JpetStore 4.0 是 ibatis 的最新示例程序，基于 Struts MVC 框架（注：非传统 Struts 开发模式），以 ibatis 作为持久化层。该示例程序设计优雅，层次清晰，可以学习以及作为一个高效率的编程模型参考。本文是在其基础上，采用 Spring 对其中间层（业务层）进行改造。使开发量进一步减少，同时又拥有了 Spring 的一些好处…

**1. 前言**

JpetStore 4.0 是 ibatis 的最新示例程序。ibatis 是开源的持久层产品，包含 SQL Maps 2.0 和 Data Access Objects 2.0 框架。JpetStore 示例程序很好的展示了如何利用 ibatis 来开发一个典型的 J2EE web 应用程序。JpetStore 有如下特点：

* ibatis 数据层
* POJO 业务层
* POJO 领域类
* Struts MVC
* JSP 表示层

以下是本文用到的关键技术介绍，本文假设您已经对 Struts，SpringFramewok，ibatis 有一定的了解，如果不是，请首先查阅附录中的参考资料。

* Struts 是目前 Java Web MVC 框架中不争的王者。经过长达五年的发展，Struts 已经逐渐成长为一个稳定、成熟的框架，并且占有了 MVC 框架中最大的市场份额。但是 Struts 某些技术特性上已经落后于新兴的 MVC 框架。面对 Spring MVC、Webwork2 这些设计更精密，扩展性更强的框架，Struts 受到了前所未有的挑战。但站在产品开发的角度而言，Struts 仍然是最稳妥的选择。本文的原型例子 JpetStore 4.0 就是基于 Struts 开发的，但是不拘泥于 Struts 的传统固定用法，例如只用了一个自定义 Action 类，并且在 form bean 类的定义上也是开创性的，令人耳目一新，稍后将具体剖析一下。
* Spring Framework 实际上是 Expert One-on-One J2EE Design and Development 一书中所阐述的设计思想的具体实现。Spring Framework 的功能非常多。包含 AOP、ORM、DAO、Context、Web、MVC 等几个部分组成。Web、MVC 暂不用考虑，JpetStore 4.0 用的是更成熟的 Struts 和 JSP；DAO 由于目前 Hibernate、JDO、ibatis 的流行，也不考虑，JpetStore 4.0 用的就是 ibatis。因此最需要用的是 AOP、ORM、Context。Context 中，最重要的是 Beanfactory，它能将接口与实现分开，非常强大。目前 AOP 应用最成熟的还是在事务管理上。
* ibatis 是一个功能强大实用的 SQL Map 工具，不同于其他 ORM 工具（如 hibernate），它是将 SQL 语句映射成 Java 对象，而对于 ORM 工具，它的 SQL 语句是根据映射定义生成的。ibatis 以 SQL 开发的工作量和数据库移植性上的让步，为系统设计提供了更大的自由空间。有 ibatis 代码生成的工具，可以根据 DDL 自动生成 ibatis 代码，能减少很多工作量。

[**回页首**](https://www.ibm.com/developerworks/cn/java/j-s-s-i/#ibm-pcon)

**2. JpetStore 简述**

**2.1. 背景**

最初是 Sun 公司的 J2EE petstore，其最主要目的是用于学习 J2EE，但是其缺点也很明显，就是过度设计了。接着 Oracle 用 J2EE petstore 来比较各应用服务器的性能。微软推出了基于 .Net 平台的 Pet shop，用于竞争 J2EE petstore。而 JpetStore 则是经过改良的基于 struts 的轻便框架 J2EE web 应用程序，相比来说，JpetStore 设计和架构更优良，各层定义清晰，使用了很多最佳实践和模式，避免了很多"反模式"，如使用存储过程，在 java 代码中嵌入 SQL 语句，把 HTML 存储在数据库中等等。最新版本是 JpetStore 4.0。

**2.2. JpetStore 开发运行环境的建立**

1、开发环境

* Java SDK 1.4.2
* Apache Tomcat 4.1.31
* Eclipse-SDK-3.0.1-win32
* HSQLDB 1.7.2

2、Eclipse 插件

* EMF SDK 2.0.1：Eclipse 建模框架，lomboz 插件需要，可以使用 runtime 版本。
* lomboz 3.0：J2EE 插件，用来在 Eclipse 中开发 J2EE 应用程序
* Spring IDE 1.0.3：Spring Bean 配置管理插件
* xmlbuddy\_2.0.10：编辑 XML，用免费版功能即可
* tomcatPluginV3：tomcat 管理插件
* Properties Editor：编辑 java 的属性文件 , 并可以预览以及自动存盘为 Unicode 格式。免去了手工或者 ANT 调用 native2ascii 的麻烦。

3、示例源程序

* ibatis 示例程序 JpetStore 4.0 http://www.ibatis.com/jpetstore/jpetstore.html
* 改造后的源程序（+spring）（源码链接）

**2.3. 架构**

**图 1 JpetStore 架构图**

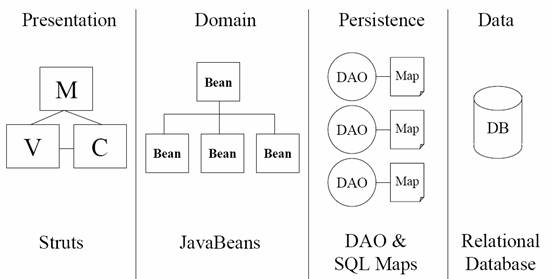


图 1 是 JPetStore 架构图，更详细的内容请参见 JPetStore 的白皮书。参照这个架构图，让我们稍微剖析一下源代码，得出 JpetStore 4.0 的具体实现图（见图 2），思路一下子就豁然开朗了。前言中提到的非传统的 struts 开发模式，关键就在 struts Action 类和 form bean 类上。

struts Action 类只有一个：BeanAction。没错，确实是一个！与传统的 struts 编程方式很不同。再仔细研究 BeanAction 类，发现它其实是一个通用类，利用反射原理，根据 URL 来决定调用 formbean 的哪个方法。BeanAction 大大简化了 struts 的编程模式，降低了对 struts 的依赖（与 struts 以及 WEB 容器有关的几个类都放在 com.ibatis.struts 包下，其它的类都可以直接复用）。利用这种模式，我们会很容易的把它移植到新的框架如 JSF，spring。

这样重心就转移到 form bean 上了，它已经不是普通意义上的 form bean 了。查看源代码，可以看到它不仅仅有数据和校验 / 重置方法，而且已经具有了行为，从这个意义上来说，它更像一个 BO(Business Object)。这就是前文讲到的，BeanAction 类利用反射原理，根据 URL 来决定调用 form bean 的哪个方法（行为）。form bean 的这些方法的签名很简单，例如：

**public String myActionMethod() {**

**//..work**

**return "success";**

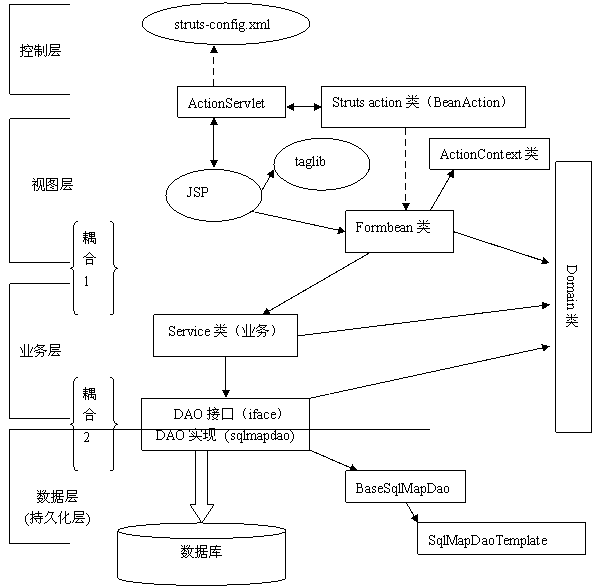
**}**

**方法的返回值直接就是字符串，对应的是 forward 的名称，而不再是 ActionForward 对象，创建 ActionForward 对象的任务已经由 BeanAction 类代劳了。**

**另外，程序还提供了 ActionContext 工具类，该工具类封装了 request 、response、form parameters、request attributes、session attributes 和 application attributes 中的数据存取操作，简单而线程安全，form bean 类使用该工具类可以进一步从表现层框架解耦。**

**在这里需要特别指出的是，BeanAction 类是对 struts 扩展的一个有益尝试，虽然提供了非常好的应用开发模式，但是它还非常新，一直在发展中。**

**图 2 JpetStore 4.0 具体实现**



**2.4. 代码剖析**

下面就让我们开始进一步分析 JpetStore4.0 的源代码，为下面的改造铺路。

* BeanAction.java 是唯一一个 Struts action 类，位于 com.ibatis.struts 包下。正如上文所言，它是一个通用的控制类，利用反射机制，把控制转移到 form bean 的某个方法来处理。详细处理过程参考其源代码，简单明晰。
* Form bean 类位于 com.ibatis.jpetstore.presentation 包下，命名规则为 \*\*\*Bean。Form bean 类全部继承于 BaseBean 类，而 BaseBean 类实际继承于 ActionForm，因此，Form bean 类就是 Struts 的 ActionForm，Form bean 类的属性数据就由 struts 框架自动填充。而实际上，JpetStore4.0 扩展了 struts 中 ActionForm 的应用： Form bean 类还具有行为，更像一个 BO, 其行为（方法）由 BeanAction 根据配置（struts-config.xml）的 URL 来调用。虽然如此，我们还是把 Form bean 类定位于表现层。

Struts-config.xml 的配置里有 3 种映射方式，来告诉 BeanAction 把控制转到哪个 form bean 对象的哪个方法来处理。

以这个请求连接为例 http://localhost/jpetstore4/shop/viewOrder.do

1. URL Pattern

<action path="/shop/viewOrder" type="com.ibatis.struts.BeanAction"

name="orderBean" scope="session"

validate="false">

<forward name="success" path="/order/ViewOrder.jsp"/>

</action>

此种方式表示，控制将被转发到"orderBean"这个 form bean 对象 的"viewOrder"方法（行为）来处理。方法名取"path"参数的以"/"分隔的最后一部分。

2. Method Parameter

<action path="/shop/viewOrder" type="com.ibatis.struts.BeanAction"

name="orderBean" parameter="viewOrder" scope="session"

validate="false">

<forward name="success" path="/order/ViewOrder.jsp"/>

</action>

此种方式表示，控制将被转发到"orderBean"这个 form bean 对象的"viewOrder"方法（行为）来处理。配置中的"parameter"参数表示 form bean 类上的方法。"parameter"参数优先于"path"参数。

3. No Method call

<action path="/shop/viewOrder" type="com.ibatis.struts.BeanAction"

name="orderBean" parameter="\*" scope="session"

validate="false">

<forward name="success" path="/order/ViewOrder.jsp"/>

</action>

此种方式表示，form bean 上没有任何方法被调用。如果存在"name"属性，则 struts 把表单参数等数据填充到 form bean 对象后，把控制转发到"success"。否则，如果 name 为空，则直接转发控制到"success"。

这就相当于 struts 内置的 org.apache.struts.actions.ForwardAction 的功能

<action path="/shop/viewOrder" type="org.apache.struts.actions.ForwardAction"

parameter="/order/ViewOrder.jsp " scope="session" validate="false">

</action>

* Service 类位于 com.ibatis.jpetstore.service 包下，属于业务层。这些类封装了业务以及相应的事务控制。Service 类由 form bean 类来调用。
* com.ibatis.jpetstore.persistence.iface 包下的类是 DAO 接口，属于业务层，其屏蔽了底层的数据库操作，供具体的 Service 类来调用。DaoConfig 类是工具类（DAO 工厂类），Service 类通过 DaoConfig 类来获得相应的 DAO 接口，而不用关心底层的具体数据库操作，实现了如图 2 中 { 耦合 2} 的解耦。
* com.ibatis.jpetstore.persistence.sqlmapdao 包下的类是对应 DAO 接口的具体实现，在 JpetStore4.0 中采用了 ibatis 来实现 ORM。这些实现类继承 BaseSqlMapDao 类，而 BaseSqlMapDao 类则继承 ibatis DAO 框架中的 SqlMapDaoTemplate 类。ibatis 的配置文件存放在 com.ibatis.jpetstore.persistence.sqlmapdao.sql 目录下。这些类和配置文件位于数据层
* Domain 类位于 com.ibatis.jpetstore.domain 包下，是普通的 javabean。在这里用作数据传输对象（DTO），贯穿视图层、业务层和数据层，用于在不同层之间传输数据。

剩下的部分就比较简单了，请看具体的源代码，非常清晰。

**2.5. 需要改造的地方**

JpetStore4.0 的关键就在 struts Action 类和 form bean 类上，这也是其精华之一（虽然该实现方式是试验性，待扩充和验证），在此次改造中我们要保留下来，即控制层一点不变，表现层获取相应业务类的方式变了（要加载 spring 环境），其它保持不变。要特别关注的改动是业务层和持久层，幸运的是 JpetStore4.0 设计非常好，需要改动的地方非常少，而且由模式可循，如下：

1. 业务层和数据层用 Spring BeanFactory 机制管理。

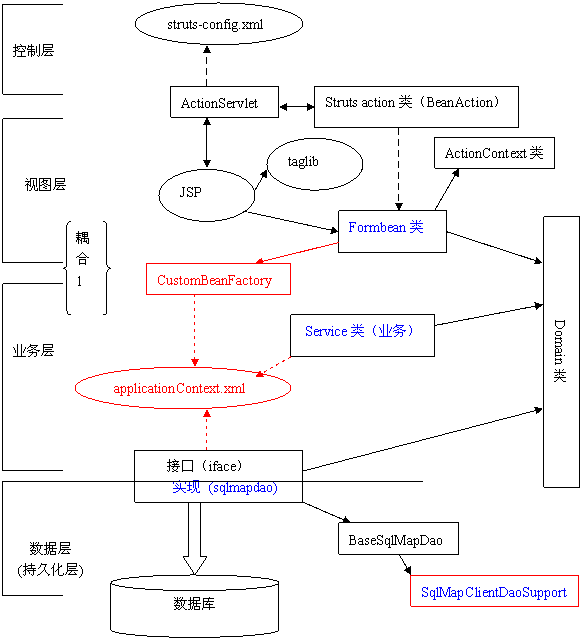
2. 业务层的事务由 spring 的 aop 通过声明来完成。

3. 表现层（form bean）获取业务类的方法改由自定义工厂类来实现（加载 spring 环境）。

[**回页首**](https://www.ibm.com/developerworks/cn/java/j-s-s-i/#ibm-pcon)

**3. JPetStore 的改造**

**3.1. 改造后的架构**

****

**其中红色部分是要增加的部分，蓝色部分是要修改的部分。下面就让我们逐一剖析。**

**3.2. Spring Context 的加载**

为了在 Struts 中加载 Spring Context，一般会在 struts-config.xml 的最后添加如下部分：

**<plug-in className="org.springframework.web.struts.ContextLoaderPlugIn">**

**<set-property property="contextConfigLocation"**

**value="/WEB-INF/applicationContext.xml" />**

**</plug-in>**

**Spring 在设计时就充分考虑到了与 Struts 的协同工作，通过内置的 Struts Plug-in 在两者之间提供了良好的结合点。但是，因为在这里我们一点也不改动 JPetStore 的控制层 ( 这是 JpetStore4.0 的精华之一 )，所以本文不准备采用此方式来加载 ApplicationContext。我们利用的是 spring framework 的 BeanFactory 机制 , 采用自定义的工具类（bean 工厂类）来加载 spring 的配置文件，从中可以看出 Spring 有多灵活，它提供了各种不同的方式来使用其不同的部分 / 层次，您只需要用你想用的，不需要的部分可以不用。**

**具体的来说，就是在 com.ibatis.spring 包下创建 CustomBeanFactory 类，spring 的配置文件 applicationContext.xml 也放在这个目录下。以下就是该类的全部代码，很简单：**

**public final class CustomBeanFactory {**

**static XmlBeanFactory factory = null;**

**static {**

**Resource is = new InputStreamResource(**

**CustomBeanFactory.class.getResourceAsStream("applicationContext.xml"));**

**factory = new XmlBeanFactory(is);**

**}**

**public static Object getBean(String beanName){**

**return factory.getBean(beanName);**

**}**

**}**

**实际上就是封装了 Spring 的 XMLBeanFactory 而已，并且 Spring 的配置文件只需要加载一次，以后就可以直接用 CustomBeanFactory.getBean("someBean") 来获得需要的对象了 ( 例如 someBean)，而不需要知道具体的类。CustomBeanFactory 类用于 { 耦合 1} 的解耦。**

**CustomBeanFactory 类在本文中只用于表现层的 form bean 对象获得 service 类的对象，因为我们没有把 form bean 对象配置在 applicationContext.xml 中。但是，为什么不把表现层的 form bean 类也配置起来呢，这样就用不着这 CustomBeanFactory 个类了，Spring 会帮助我们创建需要的一切？问题的答案就在于 form bean 类是 struts 的 ActionForm 类！如果大家熟悉 struts，就会知道 ActionForm 类是 struts 自动创建的：在一次请求中，struts 判断，如果 ActionForm 实例不存在，就创建一个 ActionForm 对象，把客户提交的表单数据保存到 ActionForm 对象中。因此 formbean 类的对象就不能由 spring 来创建，但是 service 类以及数据层的 DAO 类可以，所以只有他们在 spring 中配置。**

**所以，很自然的，我们就创建了 CustomBeanFactory 类，在表现层来衔接 struts 和 spring。就这么简单，实现了另一种方式的 { 耦合一 } 的解耦。**

**3.3. 表现层**

上面分析到，struts 和 spring 是在表现层衔接起来的，那么表现层就要做稍微的更改，即所需要的 service 类的对象创建上。以表现层的 AccountBean 类为例：

原来的源代码如下

**private static final AccountService accountService = AccountService.getInstance();**

**private static final CatalogService catalogService = CatalogService.getInstance();**

**改造后的源代码如下**

**private static final AccountService accountService =**

**(AccountService)CustomBeanFactory.getBean("AccountService");**

**private static final CatalogService catalogService =**

**(CatalogService)CustomBeanFactory.getBean("CatalogService");**

**其他的几个 presentation 类以同样方式改造。这样，表现层就完成了。关于表现层的其它部分如 JSP 等一概不动。也许您会说，没有看出什么特别之处的好处啊？你还是额外实现了一个工厂类。别着急，帷幕刚刚开启，spring 是在表现层引入，但您发没发现：**

* **presentation 类仅仅面向 service 类的接口编程，具体"AccountService"是哪个实现类，presentation 类不知道，是在 spring 的配置文件里配置。（本例中，为了最大限度的保持原来的代码不作变化，没有抽象出接口）。Spring 鼓励面向接口编程，因为是如此的方便和自然，当然您也可以不这么做。**
* **CustomBeanFactory 这个工厂类为什么会如此简单，因为其直接使用了 Spring 的 BeanFactory。Spring 从其核心而言，是一个 DI 容器，其设计哲学是提供一种无侵入式的高扩展性的框架。为了实现这个目标，Spring 大量引入了 Java 的 Reflection 机制，通过动态调用的方式避免硬编码方式的约束，并在此基础上建立了其核心组件 BeanFactory，以此作为其依赖注入机制的实现基础。org.springframework.beans 包中包括了这些核心组件的实现类，核心中的核心为 BeanWrapper 和 BeanFactory 类。**

**3.4. 持久层**

在讨论业务层之前，我们先看一下持久层，如下图所示：

****

**在上文中，我们把 iface 包下的 DAO 接口归为业务层，在这里不需要做修改。ibatis 的 sql 配置文件也不需要改。要改的是 DAO 实现类，并在 spring 的配置文件中配置起来。**

**1、修改基类**

**所有的 DAO 实现类都继承于 BaseSqlMapDao 类。修改 BaseSqlMapDao 类如下：**

**public class BaseSqlMapDao extends SqlMapClientDaoSupport {**

**protected static final int PAGE\_SIZE = 4;**

**protected SqlMapClientTemplate smcTemplate = this.getSqlMapClientTemplate();**

**public BaseSqlMapDao() {**

**}**

**}**

**使 BaseSqlMapDao 类改为继承于 Spring 提供的 SqlMapClientDaoSupport 类，并定义了一个保护属性 smcTemplate，其类型为 SqlMapClientTemplate。关于 SqlMapClientTemplate 类的详细说明请参照附录中的"Spring 中文参考手册"**

**2、修改 DAO 实现类**

**所有的 DAO 实现类还是继承于 BaseSqlMapDao 类，实现相应的 DAO 接口，但其相应的 DAO 操作委托 SqlMapClientTemplate 来执行，以 AccountSqlMapDao 类为例，部分代码如下：**

**public List getUsernameList() {**

**return smcTemplate.queryForList("getUsernameList", null);**

**}**

**public Account getAccount(String username, String password) {**

**Account account = new Account();**

**account.setUsername(username);**

**account.setPassword(password);**

**return (Account) smcTemplate**

**.queryForObject("getAccountByUsernameAndPassword", account);**

**}**

**public void insertAccount(Account account) {**

**smcTemplate.update("insertAccount", account);**

**smcTemplate.update("insertProfile", account);**

**smcTemplate.update("insertSignon", account);**

**}**

**就这么简单，所有函数的签名都是一样的，只需要查找替换就可以了！**

**3、除去工厂类以及相应的配置文件**

**除去 DaoConfig.java 这个 DAO 工厂类和相应的配置文件 dao.xml，因为 DAO 的获取现在要用 spring 来管理。**

**4、DAO 在 Spring 中的配置（applicationContext.xml）**

**<bean id="dataSource"**

**class="org.springframework.jdbc.datasource.DriverManagerDataSource">**

**<property name="driverClassName">**

**<value>org.hsqldb.jdbcDriver</value>**

**</property>**

**<property name="url">**

**<value>jdbc:hsqldb:hsql://localhost/xdb</value>**

**</property>**

**<property name="username">**

**<value>sa</value>**

**</property>**

**<property name="password">**

**<value></value>**

**</property>**

**</bean>**

**<!-- ibatis sqlMapClient config -->**

**<bean id="sqlMapClient"**

**class="org.springframework.orm.ibatis.SqlMapClientFactoryBean">**

**<property name="configLocation">**

**<value>**

**classpath:com\ibatis\jpetstore\persistence\sqlmapdao\sql\sql-map-config.xml**

**</value>**

**</property>**

**<property name="dataSource">**

**<ref bean="dataSource"/>**

**</property>**

**</bean>**

**<!-- Transactions -->**

**<bean id="TransactionManager"**

**class="org.springframework.jdbc.datasource.DataSourceTransactionManager">**

**<property name="dataSource">**

**<ref bean="dataSource"/>**

**</property>**

**</bean>**

**<!-- persistence layer -->**

**<bean id="AccountDao"**

**class="com.ibatis.jpetstore.persistence.sqlmapdao.AccountSqlMapDao">**

**<property name="sqlMapClient">**

**<ref local="sqlMapClient"/>**

**</property>**

**</bean>**

**具体的语法请参照附录中的"Spring 中文参考手册"。在这里只简单解释一下：**

**1. 我们首先创建一个数据源 dataSource，在这里配置的是 hsqldb 数据库。如果是 ORACLE 数据库，driverClassName 的值是"oracle.jdbc.driver.OracleDriver"，URL 的值类似于"jdbc:oracle:thin:@wugfMobile:1521:cdcf"。数据源现在由 spring 来管理，那么现在我们就可以去掉 properties 目录下 database.properties 这个配置文件了；还有不要忘记修改 sql-map-config.xml，去掉 <properties resource="properties/database.properties"/> 对它的引用。**

**2. sqlMapClient 节点。这个是针对 ibatis SqlMap 的 SqlMapClientFactoryBean 配置。实际上配置了一个 sqlMapClient 的创建工厂类。configLocation 属性配置了 ibatis 映射文件的名称。dataSource 属性指向了使用的数据源，这样所有使用 sqlMapClient 的 DAO 都默认使用了该数据源，除非在 DAO 的配置中另外显式指定。**

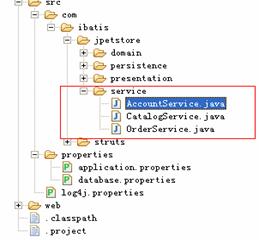
**3. TransactionManager 节点。定义了事务，使用的是 DataSourceTransactionManager。**

**4. 下面就可以定义 DAO 节点了，如 AccountDao，它的实现类是 com.ibatis.jpetstore.persistence.sqlmapdao.AccountSqlMapDao，使用的 SQL 配置从 sqlMapClient 中读取，数据库连接没有特别列出，那么就是默认使用 sqlMapClient 配置的数据源 datasource。**

**这样，我们就把持久层改造完了，其他的 DAO 配置类似于 AccountDao。怎么样？简单吧。这次有接口了：） AccountDao 接口－>AccountSqlMapDao 实现。**

**3.5. 业务层**

业务层的位置以及相关类，如下图所示：

****

**在这个例子中只有 3 个业务类，我们以 OrderService 类为例来改造，这个类是最复杂的，其中涉及了事务。**

**1、在 ApplicationContext 配置文件中增加 bean 的配置：**

**<bean id="OrderService"**

**class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">**

**<property name="transactionManager">**

**<ref local="TransactionManager"></ref>**

**</property>**

**<property name="target">**

**<bean class="com.ibatis.jpetstore.service.OrderService">**

**<property name="itemDao">**

**<ref bean="ItemDao"/>**

**</property>**

**<property name="orderDao">**

**<ref bean="OrderDao"/>**

**</property>**

**<property name="sequenceDao">**

**<ref bean="SequenceDao"/>**

**</property>**

**</bean>**

**</property>**

**<property name="transactionAttributes">**

**<props>**

**<prop key="insert\*">PROPAGATION\_REQUIRED</prop>**

**</props>**

**</property>**

**</bean>**

**定义了一个 OrderService，还是很容易懂的。为了简单起见，使用了嵌套 bean，其实现类是 com.ibatis.jpetstore.service.OrderService，分别引用了 ItemDao，OrderDao，SequenceDao。该 bean 的 insert\* 实现了事务管理 (AOP 方式 )。TransactionProxyFactoryBean 自动创建一个事务 advisor， 该 advisor 包括一个基于事务属性的 pointcut, 因此只有事务性的方法被拦截。**

**2、业务类的修改**

**以 OrderService 为例：**

**public class OrderService {**

**/\* Private Fields \*/**

**private ItemDao itemDao;**

**private OrderDao orderDao;**

**private SequenceDao sequenceDao;**

**/\* Constructors \*/**

**public OrderService() {**

**}**

**/\*\***

**\* @param itemDao 要设置的 itemDao。**

**\*/**

**public final void setItemDao(ItemDao itemDao) {**

**this.itemDao = itemDao;**

**}**

**/\*\***

**\* @param orderDao 要设置的 orderDao。**

**\*/**

**public final void setOrderDao(OrderDao orderDao) {**

**this.orderDao = orderDao;**

**}**

**/\*\***

**\* @param sequenceDao 要设置的 sequenceDao。**

**\*/**

**public final void setSequenceDao(SequenceDao sequenceDao) {**

**this.sequenceDao = sequenceDao;**

**}**

**// 剩下的部分**

**…… .**

**}**

**红色部分为修改部分。Spring 采用的是 Type2 的设置依赖注入，所以我们只需要定义属性和相应的设值函数就可以了，ItemDao，OrderDao，SequenceDao 的值由 spring 在运行期间注入。构造函数就可以为空了，另外也不需要自己编写代码处理事务了（事务在配置中声明），daoManager.startTransaction(); 等与事务相关的语句也可以去掉了。和原来的代码比较一下，是不是处理精简了很多！可以更关注业务的实现。**

[**回页首**](https://www.ibm.com/developerworks/cn/java/j-s-s-i/#ibm-pcon)

**4. 结束语**

ibatis 是一个功能强大实用的 SQL Map 工具，可以直接控制 SQL, 为系统设计提供了更大的自由空间。其提供的最新示例程序 JpetStore 4.0, 设计优雅，应用了迄今为止很多最佳实践和设计模式，非常适于学习以及在此基础上创建轻量级的 J2EE WEB 应用程序。JpetStore 4.0 是基于 struts 的，本文在此基础上，最大程度保持了原有设计的精华以及最小的代码改动量，在业务层和持久化层引入了 Spring。在您阅读了本文以及改造后的源代码后，会深切的感受到 Spring 带来的种种好处：自然的面向接口的编程，业务对象的依赖注入，一致的数据存取框架和声明式的事务处理，统一的配置文件…更重要的是 Spring 既是全面的又是模块化的，Spring 有分层的体系结构，这意味着您能选择仅仅使用它任何一个独立的部分，就像本文，而它的架构又是内部一致。