《现代软件开发——使用.NET与C#》系列第四讲

C# 中类的设计

俞晖
MSDN讲师
dolphin@vip.163.com

《现代软件开发——使用.NET与C#》系列

- 回顾上节课的内容（*现代面向对象的程序设计*）
- 本节课讲述在**C#**中的类的设计
- **LEVEL 200**

Date	Topic
2005年7月7日 10:00-11:30	*C#中类的设计*
2005年7月21日	*C# 中的异常处理*
.	.
.	.

日程

- 类的设计
  - 构造函数
  - 属性
  - 索引器
  - 代理/委托
  - 事件
  - **Clone***
  - **Close & Dispose***
  - **XML 文档索引***

构造函数 Constructor （1）

- 类中特殊的方法，多用于初始化实例的数据成员，在实例化 new 时被自动调用
- 函数名与类名总是相同，没有返回值

```
BankCustomer  c;
c = new BankCustomer();

c = new BankCustomer("kim", "lee", 1000.0M);
```

X

```
public class BankCustomer
{
    private string   FirstName;
    private string   LastName;
    private decimal  Balance;
    .
    .
    .

    public BankCustomer(string fn, string ln, decimal bal)
    {
        this.FirstName = fn;
        this.LastName  = ln;
        this.Balance   = bal;
    }
}
```

构造函数 Constructor （2）

- 通过重载，构造函数可以任意多

```
public class BankCustomer
{

    public BankCustomer(string fn, string ln, decimal bal)
    {
       this.FirstName = fn; this.LastName = ln; this.Balance = bal;
    }

    public BankCustomer ()
    {
       this.FirstName = “”; this.LastName = “”; this.Balance = 0.0M;
    }

    public BankCustomer(string fn, string ln)   // start with 0.0 balance
    {
       this.FirstName = fn; this.LastName = ln; this.Balance = 0.0M;
    }

    public BankCustomer(System.IO.StreamReader reader)   // initialize via stream (e.g. file)
    {
       this.FirstName = reader.ReadLine();
       this.LastName  = reader.ReadLine();
       this.Balance   = decimal.Parse( reader.ReadLine() );
    }
```

构造函数 Constructor （3）

- 构造函数间的调用

```
public class BankCustomer
{
    public BankCustomer(string fn, string ln, decimal bal)
    {
       this.FirstName = fn;
       this.LastName = ln;
       this.Balance = bal;
    }

    public BankCustomer(string fn, string ln)   // start with 0.0 balance
    : this(fn, ln, 0.0M)
    { }

    public BankCustomer(System.IO.StreamReader reader)   // initialize via stream (e.g. file)
    : this(reader.ReadLine(), reader.ReadLine(), decimal.Parse(reader.ReadLine()))
    { }
```

各种构造函数的应用

demo

- 构造函数
  Constructor

构造函数 Constructor（4）

- 类中特殊的方法，多用于初始化实例的数据成员，在实例化 new 时被自动调用
  - 函数名与类名总是相同
  - 没有返回值
  - 任意数量（通过重载Overload）
  - 构造函数间的调用
  - 访问修饰符的作用（public protected private）
- 如果没有显式定义，那么系统提供一个不带任何参数的public的构造函数
- 静态构造函数

属性 Property（1）

- 内部看像函数，外部看像字段
- 读/写由get/set存取器来进行
  - 没有参数列表
- **Adv:** 可以像给公有字段赋值一定轻松，同时允许通过get/set来控制对属性的访问

属性 Property （2）


```
bc.Balance = bc.Balance – amt;
```

```
public class BankCustomer
{
    .
    .
    .
   private decimal  m_Balance;   // field
   public  decimal  Balance      // property
   {
      get { return this.m_Balance; }
      set
      {
         if (value < 0.0M)   // new balance cannot be negative!  So ignore…
           ;
         else   // update balance to new value
           this.m_Balance = value;
      }
   }
```

索引器 Indexer（1）

- 索引器就是一类特殊的属性，通过它们你就可以像处理数组一样处理对象。
- 索引器通常在代表元素集合的类中定义

```
public class BankCustomer
{
        private double balance;
        private Person[ ] relatives;
        public BackCustomer()
        {
            relatives = new Person[100];
        }
        ...
}
```

索引器 Indexer （2）

- 索引器类似于属性
- 属性名是***this***，意思是回引类的当前实例，参数列表包含在方括号而非括号之内。

```
public Person this [int index]
{
      get
      {
          return relatives [index];
      }
      set
      {
          if (value != null)
          {
              relatives [index] = value;
          }
      }
}
```

```
BankCustomer bc = new
        BankCustomer();
Bc[10] = new Person("Name");
```

属性和索引器

demo

- 了解属性的使用方法
- 了解索引器的使用方法

委托 Delegate （1）

- 想象成C++中的函数指针，但不同点在于delegate完全面向对象的——既封装方法又封装对象实例
- 定义委托实际上是定义一个类型的委托，不是一个具体的实例。
- 委托类型指定它代表的方法的返回类型和参数表
- 它代表具有相同参数列表和返回类型的任何方法

委托 Delegate （2）

- 委托的声明方式和方法的声明相似，包括的是**delegate**关键字
- 声明中必须包括委托所表示的方法的返回类型和参数列表

```
<modifiers> delegate <return_type> <delegate_name> ( argument_list )

public delegate bool ProcessAnything(double d);
```

- 创建委托实例 —— new关键字

```
ProcessAnything pa = new
    ProcessAnything(account.Withdraw);
```

- 括号里面是实例方法，此方法必须和代理声明时的返回类型和参数列表相同

委托 Delegate （3）

- 委托的调用时通过输入委托实例的名称和要传递给委托所表示的方法的参数

```
public class MoneyCompute
{
      public static double Compute(double t, DelegateCompute dc)
      {
          return dc(t);
      }
}
```

```
public delegate double DelegateCompute(double x);
```

委托 Delegate （4）

- 多播委托——引用多个方法的委托，它连续调用每个方法。
- 为了把委托的单个实例合并为一个多播委托，委托必须是同类型的，返回类型必须是**void**，不能带输出参数（可以带引用参数）。
- 多播委托应用于事件模型中

委托相关Demo

- 一般委托
- 多播委托

事件 Event （1）

- C#中使用委托模型来实现事件
- 事件处理方法不必在将生成事件的类中定义
- 需要做的事情就是把事件源和事件处理程序结合起来。
- 使用事件处理委托，简称事件委托可以定义为生成事件的类的一个成员
- 事件委托是多播的

事件 Event （2）

- 事件委托的形式：

```
public delegate void MouseEventHandler (object source, EventArgs e);
```

object source：事件源

EventArgs： System.EventArgs 类的实例或派生类实例，它包含事件的另外的信息

- .NET Framework定义了大量的事件处理委托
  - public delegate void KeyEventHandler(object source, KeyEventArgs args)
  - public delegate void MouseEventHandler(object source, MouseEventArgs args)
  - ……

事件 Event （3）

- 用户自定义事件委托

```
public delegate void NameEventHandler (object source, NewEventArgs e);
```

- 必须自己来定义NameEventArgs类

- 创建事件委托的实例
  - 不使用new关键字，使用event关键字

```
public event NameEventHandler handler;
```

事件 Event （4）

- 触发事件
  - 把事件委托的一个实例定义为类的成员
  - 确定何时生成事件代码
  - 定义生成提供事件的**EventArgs**对象的代码

事件Demo

demo

- 自定义事件
  - 定义NameList类，当调用add()方法的时候，查看事件处理机（EventHandler）是否存在，如果存在，产生事件委托，由处理机处理。

事件 Event （5）

- 优化

```
.field private class NameListEventHandler nameListEvent
.method public hidebysig specialname instance void
      add_nameListEvent(class NameListEventHandler ‘value’) cil managed
      synchronized
{
      //…
}//end of method NameList::add_nameListEvent

.method public hidebysig specialname instance void
      remove_nameListEvent(class NameListEventHandler ‘value) cil managed
      synchronized
{
      //…
}//end of method NameList::remove_nameListEvent
```

其它参考资源

- **Webcast:**
  - http://www.microsoft.com/china/msdn/events/webcasts/shared/Webcast/MSDNWebCast.aspx
- **Resources:**
  - MSDN Library
    http://www.microsoft.com/china/MSDN/library/default.mspx
  - GotDotNet

    http://www.gotdotnet.com

MSDN flash:

- http://www.microsoft.com/china/newsletter/case/MSDN.asp

感谢大家的参与!

- 非常感谢大家的参与
- 您的反馈对我们来说是非常重要的

Date	Topic
2005年6月30日	*现代面向对象的程序设计*
2005年7月7日10:00-11:30	*C#中类的设计*
2005年7月21日	C#中异常处理
.	.


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

Q&A

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。