火龙果•整理
uml.org.cn

AOP开发实践

议题

- AOP概述
- **使用AOP实现松散耦合**
- 使用AOP组合两个业务逻辑
- **对象代理和过滤器**
- **结论**

火龙果•整理
uml.org.cn

什么是AOP?

- AOP是Aspect Oriented Programming的简写，中文通常译作面向方面编程，其核心内容就是所谓的“横切关注点”。

OO都是纵向结构的

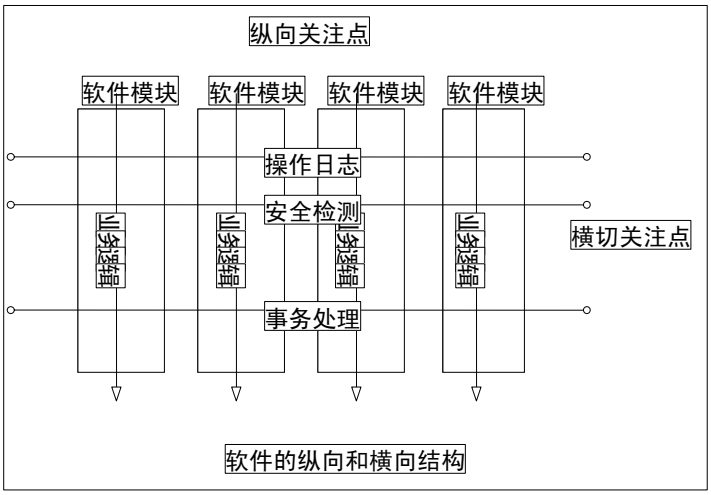
- 使用面向对象方法构建软件系统，我们可以利用OO的特性，很好的解决纵向的问题，因为，OO的核心概念，如继承等，都是纵向结构的。

火龙果•整理
uml.org.cn

AOP的目标

- 但是，在软件系统中，往往有很多模块，或者很多类共享某个行为，或者说，某个行为存在于软件的各个部分中，这个行为可以看作是“横向”存在于软件之中，他所关注的是软件的各个部分的一些共有的行为，而且，在很多情况下，这种行为不属于业务逻辑的一部分。
- 例如，操作日志的记录，这种操作并不是业务逻辑调用的必须部分，但是，我们却往往不得在代码中显式进行调用，并承担由此带来的后果（例如，当日志记录的接口发生变化时，不得不对调用代码进行修改）。
- 这种问题，使用传统的OO方法是很难解决的。AOP的目标，便是要将这些“横切关注点”与业务逻辑代码相分离，从而得到更好的软件结构以及性能、稳定性等方面的好处。

关注点切割图



火龙果•整理
uml.org.cn

AOP的自动耦合

- AOP，给我们的软件设计带来了一个新的视角和软件架构方法。使用AOP，我们可以专注于业务逻辑代码的编写，而将诸如日志记录、安全检测等系统功能交由AOP框架，在运行时刻自动耦合进来。

使用AOP技术的情景

- 通常，我们可以在如下情景中使用AOP技术：
  - Authentication 权限
  - Caching 缓存
  - Context passing 内容传递
  - Error handling 错误处理
  - Lazy loading 懒加载
  - Debugging 调试
  - logging, tracing, profiling and monitoring(记录跟踪,优化,校准)
  - Performance optimization 性能优化
  - Persistence 持久化
  - Resource pooling 资源池
  - Synchronization 同步
  - Transactions 事务

火龙果•整理
uml.org.cn

议题

- AOP概述
- **使用AOP实现松散耦合**
- 使用AOP组合两个业务逻辑
- **对象代理和过滤器**
- **结论**

Websharp Aspect

- 我们选用一个开放源代码的Web sharp Aspect框架。这个框架是基于Microsoft. Net平台的，并且是使用C#语言开发的，所以，下面的示例代码使用了C#相关的语法。
- 关于这个框架的说明以及源代码，可以从下面地址下载 www.websharp.org

火龙果•整理
uml.org.cn

- 对于应用软件系统来说，权限控制是一个常见的例子。为了得到好的程序结构，通常使用OO的方法，将权限校验过程封装在一个类中，这个类包含了一个校验权限的代码，例如：

```
public class Security
{
public bool CheckRight(User currentUser , Model accessModel ,
   OperationType operation)
{
   ....../校验权限
}
}
```


- 然后，在业务逻辑过程中进行如下调用：

```
public class BusinessClass
{
      public void BusinessMethod()
      {
            Security s = new Security();
            if (!s. CheckRight(……))
            {
                  return ;
            }
            ……//执行业务逻辑
      }
}
```

火龙果•整理
uml.org.cn

OO设计的问题

- 这种做法在OO设计中，是常见的做法。但是这种做法会带来以下的问题：
  - 不清晰的业务逻辑：从某种意义上来说，权限校验过程并不是业务逻辑执行的一部分，这个工作是属于系统的，但是，在这种情况下，我们不得不把系统的权限校验过程和业务逻辑执行过程掺杂在一起，造成代码的混乱。
  - 代码浪费：使用这种方法，我们必须所有的业务逻辑代码中用Security类，使得同样校验的代码充斥在整个软件中，显然不是很好的现象。
  - 紧耦合：使用这种方法，我们必须在业务逻辑代码中显式引用Security类，这就造成了业务逻辑代码同Security类的紧耦合，这意味着，当Security发生变化时，例如，当系统进化时，需要对CheckRight的方法进行改动时，可能会影响到所有引用代码。下面所有的问题都是因此而来。
  - 不易扩展：在这里，我们只是在业务逻辑中添加了权限校验，哪一天，当我们需要添加额外的功能，例如日志记录功能的时候，我们不得不同样在所有的业务逻辑代码中添加这个功能。
  - 不灵活：有的时候，由于某些特定的需要，我们需要暂时禁止，或者添加某项功能，采用传统的如上述的做法，我们不得不采用修改源代码的方式来实现。

- 为了解决这些问题，我们通常会采用诸如设计模式等方式，对上面的方案进行改进，这往往需要很高的技巧。利用AOP，我们可以很方便的解决上述问题。
- 我们以Websharp Aspect为例，看看如何来对上面的代码进行改动，以获得一个更好的系统结构。

火龙果•整理
uml.org.cn

- 首先，Security并不需要做任何修改。
- 然后，我们对BusinessClass做一个小小的改动：
  - 为BusinessClass添加一个名为AspectManaged的Attribute，并使得BusinessClass继承AspectObject，然后，删除代码中对Security的调用，这样，我们的代码就变成了如下的样子：

```
[AspectManaged(true)]
public class BusinessClass : AspectObject
{
      public void BusinessMethod()
      {
            ……//执行业务逻辑
      }
}
……//执行业务逻辑
```


- 然后，我们为系统增加一个SecurityAspect。

```
public class SecurityAspect : IAspect
{
      public void Execute(object[] paramList)
      {
            if(!Security.CheckRight(.....))
            {
                  throw new SecurityException("你没有权限！");
            }
      }
}
```

火龙果•整理
uml.org.cn

- 最后，我们在系统配置文件中添加必要的信息：

```
<Websharp.Aspects>
   <Aspect type="MyAPP.SecurityAspect, MyAPP" deploy-
   model="Singleton"
        pointcut-type="Method" action-position="before" match="*,*" />
</Websharp.Aspects>
```


- 我们就完成了代码的重构。当BusinessClass被调用的时候，AOP框架会自动拦截BusinessClass的BusinessMethod方法，并调用相应的权限校验方法。
- 采用这种方式，我们在BusinessClass中没有显式引用Security类及其相应方法，并且，在所有业务逻辑代码中，都没有必要引用Security类。
- 这样，借助AOP机制，我们就实现了BusinessClass和Security类的松散耦合，上面列出的所有问题都迎刃而解了。同时，这也是一种易于扩展的机制，例如，当我们需要添加日志记录功能的时候，只需要添加相应的Aspect类，然后在配置文件中进行配置即可，而无需对业务逻辑代码进行任何改动。

火龙果•整理
uml.org.cn

议题

- AOP概述
- **使用AOP实现松散耦合**
- 使用AOP组合两个业务逻辑
- **对象代理和过滤器**
- **结论**

使用AOP组合两个业务逻辑

- 使用AOP，我们不仅仅可以用来分离系统功能和业务逻辑，就象上面我们做的那样，也可以用来耦合不同的业务逻辑，得到更加灵活的软件结构。下面，我们通过一个具体的案例，来看看怎么通过AOP，组合两个业务逻辑过程。

火龙果•整理
uml.org.cn

我们假设有如下一个场景

- 我们设计了一个ERP系统，其中，库存管理系统需要同财务系统相交互，例如，当某个库存商品报废的时候，需要有相应的财务处理过程。因此，我们通常需要在库存商品报废的业务逻辑中引用相关的财务处理逻辑。这必然会造成两个部分的耦合。当然，为了使两个部分尽量耦合程度降低，我们通常会使用Façade等设计模式来进行解耦。

- 由于某些原因，我们需要将库存管理系统单独出售，这就需要我们需要从库存商品报废的业务逻辑中将引用的相关的财务处理逻辑去除，这意味着我们需要修改原有的代码。
- 为了解决这个问题，即可以随时将财务处理逻辑添加或者从库存商品报废的业务逻辑中删除，我们可以采用一些方法，例如，设置一些开关参数，在库存商品报废的业务逻辑中，根据这些开关参数的值，来判断是否需要执行财务处理逻辑。

火龙果•整理
uml.org.cn

- 问题是，这仍旧不是理想的解决方案。采用这种方式，你必须事先知道所有需要设置的开关参数，并且，在业务逻辑代码中添加相应的判断。
- 当为系统增加一个类似的需要灵活处理的部分时，开发人员不得不添加相应的参数，并且修改相应的代码（添加相应的判断代码）。
- 修改代码总是不好的事情，因为按照软件工程的要求，当有新的需求是，尽量不要修改原来的代码，而是新增相应的代码。但是，在这种情况下，你做不到。

- 使用AOP，我们可以通过一种更加自然的方式来实现这个目标。基本方法如下：
  - 首先，编写相关的库存商品报废业务逻辑，不需要添加任何其他的内容，并且，把这个逻辑的代码设置为可AOP的。
  - 其次，按照正常的方式，编写财务处理逻辑。
  - 添加一个把库存商品报废业务逻辑和财务处理逻辑组合起来的Aspect，这个Aspect可以拦截库存商品报废业务逻辑的执行，动态的加入财务处理逻辑的过程，并且，在配置文件中进行配置。
  - 这样，我们就通过一个Aspect，组合了这两个业务逻辑。并且，我们随时可以通过修改配置文件的方式把财务处理从库存商品报废业务逻辑中去除，而不用修改任何代码。

火龙果•整理
uml.org.cn

- 从上面的例子可以看出，采用AOP的好处是，我们可以独立的编写各个业务逻辑，使得系统各个部分之间的耦合度降到最低，然后，可以在系统中根据需要随时组合两个逻辑，而不用修改原来的任何代码。

议题

- AOP概述
- **使用AOP实现松散耦合**
- 使用AOP组合两个业务逻辑
- **对象代理和过滤器**
- **结论**

火龙果•整理
uml.org.cn

对象代理和过滤器

- 应该认识到，完全的AOP实现，需要开发语言的支持。因为对于AOP的研究，还正在进行之中，目前的开发语言，都还没有完全支持AOP的，但是，我们可以利用现有的一些语言功能，来实现AOP的部分功能。
- 上面所举的例子，在实现上，是利用了对象代理(Proxy)机制。所谓Proxy，就是“为其他对象提供一种代理以控制对这个对象的访问”

- 在WebsharpAspect中，当一个对象被标记为AspectManaged后，这个类的实例的创建过程，以及方法的调用会被WebsharpAspect控制。因此，当你在调用如下语句：

  BusinessClass bc = new BusinessClass()；

火龙果•整理
uml.org.cn

- 你得到的实际上并不是BusinessClass类的一个实例，而是他的一个代理（关于其中的实现机理，可以参见相关的源代码）。
- 因此，当调用这个“实例”的方法的时候，所有的调用都会被代理所捕获，代理会在实际的方法调用之前，透明的执行一些预定义的操作，然后再执行实际的方法，最后，在实际的方法调用之后，再执行一些预定义的操作。这样，就实现了AOP的功能。

- 注意，AOP并不仅仅等同于方法调用拦截，当然，这也是最常用的和非常有效的AOP功能。
- 在某些开发中，我们可能使用过滤器来完成某些AOP功能。例如，当用户访问某些资源时，我们可以对访问信息进行一些过滤处理。一个常见的场景是，在JSP开发中，为了实现对中文的正确处理，我们通常需要对浏览器同服务器之间传递的数据进行转码处理，以获得正确的文字编码。在每个Request中手工进行转码肯定不是一个好的解决方案。

火龙果•整理
uml.org.cn

- 一个比较好的例子，是为应用程序编写一个Filter，自动进行转码处理。例如，我们可以为TOMCAT写如下一个过滤器来实现转码：

```
public class SetCharacterEncodingFilter implements Filter
{
      public void doFilter(ServletRequest request,ServletResponse
      response,FilterChain chain)
            throws IOException, ServletException
      {
            request.setCharacterEncoding(“gb2312”);
            chain.doFilter(request, response);
      }
}
```


- 这样，我们就不必在具体业务处理中来进行手工的转码。实现业务逻辑同转码这样的系统功能的分离。
- 目前，常见的Web服务器都提供类似的机制。例如，在IIS中，也可以使用过滤器功能。传统的开发方式是使用VC开发一个ISAPI Filter。在.Net发布后，可以使用HttpHandler和HttpModule实现相同的功能，但是，开发难度要低很多。

火龙果•整理
uml.org.cn

- 使用过滤器的另外一个场景，可以是权限控制。例如，在客户请求某个Web页面的时候(这个Web页面通常同某个业务功能相关联)，可以使用过滤器截获这个请求，然后，判断这个用户是否具备对请求资源的访问权限。如果是，那么，过滤器可以把这个请求放过去，什么都不做，否则，过滤器可以重定向到某个页面，告诉用户不能访问的原因，或者，直接抛出异常，交由前面的处理者处理。通过这种方式，我们可以同样的分离诸如身份验证这样的系统功能和业务逻辑，实现更好的系统结构。
- 通过象Web服务器这样的应用程序环境提供的功能，我们还可以实现其他一些AOP的功能，构建更好的系统框架。

议题

- AOP概述
- **使用AOP实现松散耦合**
- 使用AOP组合两个业务逻辑
- **对象代理和过滤器**
- **结论**

火龙果•整理
uml.org.cn

结论

- AOP给了我们一个新的视角来看待软件的架构，有的时候，即使不使用AOP技术，只使用AOP的某些观念和现有的技术来搭建系统架构，分离某些本来是紧耦合的关注点，对我们也是非常有益的。