《现代软件开发——使用.NET与C#》系列第十讲

C# 中的一些设计技巧

终结篇

俞晖
MSDN Online Manager
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（*面向组件的程序设计*）
- 本节课讲述在C#中的一些设计技巧
- LEVEL 200

Date	Topic
2005/10/13	*C#中的一些设计技巧（终结篇）*

日程

- 编码习惯
- 委托再议
- Equals
- Clone
- Close & Dispose
- XML 文档索引
- String相关知识

编码习惯

1. 命名规范，接口用I开头做前缀，异常类使用Exception作为其后缀。

```
public class MyClass
{
   int myNum=0
   public MyMethod(int refObject){}
}

interface IMyInterface
{...}

public class MyEventException
{...}
```

2. 使用有意义的变量名称和名称空间。有返回值的方法GetMyObjectState()。
3. 所有的成员变量都应该声明在顶部，同时使用一个空行来将他们和属性以及方法分开。
4. 总是将大括号放在一个新行上。

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（历史）
- 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）

- <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);
```

实例


```
using System;
using System.Threading;

public class Student
{

 public  delegate string AdviseDelegate(int score);

public AdviseDelegate AdviseDelegateInstance;


                    private int score;

                        public void SetScore(int value)
   {
      if (value > 100 || value < 0)
      {
          Console.Out.WriteLine(“Wrong");
      }
      else
      {
          score = value;
          if (AdviseDelegateInstance!= null)
          {
              string result=AdviseDelegateInstance(score);
              Console.Out.WriteLine(“Result"+result);
          }
    }
  }
```

实例


```
public class Teacher
{
    public string Advise(int score)
    {
        if(score<60)
        {
            Console.Out.WriteLine(score+“Add Oil");
            return “Not Pass…";
        }
        else
        {
            Console.Out.WriteLine(score+“Good not Perfect");
            return “Pass!";
        }
    }
}
```

实例


```
class MainClass
{
    [STAThread]
    static void Main(string[] args)
    {
        Teacher teacher=new Teacher();

        Student s=new Student();

        s.AdviseDelegateInstance=new Student.AdviseDelegate(teacher.Advise);

        Console.Out.WriteLine("Student got 49");

        s.SetScore(49);

        Console.Out.WriteLine("Student got 87");

        s.SetScore(87);

        Console.ReadLine();
    }
}
```

Equals()

- 考虑覆写 *Equals(object obj)*
  - .NET 提供了默认的实现方法（引用指向同一对象）
  - .NET 将在比较和查询时使用这个方法
  - 实现自己比较的“相等”定义，你需要覆写这个方法。

```
BankCustomer  bc1, bc2;
:
:
if (bc1.Equals(bc2))
  ...;
```

```
ArrayList  customers;
customers = new ArrayList();
:
:
// we are looking for a particular customer…
BankCustomer  bc = ...;
if (customers.IndexOf(bc) >= 0)  // found it!
  ...;
```

自定义Equals

- 如果两个客户姓名相等，我们认为对象是相等的

```
public class BankCustomer
{
   public string  FirstName;
   public string  LastName;
   .
   .
   .

   public override bool Equals(object obj)
   {
      if (obj == null)
        return false;
      if ((obj.GetType().Equals(this.GetType())) == false)
        return false;

      BankCustomer  other;
      other = (BankCustomer) obj;
      return this.FirstName.Equals(other.FirstName) &&
             this.LastName.Equals(other.LastName);
   }
```

GetHashCode()

- 如果覆写了 Equals，也需要覆写 GetHashCode （警告）
  - 如果两个对象 *Equals*, GHC 必须返回相同的值
  - 根据Equals定义字段来定义GHC

```
public class BankCustomer
{
   public string  FirstName;
   public string  LastName;
   .
   .
   .

   public override int GetHashCode()
   {
      return this.FirstName.GetHashCode() + this.LastName.GetHashCode();
   }
```

Hashtable

- *Hashtable* 提供高效的搜索
  - 基于 (key, value) pairs （键/值）对– 键必须是唯一的
  - 可以通过搜索key来查找

```
using SC = System.Collections;

SC.Hashtable  customers;
customers = new SC.Hashtable();

customers.Add( 12345, new BankCustomer() );   // key, value
customers.Add( 67890, new BankCustomer() );
.
.
.
BankCustomer  c;
c = (BankCustomer) customers[67890];   // search by key

foreach (SC.DictionaryEntry hte in customers)
  ((BankCustomer)hte.Value).Balance += 250.00M;
```

IDisposable

- 一些对象需要清理：

```
System.IO.StreamWriter  file;
file = new System.IO.StreamWriter("file.txt");
file.WriteLine(...);
.
.
.
file.Dispose();    // flush writes & free file handle!
```

- *Dispose()*
  - 类中是否"抓住"资源？
  - 如果是：
    - a) 重新设计类
    - b) 实现IDisposable接口，并提供Dispose & Close 方法

例子

- 假设我们需要日志信息（logging message)：
  - #1:
    - 不“抓住”资源

```
public class Logger
{
   public void LogIt(string msg, string filename)
   {
      // open file, append msg, close file
   }
}
```

  - #2:
    - 需要在log时，保持文件打开状态

```
public class Logger
{
   private System.IO.StreamWriter  logfile;

   public Logger(string filename)  // constructor
   { this.logfile = new System.IO.StreamWriter(filename, true); }

   public void LogIt(string msg, string filename)
   { logfile.WriteLine(msg); }
}
```

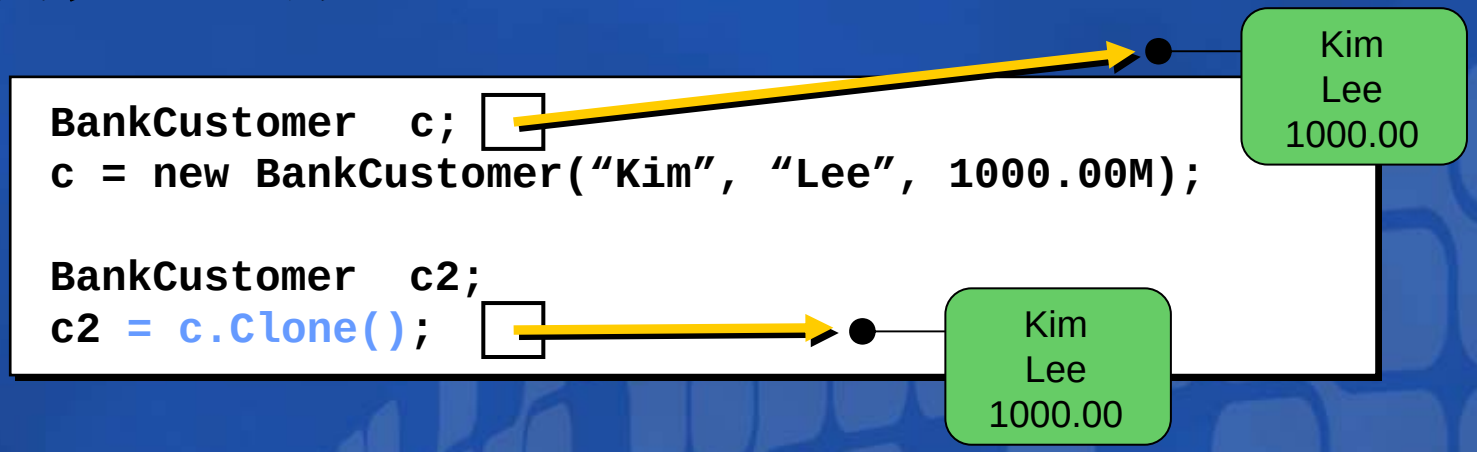
解决方案

- 加入*Dispose*:

```
public class Logger : IDisposable
{
   private System.IO.StreamWriter  logfile;

   public Logger(string filename)  // constructor
   { this.logfile = new System.IO.StreamWriter(filename, true); }

   public void LogIt(string msg, string filename)
   { logfile.WriteLine(msg); }

   public void Dispose()
   { logfile.Close(); }
}
```

ICloneable

- 如果需要实现克隆功能，那么要实现 ICloneable 接口
  - 实现clone方法

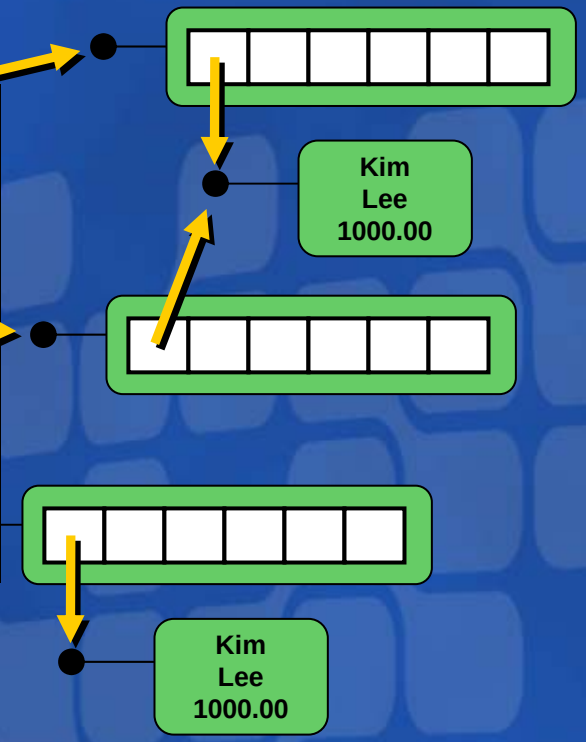


    - *你需要确定：浅表复制还是深复制*

Shallow vs. Deep

- *shallow* — 复制顶级(top-level)对象
- *deep* 复制对象和子对象

```
BankCustomer[]  customers;
.
.
.
BankCustomer[]  shallow;
shallow = customers.Clone();

BankCustomer[]  deep;
deep = customers.Clone();
```



- *参看class doc*

例子

- Shallow copies of *BankCustomer* objects:

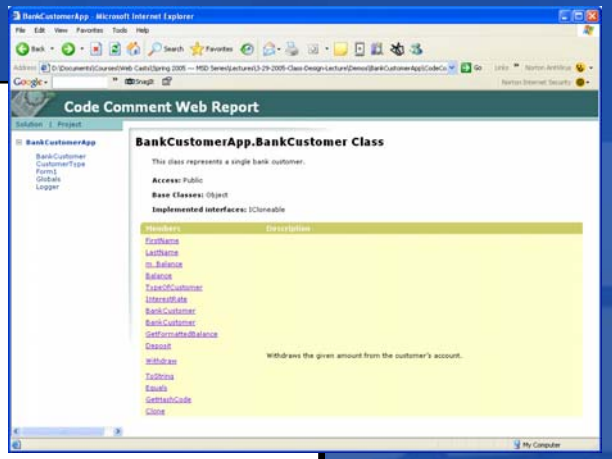
```
public class BankCustomer : ICloneable
{
    .
    .
    .
    public object Clone()
    {
       return new BankCustomer(this.FirstName, this.LastName, this.Balance);
    }
```

  - 相当于：

```
public class BankCustomer : ICloneable
{
    .
    .
    .
    public object Clone()
    {
       return this.MemberwiseClone();   // creates shallow copy for you
    }
```

文档?

- 使用XML注释，自动生成 class-level 文档!




```
/// <summary>
/// This class represents a single bank customer.
/// </summary>
public class BankCustomer
{
    .
    .
    .

    /// <summary>
    /// Withdraws the given amount from the customer's account.
    /// </summary>
    /// <param name="amt">amount to withdraw (must be >= 0)</param>
    /// <exception cref="System.Exception">Thrown if sufficient funds are not available</exception>
    public void Withdraw(decimal amt)
    {
       if (amt > this.m_Balance)
         throw new Exception("Insufficient funds!");
       else
         this.m_Balance -= amt;
    }
```

如何生成文档?

- **Visual Studio .NET:**
  - Tools menu, “Build Comment Web Pages”
  - Set project property (Config, Build) to generate .xml file

- **NDoc:**
  - http://ndoc.sourceforge.net/
  - open-source tool for generating docs
  - turns .xml file into MSDN-style docs (other formats too)

String相关知识

- string是不可变的对象。
  - 字符串连接操作并不更改当前字符串，只是创建并返回新的字符串，速度慢。
- StringBuilder的字符串连接
  - 频繁进行字符串连接操作时，使用StringBuilder类来改善性能，连接操作越频繁，差别越明显。
- 字符串驻留
  - public static Intern(String str)
  - public static isIntern(String str)

Feedback

- DEV roadmap
- C# Study Suggestion
- VS2003 VS2005
- Delegate

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

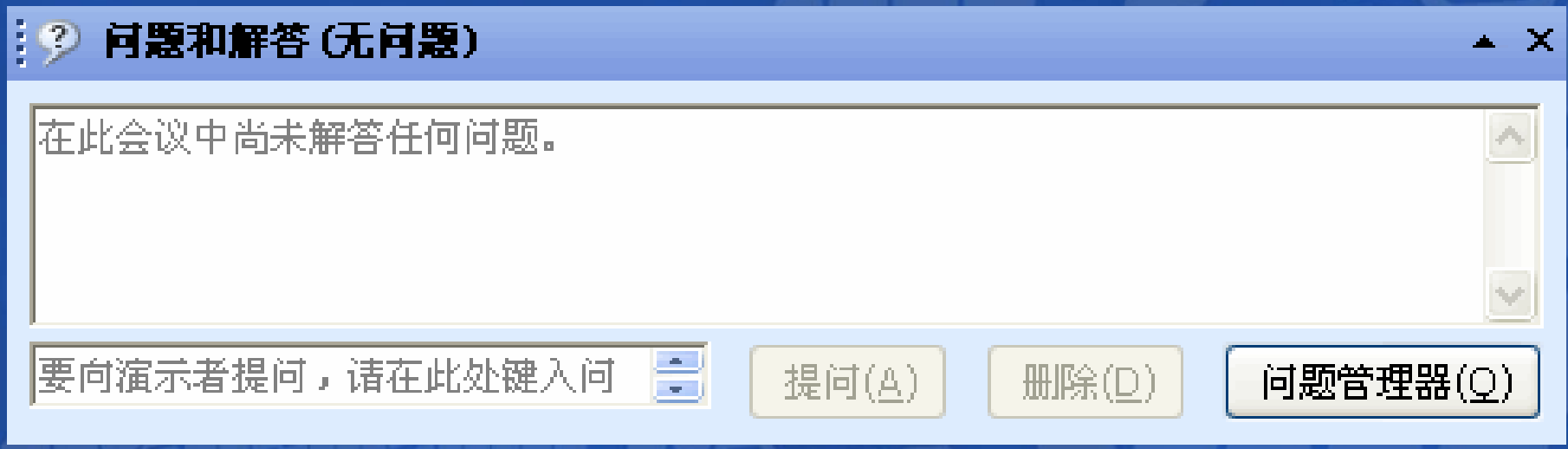
    http://www.gotdotnet.com

MSDN flash:

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

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。



感谢大家的参与!

- 非常感谢大家的支持
- 您的反馈对我们来说是非常重要的

- 祝各位朋友走出属于自己的**DEV**大道

剧终


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