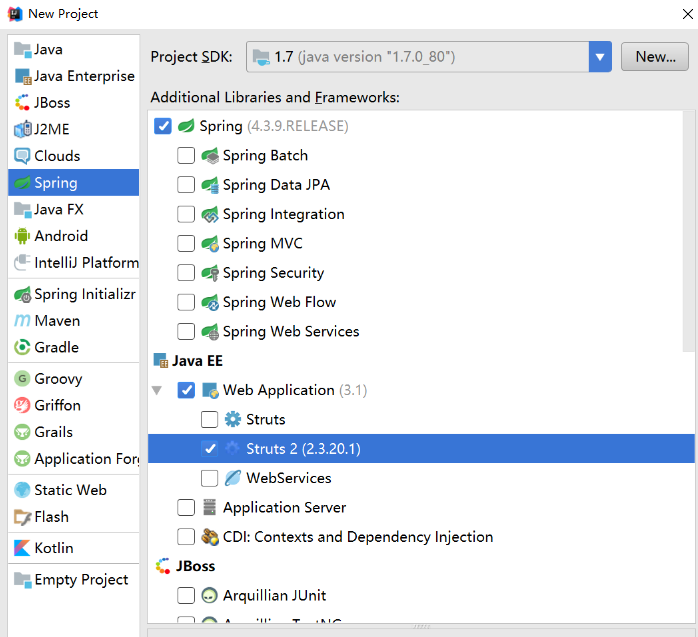
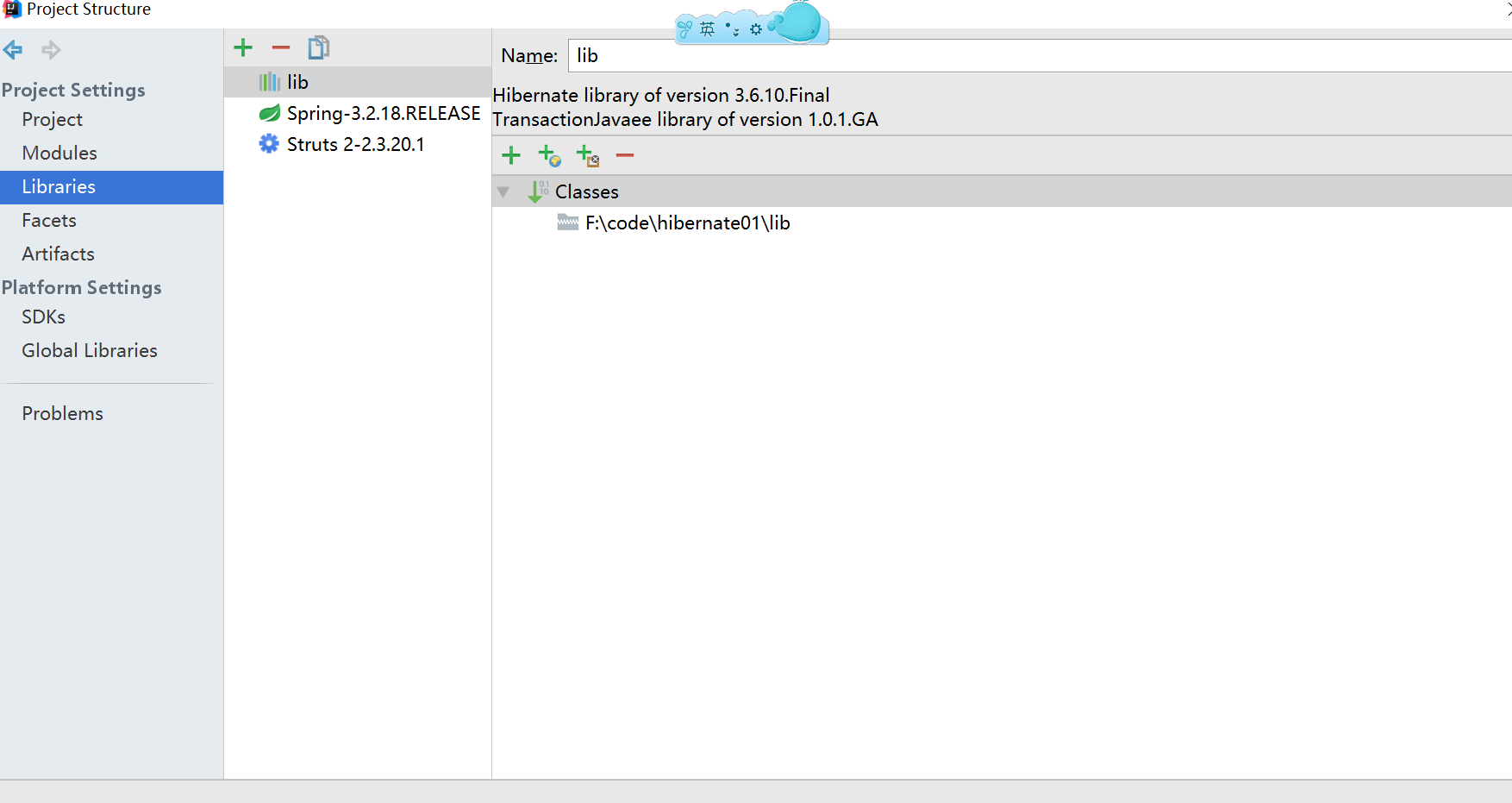
第一步：创立项目引入jar包：



选择框架，spring以及Structs2，记住spring要选择版本，3. 多少的。

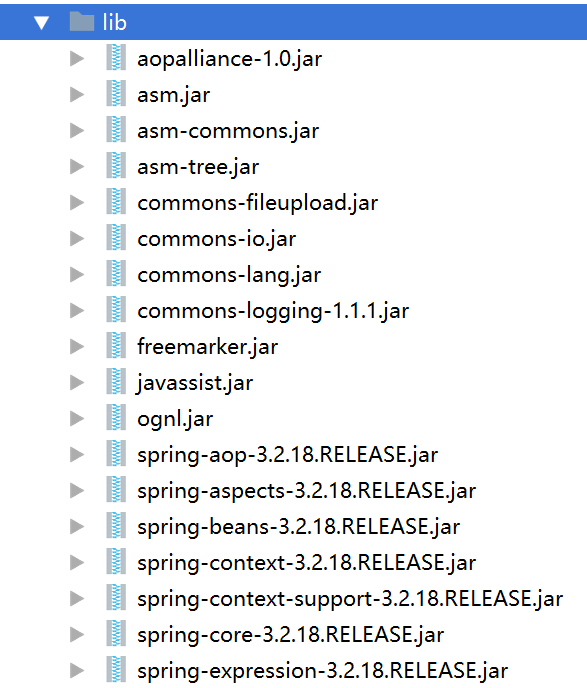
这里为什么没选择hibernate框架？ 因为莫名的hibernate直接下载不下来。

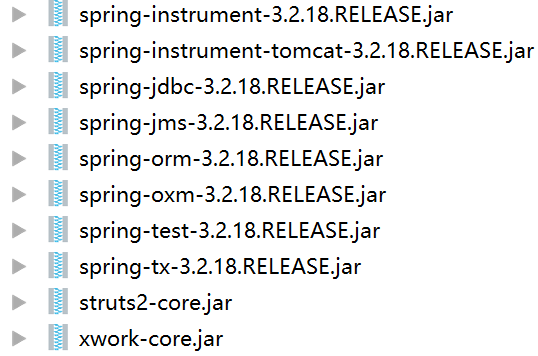
所以，要继续引入hibernate包，我们在



点击右边的+号，选择本地你建立好存放hibernate的jar包引入

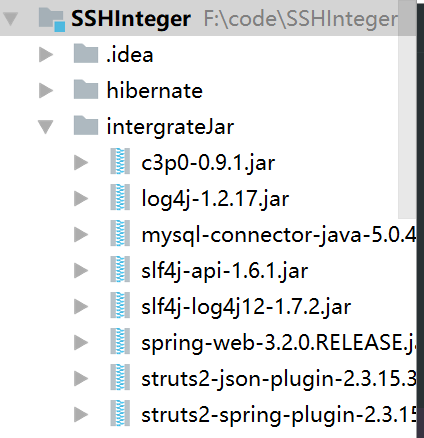
再来看一下lib包中





可以看出来，虽然struts以及spring的核心jar包都有，但是一些连接池jar包啊，驱动jar包啊，以及spring的整合包。

所以我们还要另建一个文件夹，把这些包放进去，然后引用



这种做法会出现很多莫名其妙的bug！！最理想的是，src同级下只有一个lib，而不应该把hibernate、intergrateJar 两个文件夹放在那里。

所以：

最好的解决办法就是，把所有的jar包都放进lib中，然后引入这一个jar文件夹就好了。

第二步：编写基本框架。

2.1 编写配置文件：

在src下编写这

2.1.1 struts.xml

|  |
| --- |
| *<?*xml version="1.0" encoding="UTF-8"*?>* **<!DOCTYPE** struts **PUBLIC** "-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"  "http://struts.apache.org/dtds/struts-2.3.dtd"***>*** <**struts**>  <**package** name="default" extends="struts-default" namespace="/" >  </**package**> </**struts**> |

2.1.2 hibernate.cfg.xml

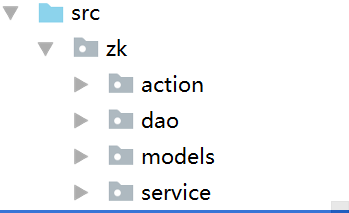
|  |
| --- |
| *<?*xml version="1.0" encoding="UTF-8"*?>* **<!DOCTYPE** hibernate-configuration **PUBLIC** "-//Hibernate/Hibernate Configuration DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd"***>*** <**hibernate-configuration**>  <**session-factory**>  <!-- 必须去配置的属性 -->  <!-- 配置数据库连接的基本信息: -->  <**property** name="hibernate.connection.driver\_class">  com.mysql.jdbc.Driver  </**property**>  <**property** name="hibernate.connection.url">  jdbc:mysql:///ssh  </**property**>  <**property** name="hibernate.connection.username">root</**property**>  <**property** name="hibernate.connection.password">0807</**property**>  <!-- Hibernate的方言 -->  <!-- 生成底层SQL不同的 -->  <**property** name="hibernate.dialect">  org.hibernate.dialect.MySQLDialect  </**property**>   <!-- 可选的属性 -->  <!-- 显示SQL -->  <**property** name="hibernate.show\_sql">true</**property**>  <!-- 格式化SQL -->  <**property** name="hibernate.format\_sql">true</**property**>   <**property** name="hibernate.connection.autocommit">false</**property**>  <!-- hbm:映射 to DDL: create drop alter -->  <**property** name="hibernate.hbm2ddl.auto">update</**property**>   <!-- C3P0连接池设定-->  <!-- 使用c3po连接池 配置连接池提供的供应商-->  <**property** name="connection.provider\_class">  org.hibernate.connection.C3P0ConnectionProvider  </**property**>   <!--在连接池中可用的数据库连接的最少数目 -->  <**property** name="c3p0.min\_size">5</**property**>  <!--在连接池中所有数据库连接的最大数目 -->  <**property** name="c3p0.max\_size">20</**property**>  <!--设定数据库连接的过期时间,以秒为单位,  如果连接池中的某个数据库连接处于空闲状态的时间超过了timeout时间,就会从连接池中清除 -->  <**property** name="c3p0.timeout">120</**property**>  <!--每3000秒检查所有连接池中的空闲连接 以秒为单位-->  <**property** name="c3p0.idle\_test\_period">3000</**property**>   <!-- 通知Hibernate加载那些映射文件 -->  <**mapping** resource="mapping的位置 用/进入下一级目录" />   </**session-factory**> </**hibernate-configuration**>  这里面的写法都是固定的，只有前面根据不同数据库不同，会配置不同。  c3p0连接池等都是固定的。 |

2.1.3 spring的application.xml

|  |
| --- |
| *<?*xml version="1.0" encoding="UTF-8"*?>* <**beans** xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:tx="http://www.springframework.org/schema/tx"  xmlns:context="http://www.springframework.org/schema/context" xsi:schemaLocation="http://www.springframework.org/schema/beans   http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/tx   http://www.springframework.org/schema/tx/spring-tx.xsd   http://www.springframework.org/schema/context   http://www.springframework.org/schema/context/spring-context.xsd">    </**beans**> |

2.2 代码分层

在src下建立四个文件夹：



一个模块写一个action，例如人员管理模块，编写一个UserAction、

UserService、UserDao 以及一个User javabean。

第三步：正式开发：

3.1 编写action

|  |
| --- |
| **package zk.action**;  **import com.opensymphony.xwork2.**ModelDriven; **import org.springframework.beans.factory.annotation.Autowired**; **import org.springframework.beans.factory.annotation.**Qualifier; **import zk.models.User**; **import zk.service.UserService**;   /\*\*  \* Created by horse on 2017/7/7.  \*/ **public class UserAction implements** ModelDriven<**User**> {  **private User** user=**new** User();  @Override  **public User** getModel() {  **return** user;  }  @Autowired  @Qualifier("userService")  **private UserService** userService;   **public String** login(){  **System**.out.println("web层的代码");  userService.save(user);  **return** "success";  } } |

问题1：struts模型驱动？

answer：实现 ModelDriven<T> 接口。

模型要new出来 private User user=new User();

不一定要用Struts2的标签。

问题2：这里怎么使用Spring呢？

answer：我们采用的是，把action也交给spring管理。同时整合spring和web项目

|  |
| --- |
| 整合spring和web：  <!-- 配置Spring的监听器 --> <**listener**>  <!-- 监听器默认加载的是WEB-INF/applicationContext.xml -->  <**listener-class**>org.springframework.web.context.ContextLoaderListener</**listener-class**> </**listener**>  <!-- 指定Spring框架的配置文件所在的位置 --> <**context-param**>  <**param-name**>contextConfigLocation</**param-name**>  <**param-value**>classpath:applicationContext.xml</**param-value**> </**context-param**>  把action交给spring管理：  即真正的action类放到spring的配置中去，而struts.xml是放一个伪类  首先在applicationContext.xml中配置真正action  <**bean** name="userAction" class="zk.action.UserAction" scope="prototype"/>  然后在struts中配置一个action，但class表示的是不是一个真正的类，而是applicationContext.xml配置的。  <**package** name="default" extends="struts-default" namespace="/" >  <**action** name="user\_\*" class="userAction" method="login">  <**result** name="success">/index.jsp</**result**>  </**action**> </**package**> |

问题3：这里通过注解注入UserService,应该怎么配置。spring如果要通过注解注入？

Answer：

需要在spring的配置文件中加入这个开启注解：

<**context:component-scan** base-package="zk"/>

3.2 编写service 以及dao

|  |
| --- |
| service  @Transactional **public class UserService** {   @Autowired  @Qualifier("userDao")  **private UserDao** userDao;   **public void** save(**User** user){  **System**.out.println("service 层代码");  userDao.save(user);  }  }  Dao  **public class UserDao extends** HibernateDaoSupport{   **public void** save(**User** user){  **System**.out.println("dao层代码");  **this**.getHibernateTemplate().save(user);  } } |

问题1：Spring要如何配置然后在service中开启事务？

Answer：

第一步：首先，事务的开启要和数据库关联，所以，我们先得把hibernate和spring整合起来。

|  |
| --- |
| 整合方法：在spring中通过LocalSessionFactoryBean配置数据源，然后引入hibernate的配置。  <!-- 零障碍整合 在spring配置文件中引入hibernate的配置文件 --> <**bean** id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">  <**property** name="configLocation" value="classpath:hibernate.cfg.xml"/> </**bean**>  然后在dao中引入 数据源，这样就不必每次都在dao层加载hibernate.cfg.xml。  <**bean** name="userDao" class="zk.dao.UserDao">  <**property** name="sessionFactory" ref="sessionFactory"/> </**bean**>  我们再看dao层继承了HibernateDaoSupport类，进入它的源码：  **private** HibernateTemplate hibernateTemplate; **public final void** setSessionFactory(SessionFactory sessionFactory) {  **if**(**this**.hibernateTemplate == **null** || sessionFactory != **this**.hibernateTemplate.getSessionFactory()) {  **this**.hibernateTemplate = **this**.createHibernateTemplate(sessionFactory);  } }  可以看到源码中有一个setSessionFactory，那么只要在配置中配置  <**bean** name="userDao" class="zk.dao.UserDao">  <**property** name="sessionFactory" ref="sessionFactory"/> </**bean**>  这样，只要继承了hibernateDaoSuppoert那么就可以通过setSeesionFactory来把sessionFactory注入到相应的dao中。而且  **this**.hibernateTemplate = **this**.createHibernateTemplate(sessionFactory);  可以看出来通过hibernateTemplate得到的就是相应的和sessionFactory有关的hibernateTemplate。  所以，只要两步：  第一：在相应的dao中通过xml配置<property name=”sessionFactory” ref=”sessionFactory”> 名字一定要是sessionFactory。  第二步：继承HibernateDaoSuppoert 。 |

整合好了之后，要在service中使用事务，那么在xml中

|  |
| --- |
| <!-- 管理事务 --> <**bean** id="**transactionManager**" class="org.springframework.orm.hibernate3.HibernateTransactionManager">  <**property** name="sessionFactory" ref="sessionFactory"/> </**bean**>  这里是配置 注解开启事务的方法 <**tx:annotation-driven** transaction-manager="**transactionManager**"/> |

这样整个配置就完成了。

3.3 OpenSessionInView一个小问题

当使用hibernate的延迟加载的时候，得到的对象会返回到web层，但是，一般事务都是在service中开启，同样事务也在service层结束了（session也结束了）。这样返回数据到web层的时候，session已经不存在了，延迟加载就没用了。所以要使用OpenSessionInView，

3.4 使用junit4

三大框架整合后，使用Junit4的时候，不知道为什么会出现sessionFactory注入的问题。

1.如果在继承了HibernateDaoSupport的dao类中使用Junit4，：

@RunWith(**SpringJUnit4ClassRunner**.**class**)  
@ContextConfiguration(locations = "classpath:applicationContext.xml")  
**public class ProductService** {  
 @Resource(name="productDao")  
 **private ProductDao** productDao;  
 @Test  
 **public void** test(){  
 **System**.out.println(productDao.getTotalCountByCid(1));  
 }

在applicationContext中也配置注入了sessionFactory：

<**bean** name="productDao" class="shop.product.ProductDao">  
 <**property** name="sessionFactory" ref="sessionFactory"/>  
</**bean**>

依旧会出现异常：未注入sessionFactory的错误。

此时只要加一个这个代码：

@Resource(name = "sessionFactory")  
**public void** setSessionFactoryOverride(SessionFactory sessionFactory)  
{  
 **super**.setSessionFactory(sessionFactory);  
}

就可以了。

2.更好的方法是在service层使用junit，调用dao层

@RunWith(**SpringJUnit4ClassRunner**.**class**)  
@ContextConfiguration(locations = "classpath:applicationContext.xml")  
**public class ProductService** {  
 @Resource(name="productDao")  
 **private ProductDao** productDao;  
 @Test  
 **public void** test(){  
 **System**.out.println(productDao.getTotalCountByCid(1));  
 }

这样就不会出先需要再次注入的问题了。