Основы платформы Microsoft .NET

Тема:
Обзор основных пространств имен и классов Microsoft .NET

Понятие пространства имен

В .NET Framawork включены сборки библиотеки классов *.NET Framework Class Library (FCL)*, содержащие несколько тысяч классов, каждый из которых представляет некоторую функциональность. Для упрощения использования библиотеки "родственные" классы скомпонованы в отдельные множества (группы), которые называются *пространствами имен (namespace)*. То есть под пространством имен можно понимать некоторое именованное логическое хранилище, объединяющее ряд классов по некоторым признакам. Пространства имен используются для облегчения поиска нужных классов и решения проблем уникальности имен. В .NET допускается существование классов с одинаковыми именами при условии, что они будут находиться в разных пространствах имен. Пространства имен могут группировать не только классы, но и другие пространства имен.

Для того, чтобы создать объект какого-либо класса, необходимо полностью указать пространство имен, в которое он входит, и само имя класса. Например, в предыдущих разделах рассказывалось о том, как можно вывести на печать строку символов. Для этого использовался класс **Console** из пространства имен **System**. Для того чтобы сослаться на этот класс, необходимо записать **System.Console**. Каждый раз, когда в программе необходимо воспользоваться функциональностью класса **Console**, писать его имя полностью (с указанием пространства имен), может оказаться неудобным и громоздким. Эту проблему решает использование директивы **using**. Эта директива указывается в

начале программы и задает набор пространств имен, которые планируется использовать в программе. После этого в программе достаточно указывать только имя класса (без указания пространства имен) при условии, что пространство имен, в которое он входит, указано в наборе директив **using**.

Обзор основных пространств имен среды .NET Framework

Библиотека классов .NET Framework – это библиотека всех классов, интерфейсов и типов значений, включенных в пакет Microsoft .NET Framework SDK. Эта библиотека, предоставляющая разработчикам доступ к системным средствам, была разработана как основа для построения приложений, компонентов и элементов управления .NET Framework.

Рассмотрим основные пространства имен библиотеки классов .NET Framework.

Пространство имен	**Краткое описание**
Microsoft.Win32	Содержит классы двух типов: классы, обрабатывающие события, вызываемые операционной системой, и классы, работающие с системным реестром
System	Содержит основные и базовые классы, чаще всего используемые в приложениях. Содержит также классы, обеспечивающие поддержку преобразования типов данных, математических операций, удаленного и локального вызова программ
System.Collections	Содержит интерфейсы и классы, определяющие различные коллекции объектов, такие как списки, очереди, битовые массивы, хеш-таблицы и словари
System.Configuration	Содержит классы и интерфейсы, позволяющие осуществлять программный доступ к параметрам конфигурации .NET Framework и обрабатывать ошибки в файлах конфигурации (файлах .config)
System.Data	Содержит в основном классы, предназначенные для работы с данными, хранимыми в различных источниках, например, базах данных
System.IO	Содержит классы, позволяющие выполнять синхронное и асинхронное чтение и запись данных в потоки или

	файлы
System.Net	Предоставляет простой программный интерфейс для многих современных сетевых протоколов. Классы WebRequest и WebResponse составляют основу так называемых подключаемых протоколов и реализации сетевых служб, позволяющей разрабатывать приложения, использующие ресурсы Интернета, не обращая при этом внимания на отдельные особенности конкретных протоколов
System.Net.Sockets	Предоставляет управляемую реализацию интерфейса сокетов Windows (Winsock) для разработчиков, которым нужно обеспечить тщательный контроль над доступом к сети
System.Reflection	Содержит классы и интерфейсы, предоставляющие управляемое представление для загруженных типов, методов и полей, а также позволяющие создавать и вызывать типы во время работы приложения (runtime)
System.Security	Содержит классы и интерфейсы, предназначенные для обеспечения безопасности приложений, а также позволяющие устанавливать права доступа к различным частям приложения
System.Text	Содержит классы, предназначенные для обработки тестовых данных, использующих различные кодировки (ASCII, Unicode, UTF-7 и UTF-8), абстрактные базовые классы для преобразования блоков знаков в блоки байтов и обратно, а также вспомогательный класс, преобразующий и форматирующий строковые объекты (String) без создания промежуточных экземпляров этих объектов
System.Threading	Содержит классы и интерфейсы, обеспечивающие создание многопоточных приложений
System.Timers	Содержит класс Timer, позволяющий разработчику вызывать событие через определенный интервал времени

System.Windows.Forms	Содержит классы для создания приложений Windows с использованием всех возможностей пользовательского интерфейса, предоставляемых операционной системой Microsoft Windows
System.Xml	Содержит классы, обеспечивающие работу с данными в формате XML

Полный список пространств имен, входящих в поставку MS .NET Framework можно найти в [1]

Обзор классов пространства имен System

Рассмотрим основные классы, входящие в пространство имен System:

Класс	**Описание**
Array	Предоставляет методы для создания, изменения, поиска и сортировки массивов, то есть выступает в качестве базового класса для всех массивов в общеязыковой среде выполнения
BitConverter	Преобразует базовые типы данных в массив байтов и массив байтов в базовые типы данных
Console	Предназначен для организации ввода/вывода информации на консоль
Convert	Предоставляем методы для преобразования (приведения) базовых типов
Enum	Предоставляет базовый класс для перечислений
GC	Управляет сборщиком мусора системы - службой, которая автоматически освобождает неиспользуемую память
Math	Предоставляет константы и статические методы для тригонометрических, логарифмических и других общих математических функций
OperatingSystem	Предоставляет информацию об операционной системе, например версию и идентификатор платформы
Random	Представляет генератор псевдослучайных чисел, устройство, которое выдает последовательность чисел, отвечающую некоторым статистическим критериям случайности
WeakReference	Представляет слабую ссылку, которая указывает на объект,

	позволяя удалять его сборщиком мусора

Полный список классов, входящих в поставку MS .NET Framework можно найти в [2]

Обработка текстовых данных

Символы

Любая *строка* – это последовательность букв или символов. В MS .NET Framework каждый символ имеет тип *System.Char* (альтернативное имя типа *char*). Этот тип позволяет хранить числовое значение (*код*) в формате Unicode, которое может изменяться от 0x0000 до 0xFFFF.

Не смотря на то, что тип char хранит числовое значение кода (целое число) символа, ему нельзя явно присвоить число. Для того, чтобы присвоить переменной типа char значение, необходимо указать явно символ, заключенный в одинарные кавычки, или использовать его числовое представление в шестнадцатеричном формате, которое также необходимо заключить в одинарные кавычки и поставить приставу \x или \u. Например:

```
class CharDemo
{
  static void Main()
  {
    System.Char ch1='A';
    System.Char ch2='\x0041';
    System.Char ch3='\u0041';
    System.Console.WriteLine(
          "ch1={0} ch2={1} ch3={2}", ch1, ch2, ch3);
  }
}
```

Кроме того, для определения некоторых специальных символов можно использовать как их шестнадцатеричный код, как показано выше, так и запись с помощью обратного слеша.

Символ	Код	Описание
\0	0x0000	Завершающий символ строки
\a	0x0007	Звуковой сигнал
\b	0x0008	Код символа "удаления слева" (BackSpace)
\t	0x0009	Горизонтальная табуляция
\n	0x000A	Переход на новую строку
\v	0x000B	Вертикальная табуляция
\f	0x000C	Переход на новую страницу
\r	0x000D	Возврат на начало строки
\"	0x0022	Двойная кавычка
\'	0x0027	Апостроф
\\	0x005C	Обратный слеш

Так же тип System.Char содержит ряд полезных методов, облегчающих программистам работу с текстовой информацией. Ниже приведены несколько таких методов:

Метод	Описание
IsUpper	Возвращает true, если символ записан в верхнем регистре, иначе возвращает false
IsLower	Возвращает true, если символ записан в нижнем регистре, иначе возвращает false
IsDigit	Возвращает true, если символ является десятичной цифрой, иначе возвращает false
IsLetter	Возвращает true, если символ является буквой алфавита, иначе возвращает false
IsSymbol	Возвращает true, если символ является знаком (не буквой и не цифрой), иначе возвращает false (например, для знака '+' этот метод вернет true)
IsLetterOrDigit	Возвращает true, если символ является буквой алфавита или десятичной цифрой, иначе возвращает false
IsPunctuation	Возвращает true, если символ является знаком пунктуации, иначе возвращает false
ToUpper	Преобразует переданный символ в верхний регистр
ToLower	Преобразует переданный символ в нижний регистр

Продемонстрируем работу этих методов на примере:

```
using System;
public class ChatMethodDemo
{
  public static void Main()
  {
    // true
    Console.WriteLine(Char.IsUpper('A'));

    // false
    Console.WriteLine(Char.IsDigit('X'));

    // true
    Console.WriteLine(Char.IsSymbol('^'));

    // true
    Console.WriteLine(Char.IsPunctuation('.'));

    // F
```

```
    Console.WriteLine(Char.ToUpper('f'));

    // y
    Console.WriteLine(Char.ToLower('Y'));
  }
}
```

Тип System.String

Для работы со строковыми данными (последовательностями символов) в MS .NET Framework используется тип *System.String* или его синоним *string*. Значение строковой переменной задается с помощью оператора присвоения, заключая последовательность символов строки в двойные кавычки. Например:

```
using System;
public class StringDemo
{
  public static void Main()
  {
    string Test = "Hello";
    Console.WriteLine(Test);
  }
}
```

В качестве символов строки могут выступать спецсимволы, описанные выше. При этом может возникнуть следующая проблема. Допустим, необходимо распечатать путь к каталогу (c:\new\test), расположенному на жестком диске компьютера. Рассмотрим пример:

```
using System;
public class StringDemoBad
{
  public static void Main()
  {
    Console.WriteLine("c:\new\test");
  }
}
```

Данная программа распечатает следующее:

```
c:
ew   est
```

Это произошло потому, что последовательности символов **\n** и **\t** были проинтерпретированы как спецсимволы перевода строки и табуляции соответственно. Есть два решения сложившейся проблемы. Первое – это использовать спецсимвол \\ для вывода символа обратного слеша:

```
Console.WriteLine("c:\\new\\test");
```

Второй – это использовать специальный символ @, который указывается в начале строки и сообщает компилятору, что все символы должны трактоваться как часть строки:

```
Console.WriteLine(@"c:\new\test");
```

Стоит обратить внимание, что все объекты класса System.String являются неизменяемыми. То есть у строки нельзя изменить длину, переставить символы местами и изменить регистр символов. Такой подход разработчиков MS .NET Framework к реализации класса System.String повышает производительность программы. Очевидно, что во многих программах необходимо производить преобразование строк. И когда это происходит, каждый раз создается новая строка, содержащая изменения исходной. Например:

```
using System;
public class StringDemo2
{
  public static void Main()
  {
    string test = @"c:\new\test";
    test = test.ToUpper();
    Console.WriteLine(test);
  }
}
```

В результате вызова метода **ToUpper** исходная строка, на которую указывает ссылка **test**, не будет преобразована к верхнему регистру, а будет создана новая преобразованная строка, на которую будет перенастроена ссылка **test**.

Сравнение строк

Для сравнения строк в классе *System.String* имеется несколько методов. Рассмотрим два наиболее часто используемых:

- **Compare** – это статический метод, который используется для лексикографического сравнения строк. Этот метод может принимать до семи различных параметров. В настоящий момент нас интересуют только три параметра: два из них – это сравниваемые строки и третий – это параметр логического типа, определяющий, нужно ли учитывать регистр символов при сравнении. Этот метод возвращает 0, если строки равны, число, большее 0, если первая строка больше второй, и число, меньшее 0, если вторая строка больше первой,
- **CompareOrdinal** – это статический метод, который используется для посимвольного сравнения кодов символов, входящих в строки. Этот метод возвращает такие же результаты, как и предыдущий.

Рассмотрим пример:

```
using System;
public class StringCompareDemo
{
  public static void Main()
  {
    string str1 = "ABC";
    string str2 = "abc";
    Console.WriteLine(String.Compare(str1, str2));
    Console.WriteLine(String.Compare(str1, str2, true));
    Console.WriteLine(String.CompareOrdinal(str1, str2));
  }
}
```

В результате работы этого примера, на экране будет распечатано три числа в следующем порядке: положительное, ноль, отрицательное. В качестве второго результата получается 0 потому, что строки "ABC" и "abc" без учета регистра символов состоят из одних и тех же символов, т.е. они равны. Метод **String.Compare(str1, str2)** вернул положительный результат (т.е. первая строка больше второй), т.к. в английском алфавите принято считать, что символы в верхнем регистре больше символов в нижнем. Таким образом, метод **Compare** сравнивает строки с учетом региональных стандартов. А вот метод **String.CompareOrdinal(str1, str2)** вернул противоположный результат, т.к. коды символов в нижнем регистре больше кодов символов в верхнем регистре, и это не зависит от региональных стандартов.

Работа с символами в строке

Для того чтобы выделить отдельный символ в строке, необходимо обратиться к строковой переменной, как к массиву, указав в квадратных скобках номер (нумерация начинается с нуля) интересующего символа. Например:

```
string test = "ABCD";
Console.WriteLine(test[1]); // напечатается символ B
```

Поиск в строке

Класс *System.String* содержит несколько методов поиска. Рассмотрим три из них:

- **IndexOf** – этот метод позволяет найти индекс первого вхождения указанной в качестве параметров строки или символа. Метод может принимать до трех параметров. Первый параметр – это строка или символ, который необходимо найти. Второй и третий параметры указывают, начиная с какого символа необходимо производить поиск и где этот поиск необходимо закончить. Метод возвращает номер символа, начиная с которого найдено совпадение с указанной строкой или символом. Если строка или символ не найдены, то метод возвращает -1,

- **StartsWith** – этот метод возвращает значение **true**, если строка начинается с подстроки, которая передается методу в качестве параметра. В противном случае возвращается **false**,
- **EndsWith** – аналогичен предыдущему методу, только проверяет конец строки на соответствие заданной подстроке.

Модификация строк

Рассмотрим четыре метода класса *System.String*, предназначенных для модификации строк:

- **Trim** – этот метод имеет две реализации. Первая реализация метода не принимает параметров и предназначена для удаления всех пробелов в начале и конце строки. Вторая реализация принимает переменное количество параметров типа *char* и удаляет все перечисленные символы в начале и в конце строки. В результате работы следующего примера на экране будут распечатаны две одинаковые строки:

```
using System;
public class StringTrimDemo
{
  public static void Main()
  {
    string str1 = "   My Test    ";
    string str2 = "abc My Testde";
    Console.WriteLine(str1.Trim());
    Console.WriteLine(str2.Trim('a', 'b', 'c', 'd', 'e', ' '));
  }
}
```

- **Remove** – этот метод предназначен для удаления заданного количества символов, начиная с заданной позиции. Первый параметр – это позиция в строке, начиная с которой нужно произвести удаление. Второй параметр – это количество удаляемых символов. Приведенный пример удаляет все символы "a", встречающиеся в строке:

```
using System;
public class StringRemoveDemo
{
  public static void Main()
  {
    string str = "aaaaaMaayaa Testaaa";
    int i = str.IndexOf('a');
    while (i != -1)
    {
      str = str.Remove(i, 1);
      i = str.IndexOf('a');
    }
    Console.WriteLine(str);
  }
```

```
}
```

- **Replace** – этот метод предназначен для замены всех указанных подстрок или символов, входящих в строку, на другую подстроку или символ. Рассмотрим предыдущий пример с использованием этого метода:

```
using System;
public class StringReplaceDemo
{
  public static void Main()
  {
    string str = "aaaaaMaayaa Testaaa";
    str = str.Replace("a","");
    Console.WriteLine(str);
  }
}
```

- **PadLeft** и **PadRight** – эти два метода позволяют дополнить строку пробелами или указанным символом слева и справа соответственно. Первый параметр – это число символов, до которого необходимо дополнить строку. Второй параметр – это символ, которым нужно дополнить строку. Если второй параметр не указан, то строка будет дополнена пробелами. Рассмотренный ниже пример выводит строку "####Test###":

```
using System;
public class StringPadDemo
{
  public static void Main()
  {
    string str = "Test";
    str = str.PadLeft(8, '#');
    str = str.PadRight(11, '#');
    Console.WriteLine(str);
  }
}
```

Литература

1. Библиотека классов (http://msdn.microsoft.com/library/rus/default.asp?url=/library/rus/cpref/html/cpref_start.asp)
2. Пространство имен System (http://msdn.microsoft.com/library/rus/default.asp?url=/library/rus/cpref/html/frlrfsystem.asp)
3. Пономарев В. Программирование на C++/C# в Visual Studio .NET 2003. – СПб.: БХВ-Петербург, 2004.