные типы

К ссылочным типам относятся следующие типы:

- типы массива,
- типы класса,
- типы делегата,
- типы интерфейса.

Главное отличие ссылочных типов от типов, характеризуемых значением, заключается в том, что переменные ссылочного типа содержат ссылки на соответствующие объекты, а не фактические данные (в отличие от типов, характеризуемых значением). В этом случае, если несколько переменных указывают на один и тот же объект, то операция, воздействующая на одну переменную, изменяет и все другие соответствующие ссылки.

Встроенные типы

C# предлагает ряд встроенных типов. В частности, есть два встроенных ссылочных типа:

- `object` — это исходный базовый тип для всех других типов,
- `string` — используется для представления строковых значений в кодировке Unicode.

Следующие типы являются встроенными типами, характеризуемыми значением:

- целочисленные типы со знаком (`int`, `long`, `sbyte` и `short`),
- целочисленные типы без знака (`byte`, `unint`, `ulong` и `ushort`),
- типы `bool`, `char` и `decimal`,
- типы с плавающим разделителем (`float` и `double`).

В следующей таблице предлагается полный список всех таких типов в C#, с объяснениями по поводу того, какие данные эти типы представляют.

Тип	Замечания

`bool`	Логический тип (допускаются только значения `true` и `false`), например,
	`bool x = true;`
	`bool y = false;`

`byte`	Целочисленный тип 8-битовой длины, например, `byte x = 13;`

`char`	Отдельный символ Unicode, например,
	`char x = 'x';`
	`char y = 'c';`

`decimal`	Десятичный тип высокой точности, минимум 28 значащих цифр, например,
	`decimal x = 1.99M;`
	`decimal y = 9.02M;`

`double`	Десятичный тип двойной точности с плавающим разделителем, например,
	`double x = 1.99;`
	`double y = 9.02D;`

`float`	Числовой тип стандартной точности с плавающим разделителем, например, `float x = 1.99F;`

`int`	Целочисленный тип 32-битовой длины со знаком, например,
	`int x = 7;`
	`int y = 17;`
	`int z = 1237;`

`long`	Целочисленный тип 64-битовой длины со знаком, например,
	`long x = 17;`
	`long y = 37L;`

`object`	Базовый тип для всех других типов, например, `object x = null;`

`sbyte`	Целочисленный тип 8-битовой длины со знаком, например,
	`sbyte x = 17;`
	`sbyte y = 37;`

`short`	Целочисленный тип 16-битовой длины со знаком, например,
	`short x = 17;`
	`short y = 37;`

`string`	Последовательность символов Unicode, например,
	`string x = "Hello, World!";`
	`string y = "37";`

`uint`	Целочисленный тип 32-битовой длины без знака, например,
	`uint x = 17;`
	`uint y = 37U;`

Окончание таблицы

Тип	Замечания

`ulong`	Целочисленный тип 64-битовой длины без знака, например,
	`ulong w = 17;`
	`ulong x = 37U;`
	`ulong y = 42L;`
	`ulong z = 54UL;`

`ushort`	Целочисленный тип 16-битовой длины без знака, например, `ushort x = 17;`

Перегрузка

Встроенные типы могут использовать перегруженные операции. Хорошим примером такого использования являются операции сравнения == и !=. Они имеют разные значения для различных встроенных типов, как объясняется ниже.

- Два выражения типа `int` равны, если они представляют одно и то же целое значение, например,

  ```
  int x = 2;
  int y = 2;
  ```
  `x == y` должно дать в результате `true`.

- Два выражения типа `object` считаются равными, если оба они ссылаются на один и тот же объект (или если оба они равны `null`), например,

  ```
  object x = null;
  object y = null;
  ```
  `x == y` должно дать в результате `true`.

- Два выражения типа `string` считаются равными, если все соответствующие символы и пробелы в строках идентичны (или если обе они равны `null`), например,

  ```
  string x = "Hello";
  string y = "Hello";
  ```
  `x == y` должно дать в результате `true`

  ```
  string x = " Hello";
  string y = "Hello ";
  ```
  `x == y` должно дать в результате `false` — эти строки отличаются пробелами.

Преобразования типов

В C# есть два вида преобразований типов.

- **Неявные преобразования.** Это преобразования, которые можно выполнить безопасно, потому что компилятору не требуется никаких дополнительных условий для того, чтобы гарантировать точность результата.
- **Явные преобразования.** Явные преобразованиям требуют больше внимания, чтобы гарантировать точность и надежность результата.

Типы массива

В C# поддерживаются как одномерные, так и многомерные массивы. Но, кроме обычных прямоугольных массивов, поддерживаются и так называемые "невыровненные" массивы. *Невыровненный* массив — это массив массивов. Невыровненные массивы в программном коде легко заметить, поскольку для таких массивов пара `[]` указывается больше одного раза подряд:

```
int[][] a2;
```

Это был массив массивов типа `int`.

```
int[][][] a3;
```

А это массив массивов массивов типа `int`.

Откуда взялись названия *прямоугольный* и *невыровненный*? Взгляните на следующий трехмерный прямоугольный массив:

```
int[,,] a1 = new int[10, 20, 30];
```

В этом примере длины трех измерений массива `a1` равны 10, 20 и 30 соответственно, и массив содержит $10 \times 20 \times 30$ элементов. Если изобразить этот массив, то должна получиться правильная форма.

Невыровненные массивы, с другой стороны, порождают не такие регулярные формы.

Переменные и параметры

Переменные представляют ячейки памяти, и каждая переменная имеет тип, определяющий, какие значения могут запоминаться в этой переменной. Локальные переменные объявляются в членах-функциях (например, методах, свойствах или индексаторах).

Для объявления локальной переменной нужно указать следующее:

- имя типа,
- оператор объявления, в котором задается имя переменной и (необязательно) ее начальное значение.

Следующий программный код демонстрирует три определения локальных переменных:

```
int x;
int y = 7;
int z = 14;
```

Один оператор объявления может определять сразу несколько переменных, например,

```
int x, y = 7, z = 14;
```

Очень важно, чтобы до того, как переменная будет использоваться, ей было присвоено значение. В противном случае будет сгенерирована ошибка компиляции. Например, попытка скомпилировать следующий программный код должна порождать ошибку компиляции (потому что в выделенной ниже строке используется переменная, которой еще не было присвоено значение):

```
class Test
{
   static void Main()
   {
      int x;
      int y = 7;
      int z = x + y;
   }
}
```

Поле — это переменная, связанная с классом или структурой или же экземпляром класса или структуры. Поле, объявленное с модификатором `static`, определяет статическую переменную, а поле, объявленное без такого модификатора, определяет переменную экземпляра. Статическое поле ассоциируется с типом, а переменная экземпляра — с экземпляром типа.

```
using Books.Data;

class Titles
{
   private static DataSet ds;
   public string Title;
   public decimal Price;
}
```

В этом примере определяется класс, в котором есть приватная статическая переменная и две общедоступные переменные экземпляра.

Объявления формальных параметров также используются в качестве определения переменных. В C# существуют четыре различных вида параметров.

- **Параметры, передаваемые по значению.** Используются для передачи значения аргумента "внутрь" соответствующего метода.
- **Ссылочные параметры.** Используются для передачи параметра "по ссылке", когда параметр выступает в качестве альтернативного имени для вызывающей стороны, обеспечивающей данный аргумент.
- **Выходные параметры.** Подобны ссылочным параметрам, но при этом начальное значение аргумента, обеспечиваемого вызывающей стороной, не важно.
- **Массивы параметров.** Объявляются с модификатором `params`. Любой метод может иметь не более чем один массив параметров, и такой массив должен быть последним в списке указанных параметров.

Выражения

Язык C# предлагает целый набор операций, которые могут использоваться в выражениях. Эти операции можно разбить на такие группы:

- унарные операции,
- бинарные операции,
- тернарная операция (такая операция здесь всего одна).

В следующей таблице имеющиеся в C# операции разделены на категории и представлены в порядке возрастания их приоритета.

Категория	Операции

Первичные операции	`x.y`
	`f(x)`
	`a[x]`
	`x++`
	`x--`
	`new`
	`typeof`
	`checked`
	`unchecked`

Унарные операции	`+`
	`-`
	`!`
	`~`
	`++x`
	`--x`
	`(T)x`

Мультипликативные операции	`*`
	`/`
	`%`

Аддитивные операции	`+` `-`
Операции сдвига	`<<` `>>`

Операции отношения (сравнения или проверки типов)	`<`
	`>`
	`<=`
	`>=`
	`is`
	`as`

Операции равенства	`==` `!=`
Операция И (операция логического умножения)	`&`
Операция исключающего ИЛИ	`^`
Операция ИЛИ (операция логического сложения)	`|`
Операция условного И	`&&`
Операция условного ИЛИ	`||`
Условная операция	`?:`

Операции присваивания	`=`
	`*=`
	`/=`
	`%=`
	`+=`
	`-=`
	`<<=`
	`>>=`
	`&=`
	`^=`
	`|=`

Когда в выражении присутствуют несколько операций, приоритет операций определяет порядок, в котором эти операции должны выполняться. Принятый по умолчанию приоритет можно изменить с помощью (круглых) скобок. Например, следующие выражения будут обработаны по-разному:

- `x + y * z`

  Здесь `y` будет умножено на `z`, а затем результат будет добавлен к `x`.

- `(x + y) * z`

  Здесь будет выполнено сложение `x` и `y`, а затем результат будет умножен на `z`.

Операторы

Вот список операторов, имеющихся в C# (многие из них будут вполне понятны тем, кто использовал C или C++):

- операторы списка и блока,
- метки операторов и операторы `goto`,
- объявления локальных констант,
- объявления локальных переменных,
- операторы выражений,
- операторы `if`,
- операторы `switch`,
- операторы `while`,
- операторы `do`,
- операторы `for`,
- операторы `foreach`,
- операторы `break`,
- операторы `continue`,
- операторы `return`,
- операторы `yield`,
- операторы `throw`,
- операторы `try`,
- операторы `checked`,
- операторы `unchecked`,
- операторы `lock`,
- операторы `using`.

Классы

Объявления классов определяют новые ссылочные типы. Класс может наследоваться из другого класса, а также может реализовывать интерфейсы. Обобщенные объявления классов имеют как минимум один параметр типа.

Классы состоят из членов и могут включать следующие объекты:

- константы,
- события,
- поля,
- финализаторы,
- индексаторы,
- конструкторы экземпляров,
- методы,
- объявления вложенных типов,
- операторы,
- свойства,
- статические конструкторы.

Каждый член характеризуется своим уровнем доступности, что используется для управления областями программного кода, доступными для данного члена. Различается пять возможных уровней доступности членов:

- `public` — доступ не ограничен;
- `protected` — доступ разрешен только для класса, содержащего член, и производных этого класса;
- `internal` — доступ разрешен только для соответствующей программы;
- `protected internal` — доступ разрешен только для соответствующей программы и производным того типа, который содержит данный член;
- `private` — доступ разрешен только для класса, содержащего член.

Константы

Константа — это член класса, который, как и предлагает название, используется для представления постоянного значения. Такое постоянное значение можно либо объявить, либо вычислить во время компиляции. Константы могут зависеть от других констант той же программы, при этом запрещены только циклические зависимости (где *A* зависит от *B*, в то время как *B* определено и зависит от *A*).

Поля

Полем называется член, используемый для представления переменной, связанной с объектом или классом.

Методы

Методом называют член, реализующий действие, которое может выполняться объектом или классом.

Методы имеют

- список формальных параметров (который может быть пустым),
- возвращаемое значение (если только возвращаемым типом не является `void`).

Методы могут быть статическими и нестатическими:

- статические методы доступны через класс,
- нестатические методы доступны через экземпляры класса.

Нестатические методы называют также методами экземпляра.

Свойства

Свойством называют член, обеспечивающий доступ к конкретной характеристике объекта или класса (например, такой как длина строки). Свойства в определенном смысле аналогичны полям, но, в отличие от полей, не указывают на области в памяти. Свойства имеют аксессоры (средства доступа), указывающие операторы, которые должны выполняться при доступе к свойству для чтения или записи.

События

Событие представляет собой член, обеспечивающий объекту или классу возможность посылать уведомления. Класс определяет событие с помощью объявления события (которое задает тип делегата) и необязательного набора аксессоров события.

Операции

Операция представляет собой член, определяющий значение выражения, которое можно применять к экземплярам данного класса. Можно определить следующие три вида операций:

- бинарные операции,
- операции преобразования,
- унарные операции.

Индексаторы

Индексатор представляет собой член, позволяющий индексацию объекта и соответствующий доступ к нему, аналогичный доступу к массиву.

Конструкторы экземпляров

Конструктор экземпляра — это член, выполняющий действия, необходимые для инициализации экземпляра класса.

Финализаторы

Финализатор — это член, выполняющий действия, необходимые для завершения использования экземпляра класса. Соответствующие действия выполняются тогда, когда класс больше не нужен.

Финализаторы не могут использовать

- параметры,
- модификаторы доступности.

Финализаторы не могут быть вызваны явно. Финализатор любого экземпляра вызывается автоматически в процессе сборки мусора, выполняемого средствами .NET Framework.

Статические конструкторы

Статический конструктор является членом, выполняющим действия, необходимые для инициализации класса. Статические конструкторы не могут использовать

- параметры,
- модификаторы доступности.

Статические конструкторы не могут быть вызваны явно, а вызываются автоматически.

Наследование

Классы поддерживают единичное наследование (т.е. они могут наследоваться от одного класса, называемого также суперклассом, — поэтому в программном коде невозможно появление некоторых слишком сложных структур). Тип `object` является базовым классом для всех классов.

Методы, свойства и индексаторы могут быть виртуальными. Это значит, что их реализации могут переопределяться в производных классах.

Статические классы

Статические классы не предполагают создание их экземпляров, и такие классы содержат только статические члены. Статические классы являются неявно изолированными, они не имеют конструкторов экземпляров.

Структуры

Структуры подобны классам. Главными особенностями структур является то, что

- структуры являются типами, характеризуемыми значением, а не ссылочными типами;
- структуры не поддерживают наследование.

Зачем использовать структуры? Главная причина в повышении производительности: поскольку значения запоминаются в стеке, структуры имеют преимущество в производительности по сравнению с классами. Однако, с учетом известных ограниче-

ний на их значения, некоторые программисты предпочитают использовать не структуры, а классы.

Интерфейсы

Интерфейс используется для определения *контракта*. Что такое контракт? Контракт интерфейса — это гарантия объекта в том, что он будет поддерживать все элементы данного интерфейса. Этот контракт создается с помощью ключевого слова `Interface`, объявляющего ссылочный тип, который инкапсулирует контракт. Класс или структура, реализующие интерфейс, должны выполнять контракт, иначе возникает ошибка.

Интерфейсы могут содержать следующие члены:

- события,
- индексаторы,
- методы,
- свойства.

Делегаты

Делегаты позволяют программисту использовать такие возможности C#, которые в других языках реализуются с помощью указателей. Между делегатами и указателями есть два главных отличия:

- делегаты обеспечивают типовую безопасность,
- делегаты обеспечивают объектно-ориентированный подход.

Объявление делегата определяет некоторый класс, производный от класса `System.Delegate`. Экземпляр делегата инкапсулирует один или несколько методов, и каждый из этих методов является единицей, доступной для вызова. Когда дело доходит до методов экземпляра, доступная для вызова единица представляется экземпляром и методом этого экземпляра. Для статических методов доступная для вызова единица представляется собственно методом.

Перечни

Объявление типа перечня определяет имя типа для связанной группы символьных констант. Перечни используются в тех ситуациях, когда программисту желательно иметь фиксированный набор опций выбора. Окончательный выбор делается в среде выполнения из набора опций, которые должны быть известны во время компиляции.

Обобщения

Обобщения представляют собой не одну, а целую группу возможностей, предлагаемых в рамках языка C#. С помощью обобщений C# обеспечивает параметризацию классов, структур, интерфейсов и методов на основе типов данных, которые в них хранятся и обрабатываются.

Многие общие классы и структуры допускают параметризацию по типам хранимых и обрабатываемых в них данных. Параметризованные классы называют обобщенными объявлениями классов, а параметризованные структуры — обобщенными объявлениями структур.

Многие интерфейсы тоже определяют контракты, допускающие параметризацию по типам используемых данных. Такие интерфейсы называют обобщенными объявлениями интерфейсов.

Итераторы

В C# для прохода по элементам, содержащимся в некоторой перечислимой коллекции, используется оператор `foreach`. Чтобы быть перечислимой, коллекция должна использовать метод `GetEnumerator`, возвращающий некоторый перечень.

Заметьте, что метод `GetEnumerator` является методом, не имеющим параметров.

Итератор представляет собой блок операторов, используемых для вывода упорядоченной последовательности значений. Итераторы в программном коде обнаружить легко, поскольку в них используются операторы `yield`. Это операторы

- `yield return` — генерирует следующее значение итерации;
- `yield break` — используется для указания момента завершения итераций.

Типы с разрешением принимать значение null

В C# предлагается поддержку пользовательских типов с разрешением принимать значение `null` (типы nullable). Эти типы обеспечивают поддержку отсутствия значения для всех типов, характеризуемых значением.

Типы с разрешением принимать значение `null` создаются с использованием модификатора `?`. Например, `int?` является формой типа `int` с разрешением принимать значение `null`. Точно так же `bool?` является формой типа `bool` с разрешением принимать значение `null`, а `char?` — соответствующей формой типа `char`.

Для типа с разрешением принимать значение `null` лежащий в основе тип должен быть типом, не допускающим использования `null` в качестве значения.

Соответствующие расширения преобразований позволяют встроенным и пользовательским операциям для стандартных значений типов работать с версиями этих типов с разрешением принимать значение `null`.

Расширения преобразований позволяют встроенным и пользовательским преобразованиям работать как с типами, не допускающими значение `null`, так и с версиями этих типов с разрешением принимать значение `null`.

Расширения операций позволяют встроенным и пользовательским операциям для стандартных значений типов работать с версиями этих типов с разрешением принимать значение `null`.

Резюме

В этой главе вам был предложен очень краткий обзор языка программирования C#. Были рассмотрены следующие вопросы.

- Что такое язык C# и каково его происхождение
- Основы C#
- Типы в C#, перегрузка операций и преобразования типов
- Переменные
- Параметры
- Выражения
- Операторы
- Классы
- Структуры
- Интерфейсы
- Делегаты
- Перечни
- Обобщения
- Итераторы

Если у вас нет опыта работы с C#, рекомендуется прочитать эту главу, в ином случае вы можете поступить так, как сочтете необходимым.

В главе 4 мы собираемся рассмотреть структуру языка C#.