用 Spring 搭建 Web 应用的轻量级解决方案

黄永欣[1],周淑秋[2]
(1.首都师范大学 信息工程学院,北京 100037;
2.中国劳动关系学院,北京 100037)

摘 要:分析了 EJB 作为企业级组件在今天不断有新技术产生的情况下暴露的种种缺点,并提出了替代 EJB 的一种轻量级解决方案:Spring 结合其他开源框架 Struts 和 Hibernate 来搭建 Web 应用系统。介绍了 Spring 的核心技术控制反转(IoC)和面向切面编程(AOP),最后结合一个销售游戏点卡的 B2C 电子商务系统介绍了如何实施该轻量级解决方案。

关键词:J2EE;企业 JavaBean;轻量级 Spring;控制反转;面向切面编程
中图分类号:TP319 **文献标识码**:A **文章编号**:1673-629X(2007)01-0189-04

The Lightweight Solution of Using Spring to Design Web Application

HUANG Yong-xin[1],ZHOU Shu-qiu[2]
(1. Information Technology Institute, Capital Normal University, Beijing 100037, China;
2. China Institute of Industrial Relations, Beijing 100037, China)

Abstract: Analyses multitudinous disadvantages of EJB as an enterprise component in the situation of incessant producing of the new technology nowadays. And brings forward the lightweight solution as a substitute of EJB that it integrates with Spring, Struts and Hibernate making the Web application. The emphasis is about the core technology of Spring IoC(Inversion of Control) and AOP(Aspect Oriented Programming). At last, introduce how to implement this lightweight solution via combination with the B2C e-business system that sells game card.

Key words: J2EE; EJB; lightweight Spring; IoC; AOP

0 引 言

基于 J2EE 的 Web 应用系统以其层次性、平台无关性逐渐被大多数的公司所认同,已经成为电子商务的主要解决方案。J2EE 技术非常强调分层原则,大部分的 Web 应用在职责上一般分为表示层、业务层和持久层 3 层,传统的基于 J2EE 的业务逻辑层通常采用 EJB[1,2] 实现,在今天新技术层出不穷,对于诞生于 1998 年的中间件解决方案 EJB 技术,其提供的很多服务已经可以用近几年出现的新技术以更加优雅的方式实现,这些新技术包括轻量级 IoC 容器、Web Service 和 AOP(面向切面编程)。EJB 容器是一个庞大而复杂的系统软件,经过几年残酷的市场竞争之后,目前实际上已经只有很少的几个组织提供 EJB 容器,在这些硕果仅存的 EJB 容器产品中,IBM, BEA, Borland, Oracle 等大厂商的应用服务器售价皆不菲,JBoss, Resin 等以"开源"标榜自己的产品实际上也对商业应用有着诸多限制。在这样的情况下特别是对于中小企业应该采用某种基于开源的轻量级架构取代 EJB 处理业务逻辑。

文中主要阐述了 EJB 的一些缺点,及其推崇的一些有价值服务仍可以采用其他方案替代,并总结了选择 EJB 技术的一个基本原则,最后给出了作为其轻量级解决方案的开源框架 Spring[3] 的优势,同时结合一个具体的游戏点卡销售的 B2C 电子商务系统讲述如何用 Spring 结合其他优秀开源框架 Struts[4] 和 Hibernate[5] 搭建 Web 应用系统。

1 中间件 EJB 技术

EJB 提供了一种现代组件体系结构,以定义和编辑的方式提供了对上述服务的支持,这使得业务组件的开发成为一件相对容易的事情。然而,在 EJB 诞生后,随后几年的技术发展,使得 EJB 不再像当初那样具有独到的优势。

其缺点如下:

①作为组件已过时。由于 Java 语言的进步,J2SE1.3 对 Java 进行了重大的改进,例如动态代理(dynamic proxy),使得任何接口可以被一个运行时生成的代理所实现,这样 EJB 所采纳的那种"不完全匹配组件接口"的实现方式看起来越发笨拙。

②由于近几年基于 XML 技术的 Web Service 提供了比过去大得多的开放性,是真正的互操作,使得 EJB 作为

收稿日期:2006-03-27
基金项目:北京市教委面上科技发展项目(KM200611417001)
作者简介:黄永欣(1980-),女,北京人,硕士研究生,主要研究方向为语义 Web、数字博物馆;周淑秋,副教授,硕士生导师,主要研究方向为计算机仿真。

分布式对象的优势不再明显。

③由于敏捷方法学的兴起,测试先行的开发其价值在很多项目中得到了证实,EJB 严重依赖容器的服务,使得有效的单元测试非常困难。

而 EJB 吹捧的一些有价值的服务也可以找到相应的替代方案,比如在业务层中最重要的事务管理方面,在 EJB 中如果只有一个方法需要复杂的事务行为,所有的其他方法都必须承担 BMT 和 JTA 的复杂性,而 Spring 这种更少侵入性的事务基础框架提供的基于 AOP 的声明性事务管理可以通过配置、在支持多个数据库的 JTA、或者 JDBC、或者其他的特定资源的事务管理 API(例如 JDO 事务管理)之间切换,而不需要对业务代码做任何修改;在业务对象管理方面,EJB 容器也是一种工厂:它借助 JNDI 为服务对象(通常是 SLSB)提供目录服务。EJB 部署描述文件提供了一个从服务名称(JNDI 名称)到实现对象的映射,从而消除了业务接口(EJB 的组件接口)与实现类(EJB 实现类)之间的耦合。EJB 的一个最大优点就是强制调用者使用业务接口而不是直接使用实现类,这是一个很好的编程实践。Java 语言把接口作为头等公民来对待,从而提供了完美的解耦能力,如果在 Java 代码中明确指定"使用接口的哪个实现类",那么针对接口编程带来的好处也就被抵消了,因此需要一些基础设施提供帮助,从而可以在 Java 代码外部选择接口的实现类。EJB 容器就是这样的基础设施,但是并不是唯一的,比如 Spring 框架以一种非常轻量的方法帮助开发者实践"针对接口编程、而非针对类编程"的原则。在远程调用、集群、线程管理、EJB 实例池、资源池、安全等其他方面均有相应的更好解决方案。

综上所述,EJB 帮助证实和普及了很多有价值的想法,但这并不意味着 EJB 的实现就是完美的。所需要的解决方案原本可以不必这么复杂,这正是 EJB 的轻量级替代品出现的理由。

然而,如果符合下列条件之一的话,现在使用 EJB 也是恰当的[6]:

* 开发以中间件为核心的应用,其中 Web 界面只扮演很少的角色,并且需要一个分布式(而不是简单的集群)体系架构,例如:大型金融应用;

* 开发一个基于 RMI 的分布式应用;

* 确实需要有状态 session bean,不能借助 Http Session 对象或者厚客户端来满足这个需求,譬如:同时保存多种客户端的状态。

2 Spring 框架

Spring[3]是一个 2003 年 2 月才开始投入使用的开源项目,是一个服务于所有层面的基于 POJO(Plain Ordinary Java Object)、IoC(Inversion of Control,控制反转)和 AOP(Aspect Oriented Programming,面向切面编程)的开源的轻量级 J2EE 应用框架,不依赖任何的组件,可以有效地组织中间件对象而不管是否还使用了 EJB 技术。同时 Spring 即是全面的又是模块化的(其结构见图 1),不是一个强制性框架,可以选择某个组件独立使用,比如在我们的系统中 MVC 模式的 Web 层框架选择的是开源的 Struts,没有选择 Spring 自己的 MVC Web 框架。Spring Core 是 Spring 的基础,它提供了一个 Ioc 容器管理业务组件,而 JDBC and DAO 模块和 O/R Mapping 模块不仅提供数据访问的抽象,还特别集成了对 Hibernate 的支持,并且提供了缓冲连接池、事务处理等重要的服务功能,保证了系统的性能和数据的完整性,在本系统中持久层则采用 Hibernate 技术。Spring 提供了管理业务对象的一致方法,并可通过对接口编程而不是对类编程去实现,Spring 的框架基础是基于使用 JavaBean 属性的 IoC 容器,IoC 容器提供了一种无侵入式的高扩展性框架,无需代码中涉及 Spring 专有类,即可将其纳入 Spring 容器进行管理,也就是由容器控制组件之间的关系,而非传统实现中由程序直接操控,这种控制由程序代码到外部容器的转移称为"反转",而 DI(Dependence Injection,依赖注入)是对 IoC 更形象的解释,即由容器在运行时期中动态地将依赖关系(比如构造参数、构造对象或接口)注入到组件之中。IoC/DI 机制有三种实现:

①接口注入(即工厂模式)。

②构造子注入(在构造方法中实现依赖)。

③设值注入(使用 Setter 方法),Spring 采取这种方式,通过配置文件管理组件协作对象,创建可以构造组件的 IoC 容器,这样不需要编写工厂模式或者其他构造的方法,就可以通过容器直接获取所需的业务组件。

系统在实现时需要自主开发许多 JavaBean,比如各种持久层 DAO Bean、业务层 Service Bean,然后在相应的 Bean 配置文件(XML 格式)中修改相应的属性,这些 JavaBean 就可以直接嵌入 IoC 容器,成为新的业务方法层组件,与 EJB 组件相比十分简洁方便,充分体现出 IoC 容器的无侵入性。轻量级容器依靠反转控制(Inversion of Control,IoC)模式不仅可以灵活地装配业务对象,更重要的是它有机会在一个完整的环境中进行工作,业务开发不用再考虑"环境"因素。使用 Spring 框架,业务组件都是通过 Spring IoC 容器注入。

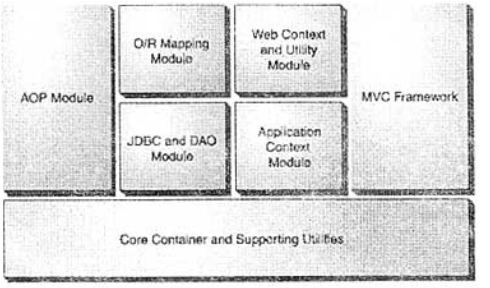

图 1 Spring 模块图

另外 Spring 还提供了一个用标准 Java 语言编写的 AOP 框架,给 POJO 提供声明式的事务管理和其他企业事

务,使得事务管理相对简单清晰。AOP 是一种超越 OOP 的编程模式,它允许程序员将横切关注点(Crosscutting Concerns,散布在多个模块中的一致概念,如同步处理、事务处理、持久化、日志等都是典型的横切关注点)封装成清晰的可重用模块,然后通过组合这些模块和功能性组件获得系统的实现。Spring 和其他框架 AOP(除了 JBoss AOP 外)使用了源码级元数据。借助 AOP 的横切服务甚至可以把事物 try/catch 这样"例行公务"的事情放在编程范围之外。从概念上来说,JNDI 查找、事务、安全之类的基础设施也就是所谓的环境都与业务逻辑横切的。由于业务组件都是由轻量级容器负责管理生命周期,使用者只要通过容器进行组件的访问,就应该让容器插入额外的代码来管理横切的基础设施。Spring 提供了将基础设施进行外部声明(在 XML 文件中配置),不让它们"侵入"到应用代码。Spring 结合 AOP 框架概念提供声明性事务管理,是 Spring 提供企业服务的重要表现之一。

本系统以 Spring 框架(IoC,AOP)取代 EJB 容器作为业务逻辑层的基础构架,Hibernate 取代 CMP 构成轻量级 B2C 电子商务平台。业务层是基于 IoC 容器的 BeanFactory 或 ApplicationContext 以及在此基础上基于 AOP 框架的 JavaBean 组件,同时只需在 JavaBean 的配置文件 Bean.xml 或者 Config.xml 中修改有关属性,就可以完成业务方法层组件的配置和加载。

3　基于 Spring 的系统实现

本系统的架构为 Velocity + Struts + Spring + Hibernate。Velocity[7]是基于 Java 的模板引擎,它的作用等同于 JSP;Struts 是工作在 Web 层表示层的 MVC 开源框架;Hibernate 是工作在持久层的一个 O/R Mapping 开源框架,这里不做具体介绍。本系统是实现一个类似云网的一个网上销售游戏点卡的 B2C 电子商务平台,系统采用 IBM 的 WebShpere 作为 Web 服务器,数据库采用 Oracle 9i。

下面以系统中用户注册模块描述上述的轻量级方案在实际中是如何实现的。

系统的业务流程(见图 2)是:在 View(UI 界面)层,通过 Velocity 页面实现交互界面,负责传送请求(request)和接受响应(response)。表示层引入 Struts 控制分发请求,根据 Struts ActionServlet 接受到的 Request 委派相应的 Action,Action 只负责向业务层传递参数和指定调用业务逻辑的 Service,管理服务组件的 Spring IoC 容器负责向 Action 提供其所指定的业务模型(就是系统的 Service)组件,在业务逻辑 Service 层,根据具体的业务逻辑要组装不同的 DAO(数据访问对象)来完成一个业务模块功能,即 Service 中每个方法对应一个业务逻辑,而每个业务逻辑的实现是由不同的 DAO 组装起来协同实现的,比如注册用户的时候需要调用 UserDAO 像数据库中添加一条用户信息,同时也要调用 AccountDAO 完成对该用户开账户,并把账户初始金额置为系统默认值。在 Service 仍旧使用 IoC 提供所需要的 DAO 组件完成业务逻辑,并提供事务管理、缓冲池等容器组件提升系统性能和保证数据完整性。而持久层的 DAO 依赖于 Hibernate 的对象化映射和数据库交互以处理 DAO 组件请求的数据,并返回处理结果。

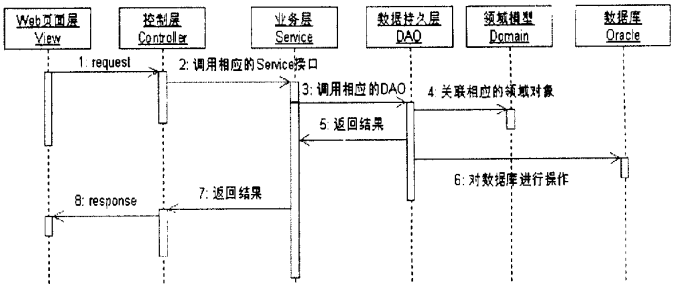

图 2　基本的业务流程图

核心代码如下:

●Velocity 页面中填写用户信息的表单:

```
<form method="post" action="/regAction.do">
  <input type="text" name="name"/>
  ……
</form>
```

●Action 层接收请求传来的参数,同时调用相应的业务组件:

```
public class RegAction extends Action{
UserService service = null;//关于用户的业务组件加载 Spring 配置文件的信息
ApplicationContext ctx = WebApplicationContextUtils
  getWebApplicationContext(
    this.getServlet().getServletContext());
service =
(UserService)ctx.getBean("UserServiceProxy");
//将 Velocity 中填写的用户信息组装成一个领域模型对象
User user = new User();
user.setName((String) dynaActionForm.get("name"));
……//设置用户其他信息
//调用业务组件层添加用户
ReturnInfo ret = service.regUser(user);
  if(ret.getFlag()){// 注册成功
  return mapping.findForward("success");
  }else{//失败
  return mapping.findForward("fail");
  }
}
```

●业务逻辑层 Service:

```
public interface UserInfo{//处理用户业务的接口注册用户业务
  public ReturnInfo regUser(User user);
public ReturnInfo getUsers();
  ……//其他业务逻辑
```

```
}
```

●用户业务的接口实现类：

```
public class UserInfoImpl implements UserInfo{
private UserDAO userDAO;//处理用户的 DAO
private AccountDAO accountDAO;//处理账户 DAO
  public ReturnInfo regUser(User user){
  user = userDAO.getUserByName(user.getName());
  if(user = = null){//没有该用户,将用户信息入库
  //用户密码加密
user.setPassword(new MD5().getStr(user.getPassword()));
//设置该用户账户存款为 0
accountDAO.set(user,0);
userDAO.add(user);
return new ReturnInfo(true,"1000001","success");
}else//该用户已经注册,返回提示信息
  return new ReturnInfo(false,"1000001","has user");
}
  ……//其他业务逻辑模块实现
Public UserDAO getUserDAO();
Public void setUserDAO(UserDAO dao);
……//其他 DAO 属性的 set/get 方法
}
```

●基于 Hibernate 技术的持久层 DAO,提供一系列关于数据库的增删改查方法：

```
public interface UserDAO{//用户 DAO 接口对数据库的增删改查
  public int addUser(User user);
  public User getUserById(int id);
  ……
}
//用户 DAO 接口实现类
public class UserDAOImpl implements UserDAO extends Hibernate-
DaoSupport{
//对数据库 User 表的增删改查
  public int addUser(User user){
  getHibernateTemplate().add(user);
  }
  ……
}
```

●Spring 配置文件中 Bean 的信息：

```
<bean id="UserDAO"
class="com.bmips.user..UserDAOImpl">
    <property name="sessionFactory">
        <ref bean="mySessionFactory" />
    </property>
</bean>
<bean id="UserService"
  class="com.bmips.user.UserServiceImpl">
    <property name="userDAO">
      <ref bean="UserDAO" /></property>
    <property name="accountDAO">
      <ref bean="AccountDAO" /></property>
</bean>
<bean id="transactionManager" //Spring 事务管理
class = "org.springframework.orm.hibernate.HibernateTransactionManager">
<property name="sessionFactory">
    <ref bean="mySessionFactory" /></property>
</bean>
<bean id="UserServiceProxy"
class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
<property name="transactionManager">
    <ref bean="transactionManager" /></property>
    <property name="target">
    <ref bean="UserService" /></property>
    <property name="transactionAttributes">
<!--指定使用事务的方法前缀-->
    <props><prop key="reg*">
PROPAGATION_REQUIRED,-Exception</prop>
      ……</props></property>
</bean>
```

4 结论与展望

目前,该模型系统已经在河北网通游戏运营部中加以使用。Spring 是一个集成 MVC 模式、IoC 模式、AOP 编程等多项前沿技术于一身的新兴框架,经过近 2 年在很多实际项目中使用,技术趋于成熟,已经得到很多大厂商如 IBM 等的重视,并将其纳入到自己的产品中。去年盛行的基于动态语言 Ruby 的框架 Ruby On Rails[8] 以其开发速度快、零配置文件等特点引起 J2EE 开发社群的重视,其架构概念和 J2EE 这种轻量级架构很相似。至于 Ruby On Rails 是否会对 Spring 技术带来冲击还需要实践的检验。

参考文献：

[1] Allamaraju S,Buest C,Davies J,et al. Professional Java Server Programming J2EE 1.3 Edition[M]. [s.l.]:Sun Microsystems Inc,2002:649-692,757-829.

[2] Crawford W,Kaplan J. J2EE Design Patterns[M]. America:Wrox,2003:41-49.

[3] Spring[EB/OL]. 2005-06. http://www.springframework.org.

[4] Struts[EB/OL]. 2004-01. http://struts.apache.org/.

[5] Hibernate[EB/OL]. 2004-06. http://www.hibernate.org/.

[6] Johnson R,Hoeller J. J2EE Development without EJB[M]. 北京:电子工业出版社. 2005:11-12.

[7] Velocity[EB/OL]. 2005-01. http://jakarta.apache.org/velocity/.

[8] Ruby On Rails[EB/OL]. 2006-01. http://www.rubyonrails.org/.