精简自己 20% 的代码 - 文章

持续重构，其乐无穷。

一：发现问题

先来说如何重构业务层的try{}catch{}finally{}代码块，我看过很多代码，异常处理这一块大致分为两种情况，一种是每个方法都大量的充斥着try{}catch{}finally{ }，这种方式的编程已经考虑到了异常处理，还有一种就是没有try{ }catch{ }finally{ }的代码，因为根本就没有考虑代码的异常处理。

每当我看到这样的代码，我都很忧伤。从程序的健壮性来看第一种还是要比第二种情况好，至少在编程意识中，随时想到了异常情况，有一种基本的编程思想。比如：一个业务单据的多表插入，关联修改，虚拟删除等都是一些基本的操作，但是又是比较容易引起错误的操作，在这些方法上都会加上try{ }catch{ }finally{}对代码进行有效的防错处理。早期的代码是这样的。

```
Java
public Boolean Save(AccountModel accountData)
{
        Boolean result = false;
        try
        {
                //TODO ...
                result = true;
        }
        catch
        {
        }
        finally
        {
        }
        return result;
}
public Boolean Edit(AccountModel accountData)
{
        Boolean result = false;
        try
        {
                //TODO ...
                result = true;
```

```
        }
        catch
        {
        }
        finally
        {
        }
        return result;
}
public Boolean VirDelete(AccountModel accountData)
{
        Boolean result = false;
        try
        {
                //TODO ...
                result = true;
        }
        catch
        {
        }
        finally
        {
        }
        return result;
}
```

仅仅定义了添加，修改，删除几个空方法，就写了三四十行代码，如果业务稍微复杂些，异常处理的代码很快就会突破百行大关。虽然复制，粘贴try{}catch{}finally{}很好使，但是业务逻辑代码大量充斥着这样的try{}catch{}finally{}代码，确实显得做事不够利落。

二：解决问题

那怎样来解决这件棘手的事呢，首先定义一个公用的try{}catch{}finally{}，如下如示：

Java

```
public class Process
{
        public static bool Execute(Action action)
        {
                try
                {
                        action.Invoke();
                        return true;
                }
                catch (Exception ex)
```

```
                {
                        //1，异常隐藏
                        //2，异常替换
                        //3，异常封装
                        //写日志
                        return false;
                }
                finally
                {
                }
        }
}Java
protected void Page_Load(object sender, EventArgs e)
{
        AccountModel accountData = new AccountModel();          //准备传入的参数
        Boolean result = false;                                                         //接
收返回的值
        Process.Execute(() => result = Save(accountData)); //执行方法
}
public Boolean Save(AccountModel accountData)
{
        Boolean result = false;
        //TODO ...
        result = true;
        return result;
}
public Boolean Edit(AccountModel accountData)
{
        Boolean result = false;
        //TODO ...
        result = true;
        return result;
}
public Boolean VirDelete(AccountModel accountData)
{
        Boolean result = false;
        //TODO ...
        result = true;
        return result;
}
```

这样的精简过的代码，是不是感觉心情很舒畅。

三：提升与扩展

对于知足者常乐的人来说，到第二个步骤就可以洗洗睡了。但是对于精益求精的人来说，问题仍然没有完。我们来说一个应用场景，在WCF中的应用，我们知道WCF服务端的异常，不经过的设置，服务端的异常是无法抛到客户端的。但是在正式环境中，不可能对进行serviceDebug的配置。正确的处理是在服务端对异常进行隐藏，替换，或者封装。

比如我们在服务端捕获了一个已知异常，但是这个异常会暴露一些敏感的信息，所以我们对异常进行替换，抛出新的异常后，我们还要把这个异常怎样传输给客户端。首先们要明确WCF中的一些基本常识，就是WCF中的数据传递要遵循WCF的数据契约，服务端抛到客户端的异常（异常其实也是数据），所以必须要给异常定义异常契约。

Java

```
public static bool Execute(Action action)
{
        try
        {
                action.Invoke();
                return true;
        }
        catch (Exception ex)
        {
                //1，异常隐藏
                //2，异常替换
                //3，异常封装
                //写日志
                WCFException exception = new WCFException
                {
                        Type = "Error"
                        ,
                        StackTrace = ex.StackTrace
                        ,
                        Message = ex.Message
                };
                throw new FaultException(exception
                        , new FaultReason("服务端异常:" + ex.Message)
                        , new FaultCode(ex.TargetSite.Name));
        }
        finally
        {
        }
}
```

这样在服务端抛出的异常，就能在客户端捕捉到。现在是不是感觉自己又提升了一些，想成为编程高手是指日可待了。

四：举一反三

异常的处理也不过如此，那是不是应该举一反三，看看事务的处理应该怎么办？比如现在大量的访求都用到了事务，如下代码：

Java

合作联系

Email：bd@Jobbole.com

QQ： 2302462408 （加好友请注明来意）

更多频道

小组 - 好的话题、有启发的回复、值得信赖的圈子
头条 - 分享和发现有价值的内容与观点
相亲 - 为IT单身男女服务的征婚传播平台
资源 - 优秀的工具资源导航
翻译 - 翻译传播优秀的外文文章
文章 - 国内外的精选文章
设计 - UI,网页，交互和用户体验
iOS - 专注iOS技术分享
安卓 - 专注Android技术分享
前端 - JavaScript, HTML5, CSS
Java - 专注Java技术分享
Python - 专注Python技术分享