您的潜力，我们的动力

Microsoft
微软（中国）有限公司

Visual Studio 2005 系列课程

跟我一起学 Visual Studio 2005
C# 2.0 语言和编译器新增功能介绍

徐长龙

vsts_china@hotmail.com

msdn

MSDN Webcasts

前提

- 熟悉C# 1.1
- Level:200

目录

- 分部类 (Partial Classes)
- 迭代器(Iterators)
- 可空类型(Nullable Types)
- 匿名方法(Anonymous Methods)
- 属性访问器可访问性(Property Accessor Accessibility)
- 委托中的协变与逆变(Covariance and Contravariance in Delegates)
- 命名空间别名限定符(namespace alias qualifier)

分部类(Parital Classes)

- C# 2.0 可以将类、结构或接口的定义拆分到两个或多个源文件中，在类声明前添加partial关键字

```
class PartialTest
{
   string strFieldTest;
   int intFieldTest;

   public void DoTest()
   {
      Debug.Print("Test");
   }
}
```

```
partial class PartialTest
{
   string strFieldTest;
   int intFieldTest;
}
partial class PartialTest
{
    public void DoTest()
    {
        Debug.Print("Test");
    }
}
```

分部类(Parital Classes)

- 什么情况下使用分部类？
  - 处理大型项目时，使一个类分布于多个独立文件中可以让多位程序员同时对该类进行处理
  - 使用自动生成的源时，无需重新创建源文件便可将代码添加到类中。**Visual Studio** 在创建 **Windows** 窗体、**Web** 窗体时都使用此方法。您无需编辑 **Visual Studio** 所创建的文件，便可创建使用这些类的代码

分部类(Parital Classes)

- 使用 **partial** 关键字表明可在命名空间内定义该类、结构或接口的其他部分
- 所有部分都必须使用 **partial** 关键字
- 各个部分必须具有相同的可访问性，如 **public**、**private** 等
- 如果将任意部分声明为抽象的，则整个类型都被视为抽象的
- 如果将任意部分声明为密封的，则整个类型都被视为密封的
- 如果将任意部分声明为基类型，则整个类型都将继承该类

分部类(Parital Classes)

- 指定基类的所有部分必须一致，但忽略基类的部分仍继承该基类型
- 各个部分可以指定不同的基接口，最终类型将实现所有分部声明所列出的全部接口
- 在某一分部定义中声明的任何类、结构或接口成员可供所有其他部分使用

分部类(Parital Classes)

- 嵌套类型可以是分部的，即使它们所嵌套于的类型本身并不是分部的也如此（右方框）

```
class Container
{
   partial class Nested
   {
      void Test() { }
   }
   partial class Nested
   {
      void Test2() { }
   }
}
```

- 编译时将对分部类型定义的属性进行合并

```
[System.SerializableAttribute]
partial class Moon { }

[System.ObsoleteAttribute]
partial class Moon { }
```

=

```
[System.SerializableAttribute]
[System.ObsoleteAttribute]
class Moon { }
```

分部类(Parital Classes)

- 限制：
  - 要作为同一类型的各个部分的所有分部类型定义都必须使用 **partial** 进行修饰

```
public partial class A { }
//public class A { } // Error, must also be marked partial
```

  - **partial** 修饰符只能出现在紧靠关键字 **class**、**struct** 或 **interface** 前面的位置
  - 要成为同一类型的各个部分的所有分部类型定义都必须在同一程序集和同一模块（**.exe** 或 **.dll** 文件）中进行定义。分部定义不能跨越多个模块
  - 类名和泛型类型参数在所有的分部类型定义中都必须匹配。泛型类型可以是分部的。每个分部声明都必须以相同的顺序使用相同的参数名。

Demo

- 代码演示
  - 按 (Ctrl + H 在窗口模式和全屏模式之间切换)

目录

- 分部类 (Partial Classes)
- **迭代器(Iterators)**
- 可空类型(Nullable Types)
- 匿名方法(Anonymous Methods)
- 属性访问器可访问性(Property Accessor Accessibility)
- 委托中的协变与逆变(Covariance and Contravariance in Delegates)
- 命名空间别名限定符(namespace alias qualifier)

迭代器(Iterators)

- 什么是迭代器?
  - 迭代器是方法、get 访问器或运算符，它使您能够在类或结构中支持 foreach 迭代，而不必实现整个 IEnumerable 接口

```
List<int> objList = new List<int>();
for (int i = 0; i < 5; i++)
{
    objList.Add(i);
}

foreach (int i in objList)
{
    Debug.Print(i.ToString());
}
```

任何实现了IEnumerable,IEnumerator接口的类都可以进行foreach遍历操作

迭代器(Iterators)

- 迭代器例子

```
public class YieldTest
 {
    public static IEnumerable Power(int number, int exponent)
    {
       int counter = 0;
       int result = 1;
       while (counter++ < exponent)
       {
          result = result * number;
          yield return result;
       }
    }
 }
```

yield 关键字用于指定返回的值。到达 yield return 语句时，会保存当前位置。下次调用迭代器时将从此位置重新开始执行。

迭代器(Iterators)

- 迭代器是可以返回相同类型的值的有序序列的一段代码
- 迭代器可用作方法、运算符或 get 访问器的代码体
- 迭代器代码使用 yield return 语句依次返回每个元素。yield break 将终止迭代
- 可以在类中实现多个迭代器。每个迭代器都必须像任何类成员一样有唯一的名称，并且可以在 foreach 语句中被客户端代码调用
- 迭代器的返回类型必须为 IEnumerable、IEnumerator、IEnumerable<T> 或 IEnumerator<T>

Demo

- 迭代器
  - 代码演示
  - 按 (Ctrl + H 在窗口模式和全屏模式之间切换)

目录

- 分部类 (**Partial Classes**)
- 迭代器(Iterators)
- **可空类型(Nullable Types)**
- 匿名方法(Anonymous Methods)
- 属性访问器可访问性(Property Accessor Accessibility)
- 委托中的协变与逆变(Covariance and Contravariance in Delegates)
- 命名空间别名限定符(namespace alias qualifier)

可空类型(Nullable Types)

- 可空类型是 System.Nullable<T> 结构的实例
- 可空类型可以表示其基础值类型正常范围内的值，再加上一个 null 值
  - 例如:Nullable<Int32>，读作“可空的 Int32”，可以被赋值为 -2147483648 到 2147483647 之间的任意值，也可以被赋值为 null 值
  - Nullable<bool> 可以被赋值为 true 或 false，或 null

```
Nullable<int> num = null;
```

=

```
int? num = null;
```

可空类型(Nullable Types)

- 在处理数据库和其他包含可能未赋值的元素的数据类型时，将 null 赋值给数值类型或布尔型的功能特别有用。
  - 例如，数据库中的布尔型字段可以存储值 true 或 false，或者，该字段也可以未定义
  - 数据库中的日期类型可以为null，现在C#日期也可以为null

```
//可空的布尔型
bool? b = null;

//可空的日期型
DateTime? d = null;
```

可空类型(Nullable Types)

- System.Nullable<T>结构
  - public **bool HasValue** { get; }
    - 获取一个值，指示当前的 System.Nullable<T> 对象是否有值
  - public **T Value** { get; }
    - 获取当前的 System.Nullable<T> 值
  - public **T GetValueOrDefault()**
    - 检索当前 System.Nullable<T> 对象的值，或该对象的默认值

可空类型(Nullable Types)

- 使用 ?? 运算符分配默认值，当前值为空的可空类型被赋值给非空类型时将应用该默认值

```
int? x = null;
int y = x ?? -1;
```

- 不允许使用嵌套的可空类型
  - 将不编译下面一行：Nullable<Nullable<int>> n

Demo

- 可空类型
  - 代码演示
  - 按 (Ctrl + H 在窗口模式和全屏模式之间切换)

目录

- 分部类 (**Partial Classes**)
- 迭代器(Iterators)
- 可空类型(Nullable Types)
- **匿名方法(Anonymous Methods)**
- 属性访问器可访问性(Property Accessor Accessibility)
- 委托中的协变与逆变(Covariance and Contravariance in Delegates)
- 命名空间别名限定符(namespace alias qualifier)

匿名方法 (Anonymous Methods)

- 在 2.0 之前的 C# 版本中，声明委托的唯一方法是使用命名方法

```
this.Load += new System.EventHandler(this.Form1_Load);

private void Form1_Load(object sender, EventArgs e)
{
    MessageBox.Show("Form1_Load!");
    ...
}
```

```
public delegate void EventHandler(object sender, EventArgs e);
```

- 要将代码块传递为委托参数，创建匿名方法则是唯一的方法

```
This.Load += delegate(System.Object o, System.EventArgs e) {MessageBox.Show("Form1_Load!"); };
```

匿名方法 (Anonymous Methods)

- 如果使用匿名方法，则不必创建单独的方法，因此减少了实例化委托所需的编码系统开销
- 启动新线程范例：

```
void StartThread()
{
    System.Threading.Thread t1 = new System.Threading.Thread
      (delegate()
          {
              System.Console.Write("Hello, ");
              System.Console.WriteLine("World!");
          });
    t1.Start();
}
```

匿名方法 (Anonymous Methods)

- 如果局部变量和参数的范围包含匿名方法声明，则该局部变量和参数称为该匿名方法的外部变量或捕获变量。例如，下面代码段中的 n 即是一个外部变量：

```
int n = 0;
Del d = delegate() { System.Console.WriteLine("Copy #:{0}", ++n); };
```

- 与局部变量不同，外部变量的生命周期一直持续到引用该匿名方法的委托符合垃圾回收的条件为止。对 n 的引用是在创建该委托时捕获的。
- 匿名方法不能访问外部范围的 ref 或 out 参数
- 在匿名方法块中不能访问任何不安全代码

Demo

- 匿名方法
  - 代码演示
  - 按 (Ctrl + H 在窗口模式和全屏模式之间切换)

目录

- 分部类 (**Partial Classes**)
- 迭代器(Iterators)
- 可空类型(Nullable Types)
- 匿名方法(Anonymous Methods)
- **属性访问器可访问性(Property Accessor Accessibility)**
- 委托中的协变与逆变(Covariance and Contravariance in Delegates)
- 命名空间别名限定符(namespace alias qualifier)

属性访问器可访问性(Property Accessor Accessibility)

- 属性或索引器的 get 和 set 部分称为“访问器”。默认情况下，这些访问器具有相同的可见性或访问级别：其所属属性或索引器的可见性或访问级别

```
string str = "";
public string TestProp
{
    get { return str; }
    set { str = value; }
}
```

```
string[] astr = new string[] { "A", "B", "C" };
public string this[int i]
{
    get { return astr[i]; }
    set { astr[i] = value; }
}
```

- 有时限制对其中某个访问器的访问会很有用。通常是在保持 get 访问器可公开访问的情况下，限制 set 访问器的可访问性

```
string str = "";
public string TestProp
{
    get { return str; }
    protected set { str = value; }
}
```

```
string[] astr = new string[] { "A", "B", "C" };
public string this[int i]
{
    get { return astr[i]; }
    protected set { astr[i] = value; }
}
```

属性访问器可访问性(Property Accessor Accessibility)

- 不能对接口或显式接口成员实现使用访问器修饰符
  - 使用访问器实现接口时，访问器不能具有访问修饰符。但是，如果使用一个访问器（如 get）实现接口，则另一个访问器可以具有访问修饰符

```
public interface ISomeInterface
{
    int TestProperty
    {
        get;
    }
}

public class TestClass : ISomeInterface
{
    public int TestProperty
    {
        get { return 10; }

        protected set { }
    }
}
```

接口不能使用修饰符

实现接口不能使用修饰符

接口未定义的访问器可以使用修饰符

属性访问器可访问性(Property Accessor Accessibility)

- 仅当属性或索引器同时具有 set 和 get 访问器时，才能使用访问器修饰符。这种情况下，只允许对其中一个访问器使用修饰符
- 访问器的可访问性级别必须比属性或索引器本身的可访问性级别具有更严格的限制
- 如果属性或索引器具有 override 修饰符，则访问器修饰符必须与重写的访问器的访问器（如果有的话）匹配

```
public class Parent
{
   public virtual int TestProperty
   {
      // Notice the accessor accessibility level.
      protected set { }

      // No access modifier is used here.
      get { return 0; }
   }
}
```

```
public class Kid : Parent
{
   public override int TestProperty
   {
      // Use the same accessibility level
      // as in the overridden accessor.
      protected set { }

      // Cannot use access modifier here.
      get { return 0; }
   }
}
```

Demo

- 属性访问器可访问性
  - 代码演示
  - 按 (Ctrl + H 在窗口模式和全屏模式之间切换)

目录

- 分部类 (**Partial Classes**)
- 迭代器(Iterators)
- 可空类型(Nullable Types)
- 匿名方法(Anonymous Methods)
- 属性访问器可访问性(Property Accessor Accessibility)
- **委托中的协变与逆变(Covariance and Contravariance in Delegates)**
- 命名空间别名限定符(namespace alias qualifier)

委托中的协变与逆变(Covariance and Contravariance in Delegates)

- 委托中的协变
  - 当委托方法的返回类型具有的派生程度比委托签名更大时，就称为协变委托方法。因为方法的返回类型比委托签名的返回类型更具体，所以可对其进行隐式转换。这样该方法就可用作委托。
  - 协变使得创建可被类和派生类同时使用的委托方法成为可能。

```
class Animals{}
class Dogs : Animals {}
class Program
{
   public delegate Animals HandlerMethod();
   public static Animals FirstHandler()  {   return null;  }
   public static Dogs SecondHandler()  {  return null;  }
   static void Main()
   {
      HandlerMethod handler1 = FirstHandler;
      // Covariance allows this delegate.
      HandlerMethod handler2 = SecondHandler;
   }  }
```

委托中的协变与逆变(Covariance and Contravariance in Delegates)

- 委托中的逆变
  - 当委托方法签名具有一个或多个参数，并且这些参数的类型派生自方法参数的类型时，就称为逆变委托方法。因为委托方法签名参数比方法参数更具体，因此可以在传递给处理程序方法时对它们进行隐式转换。
  - 这样逆变使得可由大量类使用的更通用的委托方法的创建变得更加简单。

```
class Animals{}
class Dogs : Animals {}
class Program
{
    public delegate void HandlerMethod(Dogs sampleDog);
    public static void FirstHandler(Animals elephant) { }
    public static void SecondHandler(Dogs sheepDog) { }
    static void Main(string[] args)
    {
        // Contravariance permits this delegate.
        HandlerMethod handler1 = FirstHandler;
        HandlerMethod handler2 = SecondHandler;
    }  }
```

Demo

- 委托中的协变与逆变
  - 代码演示
  - 按 (Ctrl + H 在窗口模式和全屏模式之间切换)

目录

- 分部类 (**Partial Classes**)
- 迭代器(Iterators)
- 可空类型(Nullable Types)
- 匿名方法(Anonymous Methods)
- 属性访问器可访问性(Property Accessor Accessibility)
- 委托中的协变与逆变(Covariance and Contravariance in Delegates)
- **命名空间别名限定符(namespace alias qualifier)**

命名空间别名限定符 (namespace alias qualifier)

- 当成员可能被同名的其他实体隐藏时，能够访问全局命名空间中的成员非常有用
- 别名限定符是双冒号(::)
- 命名空间别名限定符可以是 global。这将调用全局命名空间中的查找，而不是在别名命名空间中

```
class System{}

global::System.Console.WriteLine("Hello World");
```

```
using colAlias = System.Collections;

// Searching the alias:
colAlias::Hashtable test = new colAlias::Hashtable();
```

Demo

- 命名空间别名限定符
  - 代码演示
  - 按 (Ctrl + H 在窗口模式和全屏模式之间切换)

小结

- 分部类 (**Partial Classes**)
- 迭代器(Iterators)
- 可空类型(Nullable Types)
- 匿名方法(Anonymous Methods)
- 属性访问器可访问性(Property Accessor Accessibility)
- 委托中的协变与逆变(Covariance and Contravariance in Delegates)
- 命名空间别名限定符(namespace alias qualifier)

您的潜力，我们的动力

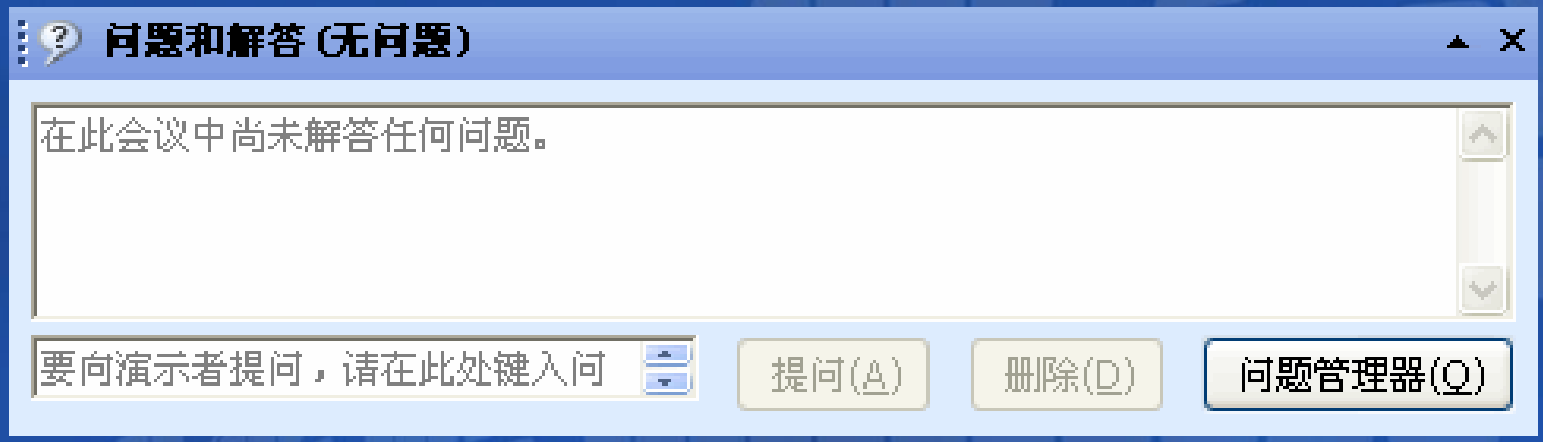
Microsoft
微软（中国）有限公司

获取更多MSDN资源

- MSDN中文网站
  http://www.microsoft.com/china/msdn
- MSDN中文网络广播
  http://www.msdnwebcast.com.cn
- MSDN Flash
  http://www.microsoft.com/china/newsletter/case/msdn.aspx
- MSDN开发中心
  http://www.microsoft.com/china/msdn/DeveloperCenter/default.mspx

Question&Answer

- 如需提出问题，请单击“提问”按钮并在随后显示的浮动面板中输入问题内容。一旦完成问题输入后，请单击“提问”按钮。




您的潜力，我们的动力
Microsoft
微软（中国）有限公司
Microsoft
msdn
MSDN Webcasts