**[.NET中AOP的几种实现方案](http://www.cnblogs.com/tenghoo/archive/2010/12/22/aop.html)**

AOP在.NET中的应用，个人也属于学习阶段，欢迎大家拍砖！

本文的例子模拟用户注册的场景，主要通过代码演示几种方案的实现方式。

**静态代理**

通过代理模式实现静态代理，大家一看代码基本就明白了。

用户注册接口和实现

[复制代码](javascript:void(0);)

    public interface IUserProcessor  
    {  
        void RegUser(User user);  
    }  
    public class UserProcessor : IUserProcessor  
    {  
        public  void RegUser(User user)  
        {  
            Console.WriteLine("用户已注册。Name:{0},PassWord:{1}", user.Name, user.PassWord);  
        }  
    }

[复制代码](javascript:void(0);)

通过静态编写代码的方式，装饰上面的用户注册

[复制代码](javascript:void(0);)

public class UserProcessorDecorator:IUserProcessor  
    {  
        public IUserProcessor UserProcessor { get; set; }  
        public UserProcessorDecorator(IUserProcessor userprocessor)  
        {  
            UserProcessor = userprocessor;  
        }  
        public  void RegUser(User user)  
        {  
            PreProceed(user);  
            UserProcessor.RegUser(user);  
            PostProceed(user);  
        }  
        public void PreProceed(User user)  
        {  
            Console.WriteLine("方法执行前");  
        }  
  
        public void PostProceed(User user)  
        {  
            Console.WriteLine("方法执行后");  
        }  
    }

[复制代码](javascript:void(0);)

客户端调用

[复制代码](javascript:void(0);)

    public class Client  
    {  
        public static void Run()  
        {  
            try  
            {  
                User user = new User() { Name = "lee", PassWord = "123123123123" };  
                IUserProcessor userprocessor = new UserProcessorDecorator(new UserProcessor());  
                userprocessor.RegUser(user);  
            }  
            catch (Exception ex)  
            {  
                throw ex;  
            }  
        }  
    }

[复制代码](javascript:void(0);)

输出

方法执行前  
用户已注册。Name:lee,PassWord:123123123123  
方法执行后

**动态代理**

1、使用.Net Remoting/RealProxy

采用TransparentProxy和RealProxy实现对象的代理，实现思路如下：Client -TransparentProxy - RealProxy - Target Object

下面实现自定义的TransparentProxy和RealProxy

[复制代码](javascript:void(0);)

using System.Runtime.Remoting.Proxies;  
using System.Runtime.Remoting.Messaging;  
    //RealProxy  
    public class MyRealProxy<T>:RealProxy  
    {  
        private T \_target;  
        public MyRealProxy(T target) : base(typeof(T))  
        {  
            this.\_target = target;  
        }  
       public override IMessage Invoke(IMessage msg)  
       {  
            PreProceede(msg);  
            IMethodCallMessage callMessage = (IMethodCallMessage)msg;  
            object returnValue = callMessage.MethodBase.Invoke(this.\_target, callMessage.Args);  
            PostProceede(msg);  
            return new ReturnMessage(returnValue, new object[0], 0, null, callMessage);  
        }  
       public void PreProceede(IMessage msg)  
       {  
           Console.WriteLine("方法执行前");  
       }  
       public void PostProceede(IMessage msg)  
       {  
           Console.WriteLine("方法执行后");  
       }  
    }  
   //TransparentProxy  
   public static class TransparentProxy  
   {  
        public static T Create<T>()  
        {  
           T instance = Activator.CreateInstance<T>();  
           MyRealProxy<T> realProxy = new MyRealProxy<T>(instance);  
           T transparentProxy = (T)realProxy.GetTransparentProxy();  
           return transparentProxy;  
        }  
   }

[复制代码](javascript:void(0);)

用户注册接口和实现

[复制代码](javascript:void(0);)

  public interface IUserProcessor  
    {  
        void RegUser(User user);  
    }  
  
    public class UserProcessor : MarshalByRefObject, IUserProcessor  
    {  
        public void RegUser(User user)  
        {  
            Console.WriteLine("用户已注册。");  
        }  
    }

[复制代码](javascript:void(0);)

客户端调用

[复制代码](javascript:void(0);)

 public class Client  
    {  
        public static void Run()  
        {  
            try  
            {  
                User user = new User() { Name = "lee", PassWord = "123123123123" };  
                UserProcessor userprocessor = TransparentProxy.Create<UserProcessor>();  
                userprocessor.RegUser(user);  
            }  
            catch (Exception ex)  
            {  
                throw ex;  
            }  
        }  
    }

[复制代码](javascript:void(0);)

输出

方法执行前  
用户已注册。Name:lee,PassWord:123123123123  
方法执行后  
2、使用EntLib\PIAB  
自定义CallHandler，这里定义两个CallHandler分别用于参数检查和日志记录。

[复制代码](javascript:void(0);)

using Microsoft.Practices.Unity.InterceptionExtension;  
  
 public class UserHandler:ICallHandler  
    {  
        public int Order { get; set; }  
        public IMethodReturn Invoke(IMethodInvocation input, GetNextHandlerDelegate getNext)  
        {  
            User user = input.Inputs[0] as User;  
            if (user.PassWord.Length < 10)  
            {  
                return input.CreateExceptionMethodReturn(new UserException("密码长度不能小于10位"));  
            }  
            Console.WriteLine("参数检测无误");  
            return getNext()(input, getNext);  
        }  
    }  
  
    public class LogHandler:ICallHandler  
    {  
        public int Order { get; set; }  
        public IMethodReturn Invoke(IMethodInvocation input, GetNextHandlerDelegate getNext)  
        {  
            User user = input.Inputs[0] as User;  
            Log log = new Log() { Message = string.Format("RegUser:Username:{0},Password:{1}", user.Name, user.PassWord), Ctime = DateTime.Now };  
            Console.WriteLine("日志已记录，Message:{0},Ctime:{1}",log.Message,log.Ctime);  
            var messagereturn  = getNext()(input, getNext);   
            return messagereturn;  
        }  
    }

[复制代码](javascript:void(0);)

定义对应的HandlerAttribute

[复制代码](javascript:void(0);)

using Microsoft.Practices.Unity.InterceptionExtension;  
using Microsoft.Practices.Unity;  
  
    public class UserHandlerAttribute : HandlerAttribute  
    {  
        public override ICallHandler CreateHandler(IUnityContainer container)  
        {  
            ICallHandler handler = new UserHandler(){Order=this.Order};  
            return handler;  
        }  
    }  
  
    public  class LogHandlerAttribute:HandlerAttribute  
    {  
        public int Order { get; set; }  
        public override ICallHandler CreateHandler(IUnityContainer container)  
        {  
            return new LogHandler() { Order = this.Order };  
        }  
    }

[复制代码](javascript:void(0);)

用户注册接口和实现，这里通过为接口添加attribute的方式实现。order值表示执行顺序，值小的先执行。

[复制代码](javascript:void(0);)

    [LogHandlerAttribute(Order=2)]  
    [UserHandlerAttribute(Order=1)]  
    public interface IUserProcessor  
    {  
         void RegUser(User user);  
    }  
  
    public class UserProcessor : MarshalByRefObject,IUserProcessor  
    {  
        public  void RegUser(User user)  
        {  
            Console.WriteLine("用户已注册。");  
        }  
    }

[复制代码](javascript:void(0);)

客户端调用

[复制代码](javascript:void(0);)

using Microsoft.Practices.EnterpriseLibrary.PolicyInjection;   
     
    public class Client  
    {  
        public static void Run()  
        {  
            try  
            {  
                User user = new User() { Name = "lee", PassWord = "123123123123" };  
                UserProcessor userprocessor = PolicyInjection.Create<UserProcessor>();  
                userprocessor.RegUser(user);  
            }  
            catch(Exception ex)  
            {  
                throw ex;  
            }  
        }  
    }

[复制代码](javascript:void(0);)

输出：

参数检测无误  
日志已记录，Message:RegUser:Username:lee,Password:123123123123,Ctime:2010-12-22  
6:14:59  
用户已注册。  
3、使用Castle\DynamicProxy  
自定义Interceptor

[复制代码](javascript:void(0);)

 public class MyInterceptor : IInterceptor  
    {  
        public void Intercept(IInvocation invocation)  
        {  
            PreProceed(invocation);  
            invocation.Proceed();  
            PostProceed(invocation);  
        }  
        public void PreProceed(IInvocation invocation)  
        {  
            Console.WriteLine("方法执行前");  
        }  
  
        public void PostProceed(IInvocation invocation)  
        {  
            Console.WriteLine("方法执行后");  
        }  
    }

[复制代码](javascript:void(0);)

用户注册接口和实现

[复制代码](javascript:void(0);)

 public interface IUserProcessor  
    {  
        void RegUser(User user);  
    }  
  
    public class UserProcessor : IUserProcessor  
    {  
        public virtual void RegUser(User user)  
        {  
            Console.WriteLine("用户已注册。Name:{0},PassWord:{1}", user.Name, user.PassWord);  
        }  
    }

[复制代码](javascript:void(0);)

客户端调用

[复制代码](javascript:void(0);)

 public class Client  
    {  
        public static void Run()  
        {  
            try  
            {  
                ProxyGenerator generator = new ProxyGenerator();  
                MyInterceptor interceptor = new MyInterceptor();  
                UserProcessor userprocessor = generator.CreateClassProxy<UserProcessor>(interceptor);  
                User user= new User() { Name = "lee", PassWord = "123123123123" };  
                userprocessor.RegUser(user);  
            }  
            catch (Exception ex)  
            {  
                throw ex;  
            }  
        }  
    }

[复制代码](javascript:void(0);)

输出

方法执行前  
用户已注册。Name:lee,PassWord:123123123123  
方法执行后

关于上面各方案的详细介绍园子里都有很好的文章，我就不班门弄斧了。

另，公司最近在招聘，做个广告，欢迎有意者投递简历。

http://job.cnblogs.com/offer/10501/