[**.NET中的设计模式---代理模式(Proxy)**](http://www.cnblogs.com/niyw/archive/2010/08/16/1800980.html)

　　一个称职的代理人是什么呢？显然它不仅要向客户提供供应者完善的服务，也要对这种服务进行有效的控制，同时又要方便客户使用。设计模式中的代理模式正式要充当这个代理人，那么.NET中该如何实现该代理人呢？且看下文。

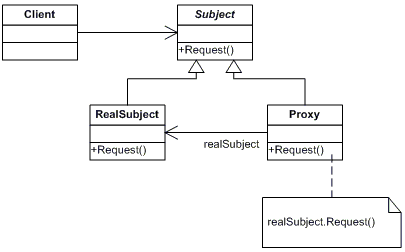
　　代理模式是GOF23种设计模式中结构型模式之一，意图是**为其他对象提供一种代理以控制对这个对象的访问**，UML类图如下：

图1

由此图也许我们看不出代理模式意图要表达真实意思，那么我们在看一副图，如下：

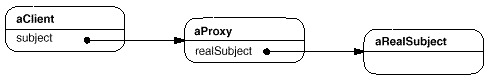


图2

由此图联系其意图，我们不难理解代理模式的意图了。图中aClient是客户，要访问aRealSubject对象，然而由于某种原因我们需要对这种访问进行控制。因此我们引入了代理aProxy，代理aProxy提供了一个虚拟的aRealSubject供aClient访问，并且能够对这种访问进行控制，而对aClient来说，这个虚拟的对象与真实的对象完全一样。

　　上面我们明白了什么是代理模式，那么在.NET中如何实现呢？根据图1也许你能很快写出一段示例代码，<C#设计模式>一书中就是这么实现的。但是示例代码并不能解决我们的实际问题，我们也并不需要关注UML图中的每一个细节，更不一定必须按UML图中的结构类实现。那么在实际.NET开发中该如何实现代理模式呢？幸运的是微软在.NET中为我们提供了实现代理模式的基本框架，我们稍作改动即可在代码中直接使用。

　　.NET中与实现代理模式有关的类有以下几个:

1. **ContextBoundObject**：定义所有上下文邦定类的基类；
2. **RealProxy**：提供代理的基本功能；
3. **ProxyAttribute**：指示对象类型需要自定义代理；

　　如果我们要定义一个需要被代理的类，那么仅需要从ContextBoundObject继承即可，如下：

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | [ProxyTag]  public  class RealClass : ContextBoundObject {      public RealClass() {          Console.WriteLine("构造一个RealClass!");      }  } |

　　如果要为RealClass定义代理，那么需要从RealProxy继承自己的代理类，并重载Invoke方法：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | public override IMessage Invoke(IMessage msg) {      //自定义构造对象前进行处理      PreProcess(msg);      IMessage retMsg=null;      if (msg is IConstructionCallMessage) {          IConstructionCallMessage ccm = (IConstructionCallMessage)msg;            retMsg = EnterpriseServicesHelper.CreateConstructionReturnMessage(ccm, (MarshalByRefObject)this.GetTransparentProxy());      }      //自定义构造对象后进行处理      PostProcess(retMsg);      return retMsg;  } |

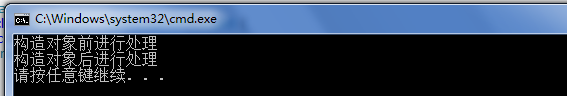
　　在定义完代理类后，需要定义二者关联的Attribute，才能完成需要的代理，此时需要从ProxyAttribute继承实现自己的标签，并重载CreateInstance方法：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | public override MarshalByRefObject CreateInstance(Type serverType) {      MarshalByRefObject mobj = base.CreateInstance(serverType);      if (aspectManaged) {          RealProxy realProxy = new ProxyClass(serverType, mobj);          MarshalByRefObject retobj = realProxy.GetTransparentProxy() as MarshalByRefObject;          return retobj;      }      else {          return mobj;      }  } |

　　经过以上步骤，我们就完成了一个可以直接使用的代理类。写一行测试代码验证一下我们例子。

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | try {      RealClass pc = new RealClass();  }  catch (Exception ex) {      Console.WriteLine(ex.Message);  } |

运行结果为：



根据类的定义，我们在创建一个RealClass实例时会输出“构造一个RealClass!”，而实际测试结果如上。由此可知代理RealProxy达到了我们预期的目的。

注：

　　1，阅读此文需要理解代理模式，如有不明的地方，请留言。

　　2，如需Demo,请留下邮箱（不知道怎么把Demo传上来 :( ）。

分类: [.NET与设计模式](http://www.cnblogs.com/niyw/category/255528.html)

标签: [.NET](http://www.cnblogs.com/niyw/tag/.NET/), [代理模式](http://www.cnblogs.com/niyw/tag/%E4%BB%A3%E7%90%86%E6%A8%A1%E5%BC%8F/), [设计模式](http://www.cnblogs.com/niyw/tag/%E8%AE%BE%E8%AE%A1%E6%A8%A1%E5%BC%8F/)