Struts+Spring+Hibernate 整合教程 v1.0

作者: 陈文光<chenjacken@gmail.com>

作者 Blog: http://www.jacken.com.cn

声明

开源传万世，只因你我皆参与!

```
While (isAlive()) {
        goodGoodstudy();
        dayDayUp();
    }
```

日期: 2008-01-06

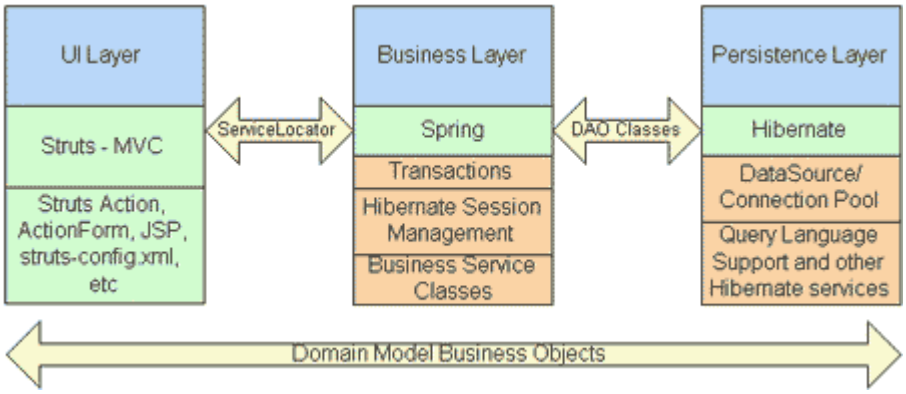
1 SSH 整合理念

1.1 框架

用 java 来建立一个很有价值的 web 应用不是一个简单的任务。在架构这个应用时要考虑很多的因素和问题。从更高的层次来看，开发人员面临着关于如何构建用户接口，何处驻留业务逻辑，以及如何实现数据持久性这些问题。这 3 层都有各自的问题需要回答。而每一层又需要实现那些技术？应用如何设计来进行松散耦合并能进行灵活变更？应用架构是否允许某一层变更而不影响到其它的层次？应用应该如何处理容器一级的服务比如事务？

在为你的应用创建一个架构之前有许多问题需要澄清。幸运的是，有很多开发者都意识到这个问题，并建立了很多框架来解决这些问题。一个良好的框架可以让开发人员减轻重新建立解决复杂问题方案的负担和精力；它可以被扩展以进行内部的定制化；并且有强大的用户社区来支持它。框架通常能很好的解决一个问题。然而，你的应用是分层的，可能每一个层都需要各自的框架。仅仅解决 UI 问题并不意味着你能够很好的将业务逻辑和持久性逻辑和 UI 组件很好的耦合。例如，你不应该使具有 JDBC 代码的业务逻辑放入控制器之中，这不是控制器应该提供的功能。一个 UI 控制器应该是轻量化的组件，由它代表对 UI 范围之外的其它应用层的服务调用。良好的框架自然地形成代码分离的原则。更为重要的是，框架减轻了开发人员从头构建持久层代码的精力，从而集中精力来应用逻辑上，这对客户端来说更为重要。

本文讨论了如何结合几个著名的框架来达到松散耦合，如何设计你的架构，以及如何达到各个层次的一致性设计。面临的挑战是，将框架整合起来，以使每一层都向另外的层次以一种松散的方式来暴露接口，而不管底层功能使用的是什么技术。本文还讨论整合 3 种著名开源框架的一种策略。对表现层，我们使用 Struts；业务层使用 Spring；对于持久层我们使用的是 Hibernate。你尽可以取代这里的某个框架而使用你喜欢的框架已达到同样的效果。图 1 显示了框架被整合起来时，从最高层次看到的视图。



1.2 应用层

许多设计良好的 web 应用，可以被按职责分为四层。这些层次是表现层、持久层、业务层、和领域模型层。每一个层次都有其独特的职责，不能把各自的功能与其它层次相混合。每一个应用层都应该和其它层隔离开来，但允许使用接口在层间进行通信。我们开始来看看每个层，并讨论一下它们各自都应该提供什么和不应该提供什么。

1.2.1 表现层

一个典型的 web 应用的末端是表现层。许多 Java 开发者都知道 Struts 提供了什么东西。然而，太多时候，耦合代码比如业务逻辑被放进 org.apache.struts.Action 中。所以，我们先总结一下 Struts 之类的框架应该提供什么。下面就是 Struts 的职责所在：

- 管理用户的请求和响应
- 提供一个控制起来将调用委托到业务逻辑和其他上游处理
- 将来自于抛出例外的其他层的例外处理到 Struts Action 中
- 组装可以在视图中表现的模型对象
- 执行 UI 校验

下面是一些经常可以使用 Struts 进行编码但是不应该和表现层关联的事情：

- 直接和数据库交互，比如 JDBC 调用
- 与应用相关的业务逻辑和校验
- 事务管理

在表现层中引入这些类型的代码将导致类型耦合和维护负担。

1.2.2 持久层

一个典型 Web 应用的另一端是持久层。这也是应用中最容易很快失控的地方。开发者通常低估了自己构建自己的持久层框架的挑战。一个定制的，内部开发的持久层不仅需要大量的开发时间，并且通常缺乏功能和难以管理。目前有许多解决这些问题的开源对象关系映射 (ORM) 框架。特别地，Hibernate 框架就允许 Java 中的对象-关系的持久性和查询服务。Hibernate 对已经熟悉了 SQL 和 JDBC API 的 Java 开发者来或具有中度的学习曲线。Hibernate 的持久对象基于 POJO 和 Java 群集（collections）。此外，使用 Hibernate 不和你的 IDE 接口。下面列出了你需要在持久性框架中编写的代码类型：

- 查询关系信息到对象中。Hibernate 是通过称为 HQL 的 OO 查询语言，或者使用更有表现能力的规则 API，来完成这个工作的。除了使用对象而不是表，使用字段而不是列的方式，HQL 非常类似于 SQL。也有一些新的特定的 HQL 语言特征需要学习；但是，它们是很容易理解和良好编写的。HQL 是一种用于查询对象的自然语言，而对象，只需要很少的学习曲线吧。.
- 存储、更新和删除存储在数据库中的信息
- 高级的对象关系映射框架比如 Hibernate 支持大部分主流 SQL 数据库，它们支持父/子关系，事务，继承和多态。

下面是应该在持久层避免的一些事情：

- 业务逻辑应该置于应用的更高层中。这里只允许数据访问方法。
- 不应该使持久逻辑和表现逻辑耦合。避免表现组件如 JSP 或者基于 servlet 的类中的逻辑直接和数据访问进行通信。通过将持久性逻辑隔离在其自己的层中，应用将具有更加灵活的修改性而不影响到其他层的代码。例如， Hibernate 可以使用其他持久框架和 API 代替，而不需要修改其它层中的代码。

1.2.3 业务层

典型的 Web 应用的中间组件一般是业务层和服务层。从编程的角度来说，service layer 经常被忽略。这种类型的代码散布于 UI 表现层和持久层并不是不多见。这些都不是正确的地方因为它导致了紧密耦合的应用和难以维护的代码。幸运的是，大多数框架都解决了这个问题。这个空间内最流行的两个框架是 Spring 和 PicoContainer。它们都被视为是具有非常小的足迹（footprint）并且决定如何将你的对象整合在一起的微容器（microcontainer）。这些框架都建立在一种叫做依赖性注入（dependency injection） (也称控制反转（inversion of control：IOC）)的简单概念之上。我们将关注 Spring 中通过针对命名配置参数的 bean 属性的 setter 注入的使用。Spring 也允许一种更加高级的构造器注入（constructor injection）形式作为 setter injection 的可选替代。对象通过简单的 XML 文件进行连接，该配置文件包含对各种对象的引用，比如事务管理处理器（transaction management handler）,对象工厂，包含业务逻辑的服务对象，以及数据访问对象(DAO)。

我们随后会用一些例子来澄清 Spring 中使用这些改变的方式。

业务层应该负责下面的问题：

- 处理应用的业务逻辑和业务校验
- 管理事务
- 允许与其他层进行交互的接口

- 管理业务级对象之间的依赖性
- 加入了表现和持久层之间的灵活性，以便它们不需要彼此进行直接通信
- 从表现层暴露上下文给业务层以获得业务服务
- 管理从业务层到表现层的实现

1.2.4 领域模型层

最后，因为我们要解决实际的问题的 web 应用，我们需要一套在不同的层间移动的对象。领域模型层包含的是表达实际业务对象的对象，比如 Order, OrderLineItem, Product 等等。这一层允许能让开发者不再构建和维护不必要的数据传输对象 DTO 来匹配其领域对象。例如， Hibernate 允许你读取数据库信息到一个领域对象的对象图中，以便你可以在离线的情况下将其表现在 UI 层中。这些对象可以被更新并跨过表现层发送回去，然后进行数据库更新。另外，你不再需要将对象转变成 DTO，因为它们在不同的层间移动时可能会丢失事务。这种模型允许 Java 开发者能够以 OO 风格的方式很自然的处理对象，而不用编写额外的代码。

1.3 整合一个简单的例子

到此，应该对各种层次和组件有一个高层的理解了罢。可以开始一些实践了。再次说明。我们的例子整合了 Struts, Spring, 和 Hibernate 框架。每个框架都包含大量的内容细节，我们不会多述。我们的目的使用一个例子向你说明如何将它们整合在一起构建一个优雅的 Web 应用架构。实例将演示一个请求是如何得到各层的服务的。此应用的用户可以将一个订单保存在数据库中并且察看数据中的已有订单。进一步的增强允许将用户更新和删除现有订单。

首先，我们将常见我们的领域对象，因为它们是要和各层沟通的。这些对象将允许我们能够定义那些对象需要持久化，那些业务逻辑需要提供，以及应该设计那些表现接口。接下来，我们将使用 Hibernate 来为领域对象配置持久层和定义对象关系映射。然后，我们将定义和配置我们的业务层。在完成这些组件后，我们将讨论如何使用 Spring 将这些层关联起来。最后，我们将提供一个表现层，它知道如何与业务服务层通信以及如何处理来自于其他层的例外。

2 Spring 整合 Struts

3.1 三个小窍门

- 接下来的每种整合技术（或者窍门）都有自己的优点和特点。我偏爱其中的一种，但是我知道这三种都能够加深您对 Struts 和 Spring 的理解。在处理各种不同情况的时候，这将给您提供一个广阔的选择范围。方法如下：

1. **使用 Spring 的 ActionSupport 类整合 Structs**
2. **使用 Spring 的 DelegatingRequestProcessor 覆盖 Struts 的 RequestProcessor**
3. **将 Struts Action 管理委托给 Spring 框架**

- 装载应用程序环境

无论您使用哪种技术，都需要使用 Spring 的 ContextLoaderPlugin 为 Struts 的 ActionServlet 装载 Spring 应用程序环境。就像添加任何其他插件一样，简单地向您的 struts-config.xml 文件添加该插件，如下所示：

```
   <plug-in
 className="org.springframework.web.struts.ContextLoaderPlugIn">
       <set-property property="contextConfigLocation"
           value="/WEB-INF/beans.xml" />
   </plug-in>
```

3.2 窍门 1. 使用 Spring 的 ActionSupport

- 步骤：

1. Action 直接继承 ActionSupport
2. 使用 ApplicationContext ctx = getWebApplicationContext();取得 Spring 上下文
3. 取得相应 Bean

```
  //1继承ActionSupport
  public class SearchSubmit extends ActionSupport {
   public ActionForward execute(ActionMapping mapping, ActionForm form,
           HttpServletRequest request, HttpServletResponse response)
           throws IOException, ServletException {
       // 2.使用ApplicationContext ctx = getWebApplicationContext();取得
  Spring上下文
       ApplicationContext ctx = getWebApplicationContext();
       // 3.取得相应Bean
       BookService bookService =(BookService) ctx.getBean("bookService");
   }
}
```

- 优点：

1) 简单

- 缺点：

1) 耦合高

2) 违反 IOC

3) 无法使用多方法的 Action

3.3 窍门 2. 覆盖 RequestProcessor

- 步骤：

1. Action 中，使用 IOC 获得服务，配置 struts-config.xml：

```
    <!-- 一个普通的Action-->
       <action path="/searchSubmit"
           type="ca.nexcel.books.actions.SearchSubmit"
input="/searchEntry.do"
           validate="true" name="searchForm">
           <forward name="success" path="/WEB-INF/pages/detail.jsp" />
           <forward name="failure" path="/WEB-INF/pages/search.jsp" />
       </action>
  <!-- 利用了 <controller> 标记来用 DelegatingRequestProcessor覆盖默认的
  Struts RequestProcessor -->
   <controller
       processorClass="org.springframework.web.struts.
     DelegatingRequestProcessor" />
```

2. Spring 配置文件中注册该动作：

```
  <bean id="bookService"
       class="ca.nexcel.books.business.BookServiceImpl" />
   <bean name="/searchSubmit"
       class="ca.nexcel.books.actions.SearchSubmit">
       <property name="bookService">
           <ref bean="bookService" />
       </property>
   </bean>
```

3. 写具有 JavaBean 属性的 Struts 动作：

```
public class SearchSubmit extends Action {
   // 一个Service属性
   private BookService bookService;
   // getter...
   public BookService getBookService() {
       return bookService;
   }
   // setter...
   public void setBookService(BookService bookService) {
       this.bookService = bookService;
   }
   public ActionForward execute(ActionMapping mapping, ActionForm form,
           HttpServletRequest request, HttpServletResponse response)
           throws IOException, ServletException {
       // 调用bookService,不需要new
       Book book = getBookService().read(isbn.trim());
   }
}
```

3.4 窍门 3. 将动作管理委托给 Spring

- 步骤:

1. Action 中，使用 IOC 获得服务，配置 struts-config.xml：

```
<!-- 一个Action 注意其type Spring 代理类 -->
      <action path="/searchSubmit"
          type="org.springframework.web.struts.DelegatingActionProxy"
          input="/searchEntry.do" validate="true" name="searchForm">
          <forward name="success" path="/WEB-INF/pages/detail.jsp" />
          <forward name="failure" path="/WEB-INF/pages/search.jsp" />
          </action>
```

2. Spring 配置文件中注册该动作：

```
 <bean id="bookService"
       class="ca.nexcel.books.business.BookServiceImpl" />
    <bean name="/searchSubmit"
       class="ca.nexcel.books.actions.SearchSubmit">
       <property name="bookService">
          <ref bean="bookService" />
       </property>
</bean>
```

- 优点

1. 动作委托解决方法是这三种方法中最好的。
2. 不使用 Spring api 编写 Action
3. 利用了 IOC 装配

3.5 拦截 Struts

2.5.1. 前提：

将 Struts 动作委托给 Spring 框架可以将 Spring 的 AOP 拦截器应用于 Struts 动作。

2.5.2. 使用拦截器的步骤：

1. 创建拦截器。
2. 注册拦截器。
3. 声明在何处拦截代码。

2.5.3. 一个例子：

- 一个简单的日志记录拦截器

```
//创建拦截器
public class LoggingInterceptor implements MethodBeforeAdvice {
   public void before(Method method, Object[] objects, Object o)
           throws Throwable {
       System.out.println("logging before!");
   }
 }
```

- 注册拦截器

```
    <!--  Interceptors -->
  <!-- 注册了这个刚才建的拦截器 -->
  <bean name="logger"
      class="ca.nexcel.books.interceptors.LoggingInterceptor" />
  <!-- AutoProxies -->
  <!-- 创建了一个 bean 名称自动代理，它描述如何应用拦截器。还有其他的方法定义拦截
点，但是这种方法常见而简便。 -->
  <bean name="loggingAutoProxy"
      class="org.springframework.aop.framework.autoproxy.
         BeanNameAutoProxyCreator">
      <property name="beanNames">
      <!-- 将 Struts 动作注册为将被拦截的 bean -->
          <value>/searchSubmit</value>
      </property>
      <property name="interceptorNames">
          <list>
          <!-- 当拦截发生时，我执行了在 (1) 处创建的拦截器 bean 的名称 -->
              <value>logger</value>
          </list>
      </property>
  </bean>
```

3 Spring 整合 Hibernate

3.1 为什么要整合?

时至今日，可能极少有 J2EE 应用会直接以 JDBC 方式进行持久层访问。毕竟，用面向对象的程序设计语言来访问关系型数据库，是一件让人沮丧的事情。大部分时候，J2EE 应用都会以 ORM 框架来进行持久层访问，在所有的 ORM 框架中，Hibernate 以其灵巧、轻便的封装赢得了众多开发者的青睐。

在 Spring 框架中，像 JDBC DataSource 或 Hibernate SessionFactory 这样的资源，在应用程序上下文中可以用 bean 实现。需要访问资源的应用程序对象只需通过 bean 引用得到这类预先定义好的实例的引用即可。步骤如下：

1. **配置数据源，有三种方式的数据源**
2. **配置 sessionfactory**
3. **配置事务**

3.2 配置数据源

- **方式一：Spring 内置实现 DriverManagerDataSource**

```
    <!-(1)配置数据源-方式一-->
    <bean id="myDataSource"
    class="org.springframework.jdbc.datasource.DriverManagerDataSource"
>
        <property name="driverClassName"
            value="com.mysql.jdbc.Driver">
        </property>
        <property name="url"
            value="jdbc:mysql://127.0.0.1:3306/prefo">
        </property>
        <property name="username" value="root"></property>
        <property name="password" value=""></property>
    </bean>
```

- **方式二：DBCP 提供的 BasicDataSource**

```
  <!-(1)配置数据源-方式二-->
    <bean id="myDataSource"
        class="org.apache.commons.dbcp.BasicDataSource">
        <property name="driverClassName"
            value="com.mysql.jdbc.Driver">
        </property>
        <property name="url"
            value="jdbc:mysql://127.0.0.1:3306/prefo">
        </property>
        <property name="username" value="root"></property>
        <property name="password" value=""></property>
    </bean>
```

- **方式三：JNDI 数据源**(mysql5,tomcat6 为例)

1. **在 Tomcat 根目录下的 conf\server.xml 配置 Resource：**

```
    <GlobalNamingResources>
        <!-- Editable user database that can also be used by
            UserDatabaseRealm to authenticate users
        -->
        <Resource name="jdbc/mydatasource" auth="Container"
            description="DB Connection" type="javax.sql.DataSource"
            username="root" password=""
            driverClassName="com.mysql.jdbc.Driver"
            url="jdbc:mysql://localhost:3306/prefo" maxActive="5" />
            <!--
            name:资源的JNDI查找的名字
            type:资源
            driverClassName:JDBC驱动程序
            ...
             -->
    </GlobalNamingResources>
```

2. 在 Tomcat 根目录下的 conf\context.xml 配置:

```
<ResourceLink name="jdbc/mydatasource" global="jdbc/mydatasource"
      type="javax.sql.DataSourcer" />
```

3. 在 Spring 的配置文件中配置:

```
<!-(1)配置数据源-方式三,使用JNDI-->
  <bean id="myDataSource"
      class="org.springframework.jndi.JndiObjectFactoryBean">
      <property name="jndiName">
          <value>java:comp/env/jdbc/mydatasource</value>
       <!--java:comp/env/Tomcat的前缀,后为JNDI的名称  -->
      </property>
  </bean>
```

✓ 另外,第 1,2 步可以整合在 conf\context.xml 中配置:

```
<Resource name="jdbc/mydatasource" type="javax.sql.DataSource"
      driverClassName="com.mysql.jdbc.Driver" username="root"
password=""
      url="jdbc:mysql://localhost:3306/prefo" maxIdle="2"
maxWait="5000"
      maxActive="4" />
```

3.3 配置 sessionfactory

- 配置好数据源(以上三种方式的一种)
- 配置 sessionFactory:

```
<!-- (2)装配SessionFactory -->
 <bean id="mySessionFactory"
 class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
    <!-- 传入dataSource -->
    <property name="dataSource">
        <ref bean="myDataSource" />
    </property>
    <!-- 添加映射文件 -->
    <property name="mappingResources">
        <list>
            <value>
                cn/com/jacken/prefo/books/vo/CatelogList.hbm.xml
            </value>
            <value>
                cn/com/jacken/prefo/books/vo/BooksList.hbm.xml
            </value>
            <value>
                cn/com/jacken/prefo/users/vo/UserList.hbm.xml
            </value>
        </list>
    </property>
    <!--hibernate属性  -->
    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.dialect">
                org.hibernate.dialect.MySQLDialect
            </prop>
            <prop key="hibernate.show_sql">true</prop>
        </props>
    </property>
 </bean>
```

✓ **如果不需要事务,直接就可以配置 Dao 且调用了**(dao 要继承 HibernateDaoSupport)

```
<!-- 装配catelogDao -->
  <bean id="catelogDao"
      class="cn.com.jacken.prefo.books.dao.CatelogDaoHibernateImpl">
      <!-- 传入sessionFactory -->
      <property name="sessionFactory">
          <ref bean="mySessionFactory" />
      </property>
  </bean>
```

3.4 配置事务

在上面的配置文件中，部署了控制器组件、业务逻辑组件、DAO 组件，几乎可以形成一个完整的 J2EE 应用。但有一个小小的问题：事务控制。

Spring 提供了非常简洁的声明式事务控制，只需要在配置文件中增加事务控制片段，业务逻辑代码无须任何改变。Spring 的声明式事务逻辑，甚至支持在不同事务策略之间切换。有以下几个步骤:

- **配置好数据源**(以上三种方式的一种)
- **配置 sessionFactory**(同 3.3)
- **配置声明式事务:**

1. **配置 Dao** (dao 要继承 HibernateDaoSupport)

```
<!-- 装配catelogDao -->
<bean id="catelogDao"
    class="cn.com.jacken.prefo.books.dao.CatelogDaoHibernateImpl">
    <!-- 传入sessionFactory -->
    <property name="sessionFactory">
        <ref bean="mySessionFactory" />
    </property>
</bean>
```

2. **装配事务管理器**

```
<!-- (3)装配事务管理器 -->
<bean id="myTransactionManager"
class="org.springframework.orm.hibernate3.HibernateTransactionManager">
    <!-- 传入session -->
    <property name="sessionFactory">
        <ref bean="mySessionFactory" />
    </property>
</bean>
```

3. **配置抽象事务代理**

```
<!-- (4)抽象事务代理 -->
   <bean id="abstractTxProxy"
   class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean"
       abstract="true">
       <!-- 传入事务管理器 -->
       <property name="transactionManager">
           <ref bean="myTransactionManager" />
       </property>
       <property name="transactionAttributes">
           <props>
               <!-- key 指定方法 -->
               <prop key="*">PROPAGATION_REQUIRED</prop>
           </props>
       </property>
   </bean>
```

4. **装配 Service 层**

```
  <!-- 装配CatelogService -->
    <bean id="catelogServiceTarget"
    class="cn.com.jacken.prefo.books.services.CatelogServiceSpringImpl">
        <!-- 传入dao -->
        <property name="catelogDao">
            <ref bean="catelogDao" />
        </property>
        <property name="booksDao">
            <ref bean="booksDao" />
        </property>
    </bean>
```

5. **装配具体事务代理**

```
    <!-- 具体代理事务代理--CatelogService事务 -->
    <bean id="catelogService" parent="abstractTxProxy">
    <!-- 传入CatelogService -->
        <property name="target">
            <ref bean="catelogServiceTarget" />
        </property>
    </bean>
```

参考资料:

培训机构教程,互联网...