面向方面编程在B/S多层架构中业务层的应用

范后军,魏慧琴
(北京交通大学 计算机与信息技术学院,北京 100044)

摘 要:介绍了基于面向方面编程(AOP)基本思想,分析了B/S多层架构中面向对象编程所面临的困境,介绍如何将AOP思想应用于B/S多层架构中的业务层,相比纯面向对象的实现突出显示了AOP在业务层事务处理上优越性。它主要体现在两个方面:系统架构清晰,可读性、复用性、扩展性增强;降低系统的耦合性,有利于团队成员分工合作,减少主业务开发人员负担。

关键词:AOP;事务;ThreadLocal变量;B/S多层架构;J2EE
中图分类号:TP311 **文献标识码**:A **文章编号**:1673-629X(2007)04-0083-03

Application of AOP in Business Layer of B/S Multilayer

FAN Hou-jun, WEI Hui-qin
(Computer and Information Technology School of Beijing Jiaotong University, Beijing 100044, China)

Abstract: Introduces the conception of the aspect oriented programming(AOP), analyzes the corner of the object oriented programming (OOP) and studys how to apply AOP in the business layer of B/S multilayer, reveals the strongpoint of AOP in dealing with transaction, mainly including: 1. more excellent architecture, strengthen the readability, reusablity and extensibility of applications; 2. loosen the coupling of system, more easily cooperating for a team, lighten some burden of developers in the business layer.

Key words: AOP; transaction; ThreadLocal variable; B/S multilayer; J2EE

0 前 言

“计算机语言的发展经历了:汇编语言、面向过程语言和面向对象语言等阶段,编程语言的的发展使我们可以构建更加复杂的软件系统”[1]。当前企业应用系统普遍采用面向对象语言(如Java)进行开发,面向对象编程(OOP)通过分析、抽象出一系列具有一定属性与行为的对象,并通过这些对象之间的协作完成系统的功能,按照功能或行为对软件系统进行分割,架构整个系统,相比传统编程模式大大解耦了软件系统。随着需求不断发展,系统越来越复杂,低耦合性、可复用性、可扩展性的重要性越来越突出,OOP作为一种成熟的编程思想也显出了自身的不足。软件系统中通常存在很多业务功能,商业业务方法(下称主业务方法)执行之前或之后总会有一些辅助业务如事务、日志、权限认证等需要处理,OOP针对这些公共的辅助业务进行封装,在每一个主业务方法体代码中都进行显示地调用,这使得辅助业务相关代码分散在各个主业务方法中,导致代码混乱,难于理解和维护。面向方面编程(AOP)能够很好地解决此类问题,它可以实现辅助业务和主业务完全分离,可以使得开发人员分别专注于主业务和辅助业务两个方面编程。相比OOP,AOP从更高层次对软件系统进行划分,OOP从单一的横方向进行分解系统,而AOP从纵横交叉两个方面分解系统,更贴近现实世界的层次上实现软件开发的模块化。

收稿日期:2006-07-18

作者简介:范后军(1979-),男,湖北大冶人,硕士研究生,研究方向为软件工程;魏慧琴,副教授,硕士研究生导师,研究方向为网络技术、MIS、教育技术。

1 多层B/S架构分析

基于J2EE的B/S多层架构可以简单地表示如图1所示[2]。

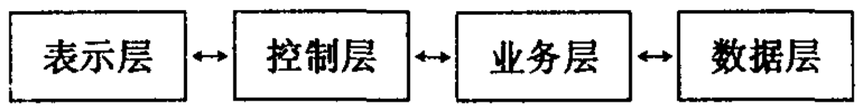

图1 J2EE多层架构

表示层采取的技术主要是JSP和HTML;控制层主要采用Servlet技术;数据层即数据库,如MySQL,Oracle等;业务层集中应用系统各种商业业务功能,该层通常包含与业务实体对应的各种POJO和诸多组件,从控制层接受客户发过来的请求,读写数据库实现相应业务操作,系统的复用性、可扩展性、低耦合性对

于该层尤为重要,在该层除了要实现主要业务功能,还有很多辅助业务需要处理,如事务、日志、权限认证等。文中就以事务处理为例来展示 AOP 的强大功能。业务层职能是向控制层提供各种商业业务操作调用,在业务层某一业务功能步骤如下:

(1)获取数据库连接。

(2)开启事务(如果只是读数据库操作,可以不需要事务)。

(3)执行商业业务操作。

(4)关闭事务或者回滚事务。

(5)释放数据库连接。

基于 J2EE 技术的 Web 系统充分利用了 Java 语言纯面向对象的特点,通常都会做好“O/R 映射”[3],简单来说,就是把关系型数据库中各实体表对应到值对象(VO,Value Object)类,这样类通常只有属性和及其 get、set 方法,然后定义一个基本 BaseBO 接口封装数据库连接,如:

```
public interface BaseBO{
  public void comit ():/提交事务
  public void rollBack (); //回滚事务
  …
}
```

在这个接口中通常还定义各种访问数据的基本操作,而开启事务的方法是可以不用暴露出来,获取数据库连接时设为事务非自动提交就可以了。目前已经出现了一些框架把这部分工作做得很好了,如:Hibernate。

有了 BaseBO,实现一些具体的业务功能就容易得多。下面就给出业务层主要类图,如图 2 所示。

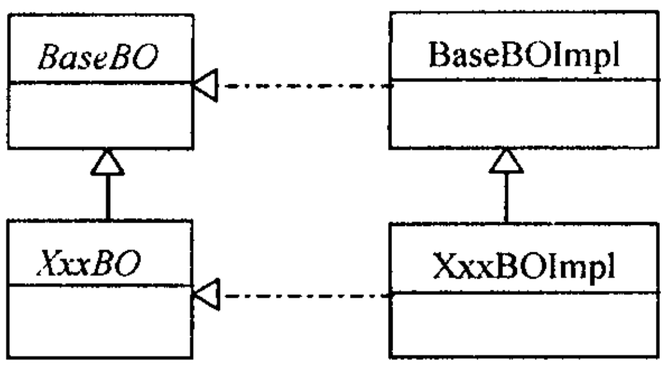

图 2 业务层类图

Web 系统通常有很多接口 XxxBO,每个接口 XxxBO 都封装一些业务操作,继承自 BaseBO,并对应一个实现 XxxBOImpl,这些实现都继承自 BaseBO 的实现 BaseBOImpl,XxxBOImpl 中还会存在互相调用。在面向对象编程中,XxxBOImpl 所提供实际业务功能很多需要事务处理,也就是说每编写一个业务方法,都需要去考虑 try{}catch{}处理事务,这就出现了面向对象语言所面临的尴尬局面:辅助业务相关代码分散在各个主业务方法中,导致代码混乱。不但增加开发人员的负担,更大大地降低了可维护性。同时,代码的复用性也受到了很大的损伤,例如:在一个银行系统中,通常需要提供对用户更新存款的业务,这个业务需要包含事务处理,而系统中的转账业务需要调用两次更新存款的操作,这两次更新必须在同一个事务中,那么已有的更新存款业务就不可以重用了,需要重新另写一个更新存款的方法。

2 面向方面编程

AOP 可以实现主业务和辅助业务彻底分离,并实现它们自动组合完成正常的业务功能,使开发人员分别专注于主业务和辅助业务两个方面进行开发。利用 AOP 的思想在技术上主要有两点需要解决:

(1)确定哪些是公共的辅助业务,将这一部分完全剥离出来,专注辅助业务方面编程。既要剥离事务处理,又要能够为实现与主业务的组合作准备。基于 J2EE 的 B/S 多层应用中,对每一个用户请求,Web 服务器都开启一个新的线程来处理,可以采用 java.lang.ThreadLocal 变量实现,它能为线程分别保存各自数据备份。具体实现如下:

```
package com.aop;
import java.sql.Connection;
import java.sql.SQLException;
public class Aspect {
  private static ThreadLocal conMap;
  //获取当前线程的数据库连接
  public static Connection getConnection() throws SQLException{
    Connection con = (Connection) conMap.get();//从 ThreadLocal 变量中获取数据库连接
    if(con==null){
      con=getConnection();//从数据库连接池中获取连接
      conMap.set(con);//存入 ThreadLocal 变量
    }
    return con;
  }
  //获取当前线程连接,提交事务,并释放连接
  public static void comit() throws SQLException{
    …
  }
  //获取当前线程连接,回滚事务,并释放连接
  public static void rollBack() throws SQLException{
    …
  }
}
```

注意这里的方法 getConnection(),该方法的功能是获取当前线程所拥有的连接,应用系统中 BaseBOImpl 直接通过该方法获取连接,以确保一个线程的一次业务操作在同一个数据库连接上进行数据操作。

(2)如何将辅助业务与主业务组合在一起,即提供一个组合器,"可以利用 Java 的动态代理机制及反射机制实现"[4],在业务层增加一个 AOP 的组合器,实现如下:

```
package com.aop;
import java.lang.reflect.Proxy;
public class AopFactory {
  private static PointHandler handler=new PointHandler();
  public static Object getInstance(Class clazz){
  try{
    handler.setDelegate(clazz.newInstance());
    //利用 java 的代理机制,获取代理对象
    Object object=Proxy.newProxyInstance(clazz.getClassLoader
(),
      clazz.getInterfaces(), handler);
    return object;
  }catch(Exception e){
    throw e;
    return null;
  }
 }
}
```

其中的 PointHander 类实现了 java.lang.reflect.InvocationHandler 接口,提供执行处理器,实现对辅助业务的调用,如下:

```
package com.aop;
import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Method;
public class PointHandler implements InvocationHandler {
  private Object delegate;
  public void setDelegate(Object delegate) {
    this.delegate=delegate;
  }
  public Object invoke(Object proxy,Method method, Object[]
args)
      throws Throwable {
    Object o=null;
    try {
      o=method.invoke(delegate,args);//调用实际业务方法
      Aspect.commit();//提交事务
    } catch (Exception e) {
    Aspect.rollBack();//回滚事务
      …
    }
    return o;
  }
}
```

在控制层调用业务层操作时通过组合器获取业务操作对象就可以实现主业务与辅助业务自动组合,示例代码:

```
XxxBO xxxBO=(XxxBO)com.aop.AopFactory.get(XxxBOImpl.
class);
```

这时从业务层外部调用任意一个业务方法,组合器 AOPFactory 会自动将主业务方法与辅助业务 comit()和 rollBack()组合在一起,实现完整的业务功能。而业务层内部各个业务实现类之间的直接调用与辅助业务是完全隔离的。

3　应用 AOP 实现 B/S 架构分析

将 AOP 应用于业务层,业务层的架构可以用图 3 作一个大致的描述。

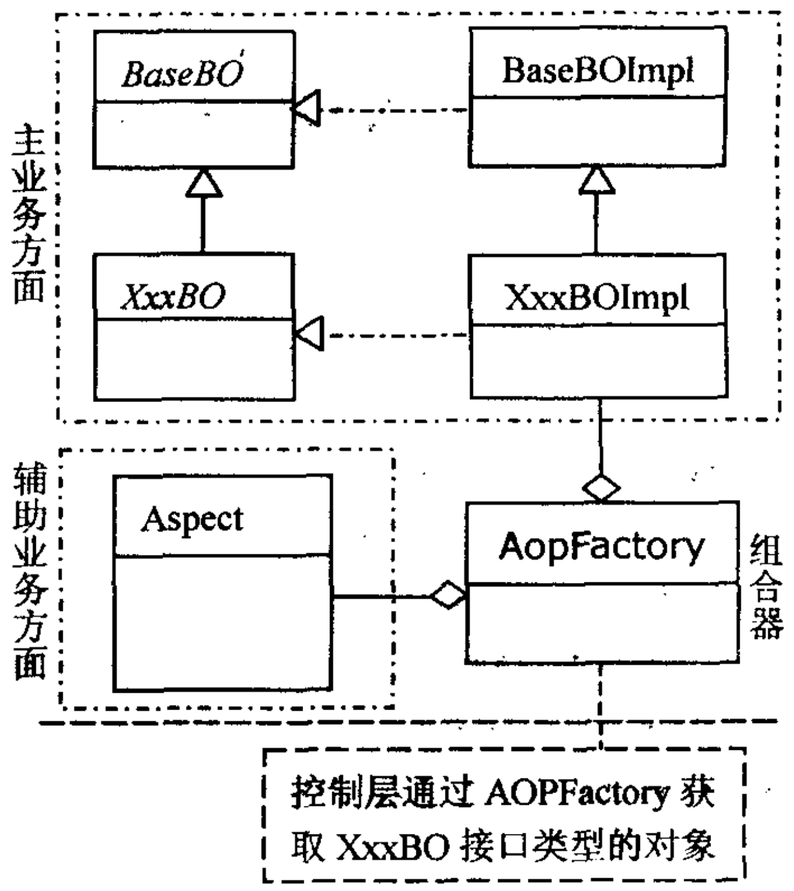

图 3　基于 AOP 业务层类图

在 B/S 多层架构中业务层通过引入 AOP 新的编程方式,AOP 的两个方面——主业务方面与辅助业务方面仍然是面向对象的。AOP 的加入除了彻底解决上述 OOP 所面临的困境以外,"同时也为系统提供了更大扩展空间"[5]。总结 AOP 所带来优越性可以体现在以下两方面:

(1)AOP 从一个较高的层次划分软件系统,提供了一个更加清晰的框架,将辅助业务从主业务中完全剥离出来,更加接近现实世界从两个方面构建系统,把辅助业务的调用集中在一个地方,实现对辅助业务的调用只写一次,这样可以非常方便对辅助业务维护和扩展,例如为每个主业务方法添加一些日志信息,只需要编写处理日志信息辅助业务方面,对 com.aop.PointHandler 做相应修改即可,而不影响主业务代码。另外,耦合性降低,代码更加清晰,可读性和重用性大大增强。

(2)降低了系统的耦合性,减轻了开发人员负担,开发人员只需要关心主业务的开发与实现,不用受辅

(下转第 135 页)

```
    <property name="gyywDao" ref="gyywDao"/>
</bean>
```

Spring 的事务配置包括两个部分:其一,定义事务管理器 transactionManagerA,使用 HibernateTransaction Manager 实现事务管理;其二,对各个业务接口进行定义,其实 gyywTransactionProxy 和 gyywService 是父子节点的关系,本来可以将 gyywTransactionProxy 定义的内容合并到 gyywService 中一起定义,但由于系统有多个业务接口需要定义,将这些业务接口定义内容的共同部分抽取到一个父节点中,然后在子节点中通过 parent 进行关联,就可以大大简化业务接口的配置了。父节点 gyywTransactionProxy 注入了事务管理器,此外还定义了业务接口事务管理的方法(允许通过通配符的方式进行匹配声明,),有些接口方法仅对数据进行读操作,而另一些接口方法需要涉及到数据的更改。对于前者,可以通过 readOnly 标识出来,这样有利于操作性能的提高,需要注意的是由于父类节点定义的 Bean 仅是子节点配置信息的抽象,并不能具体实例化一个 Bean 对象,所以需要特别标注为 abstract = "true"。GyywServiceImpl 作为一个目标类注入事务管理器中,而其所需的 gyywDao 通过注入 GyywDaoImpl 实现。

通过以上介绍可以发现正如图 1 所示,Tapestry 框架通过 Spring 框架的 Application Context 调用底层服务,而 Spring 框架通过将 Hibernate 的 sessionfactory 注入 DAO 以及将 transactionmanager 注入 service,实现对底层数据库的操作。

3　结束语

Tapestry + Spring + Hibernate 框架是一种非常优秀的基于 J2EE 平台的 Web 应用开发框架,在业界已经有了许多成功的案例。应用 Tapestry + Spring + Hibernate 框架构建 Web 应用程序,使得该系统结构层次清晰并实现了层之间的解耦,开发过程中层与层之间的工作几乎完全独立,极大地提高了系统的开发效率。同时也提高了系统的可重用性和灵活性,为日后的扩展和维护留有很大余地。

参考文献:

[1] 宋秀琴,侯殿昆,方中纯. 基于 Struts 和 Hibernate 的 Web 应用的构建[J]. 微计算机信息,2005,21(11):125 - 127.

[2] 孙　刚,孟祥武. 基于 Struts 框架的 J2EE Web 应用[C]// 第六届 JAVA 技术及应用大会论文集. 北京:电子工业出版社,2003:124 - 129.

[3] 黄　华. 框架技术在 web 系统开发中的应用[J]. 微机发展,2005,15(5):77 - 79.

[4] Egle M. Wiring Your Web Application with Open Source Java [EB/OL]. 2004 - 04 - 07. http://www.onjava.com/pub/a/onjava/2004/04/07/wirringwebapps.html.

[5] Howard M,Ship L. Tapestry in Action[M]. [s.l.]:Manning, 2004.

[6] What is Tapestry[EB/OL]. 2006 - 01 - 25. http://jakarta.apache.org/tapestry.

[7] Spring Framework 开发参考手册[EB/OL]. 2005 - 06. Spring 中文论坛.

[8] Hibernate - version 3.0.4[EB/OL]. 2005 - 03. http://www.hibernate.org.

(上接第 85 页)

助业务的干扰,整个开发团队中只需要很少的一部分人负责辅助业务方面开发即可,有利于成员之间的分工合作,发挥团队的作用。

由于 AOP 组合器的实现引入了 Java 动态代理和反射机制,不可否认会对性能产生一定的影响,但这一点点开销与它所带来的清晰架构相比是很值得的,另外还可以借鉴 EJB 对象池的思想在组合器中为业务对象建立适当缓存优化性能。

4　总　结

文中框架的设计尚显粗糙,例如,框架中对所有主业务方法都进行事务处理,而在实际应用中,很多业务方法仅仅只是读数据,无需事务处理。比较简单的解决办法是对业务方法名做一定约束,方法名以 get、find、query 等打头表示该业务不需要事务处理,否则需要,那么在 com.aop. PointHandler 中对方法名作判断可选择是否组合事务处理。

AOP 仍处在不断发展之中,受到越来越多人的关注,是 OOP 之后的又一次编程语言的重要创新,不过和当初一样,AOP 面临的还是不同的标准和想法。

参考文献:

[1] Amandaxp. AOP 技术简介[EB/OL]. 2005. http://dev2dev.bea.com.cn/bbs.

[2] Nyberg G,Patrick R. 精通 BEA WebLogic Server 构建与部署 J2EE 应用的最佳策略[M]. 北京:电子工业出版社,2004.

[3] 孙卫琴. 精通 Hibernate:Java 对象持久化技术详解[M]. 北京:电子工业出版社,2006.

[4] Wxh1028. 用 Java 动态代理实现 AOP[EB/OL]. http://www.yesky.com/SoftChannel/72342371961929728/20041014/1864192-1.shtml.

[5] Jacobson I,Pan Wei Ng. Aspect - Oriented Software Development with Use Cases[M]. 北京:电子工业出版社,2003.