# 数据采集系统

## 项目涉及

ssh: struts2 spring hibernate

Struts:开发web程序的框架，是更加整洁的mvc结构

分离关注：拦截器

Action : 原型的 线程安全性

耦合度低:和原生的servlet api

Hibernate: 持久化技术，替代了jdbc，封装了数据的访问细节，体现了oop的思想

Spring : 业务层框架，管理bean

Ioc：inverse of control 反转控制

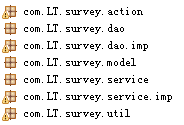
Aop: aspect oriented program ,不改变源代码，还给类增加新的功能，对oop进行增强。 学过的(前置、后置、环绕(事务管理)、异常、引入)

代理.

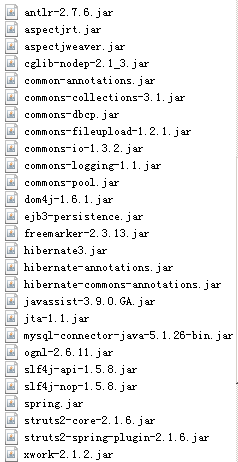
## 搭建项目:

### 1.创建web项目

### 2.创建各种包.



### 3.引入类库

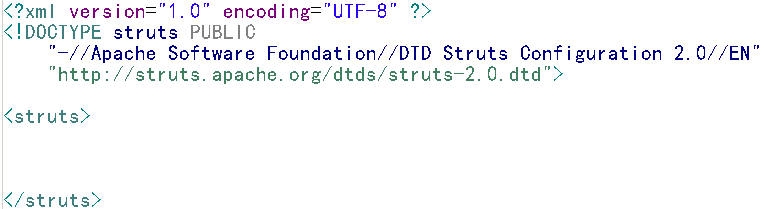


## 4.配置项目

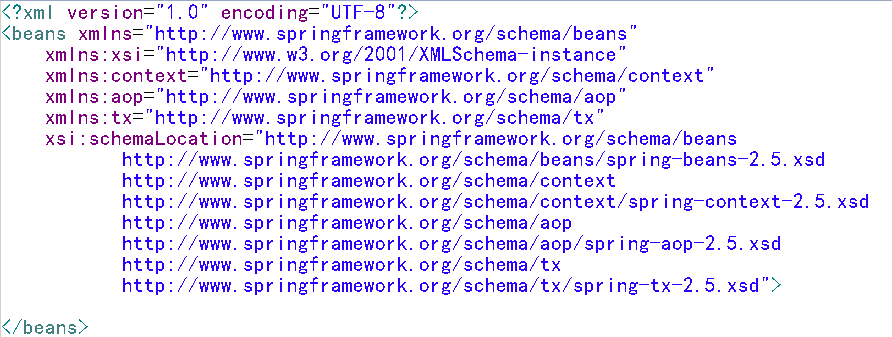
[struts2 + web]



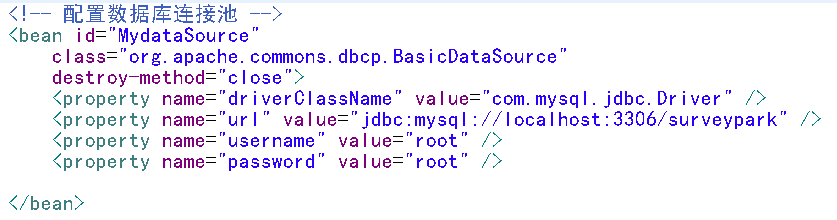
[confi/struts.xml]



[spring--config/beans.xml]



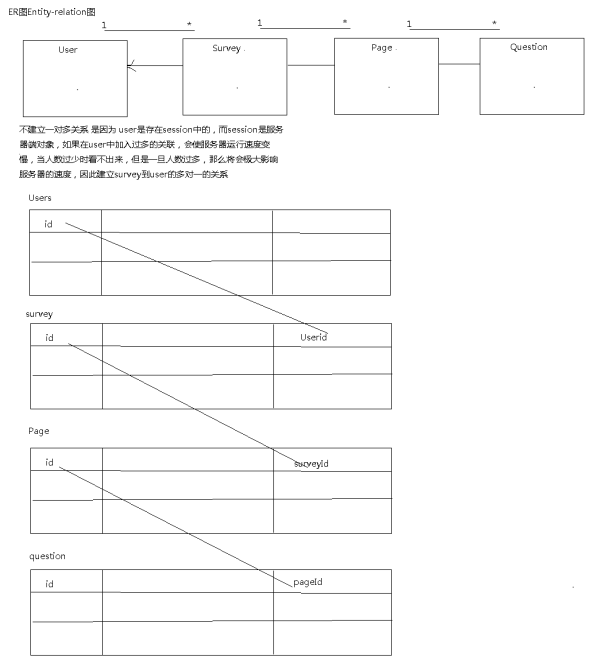
Spring中配置数据源DataSource



测试数据源

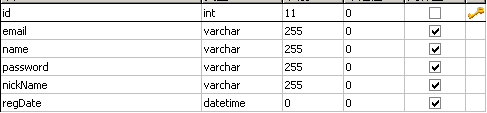
## 5.实体关系分析

类结构

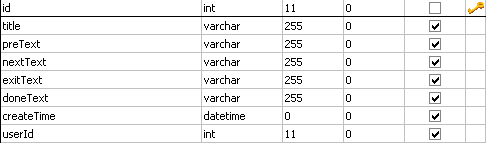


表结构

User



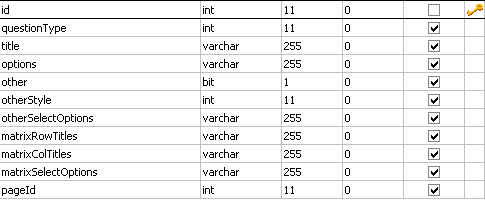
Survey



Page

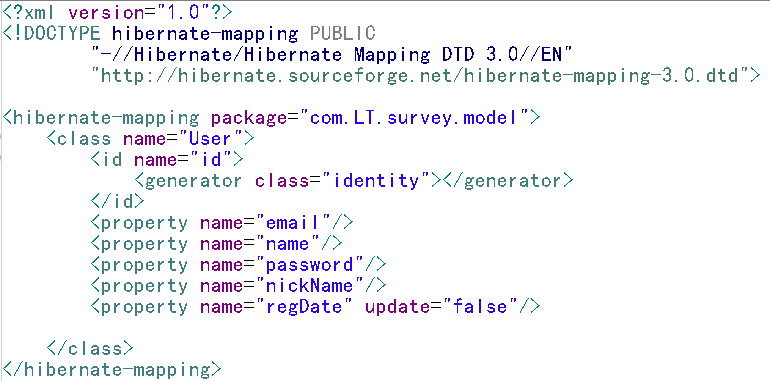


Question



映射文件

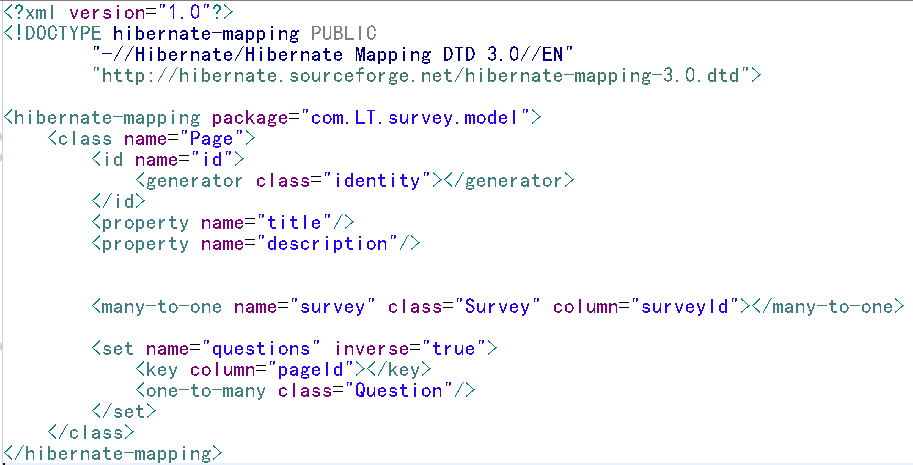
User.hbm.xml



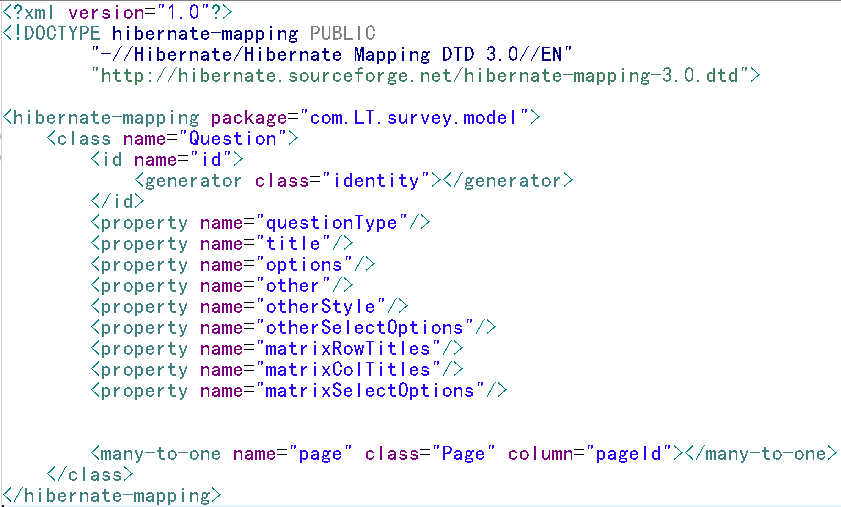
Survey.hbm.xml



Page.hbm.xml



Question.hbm.xml

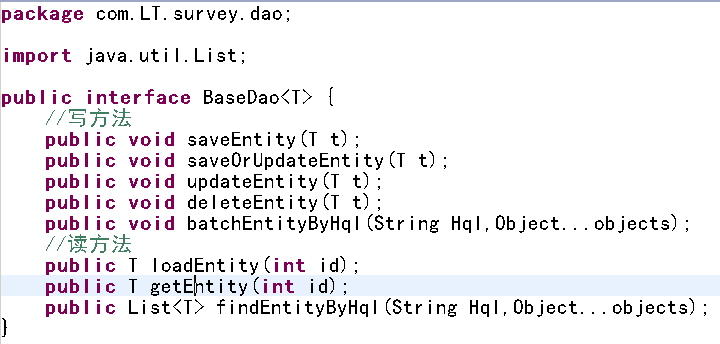


## 6.dao和实现类

dao

通过对系统整体的判断，发现所有对于实体类的操作都基本上是相同的，分为写操作和读操作，这样可以只向上抽取一个最基本的接口，定义所需要的方法，让实现类去实现这个接口即可，由于存在不同的实体类，所以可以用到java中的泛型

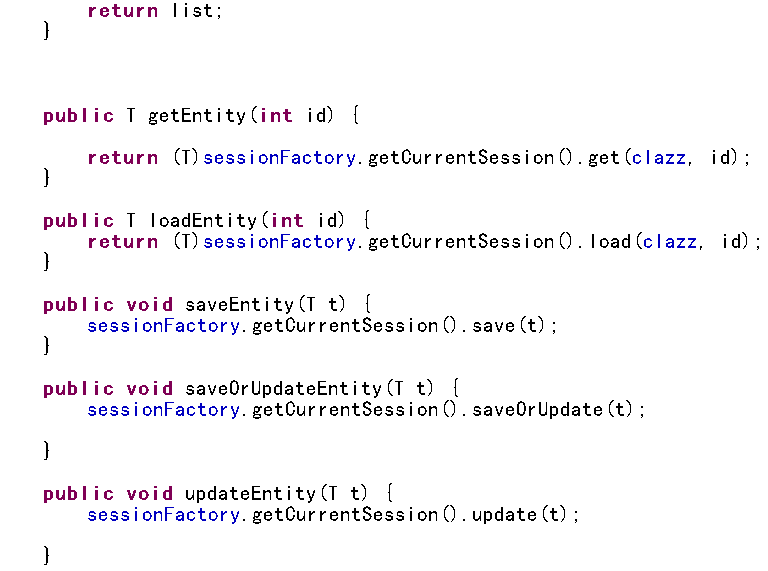
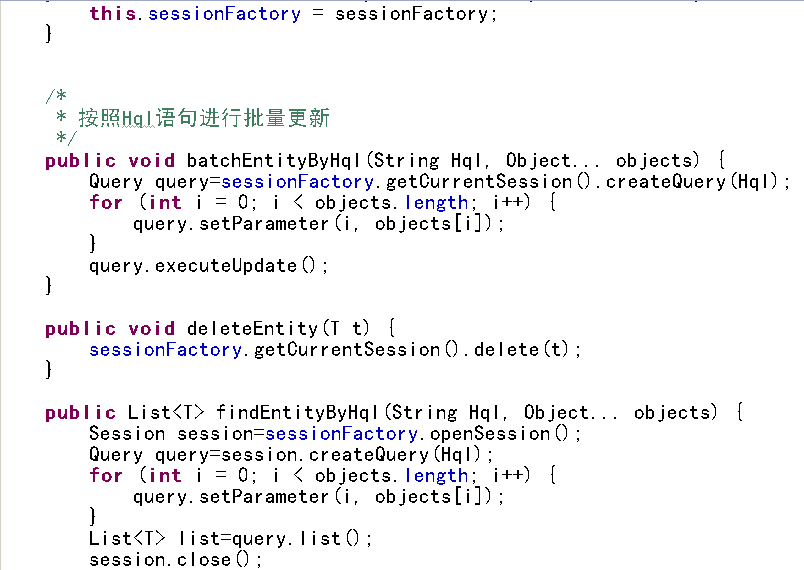
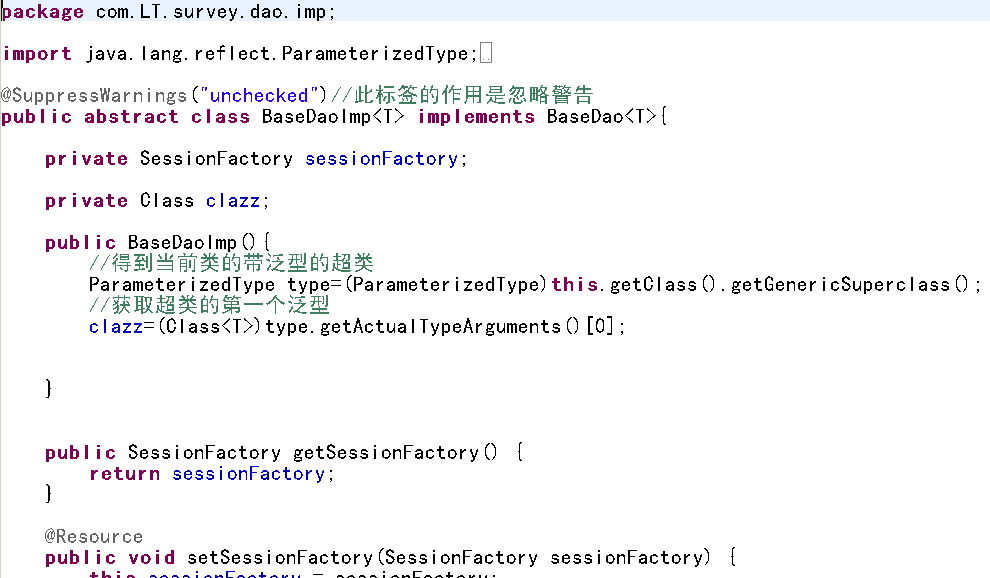
BaseDao<T>



这就是最顶层的接口，所有实现类都应该满足这个接口定义的规则

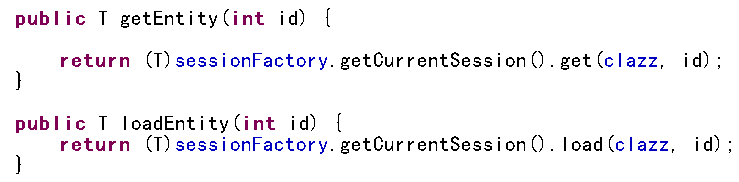
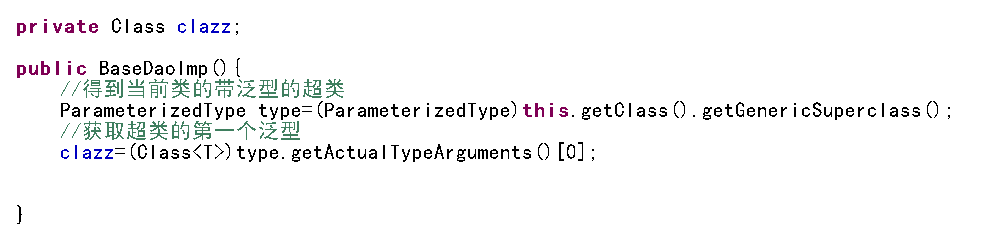
Dao的实现层，由于对各个实体类的基本操作都相同，因此可以只定义一个最基本的实现类BaseDaoImp<T>，然后让各个类的dao的实现类去继承这个基本类即可，这样当其中的实体类操作不一样时，可以在单独添加操作

BaseDaoImp<T>

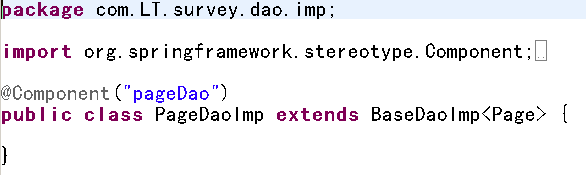


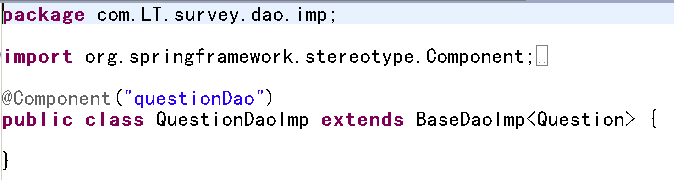
这里需要注意的是由于其中的某些方法实现时必须要得到确定的类，而不是用泛型T代替

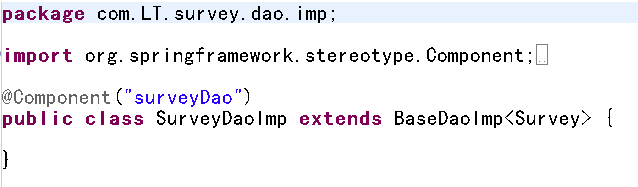
这里可以用java中的反射机制得到具体的类

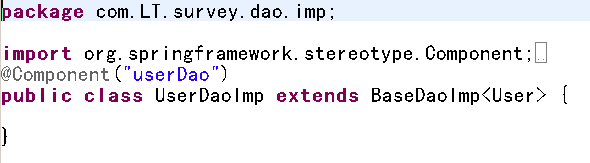


接下来就是各个实体类的具体的实现类，只需继承那个最基本的实现类即可





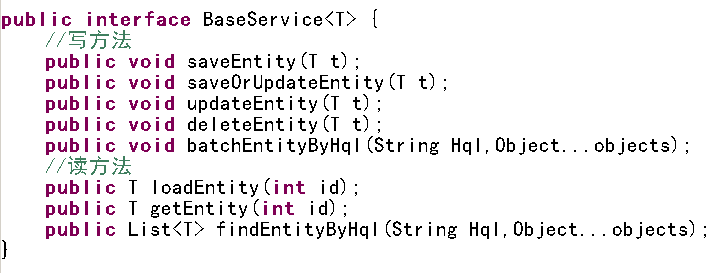




Service

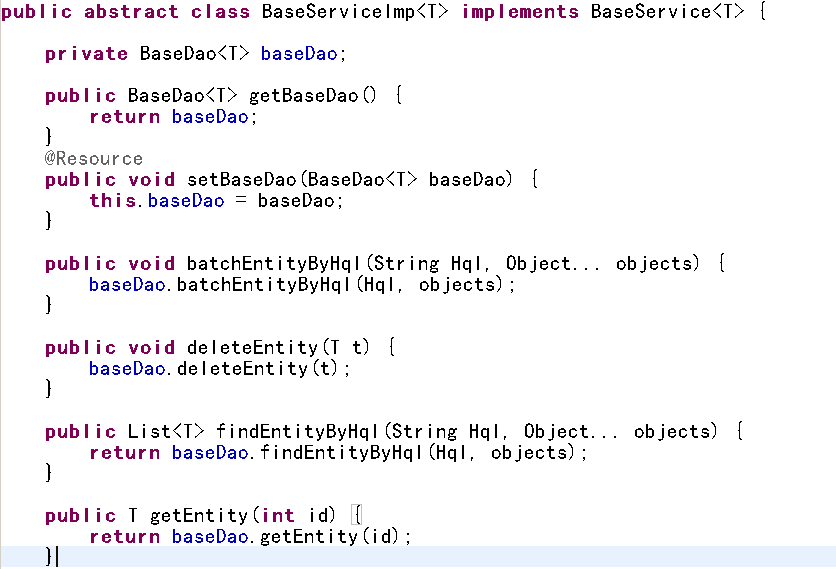
可以先抽取一个顶层的BaseService接口，里面是BaseDao的所有方法(基本上很多人默认的一个不成文的不是规定的约束->即dao层的方法，service层都有)

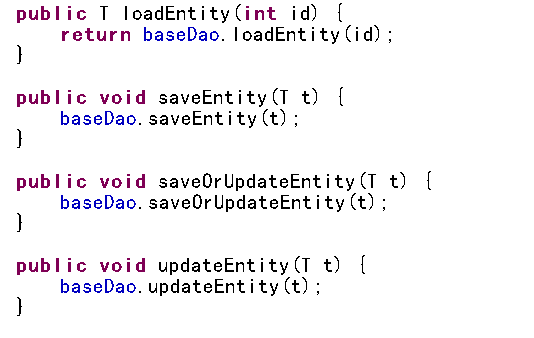
同样使用泛型



再写一个基本的Service实现类BaseServiceImp，实现BaseService接口

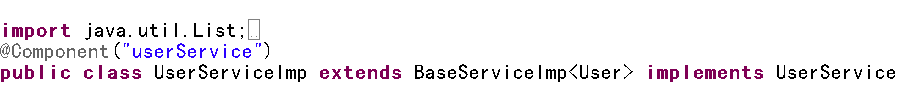
BaseServiceImp<T>



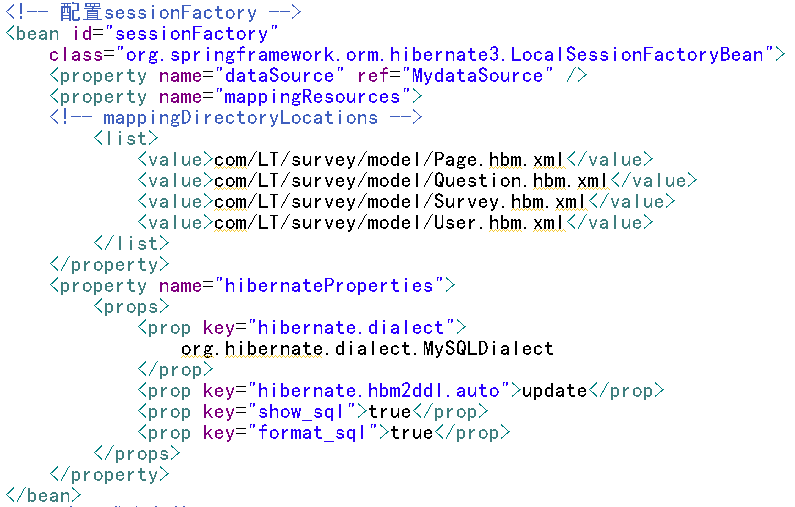


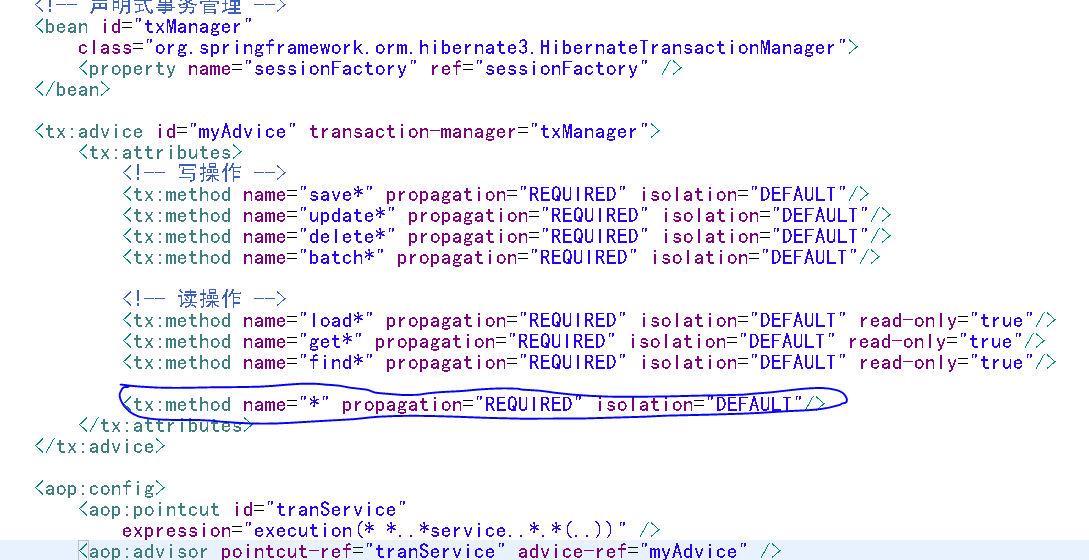
开发到这里，出现一个分歧，就是每个实体类的service实现类是不是都应该写，通过对数据采集系统整体需求的分析以及对实体类关系的映射可知，如果每个实体类都写service实现类，那将会很麻烦，因此只需要定义一个userServiceImp实现类和一个SurveyServiceImp实现类

对于user的service，首先定义一个userService接口，继承BaseService接口,然后编写userServiceImp实现类，让其继承BaseServiceImp<User>,并实现userService接口，这样当以后有多余的功能时可以在userService中添加



## 7.pring、hibernate、spring整合

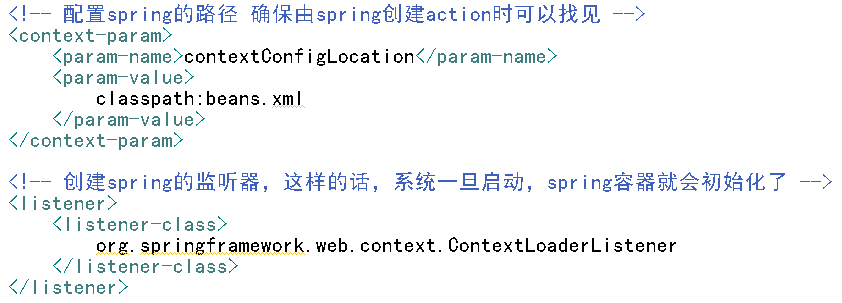


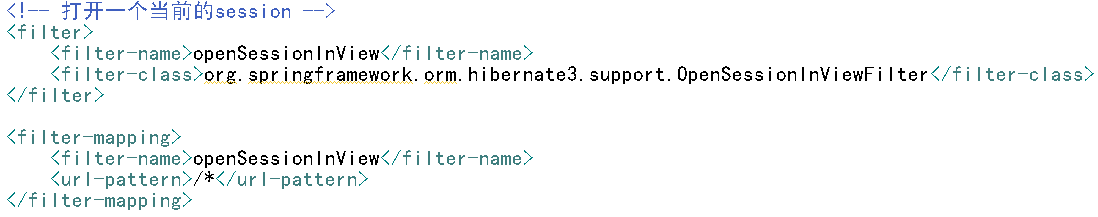


注意:在加入声明式事务管理时，一定要注意所有的service方法上都加入，加全，即最终完善的就是圈住的那句话，如果不加全，会出现一个特别恶心的错

No session bound to thread。。。。。。

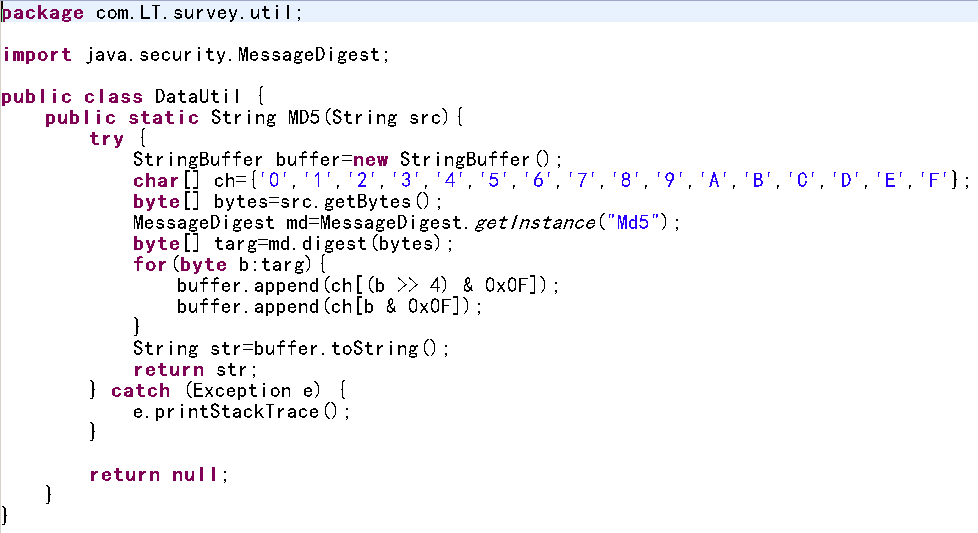
完成spring和struts2的整合，配置web.xml





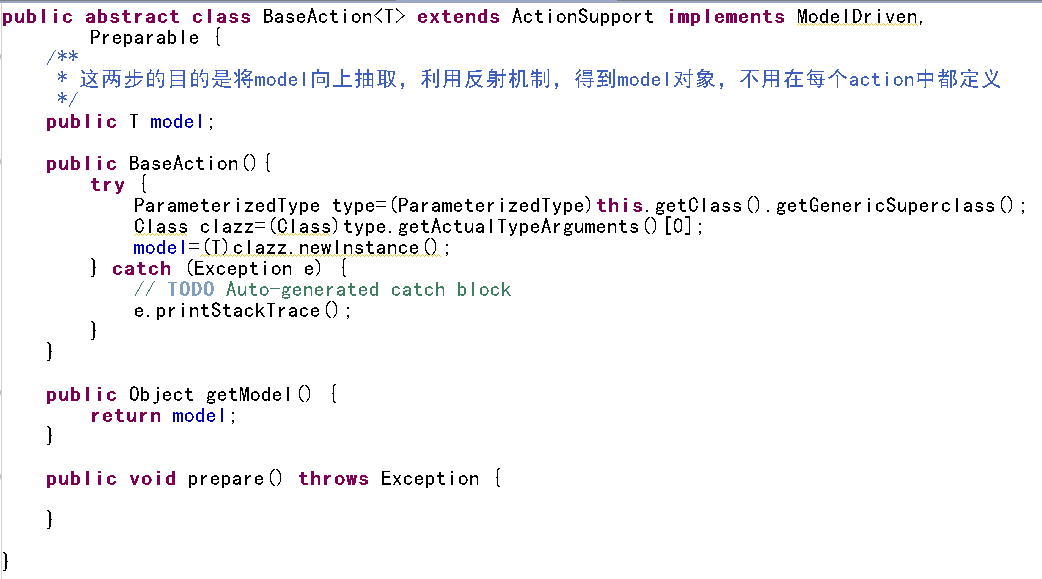
## 8.Md5加密

由于password是使用varchar的方法明文存入的，很不安全，所以可以人为编写一个md5算法，对密码进行md5加密



## 9.action的开发

向上抽取，得到一个BaseAction<T>



## 10.功能实现

10.1 用户注册和登录

注册

(1)到达注册页面

1.导航栏-->用户注册链接

2.进入RegAction.toRegPage()方法

public String toRegPage(){

return "regPage" ;

}

3.struts.xml

<action name="RegAction\_\*" class="regAction" method="{1}">

<result name="regPage">/reg.jsp</result>

</action>

4.跳转到/reg.jsp

<s:form>

<s:textfield name="email" />

...

</s:form>

以上的步骤不是多此一举，是为了之后的权限管理更方便才如此做

(2)进行用户注册

1.注册页面--->提交

2.RegAction.doReg()方法

public String doReg(){

1.接受用户信息--model

2.userService.saveEntity(model);

3.return "success" ;

}

public void validate(){

1.非空

2.密码一致性

3.email是否占用

}

3.struts.xml

<action ...>

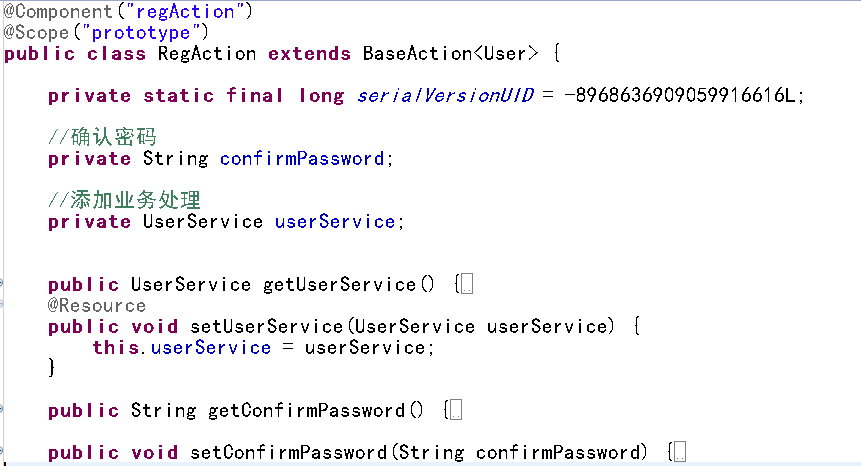
<result name="success">/index.jsp</result>

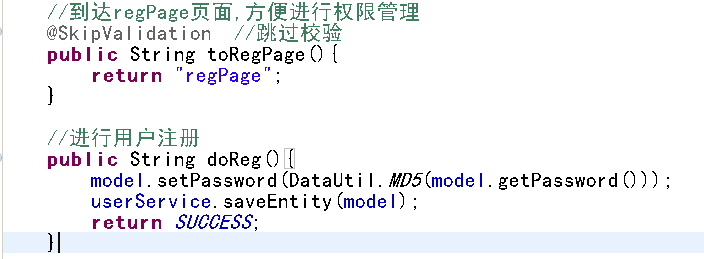
<result name="input">/reg.jsp</result>

</action>

4.跳转到/reg.jsp|/index.jsp

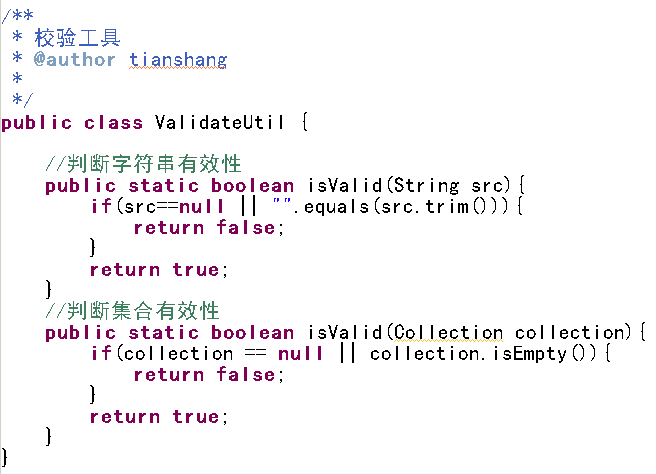
RegAction





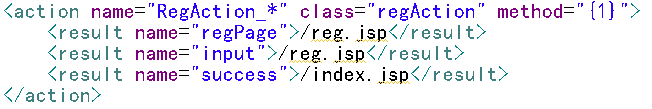


为了方便编写一个工具类--------->ValidateUtil.class



在上面的RegAction中有一个Validate方法，这是一个固定的方法，作用就是校验，它会在进入RegAction后先执行，如果不想让运行某个方法时先校验，可以用注解@SkipValidation,跳过校验

Struts.xml的相应配置



(b)登录

(1)进入登陆

1.导航栏-->首页

2.LoginAction.toLoginPage()

public String toLoginPage()

{

1.return "loginPage" ;

}

3.struts.xml

<action name="LoginAction\_\*" class="" method="">

<result name="loginPage">/index.jsp</result>

<result name="input">/index.jsp</result>

</action>

4.跳转到/index.jsp

(2)进行登陆验证

1.登陆页面-->提交

2.LoginAction.doLogin()

public String doLogin(){

return "success";

}

public void validate(){

1.验证登陆信息

2.失败:错误信息,index.jsp

3.成功:user --> session

}

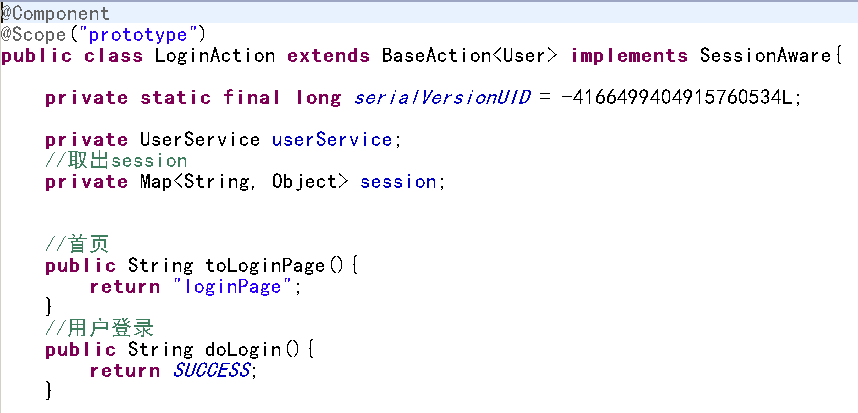
3.struts.xml

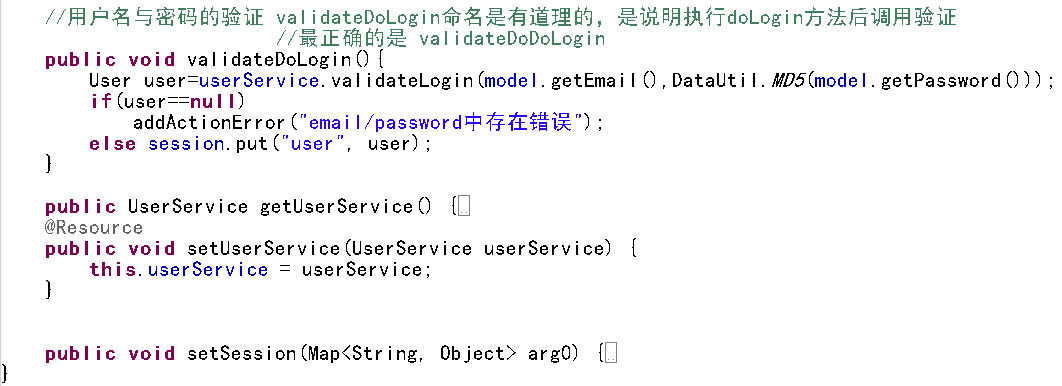
<action ..>

<result name="success">/index.jsp</result>

</action>

LoginAction

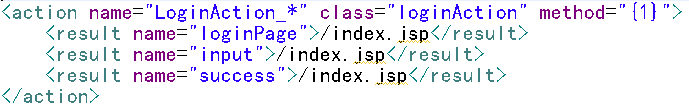




在LoginAction中要取出session，首先实现SessionAware接口，重写setSession方法，定义Map集合session即可

里面有一个validateDoLogin方法，这样是命名是有道理的，意思是在执行了doLogin方法后调用验证，最正确的写法是validateDoDoLogin

Struts.xml的相应配置



\*注意：在struts2中，所有的校验当检测到Filederror、Actionerror有值时，说明出现错误，系统会自动跳转到input视图，所以配置校验时，一定要在struts.xml中配置name为input的返回

10.2我的调查和新建调查

这是编写关于Survey的功能，先定义一个SurveyAction,继承BaseAction<Survey>,并且要加入一个SurveyService业务层(编写一个SurveyService接口，在定义一个SurveyService的实现类,按照下面的功能加方法)

我的调查

查看我的调查列表

1.导航栏-->我的调查

2.SurveyAction.mySurveys()方法

public String mySurveys(){

1.session --> user

2.this.mySurveys = surveyService.findMySurveys(user);

public List<Survey> findMySurveys(User user){

String hql = "from Survey s where s.user.id = ?" ;

return surveyDao.findEntityByHQL(hql,user.getId());

}

3.return "mySurveyListPage";

}

3.struts.xml

<.

4.跳转到/mySurveyList.jsp

<s:iterator value="mySurveys" var="s">

<s:property value="#s.id" />

<s:property value="#s.title" />

<s:date name="#s.createTime" format="yyyy-MM-dd hh:mm" />

</s:iterator>

新建调查

1.导航栏-->新建调查

2.SurveyAction.newSurvey()

public String newSurvey(){

1.session --> user

2.this.model = surveyService.newSurvey(user);

public Survey newSurvey(User user){

Survey s = new Survey();

Page p = new Page();

p.setSurvey(s);

s.getPages().add(p);

surveyDao.saveEntity(s);

pageDao.saveEntity(p);

return s ;

}

3.return "designSurveyPage" ;

}

3.struts.xml

<result name="designSurveyPage">/designSurvey.jsp</result>

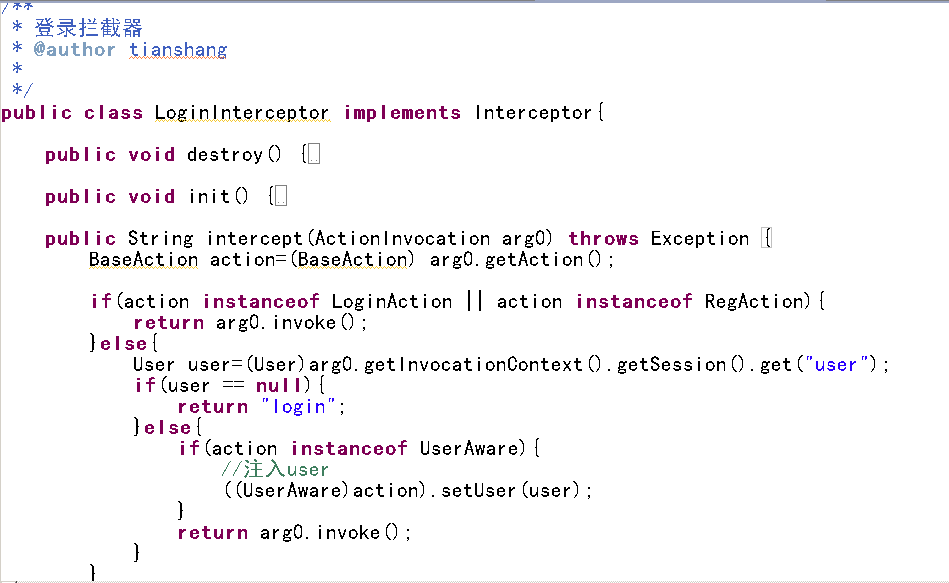
4.跳转到/designSurvey.jsp

[略]

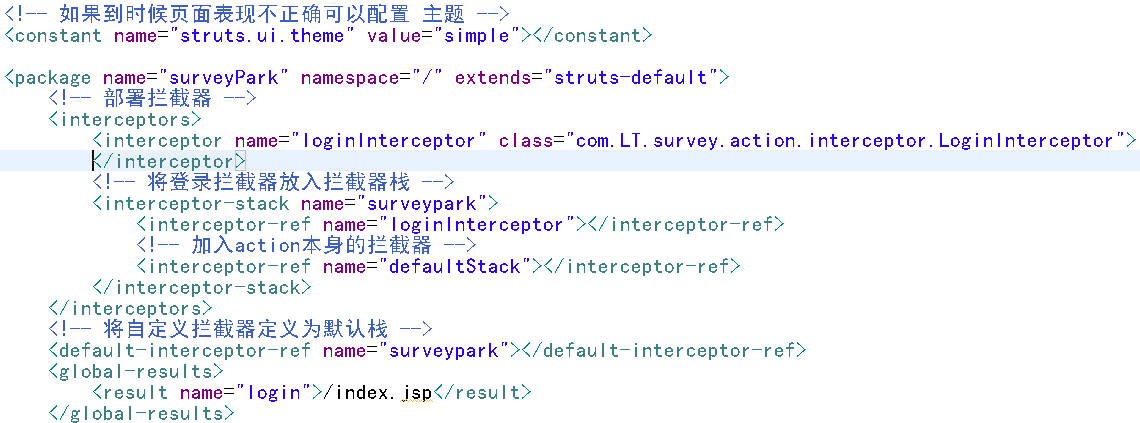
10.3登陆拦截器

有这么一种情况，当用户在首页不登录，却直接点击 我的调查 或者 新建调查，这时会出现错误，因此建立一个拦截器，作用就是，只要拦截器判断用户没有登录，那么用户的下一步动作不是注册或者登录，直接让其跳转到登录界面 同时在进行 我的调查 新建调查 时总是需要从session中取出user，为了更简便，我们可以在拦截器中直接帮我们注入user，原理是定义一个UserAware接口，只要是需要user的Action，那么就让其实现UserAware接口，然后在拦截器中判断这个action是否实现了UserAware(instanceof)只要实现就将User注入，这样简化了程序

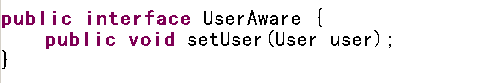
定义拦截器



注册拦截器



UserAware



10.4设计调查页面分析

<!-- 调查标题 -->

<s:property value="title" />

<!-- 输出页面集合 -->

<s:iterator value="pages" var="p">

<!-- 页面标题 -->

<s:property value="#p.title" />

<!-- 迭代问题集合 -->

<s:iterator value="#p.questions" var="q">

<!-- 问题标题 -->

<s:property value="#q.title" />

<!-- 处理选项输出 -->

<s:set var="qt" value="#q.questionType" />

<!-- 0,1,2,3 -->

<s:if test="#qt < 4">

<s:iterator value="#q.optionArr">

<input type="<s:property value="#qt < 2?'radio':'checkbox'" />"><s:property />

<!-- 是否携带br -->

<s:if test="#qt == 1 || #qt == 3"><br></s:if>

</s:iterator>

<!-- 是否带其他 -->

<s:if test="other">

<input type="<s:property value="#qt < 2?'radio':'checkbox'" />">其他

<!-- 其他项样式 -->

<s:if test="#q.otherStyle == 1">

<!-- 文本框 -->

<input type="text">

</s:if>

<s:elseif test="#q.otherStyle == 2">

<!-- 下拉列表 -->

<select>

<s:iterator value="#q.otherSelectOptions">

<option><s:property /></option>

</s:iterator>

</select>

</s:elseif>

</s:if>

</s:if>

<!-- 4,5 -->

<s:elseif test="#qt == 4 || #qt == 5">

<!-- 下拉列表 -->

<s:if test="#qt == 4">

<select>

<s:iterator value="#q.options">

<option><s:property /></option>

</s:iterator>

</select>

</s:if>

<s:elseif test="#qt == 5">

<input type="text">

</s:elseif>

</s:elseif>

<!-- 6,7,8:矩阵式问题 -->

<s:elseif test="#qt > 5">

<table>

<!-- 表头 -->

<tr>

<td></td>

<s:iterator value="#q.matrixColTitleArr">

<td><s:property /></td>

</s:iterator>

</tr>

<!-- 输出n多行 -->

<s:iterator value="#q.matrixRowTitleArr">

<tr>

<td><s:property /></td>

<s:iterator value="#q.matrixColTitleArr">

<td>

<s:if test="#qt == 6"><input type="radio"></s:if>

<s:if test="#qt == 7"><input type="checkbox"></s:if>

<s:if test="#qt == 8">

<select>

<s:iterator value="#q.matrixSelectOptionArr">

<option><s:property /></option>

</s:iterator>

</select>

</s:if>

</td>

</s:iterator>

</tr>

</s:iterator>

</table>

</s:elseif>

</s:iterator>

</s:iterator>

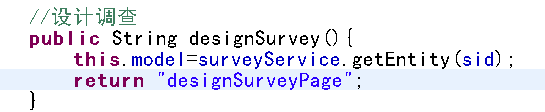
设计调查

1. 调查列表-》设计

通过在 我的调查 页面点击 设计 可以直接跳转到对应调查页面(通过传参数)

这样既可以，但是做完之后，我们会发现出现一个特别严重的问题 即:

当点击设计进入调查页面时，会突然发现，这并不是我们对应的调查页面，而是一个全新的未命名的调查。

这在struts中是一个比较经典的问题-------》Action中模型赋值的问题

##### Action中模型赋值问题

我们知道，在struts中有一系列拦截器，它们会在创建Action后先执行，在创建action时model是一个全新的对象，由于我们实现了ModelDriven接口，所以要在执行自己定义方法时首先按照拦截器顺序先执行对应的拦截器，这里就是执行ModelDriven拦截器中的getModel方法，将model直接压入栈中，之后即使执行对应的方法为model设置了新的值，但是栈中仍没有变化，故在设计页面得到的model始终是旧的model。

解决方法：

1.手动压栈.

1.耦合度高.

2.不推荐直接操作vs

3.栈中的模型对象过多

valueStack.push(newModel);

2.通过手动将新模型的属性全部赋值给旧模型.

性能比较差

oldModel.setXxxx(newModel.getXxxx());

...

3.使用paramsPrepareParamsStack + preparable拦截器配合使用

prepare拦截器先执行,先为model赋值,后调用modelDriven,在栈顶压入的model是新模型,

但是需要主要在prepare拦截器还要进行传递参数,defaultstack在prepare不能完成传参,

因此可以改换成paramsPrepareParamsStack来达到此目的.

此种方法有一个很大的缺点，就是对于所有进行传参数的方法都必须有一个prepare方法，这样会造成方法爆炸，很不便利，不推荐使用，如果没有对应的prepare方法，将无法传参

4.使用刷新机制,进行赋值,直接在action中为model赋值,使用刷新机制即可.(因为struts的拦截器是有两个过程预处理和后处理，只要在后处理之前刷新一下，就可以重新压栈)

<interceptors>

<!-- 注册登陆拦截器 -->

<interceptor name="loginInterceptor" class="com.atguigu.surveypark.struts2.interceptor.LoginInterceptor" />

<!-- 定义拦截器栈 -->s

<interceptor-stack name="surveyparkStack">

<interceptor-ref name="loginInterceptor" />

<interceptor-ref name="defaultStack">

<param name="modelDriven.refreshModelBeforeResult">true</param>

</interceptor-ref>

</interceptor-stack>

</interceptors>

##### Hibernate懒加载问题

由于在调用getSurvey时用的是hibernate的get方法，在没有对映射文件设置的情况，hibernate中fetch-Type默认是lazy，这样返回的是代理对象，session是在service层结束后就关闭，这样返回的仍然得不到

解决方法

1. 不用懒加载(不推荐使用，懒加载的设置就是为了提高性能，当访问很多时，不会变慢)

2.一劳永逸解决懒加载问题.使用spring的openSessionInViewFilter.

<!-- openSessionInViewFilter,改过滤器在view渲染时始终开启session,一劳永逸解决hibernate的懒加载问题,

该过滤器必须配置在struts2过滤器之前,不推荐使用(性能问题)

<filter>

<filter-name>openSessionInViewFilter</filter-name>

<filter-class>org.springframework.orm.hibernate3.support.OpenSessionInViewFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>openSessionInViewFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

-->

此种方法尽管可以直接解决懒加载问题，但是对性能影响极大，尤其是当访问 特别多时，故不推荐使用

3.强行在service层面是初始化代理对象.

public Survey getSurveyWithChildren(Integer sid){

//Survey s = surveyDao.getEntity(sid);

//降低耦合度

Survey s = this.getSurvey(sid);

//强行初始化pages和questions集合

for(Page p : s.getPages()){

p.getQuestions().size();

}

return s;

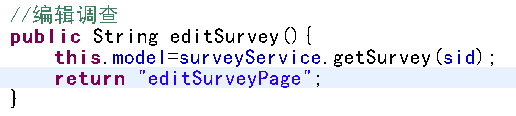
}

这种方法符合我们写代码的要求，高内聚低耦合(组件之间耦合很低，但组件内部高度聚合)

### 编辑调查

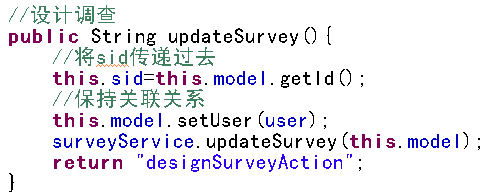
##### 编辑调查页面

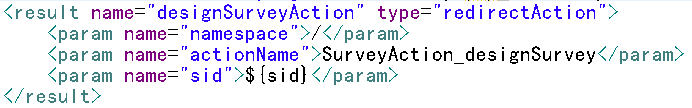
在设计调查页面点击编辑调查，进入编辑调查页面(Survey\_editSurvey)



##### 调查进行编辑

在编辑调查页面，对调查进行编辑，然后确定，便**重定向**回设计调查页面(注意要将参数绑定上,因为在action中是采用原型法，每次打开都是一个新的action，里面的参数都会重置)





**注意：**

对数据进行写操作时，一般都是重定向，防止用户重复提交

Jsp页面中操作ognl表达式用%{}

配置文件中操作ognl表达式用${}

Struts标签中 s:textfield默认找name想等的值，从action中

### 添加页面

这是关于Page的功能，则首先定义一个PageAction类用于对Page功能的管理，继承BaseAction，基本属性有sid(调查的编号) pid(所属页面号) page的model 以及SurveyService业务实现

1.设计调查-->添加页面

2.PageAction.toAddPage()

public String toAddPage(){

sid

return "addPagePage" ;

}

3.struts.xml

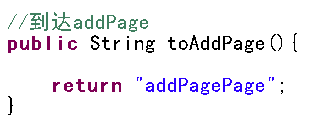
<action name="PageAction\_\*" class="pageAction" method="{1}">

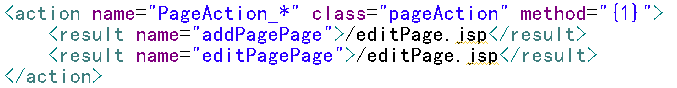
<result name="addPagePage">/editPage.jsp</result>

</action>

4.跳转到/editPage.jsp

[略]





### 保存/更新页面

------------------

1.编辑page-->提交

2.PageAction.saveOrUpdatePage()

public String saveOrUpdatePage(){

model + sid

//维护关联关系

model.setSurvey(new Survey(sid));

surveyService.saveOrUpdatePage(model);

public void saveOrUpdaetPage(Page p){

pageDao.saveOrUpdateEntity(p);

}

return "designSurveyAction" ;

}

3.struts.xml

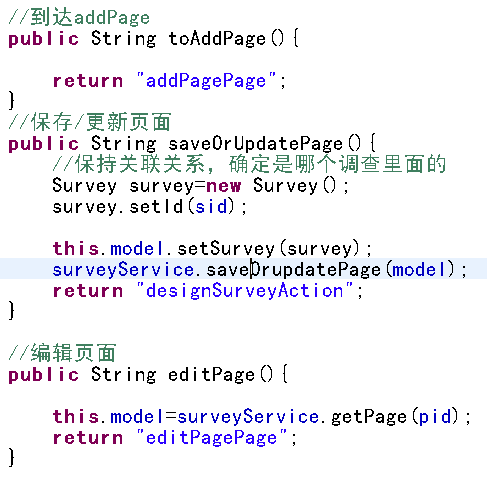
改造designSurveyAction成全局结果.

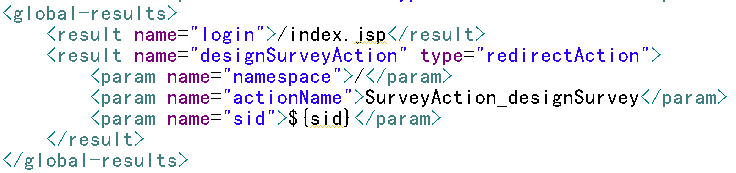
4.略

5.设置page集合按照id排序

[Survey.hbm.xml]

<set name="pages" inverse="true" order-by="id">







### 编辑页面

------------------------

1.设计调查 --->编辑页面

2.PageAction.editPage()

public String editPage(){

sid + pid

this.model = surveyService.getPage(pid);

public Page getPage(Integer pid){

return pageDao.getEntity(pid);

}

return "editPagePage" ;

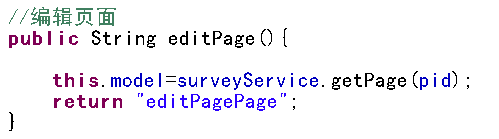
}

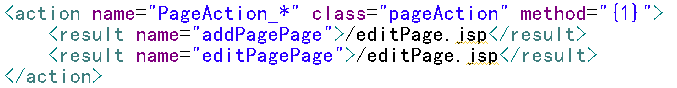
3.struts.xml

<result name>

4.跳转到/editPage.jsp

[略]





### 增加问题

这是关于问题的功能，则首先定义一个QuestionAction类用于对Question功能的管理，继承BaseAction，基本属性有sid(调查的编号) pid(所属页面号) page的model 以及SurveyService业务实现

1.设计调查---------->增加问题

2.QuestionAction\_toSelectQuestionType()方法

Public String toSelectQuestionType(){

//传参数 sid pid

return “selectQuestionTypePage”;//到达问题类型页面

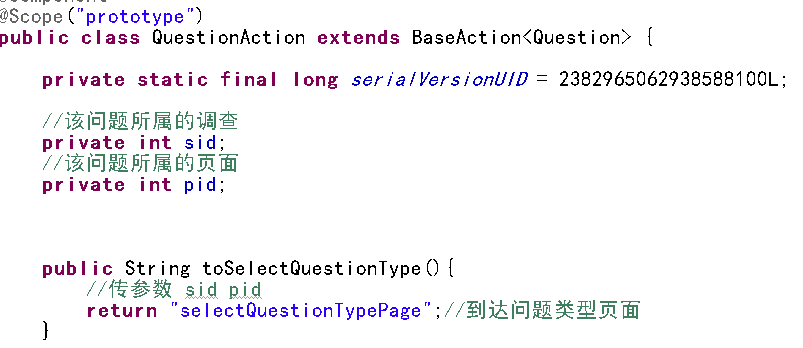
}

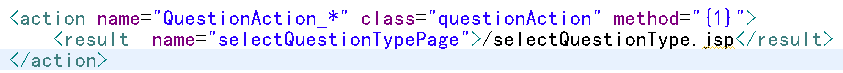
1. Struts.xml
   1. <action name=”QuestionAction\_\*” class=”questionAction” method=”{1}”>

<result name=”selectQuestionTypePage”>/selectQuestionType.jsp</result>

* 1. </action>

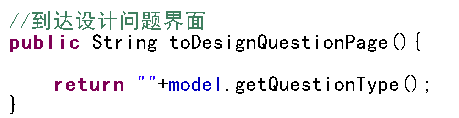
1. 跳转到selectQuestionType.jsp页面

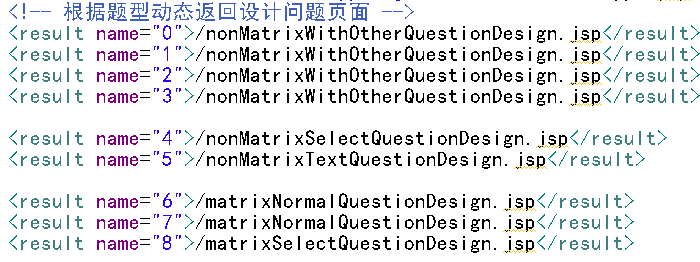




### 到达设计问题页面

1. 选择题型页面---------》提交
2. QuestionAction.toDesignQuestionPage()
3. 由于到达的页面按照分类可以分为四类，不能只用一个返回值故可以将返回值设成字符串的问题类型，在struts.xml配置即可

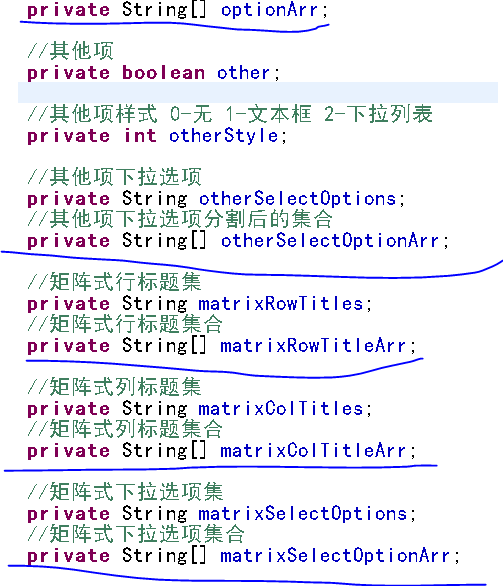




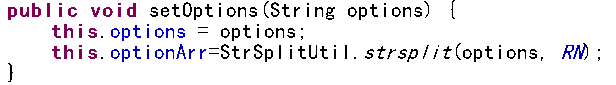
### 保存/更新问题

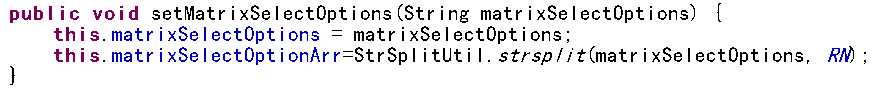
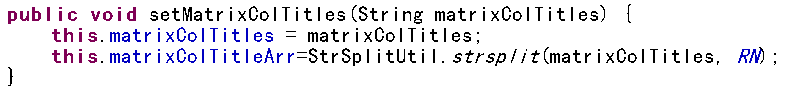
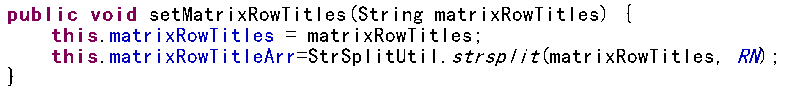
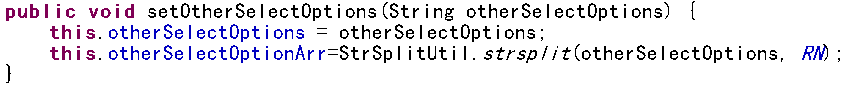
选择问题界面-------------》编辑问题

1. QuestionAction.saveOrUpdateQuestion()方法
   1. public String saveOrUpdateQuestion(){
   2. sid + pid + model
   3. //维护关联关系
   4. model.setPage(new Page(pid));
   5. surveyService.saveOrUpdateQuestion(model);
   6. public void saveOrUpdateQuestion(Question q){
   7. question.saveOrUpdateEntity(q);
   8. }
   9. return "designSurveyAction" ;
   10. }
2. 当编写完成后，测试时发现，问题可以显现出来，但是选项及其那些编写的下拉列表显现不出来，经过查看jsp界面可知，选项及其下拉列表都是有几个Arr数组中取出的，但是在Question类中并没有这些数组，故先在Question类中定义几个数组



这些数组的值都是对应的属性字符串从数据库得到时，通过字符串分割得到的，故同样要在对应的属性的set方法中加入分割并赋值





由于有多个地方进行字符串分割，故将分割字符串抽取出来成为一个单独的工具方法

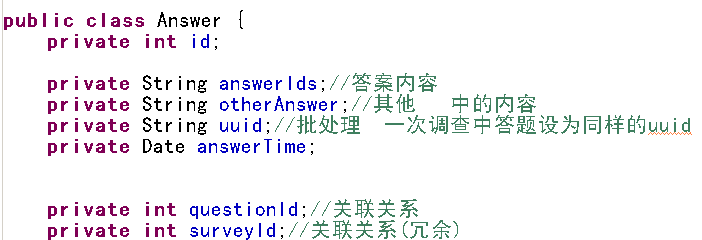
经过上述操作，再次测试后，发现可以显示

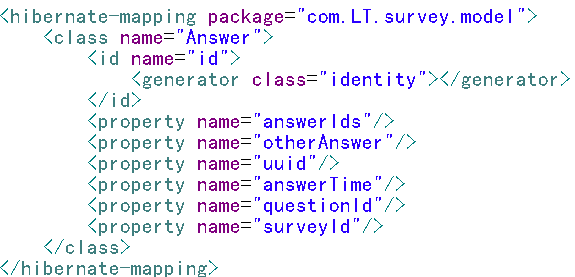
### 增加Answer实体

当调查页面设计基本完成后，现在我们的实体类有User、Survey、Page、Question四个，但是由于我们是数据采集系统，故至少应该有答案，所以我们应该在增加一个答案实体Answer，即每个问题都应该有答案，现在应该考虑问题实体与答案实体的关联，本来，一个问题可以有很多答案，一个答案只可以属于一个问题，故问题与答案的关联应该是一对多的关系，但是同样的，由于我们是数据采集，单个数据取出来完全没有用，只有大量的数据通过百分比才能得出结论，故这里我们的问题实体和答案实体不设计关联关系，但是为了保证关联，可以在Answer中设置一个qid，代表所属的问题。

再假如，如果我们将一个调查比成一棵树时，会发现从树根到叶子依次是Survey----》Page----》Question----》Answer，无论我们要删除什么，我们都应该从叶子开始删，因此为了保证关联关系，可以在Answer中在设计一个sid，这虽然是冗余字段，但是可以使得在删除Survey时，先删除叶子时，可以不用跨很多层。

故Answer实体最终为





### 删除

#### 删除问题

1.设计调查页面-----------》删除问题

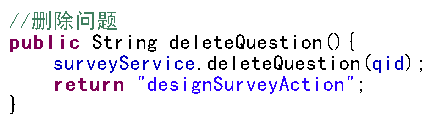
当删除问题时，过程应该是 先删除答案，再删除问题

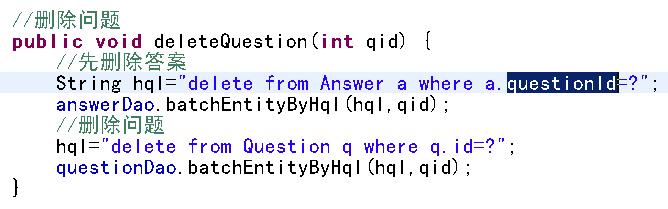
1. public String deleteQuestion(){

surveyService.deleteQuestion(qid);

Return ”designSurveyAction”;

}





#### 删除页面

1.设计调查页面-----------》删除页面

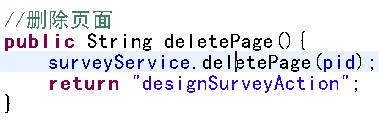
当删除页面时，过程应该是 先删除答案，再删除问题，再删除页面

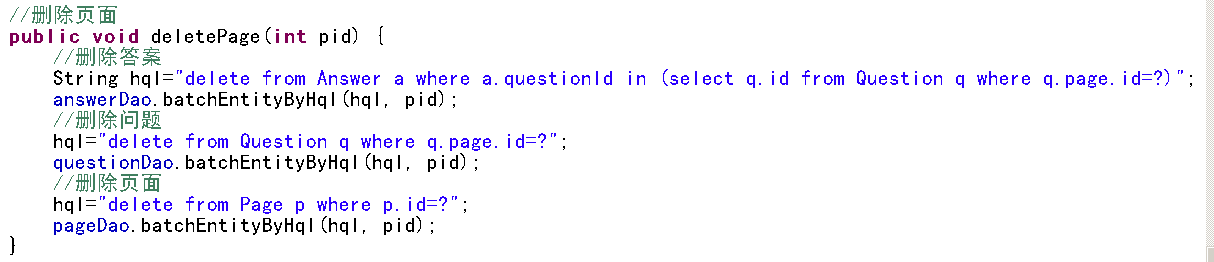
1. public String deletePage(){

surveyService.deletePage(pid);

Return ”designSurveyAction”;

}





这里需要注意的是在hibernate中查询可以多级查询，但是执行写操作时，最多不超过2级，一旦超过2级就会报错

#### 删除调查

1.我的调查页面-----------》删除调查

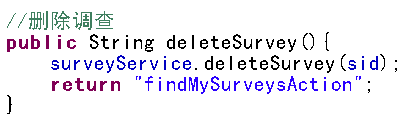
当删除调查时，过程应该是 先删除答案，再删除问题，再删除调查

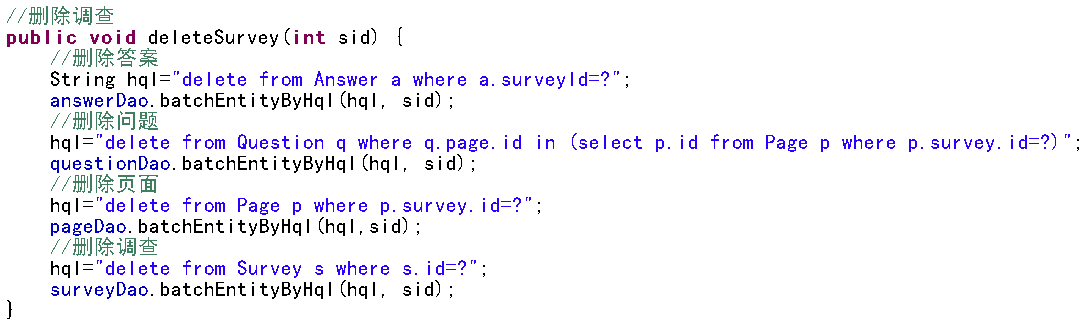
2.public String deleteSurvey(){

surveyService.deleteSurvey(pid);

Return ”findMySurveysAction”;

}





其实在hibernate中有一个级联操作，即当对一个数据进行更新时，它会将与之关联的数据全部更新，但问题是这种级联操作对于程序员是不好的，它会出现一些莫名其妙的错误，使代码脱离程序员的掌握，而且性能极度不好，所以一般删除或更新还是应用hql批处理，这样性能也比较好

### 编辑问题

1.设计调查-->编辑问题

2.QuestionAction.editQuestion()

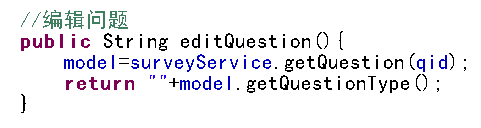
public String editQuestion(){

sid + qid

this.model = surveyService.getQuestion(qid);

return "" + model.getQuestionType() ;

}



注意：除了sid和qid，也要加入pid表示代表哪个页面，如果不加，会出现异常(尽管看似并没有哪里要接收pid，但是在设计调查页面，要显示qid时必须先进入pid才可以,这个理由也不对，搞不清楚，反正必须要保证关联要加上pid)

### 清除调查

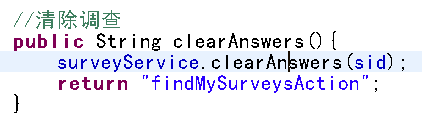
清除调查其实就是清除对应调查下的答案，有时候当设计好调查后，我们会先进行测试一下，这些答案是虚假的，无用的，所以要清除调查

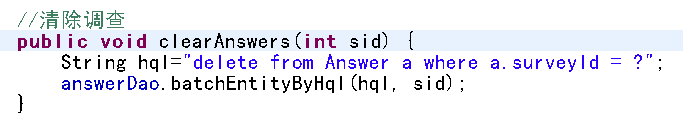
1. 我的调查----------》清除调查
2. Public String clearAnswers(){

surveyService.clearAnswers(sid);

Return “findMySurveysAction”;

}

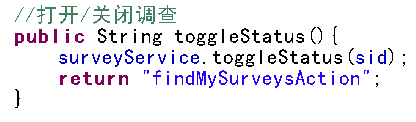


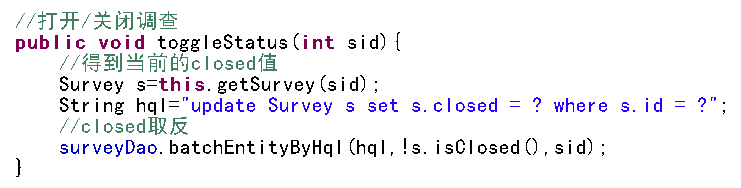


### 打开/关闭调查

这是对于调查是否可被参与调查的标记，一般情况下，只有允许公布的调查，用户才可进行参与，因此设置一个标记，用来规定是否可以被参与调查

1. 在survey实体类中加一个boolean类型的closed标记 closed为true 关闭 closed为false 开 放
2. 配置映射文件
   1. 如果closed不设置默认值，那么在数据库中对应的字段显示的是NULL，页面无法读入，会出现异常。
   2. 由于我们在编辑调查时，不希望对应的状态改变，因此在配置文件中可以设置update=”false” 这可以防止session中的save、update、saveOrUpdate方法，但是hql不能防止
3. 添加业务代码





### 增加logo

增加logo本质上就是上传图片(文件)，要实现增加logo的功能，我们的思路是将logo保存到服务器端，将logo的路径保存在数据库中，因此要在survey实体中加入一个logoPhotoPath属性，并在配置文件配置

到达logo页面

1. 设计调查-->增加logo
2. SurveyAction.toAddLogoPage()

SurveyAction.toAddLogoPage()

public String toAddLogoPage(){

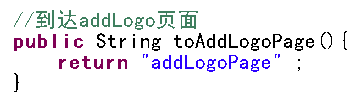
sid

return "addLogoPage" ;

}

3.struts.xml

4.跳转到/addLogo.jsp



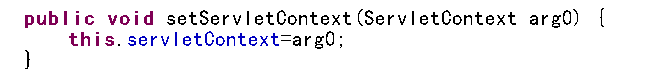


添加logo

首先为了保证能接收到客户端传过来的图片，要在SurveyAction中添加几个属性

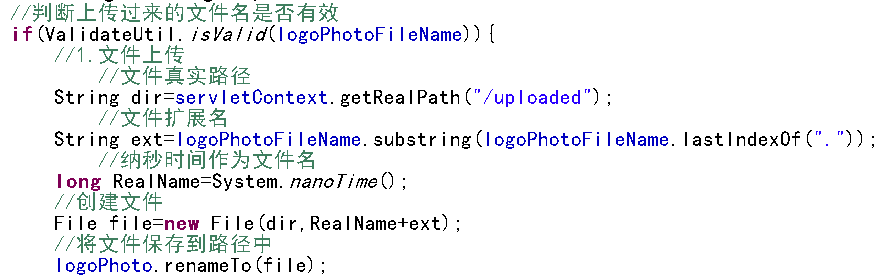


为了得到真实的路径，利用servletContext得到，因此可让SurveyAction实现ServletContextAware接口，并实现其方法，用来得到servletContext

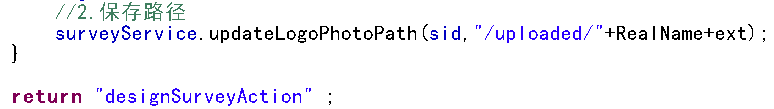


添加logo分为两个步骤，在方法中得到实现

1. 保存文件



1. 保存文件路径



在页面显示图片时用到的标签是



这里需要注意的是，由于可能出现logo并不存在，或者是路径没有，这种情况如果不加以判断，在页面上会显示一个不解析图片形状出来，为了解决这种情况，我们可以进行校验

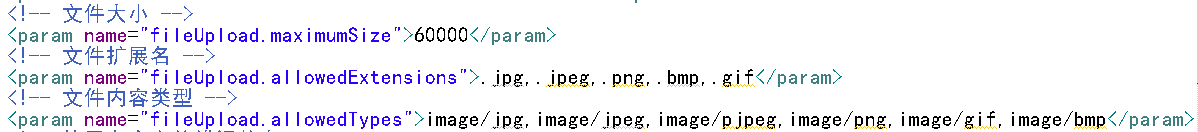
Struts的标签是很强大的，它不仅可以解析属性，也可以运行在Action中的方法，那么可以在Action中建立一个检验图片是否存在的方法，在显示页面先检验后显示





对上传文件约束

当我们增加logo时，有可能增加的类型不对，或者大小不符合，因此现在要添加对logo图片的约束，主要是从大小、扩展名、类型进行约束，对应的struts拦截器就已经设置了相应的功能，我们只需要在struts.xml配置就可以了



错误信息国际化

对上传文件约束后，检测时发现，错误信息很繁琐，不太看得懂，所以可以自定义错误信息，即错误信息国际化

在action目录下创建

1.创建SurveyAction.properties

[struts-core-x.x.x.jar/org.apache.struts/struts-message.properties]

struts.messages.error.file.too.large=The file is to large to be uploaded: {0} "{1}" "{2}" {3}

struts.messages.error.content.type.not.allowed=Content-Type not allowed: {0} "{1}" "{2}" {3}

struts.messages.error.file.extension.not.allowed=File extension not allowed: {0} "{1}" "{2}" {3}

2.创建临时文件1.txt(记事本编写(gb2312))

struts.messages.error.file.too.large=文件太大!

struts.messages.error.content.type.not.allowed=类型不对!

struts.messages.error.file.extension.not.allowed=扩展名不符!

3.对1.txt进行转码(ascii码)

cmd

d:

cd xxxx

native2ascii -encoding gb2312 1.txt 2.txt

4.刷新项目,产生一个2.txt文件

5.删除1.txt文件

6.重命名2.txt --> SurveyAction\_zh\_CN.properties

做完上述这些，还有一个十分重要的地方要去修改，我这里所用的struts2版本是2.1.6，是比较低的一个版本，在拦截器fileUpload中有一个boolean类型的变量useActionMessageBundle，它代表的意思是使不使用自定义的错误信息，默认值是false，所以需要在struts.xml文件中更改了这个拦截器变量的值，而对于较高的版本时，它已经去掉了这个变量，默认支持使用自定义错误信息



### 移动/复制页

准备工作

1. 向Page实体添加一个属性orderno，这个属性即代表了按照此属性排序
2. 修改Page.hbm.xml和Survey.hbm.xml映射文件，将orderno加入进去，并使页面排序按照这个属性，注意在映射文件添加上这个属性时，orderno必须赋初值，orderno与id是相等，由于hibernate在给属性赋值时是根据set方法赋值，所以要重写Page中的setId方法，让id等于orderno

#### 移动页

对于移动页来说，就是将一个调查里面的所有页位置可以调换

设计调查页面--------》移动/复制页

1. MoveOrCopyPageAction.toSelectTargetPage()方法

public String toSelectTargetPage(){

srcPid

this.mySurveys = surveyService.getSurveyWithPages(user);

public List<Survey> getSurveyWithPages(User user){

String hql = "from Survey s where s.user.id = ?" ;

List<Survey> list = surveyDao.findEntityByHQL(hql,user.getId());

for(Survey s : list){

s.getPages().size();

}

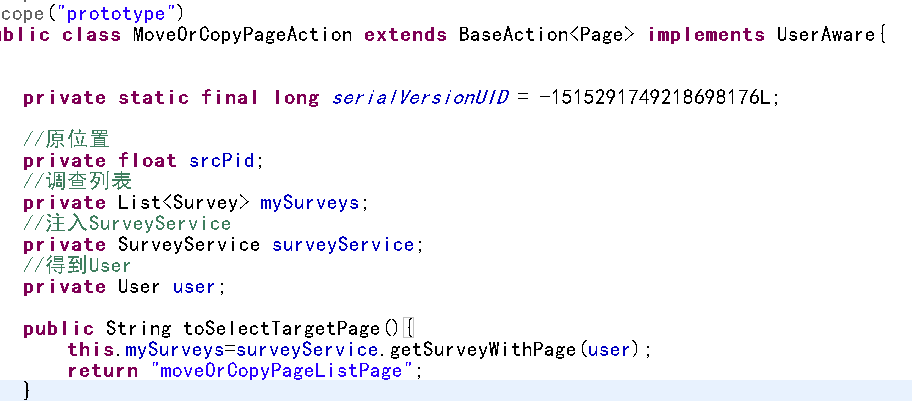
return list ;

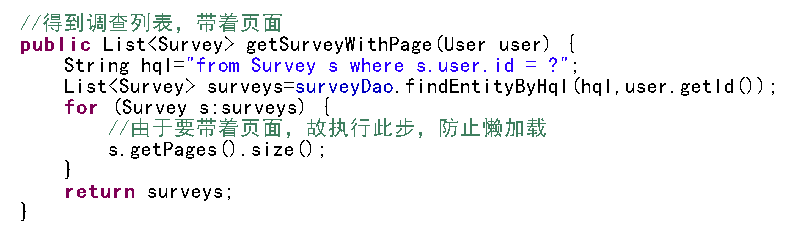
}

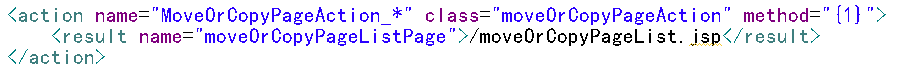
return "moveOrCopyPageListPage" ;

}

1. struts.xml配置







进行移动页面

思路如下

1. 先判断是不是同一个调查内的
   1. 同一个调查
      1. 在判断pos的值，看是向目标页前插还是目标页后插
         1. 前插
            1. 判断是不是首页

是首页

srcPage.orderno=targPage.orderno-0.01f

不是首页

得到目标页前一页prePage

srcPage.orderno=(targPage.orderno+prePage.orderno)/2

2. 后插

* + - * 1. 判断是不是首页

是首页

srcPage.orderno=targPage.orderno+0.01f

不是首页

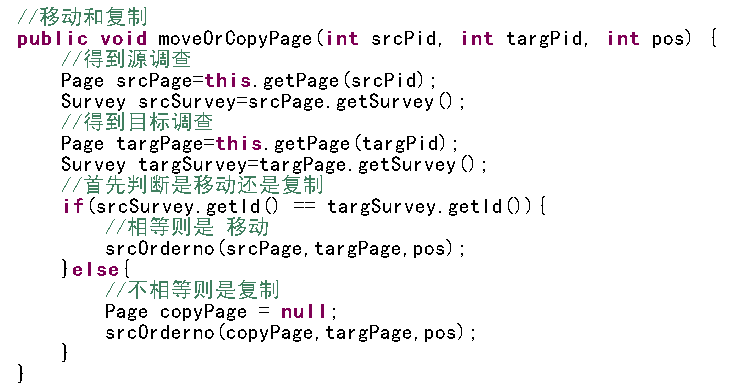
得到目标页前一页nextPage

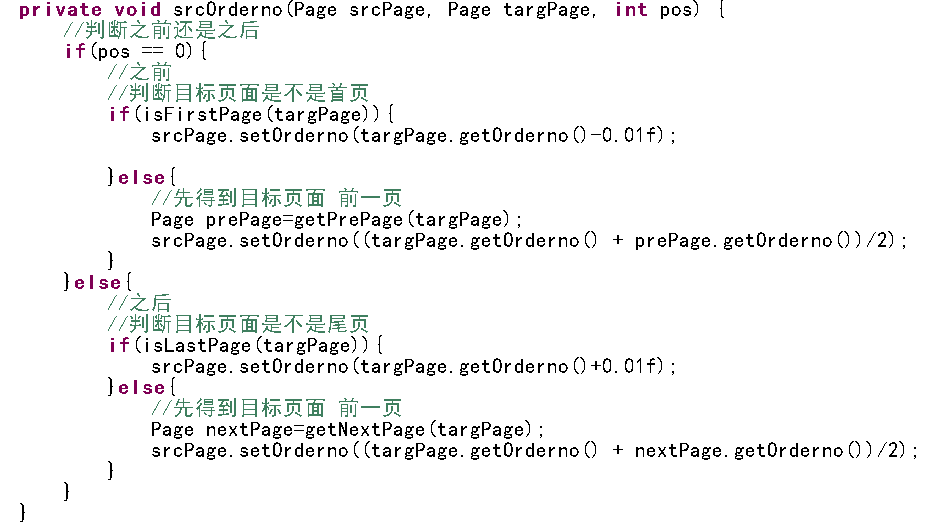
srcPage.orderno=(targPage.orderno+nextPage.orderno)/2

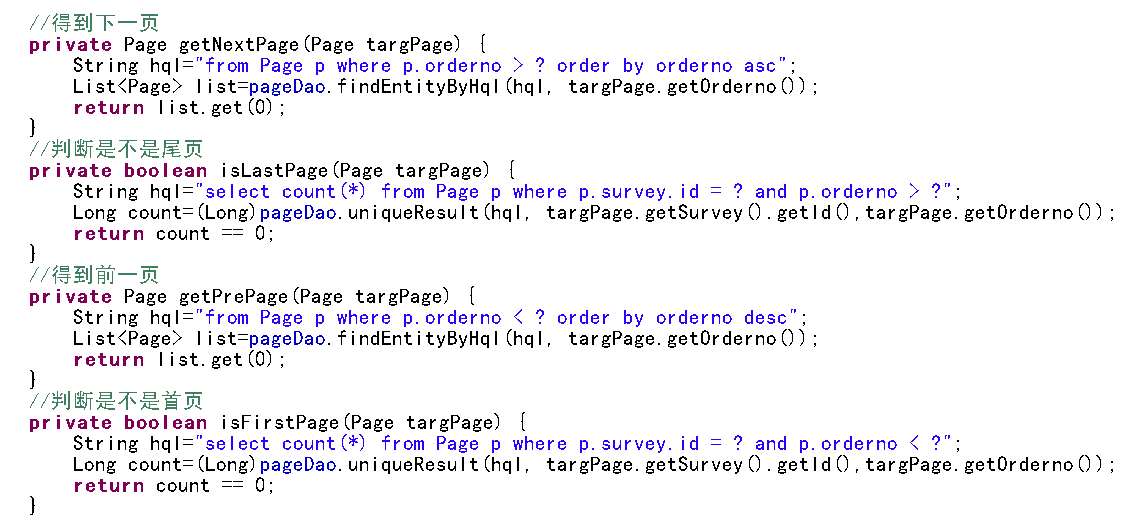
* 1. 不同调查

那说明就是复制

最后更改srcPage的值，老师的程序中并没有这一步，但是它自动保存了，我的程序必须加这一步才能成功







#### 复制页

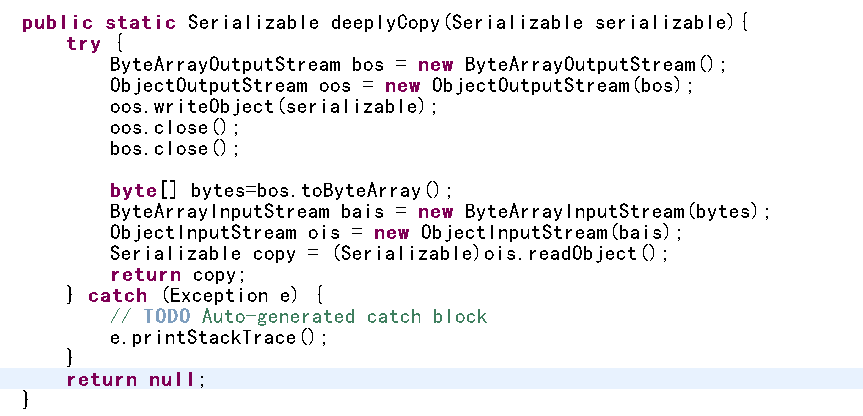
对于复制页来说，就是将不同调查里的页面可以复制进去使用

这里有第一个值得说出去的学会的知识---------------------》深度复制(java的串行操作)

深度复制不是指只是把数据赋值过去就行，深度复制指的是要复制出一个一模一样的关系，不仅是数据上的相等，而且也是地址的相等。

深度复制利用的是java的串行化技术，要使用这个技术，要进行深度复制的类，必须实现serializable 接口，要是有不想复制的类，与要复制的类之间有关联，要将要复制的类内中将不复制的类加上一个前缀 transient，这样就不会出错，也可复制，重要的是这样性能就大大提高，我们可以进行部分复制

深度复制所用的代码

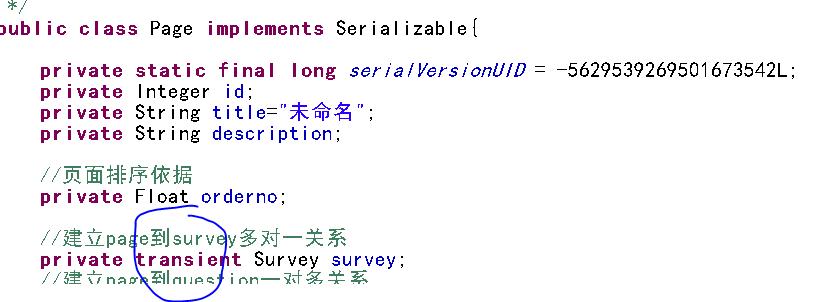


原理：

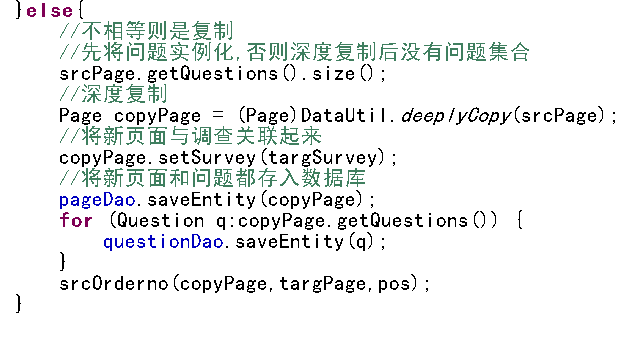
1.首先要将对象转化成字节数组，因此先定义一个字节数组输出流(ByteArrayOutputStream)，然后用对象输出流(ObjectOutputStream)封装，利用对象输出流的writeObject方法，将Serializable对象放入字节数组输出流中

2.将字节数组输出流中的字节数组再转换成对象，那么先定义一个字节数组，将字节数组输出流中的对象利用toByteArray方法放入定义好的字节数组中，这时这个字节数组就是对象的转成的，然后定义字节数组输入流(ByteArrayInputStream)，并将字节数组放入，然后用对象输入流(ObjectInputStream)封装，利用对象输入流的readObject方法，将字节数组转换成对象。这样利用java的串行化技术就完成了深度复制

由于我们要复制页，所以要将page、question类全部实现serializable接口，在page中有调查属性，这不需要复制，在前面加上transient



然后对页进行深度复制



### 参与调查

这是对已经存在的调查参与调查，因此创建一个新的Action来处理EngageSurveyAction

#### 显示可参与调查

1.导航栏-->参与调查

2.EngageSurveyAction.findAllAvailableSurveys()方法

public String findAllAvailableSurveys(){

this.surveys = surveyService.findAllAvailableSurveys();

public List<Survey> findAllAvailableSurveys(){

String hql = "from Survey s where s.closed = ?" ;

return surveyDao.findEntityByHQL(hql,false);

}

return "engageSurveyListPage" ;

}

3.struts.xml

<!-- EngageSurveyAction -->

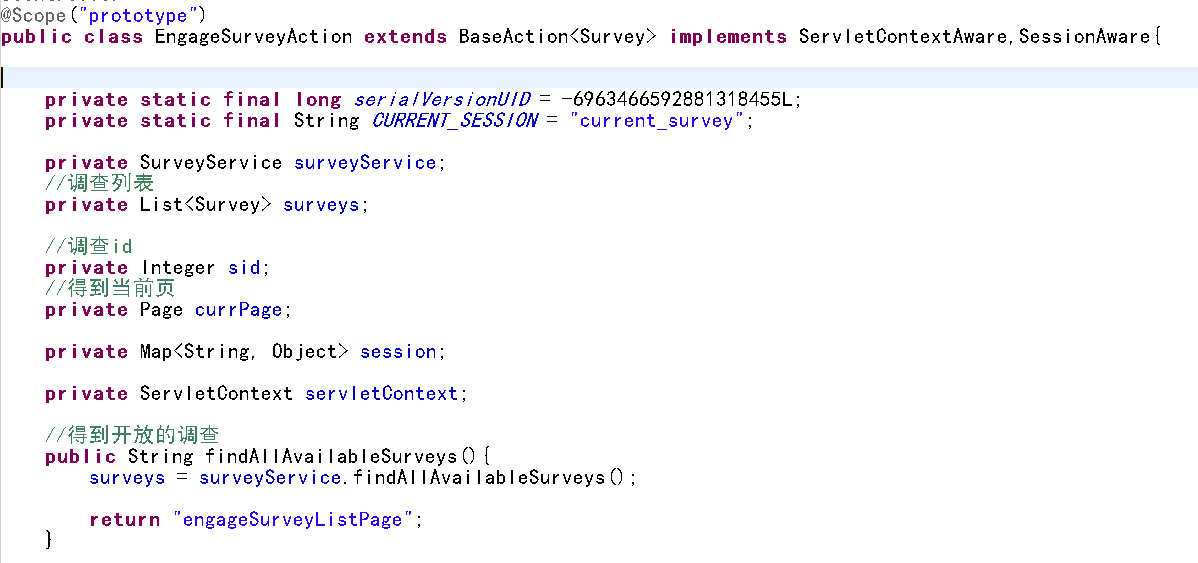
<action name="EngageSurveyAction\_\*" class="engageSurveyAction" method="{1}">

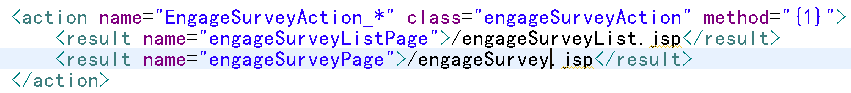
<result name="engageSurveyListPage">/engageSurveyList.jsp</result>

</action>

4.跳转到/.jsp

\*注意-----》可以显示的调查并不是所有的调查，而是打开的调查

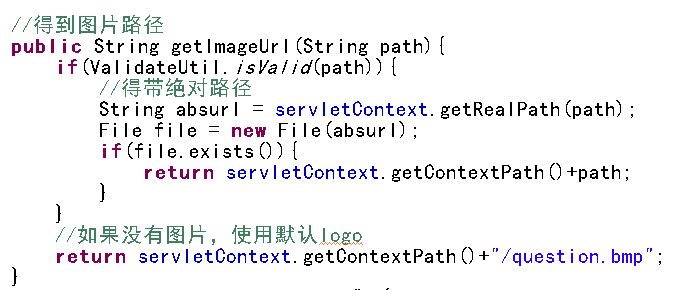




#### 得到logo

通过survey中的logoPhotoPath取得地址，如果存在，则显示，不存在，则使用默认地址

在页面利用struts的标签得到



### 进入调查参与

1.参与调查视图页面-->点击调查

2.EngageSurveyAction.entry()方法

public String entry(){

sid

//查询调查首页

this.currPage = surveyService.getFirstPage(sid);

public Page getFirstPage(Integer sid){

String hql = "from Page p where p.survey.id = ? order by p.orderno asc";

List<Page> list = pageDao.findEntityByHQL(sid);

Page p = list.get(0);

p.getQuestions().size();//初始化问题集合

p.getSurvey().getTitle();//初始化调查对象

return p ;

}

//查询当前调查存放到session中.

sessionMap.put("current\_survey",currPage.getSurvey());

return "engageSurveyPage" ;

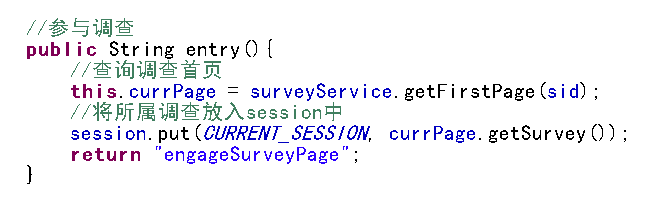
}

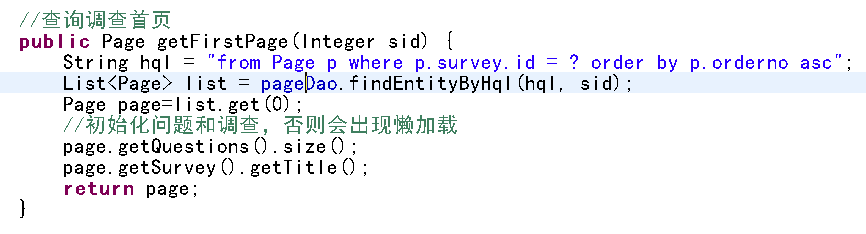
3.struts.xml

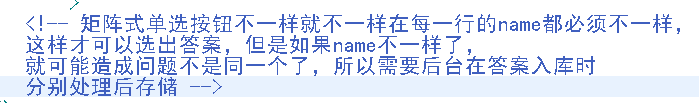
4.跳转到/engageSurvey.jsp

[类似于designSurvey.jsp]

进入页面后显示的是调查的首页，因此先查询出调查的首页，而在得到首页后一定要记得初始化问题集合和调查集合，否则会出现懒加载问题，出现错误

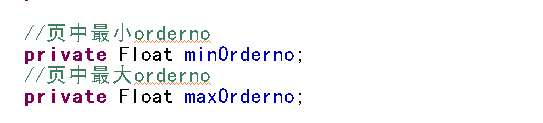






### 最大页序和最小页序的处理方式

在进入参与页面后，有一个上一页 和下一页等按钮， 这里遵循的规则是 如果是首页，则上一页按钮不存在，如果是尾页，则下一页按钮不存在，因此可以在survey实体中加入两个属性，最大页和最小页，利用sql的聚集函数求出





### 上一页/下一页/完成/退出

当我们进入参与调查页面时，显示的是调查的首页，可以通过点击下一页得到调查的下一页，点击退出退回到调查列表，每个按钮都是一个提交，所以可以根据提交按钮的名字分别处理

doEngageSurvey(){

String submitName = getSubmitName();

If(上一页){

}else if(下一页){

}else if(完成){

}else if(退出){

}

}

每一次点击按钮，除非是完成，否则传送过来的数据都不能直接入库，因此可以将这些数据都先存入session中，而且为了返回支持来回返回回显，结构设计成如下

//这页具体的参数

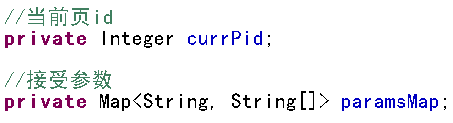
Map<String ,String[]> paramsMap

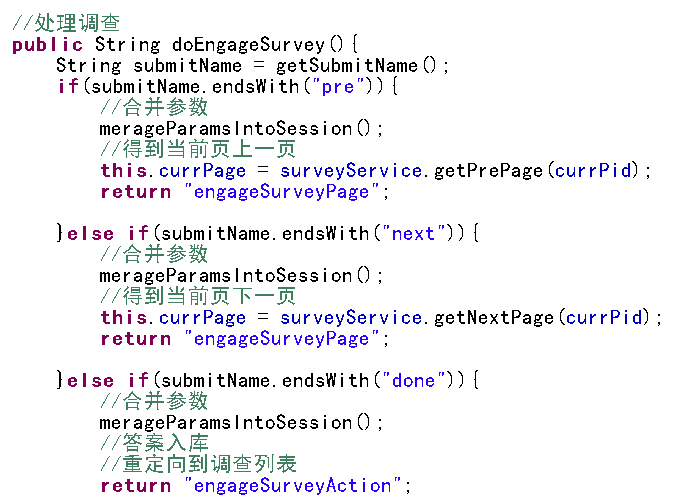
//第几页参数

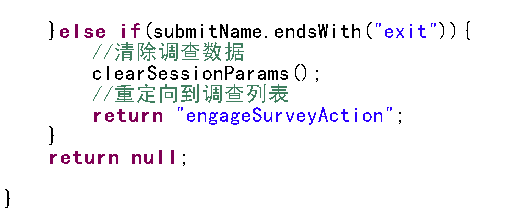
Map<Integer,Map<String ,String[]>> allParasMap;

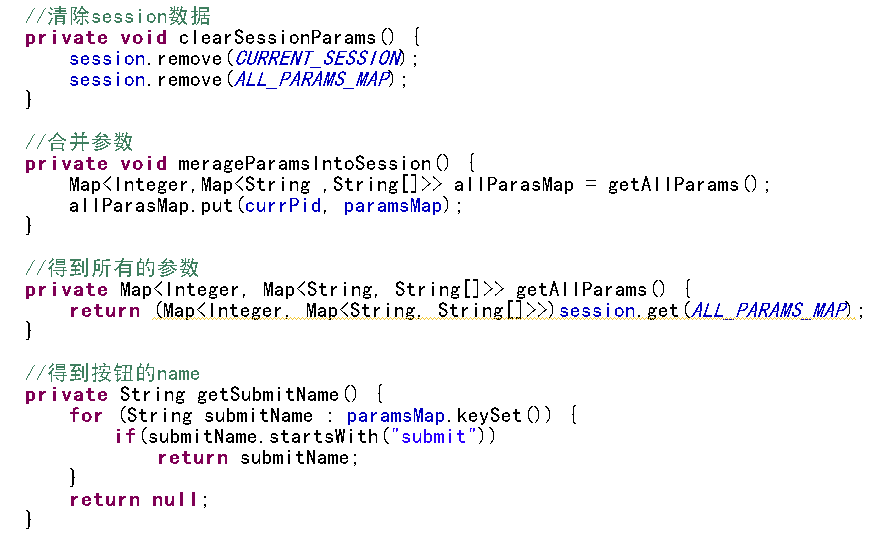
//总共的参数

Session.put(ALL\_PARAMS\_MAP,allParasMap);









#### 重写删除页

删除最后一个页面时,会出现无法将null赋值给基本类型的属性 for minOrderno来说.

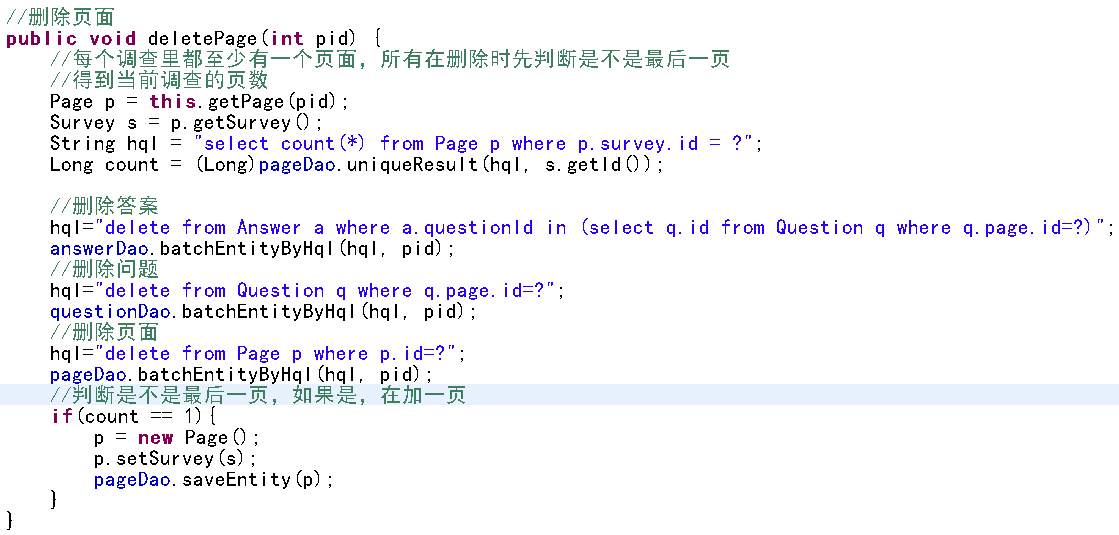
-----------------------

1.每个调查至少有一个页面,删除时需要做判定.

2.

3.

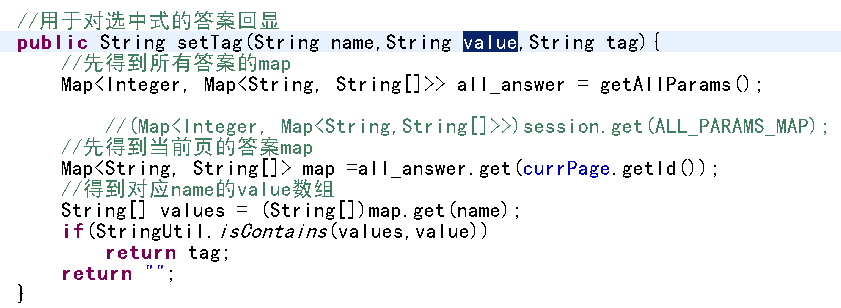
4.

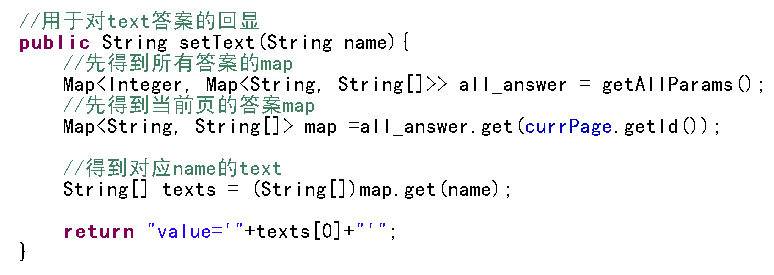


### 答案回显

这里可以分为两种情况，一种是选中式回显，一种是文本回显，对于选中式回显来说，它要存在两种情况，一种是单选和多选的checked，一种是下拉选项中的selected，文本回显就是把输入的文本取出。

针对这两种情况，我们分别定义一个函数，setTag和setText ，它们在最开始都是要先通过session获得答案map，对于选中式回显，是查看map中相应的值中包不包含，若包含，返回对应的checked或者是selected。对于文本框回显，是查看map中相应的值中存不存在，，存在返回值





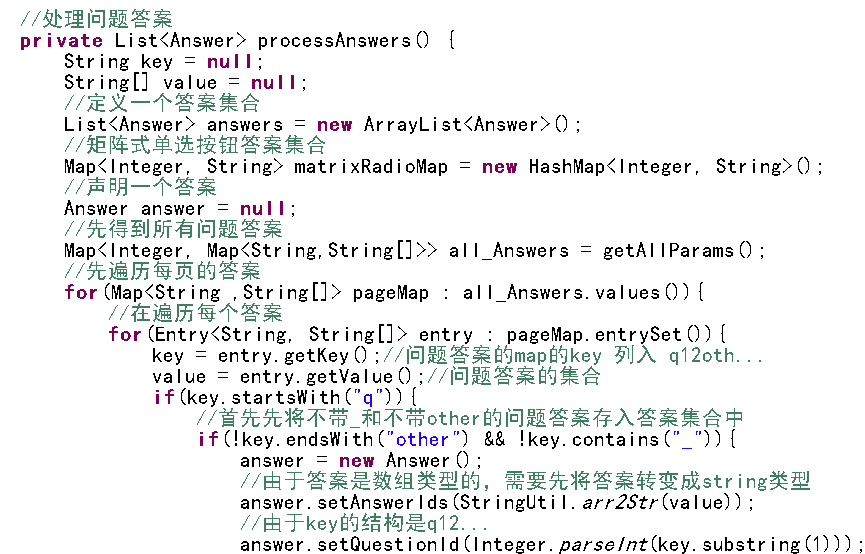
### 处理答案

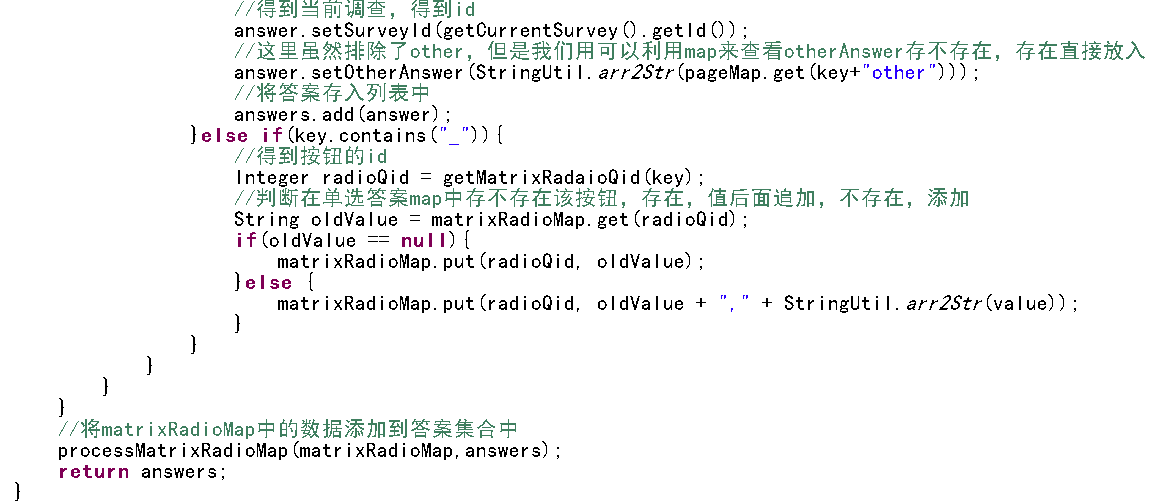
答案存储在session中，在最后完成时需要入库

步骤是

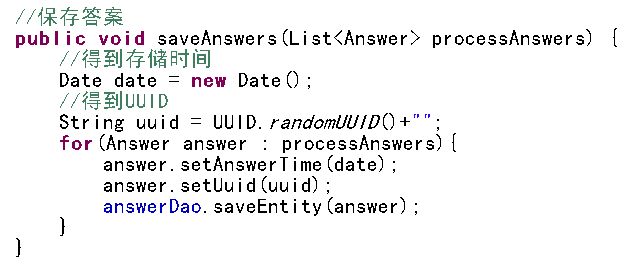
1. 建立一个answer列表
2. 将map中的答案一个一个取出，处理后，放入answer列表中
3. 调用surveyService中保存答案方法，将答案入库











### 问题统计

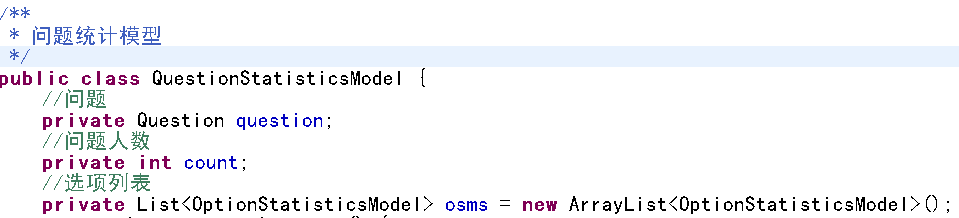
Sql语句 查看是否选中某个 用模糊查询”%,0,%”

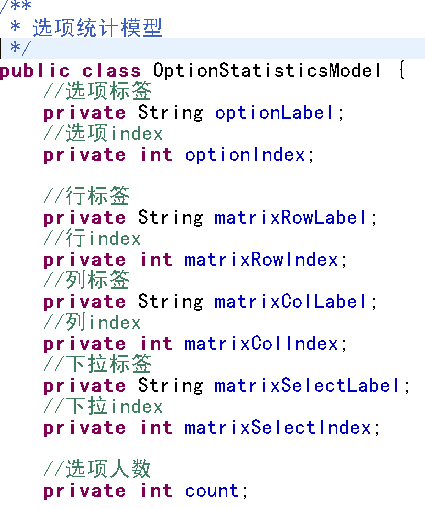
要将对应问题答案上前后都加上’，’，采用mysql的函数concat(‘,’ , answerids , ’,’)

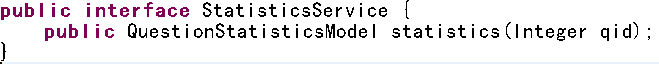
问题统计分为两方面

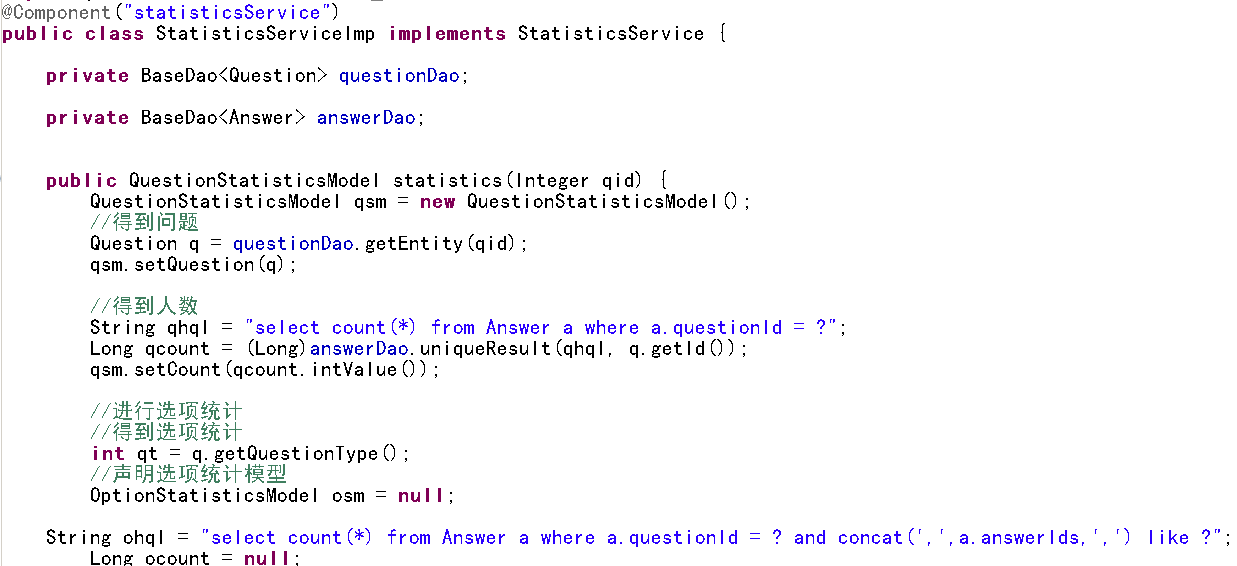
1. 对于每个问题参与人数的统计
2. 对于每个问题的答案统计

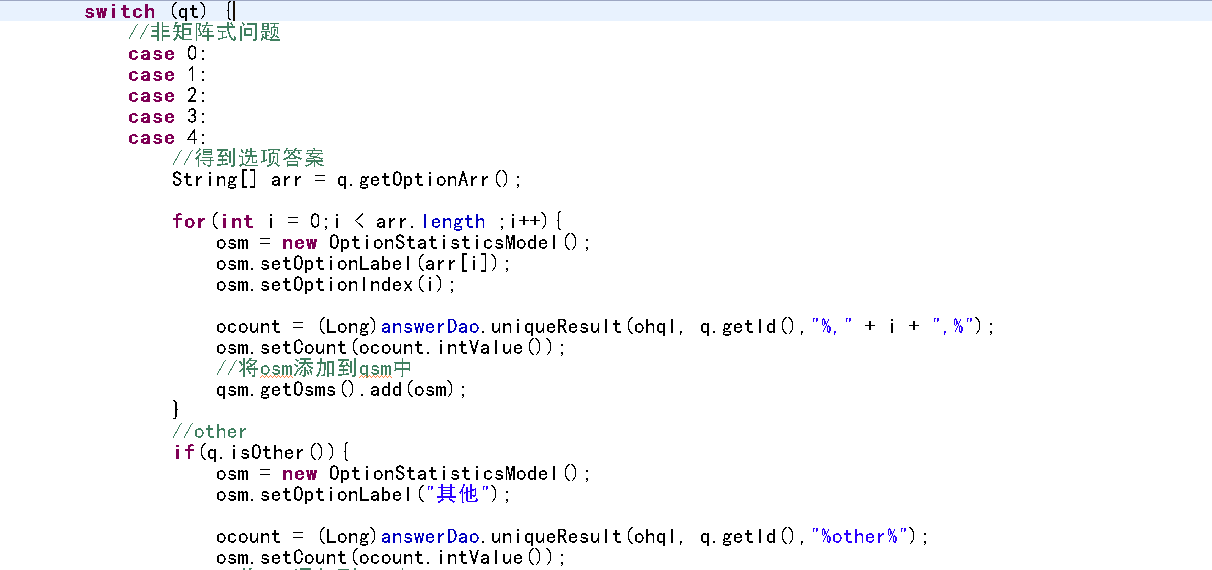
定义两个类，分别进行问题统计与答案统计，然后在定义一个进行问题统计的service



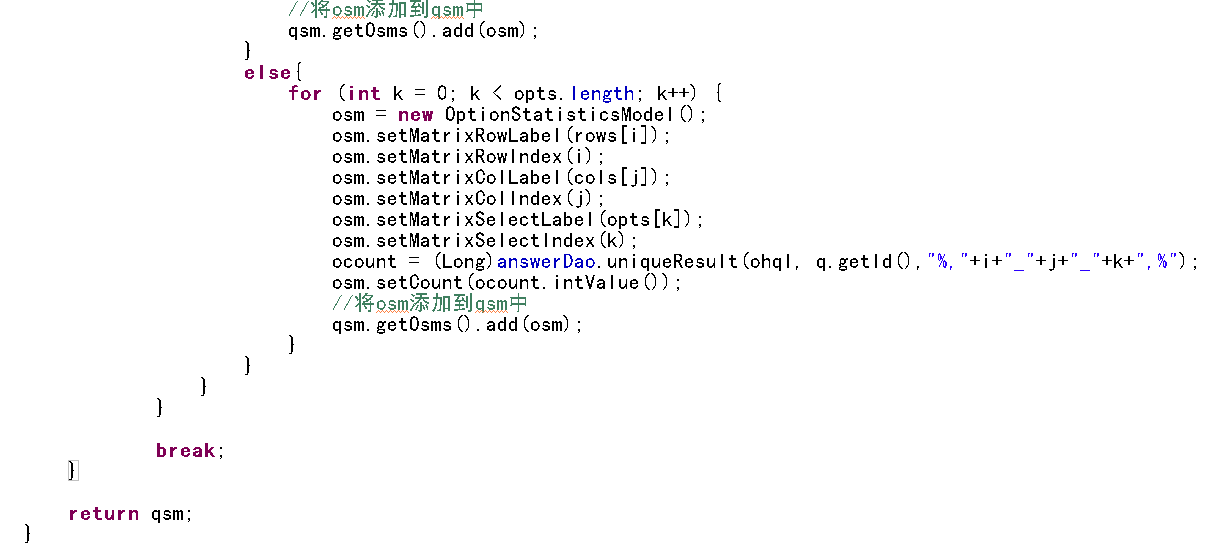






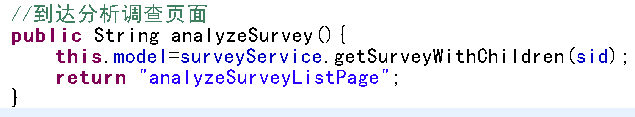






### 分析调查

1. 我的调查-----》分析
2. 通多suryveyId查出调查及其页面和问题

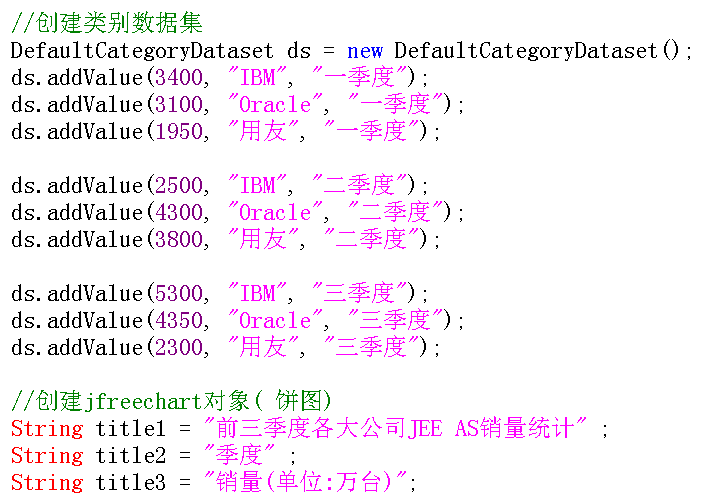


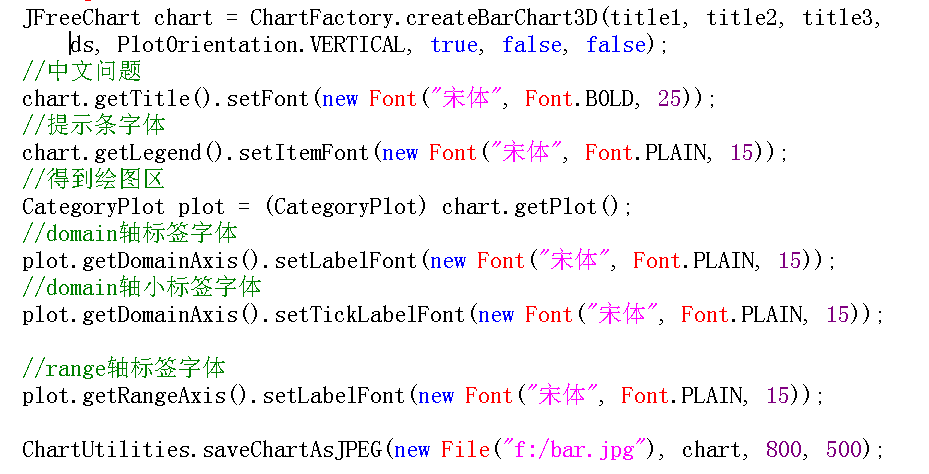


### 生成报表工具-----JFreechart应用

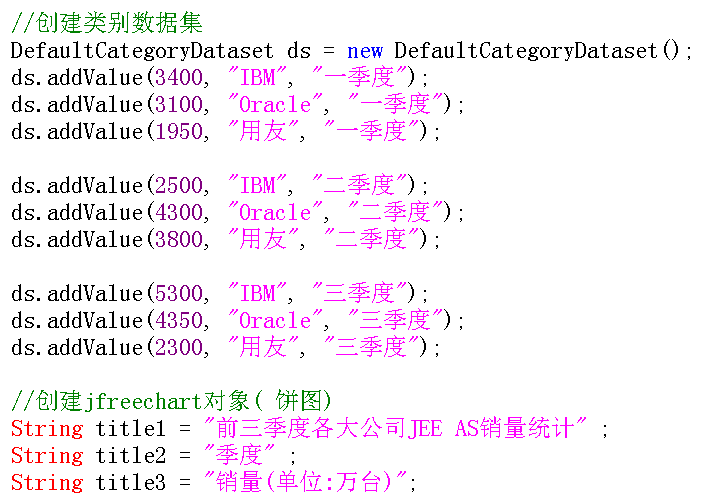
JFreechart是专门用于生成报表的插件

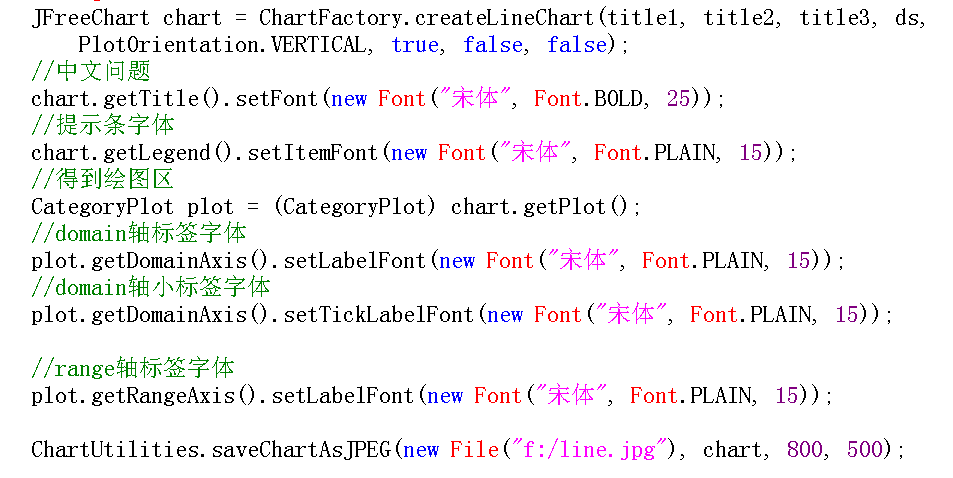
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*生成柱状图\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*



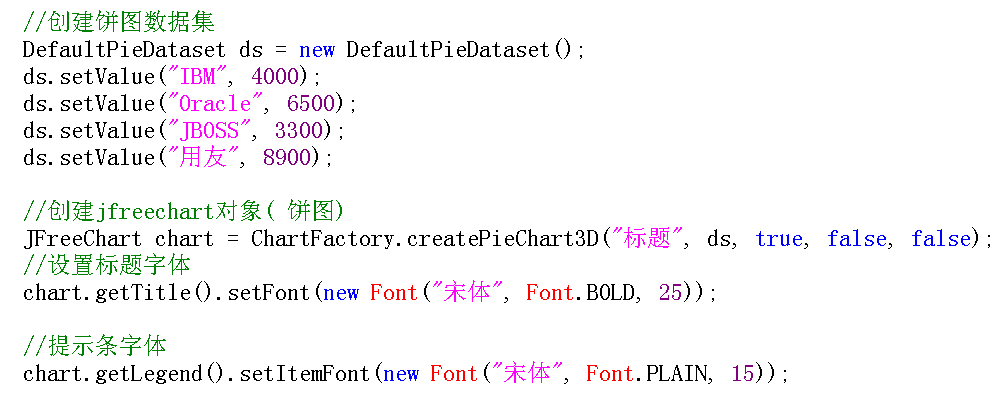


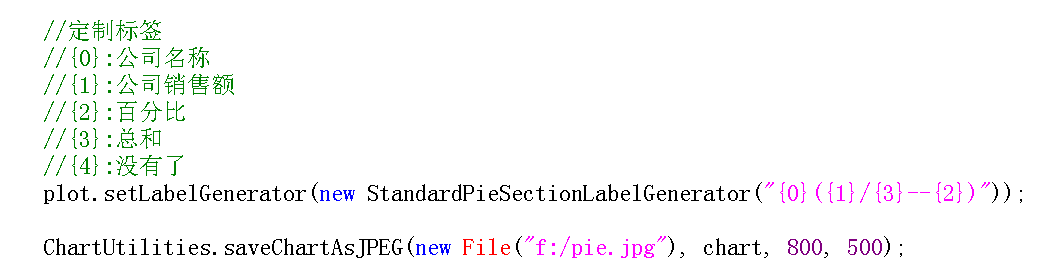
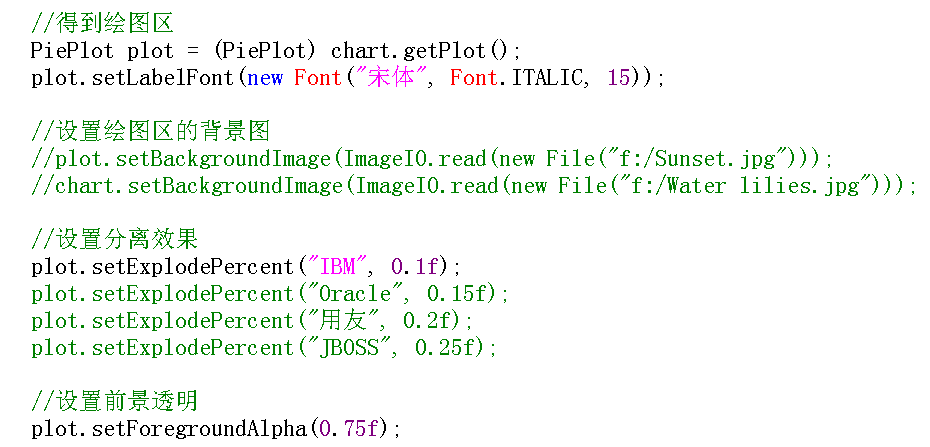
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*生成折线图\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*





\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*生成饼图图\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*





#### JFreechart整合

1.引入类库

iText-2.1.5.jar

jcommon-1.0.18.jar

jfreechart-1.0.15.jar

2.分析列表页面-->提交

3.ChartOutputAction.execute().

public String execute(){

return success ;

}

public InputStream getIs(){

//使用jfreechart生成图标,并存放在inputStream中.

}

4.struts.xml

<action name="ChartOutputAction" class="chartOutputAction">

<result name="success" type="stream">

<param name="contentType">image/jpeg</param>

<param name="inputName">is</param>

<param name="bufferSize">1024</param>

</result>

</action>

1. 输出图片给浏览器

#### 插件方式整合JFreechart

在struts中自带有一个插件用来整合JFreechart，可以省去转换流部分直接将chart送入servletOutputStream中

使用struts2给出的插件整合jfreechart

-------------------------------------

1.导入插件:struts2-jfreechart-plugin-2.3.15.1.jar

2.改造action

class ChartOutputAction{

..

Jfreechart getChart(){

...

return chart ;

}

}

3.修改struts.xml文件,指定chart结果类型

<package name="surveyparkPkg" extends="struts-default,jfreechart-default" namespace="/">

...

<!-- 图表输出action -->

<action name="ChartOutputAction" class="chartOutputAction">

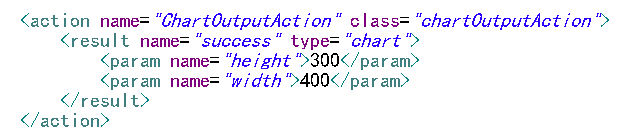
<result name="success" type="chart">

<param name="height">300</param>

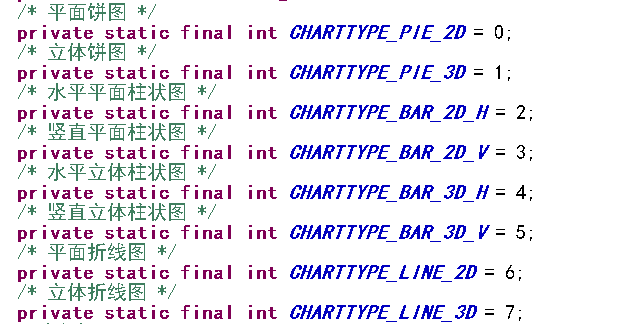
<param name="width">400</param>

</result>

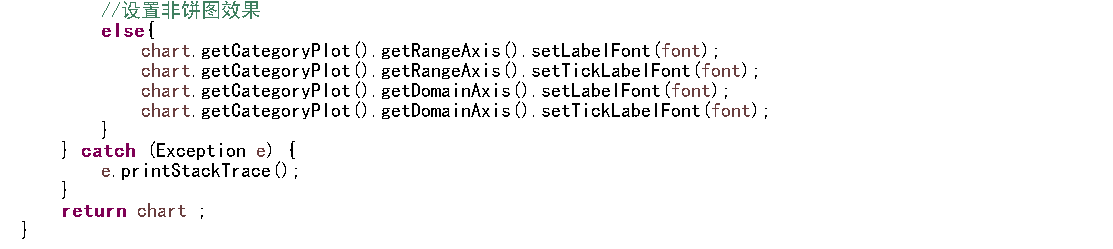
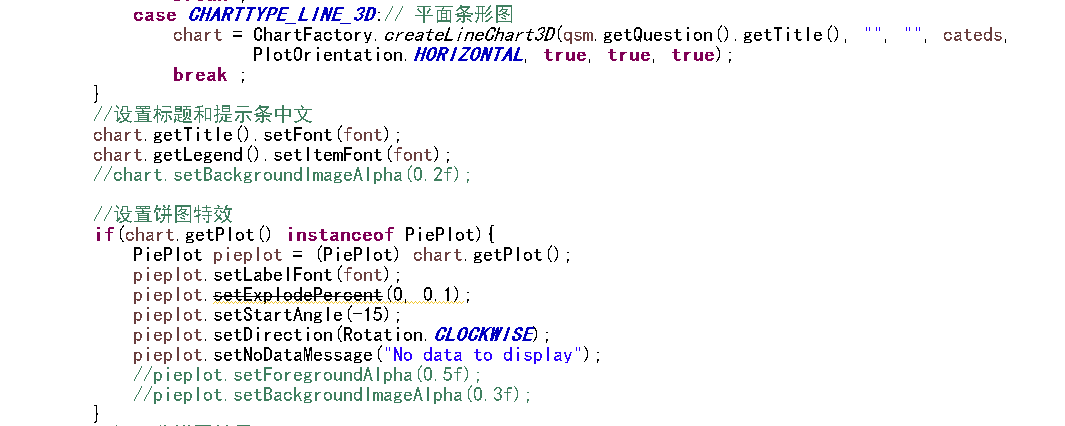
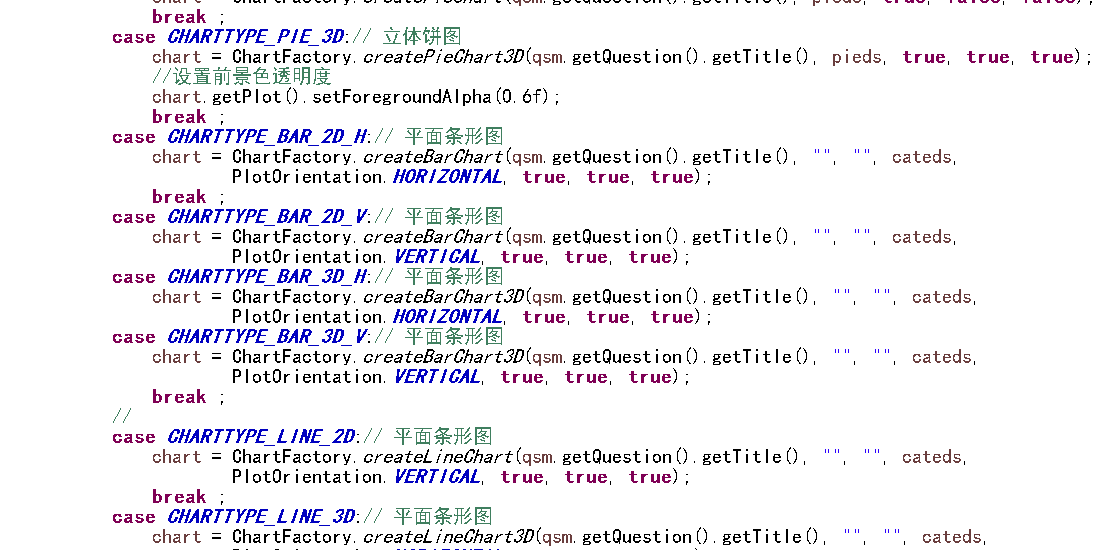
</action>



进行图形生成形状的选择，在action中定义如下常量



在getChart方法中进行选择

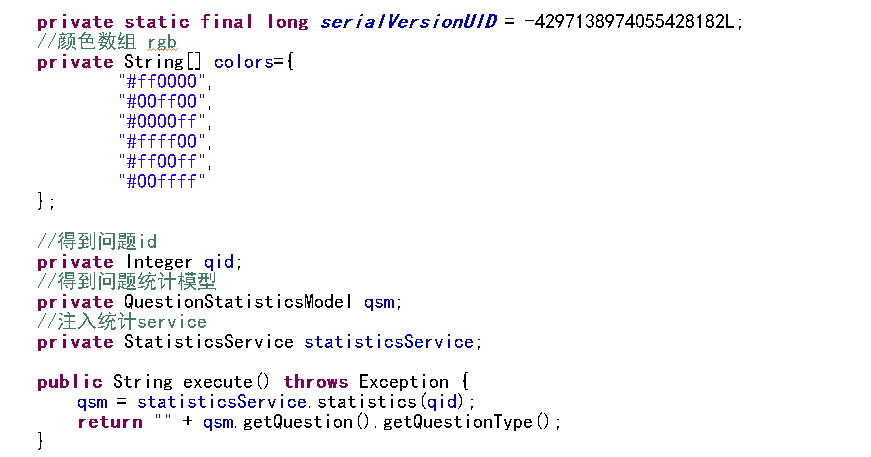


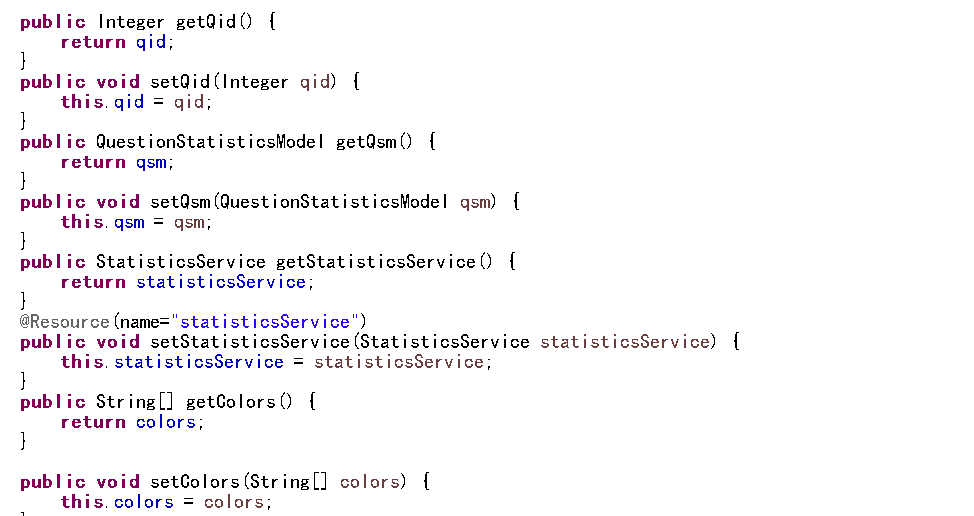
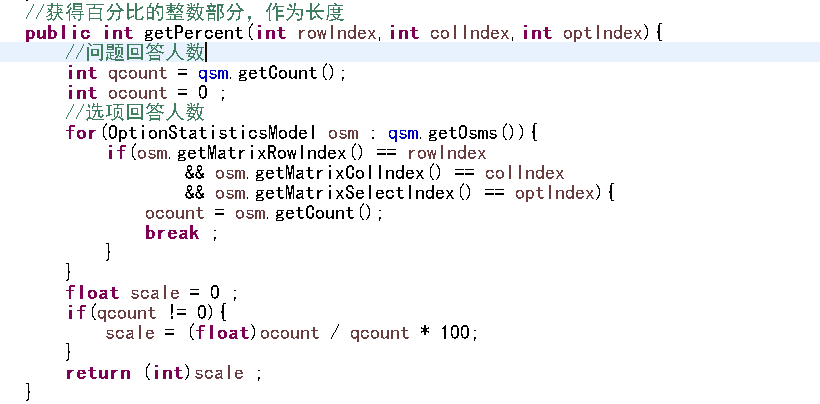
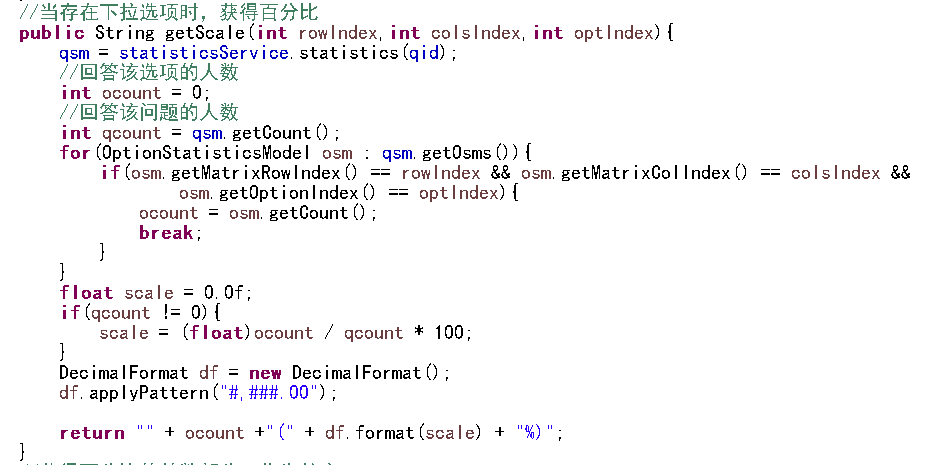
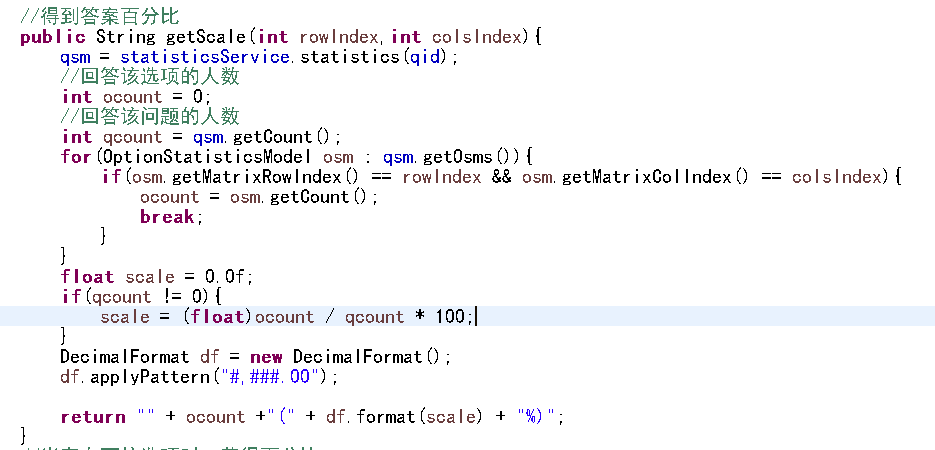
在系统中不只是存在图表的生成，对于矩阵问题，图就无能为力了，所以用表格方式表明，

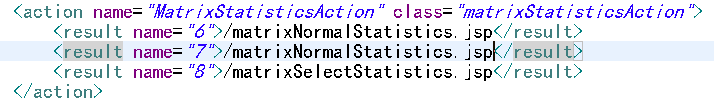
重新定义一个action，它是一个静态的action，返回值只有矩阵式三种类型 6、7、8，在6和7是生成表格，8 是生成类似于柱状图的图形

主要是依赖action中两个方法，getScale和getPercent

getScale获得百分比 getPercent获得柱状的宽度







### poi输出xls

Poi是一个专门针对xls、word、ppt的插件

步骤:

1.下载类库

commons-codec-1.5.jar

commons-logging-1.1.jar

log4j-1.2.13.jar

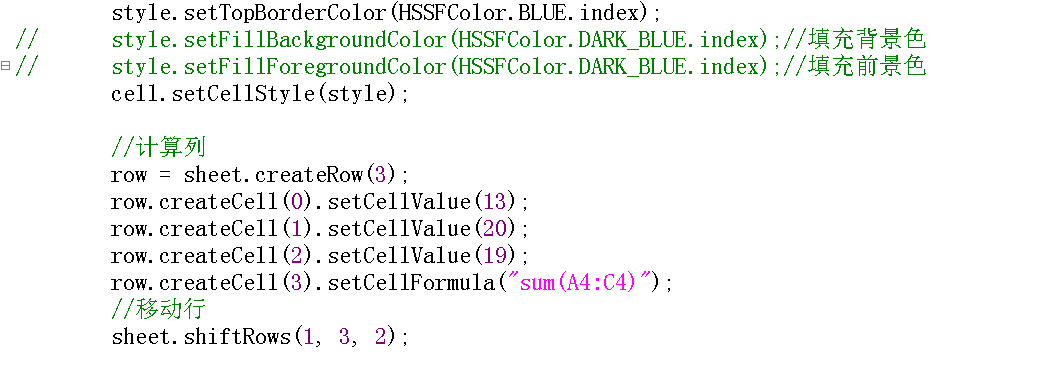
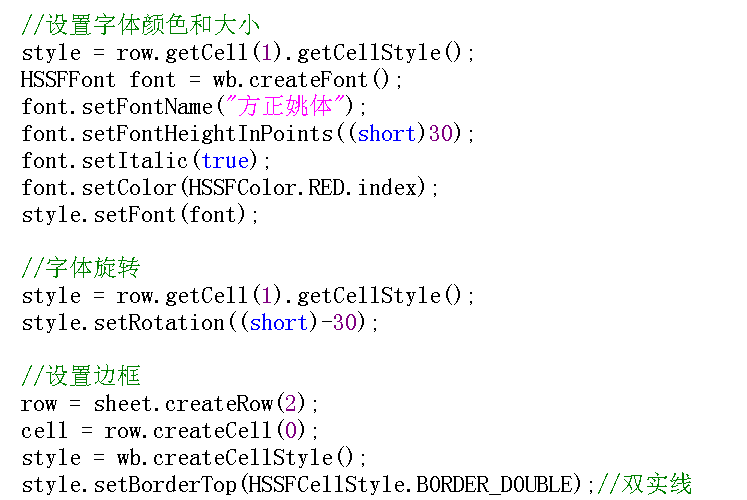
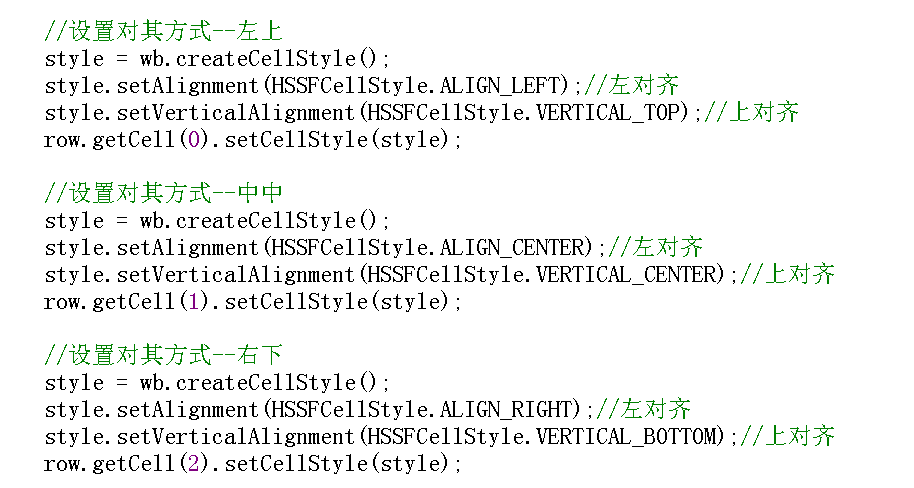
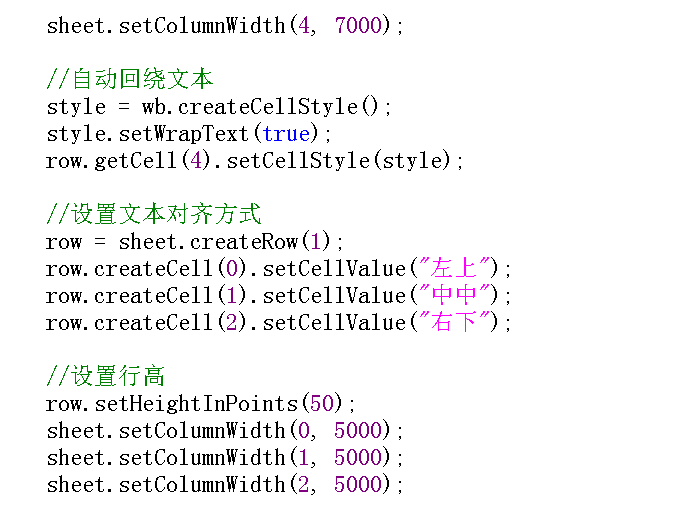
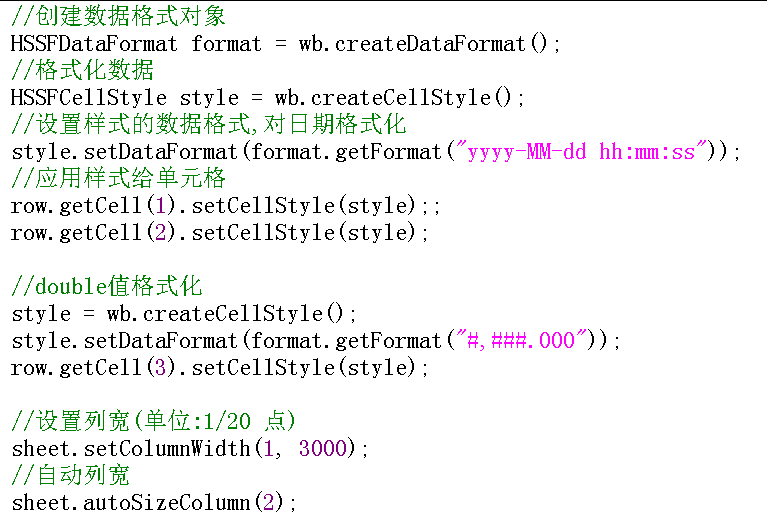
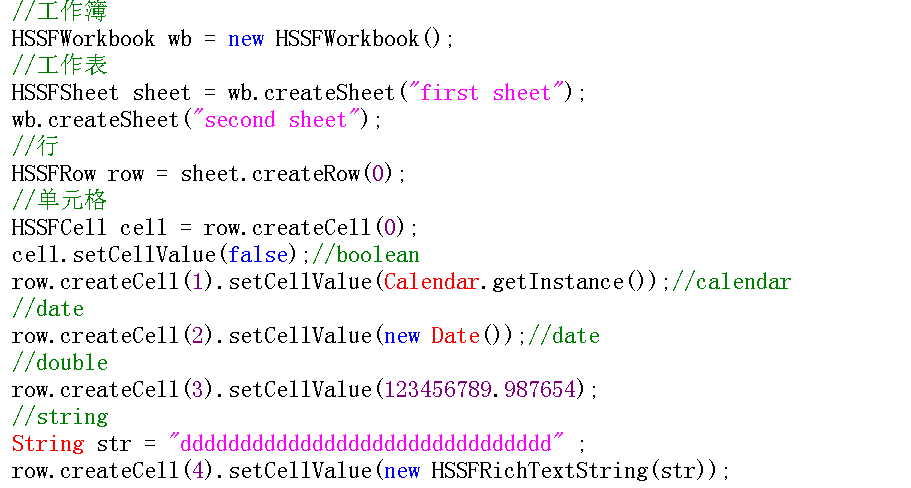
poi-3.9-20121203.jar

xls -----> sheet ---> row ---> cell

HSSFWorkbook ---> HSSFSheet --> HSSFRow ---> HSSFCell

cell.setValue(boolean|Calendar|Date|String|double|Richtext)

关于poi的这种操作



#### 集成poi输出xls

1.引入类库

commons-codec-1.5.jar

poi-3.9-20121203.jar

2.调查列表-->收集调查

3.CollectionSurveyAction.execute()

public String execute(){

sid

return "success" ;

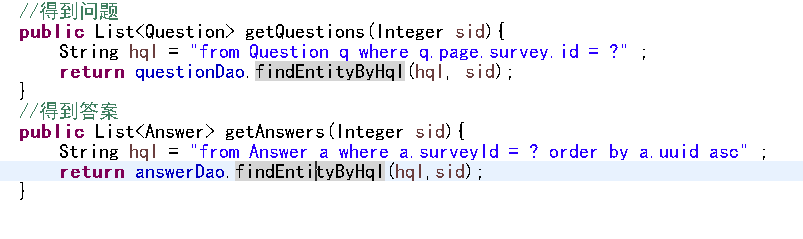
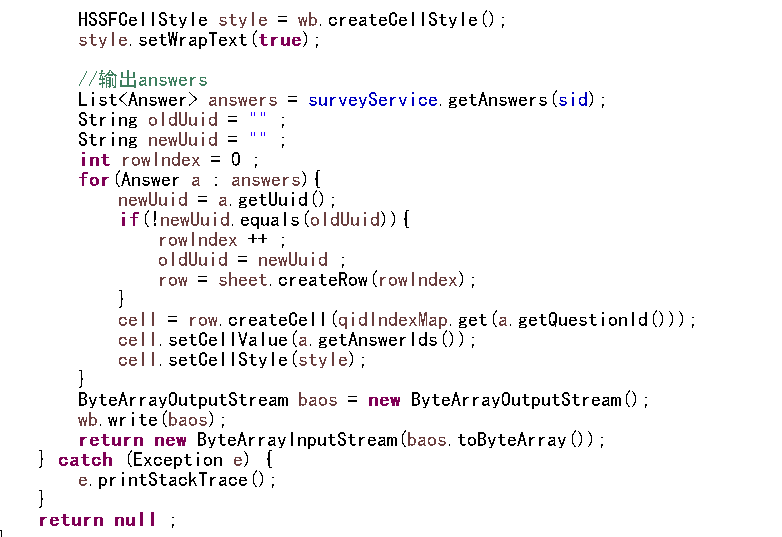
}

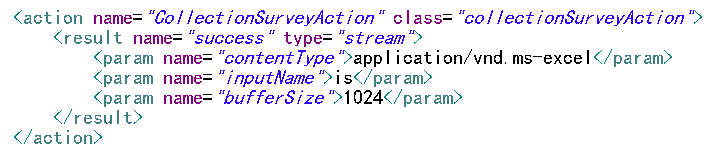
public InputStream getIs(){

poi --> inputStream

}

利用poi生成对于答案的xls文件





### 权限的分析与实现

在一个软件系统中权限的控制是一个特别重要的东西，每个系统都会有权限的存在，它代表的其实就是每个url地址，用户是否有资格去访问这个url，大的系统中会有成千上万的权限，而不同的用户因为自身不同的角色，所以有不同的权限，要进行权限的控制与管理。有三个实体

User:系统内的所有的用户

Role:系统内的不同的角色

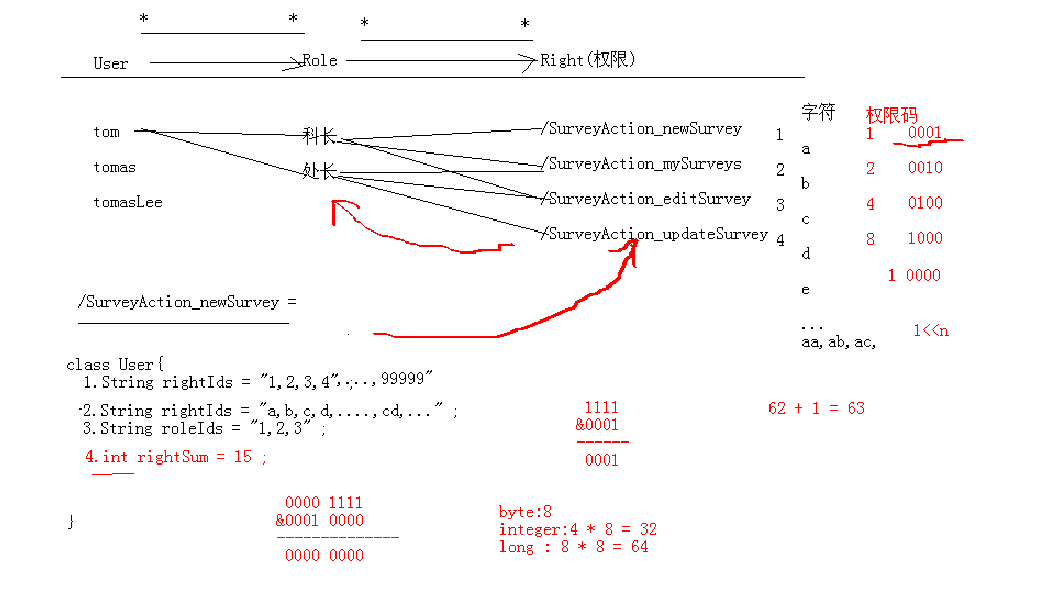
Right:系统内的权限

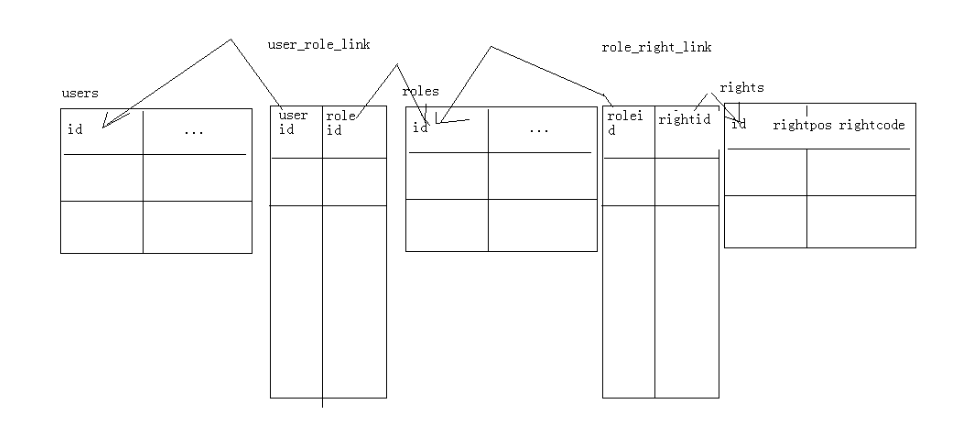
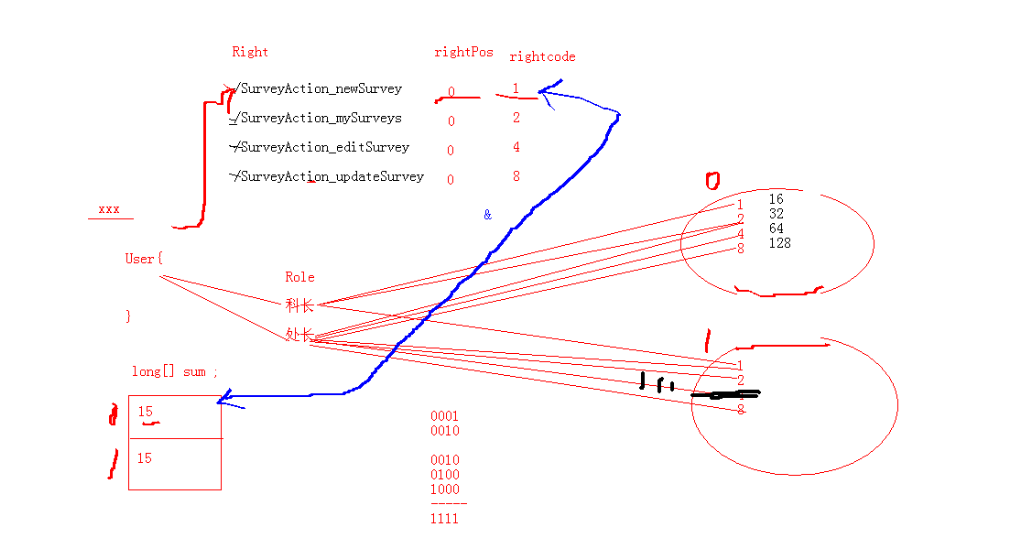
如何进行权限的设计?

在过去，很多人会在权限表上加一个字段，从1开始递增，在user表中加一个String类型的权限值，如果该用户有该权限，那么就在修改权限值，用逗号隔开，然后进行判断是否存在。但是这特别不好，因为权限动辄就是成千上万，这么多的权限都写在一个权限值后面，在进行比对，十分耗时耗力，因此不采用这种方法。后来有人进行改进，用a，b，c...26个英文字母进行标识，当字母用完，再在字母后添加新的，如aa,ab，但是这种方法本质上和第一种没区别。

现在一般采用的是另一种

在权限表中加上一个字段----权限码，它的类型是long,它的值得出原理是1每次左移一位，如1,2,4,8...，在user表中的加上一个字段----权限总和，类型是Integer，它的得出是拥有的权限进行 或 运算得出，如果到时候进行比对时，直接拿这个权限总和与要匹配的权限码进行 与 运算，如果不等于0，那么说明存在该权限，这种方法优点是效率最高，但是这有一个致命的错误，对于java中类型来说，最大的为long 是64位，意味着1最多只能向左移63位，说明权限最多只有64个，所以为了解决这种情况，又在权限表中加了一个----权限位，从0开始，它的作用相当于对权限分组，只要权限达到64个，权限位改变，权限码从1开始，这样权限的数量就不会受到限制。User表中的权限总和类型改成long[]数组类型的。要进行哪个权限组的比较，就有long[index]中的总和比较

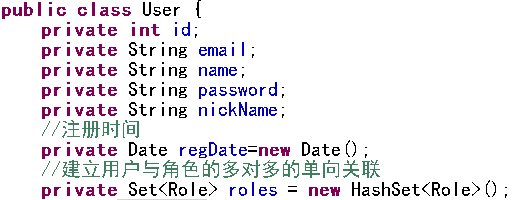


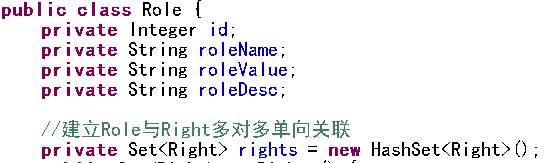


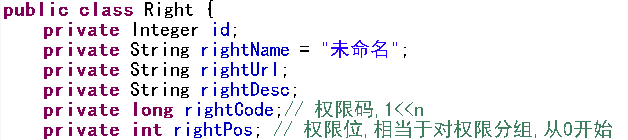
#### 准备工作

1. 建立角色表Role和权限表Right，并建立关系

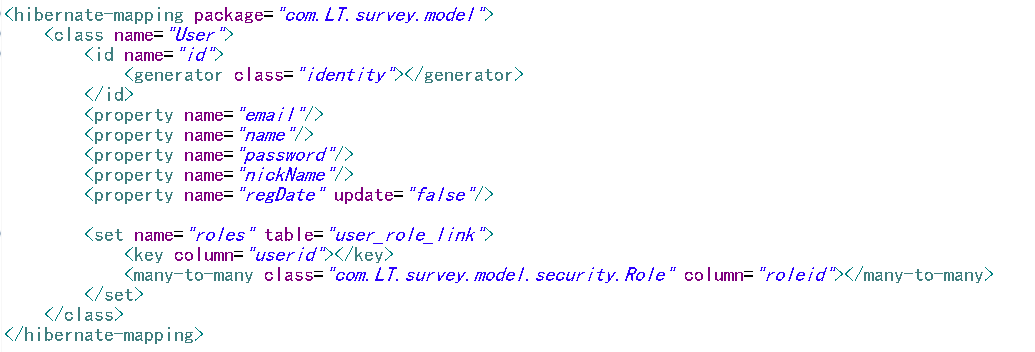
权限管理:三者之间均为多对多关联,这里做的是单向关联.

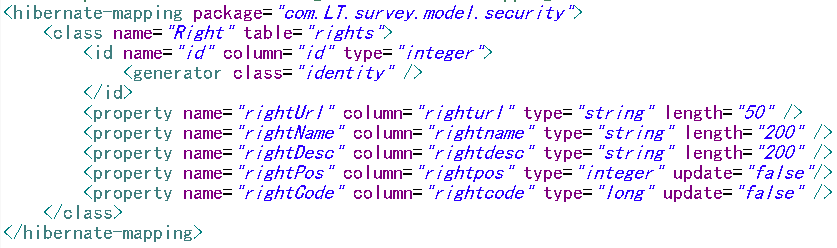
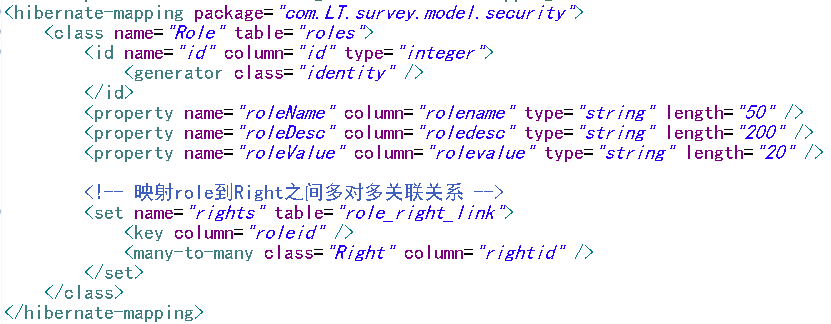






对应的hbm文件



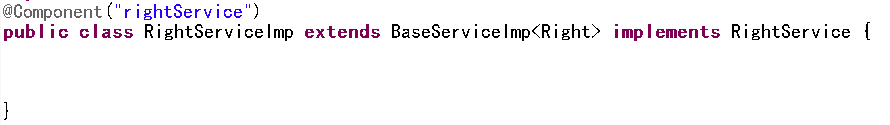
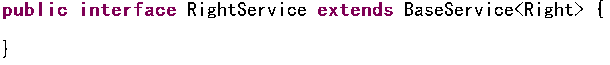


1. 建立角色和权限的dao实现类





1. 建立角色和权限的service接口和实现类



#### 权限管理

1.导航栏--->权限管理

2.RightAction.findAllRights();

public String findAllRights(){

this.rights = rightService.findAllEntities();

return "rightListPage" ;

}

3.struts.xml

<!-- RightAction -->

<action name="RightAction\_\*" class="rightAction" method="{1}">

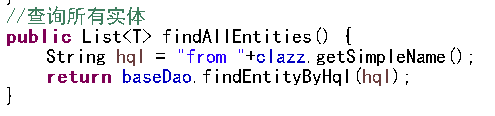
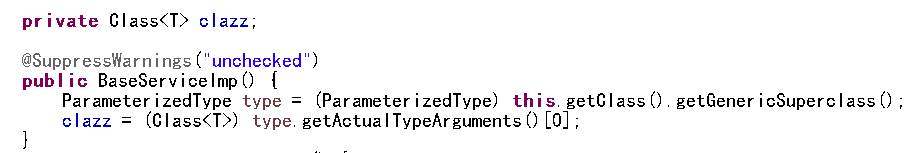
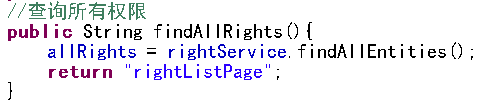
<result name="rightListPage">/rightList.jsp</result>

</action>

4.跳转到

[略]

经过分析方法 findAllEntities (得到所有实体)在所有service中都可能会被用到，这里向上抽取在baseService中定义方法，baseServiceimp中实现该方法



#### 添加权限

---------------------

1.权限列表-->添加权限

2.RightAction.toAddRightPage();

public String toAddRightPage(){

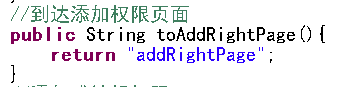
return "addRightPage" ;

}

3.struts.xml

<result name="addRightPage">/editRight.jsp</result>

1. 略



#### 保存更新权限

------------------

1.添加权限页面-->提交

2.RightAction.saveOrUpdateRight()

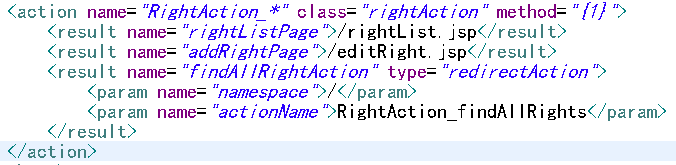
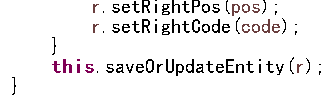
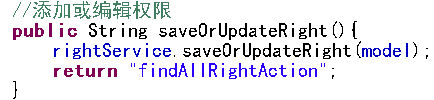
rightService.saveOrUpdateRight(model);

return "findAllRightAction" ;

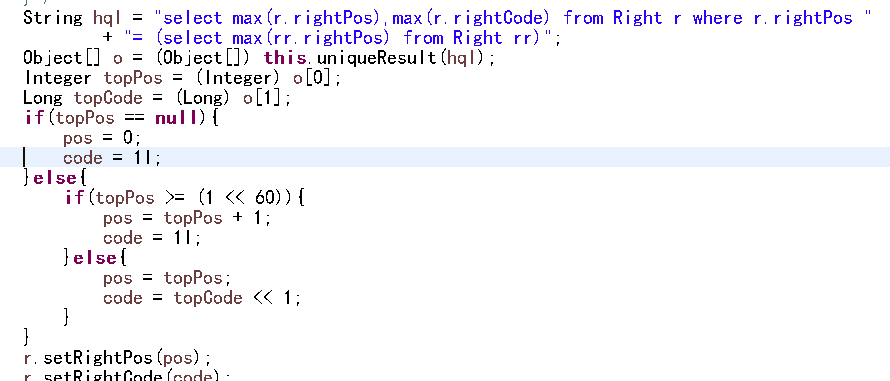
}

3.

4.

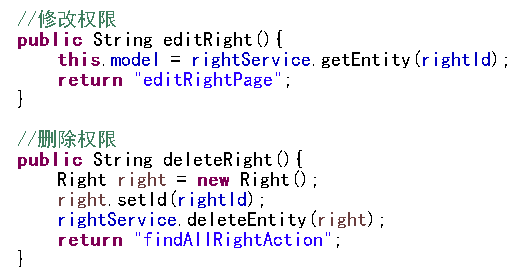


上述对于保存权限的方法存在一个问题，要是权限过多，那么系统就必须一个一个数据扫描过去，在找出，比较麻烦，所以可以利用数据库的聚集函数搞定



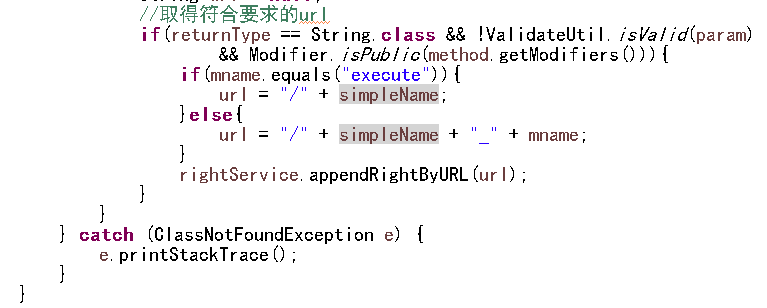
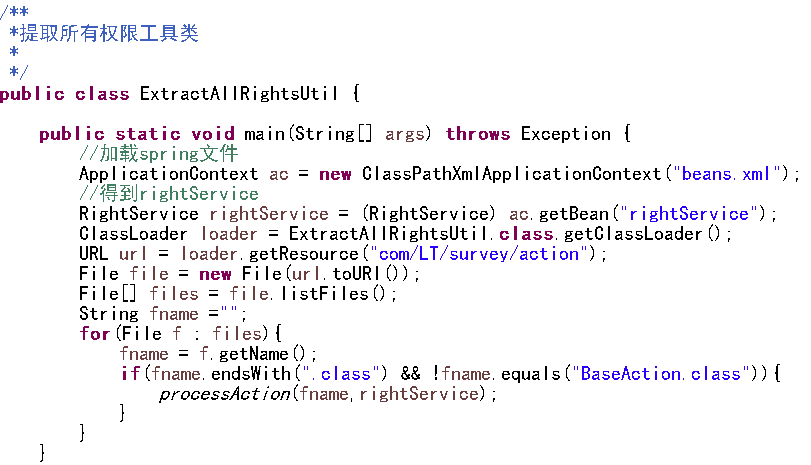
#### 修改和删除权限

利用已经编写的rightService方法中的getEntity方法和deleteEntity方法，实现对权限的修改和删除



#### 通过反射获取所有权限

由于权限较多，而且我们的权限(url)是以固定的形式展现的，因此我们可以利用java的反射机制获得所有的权限



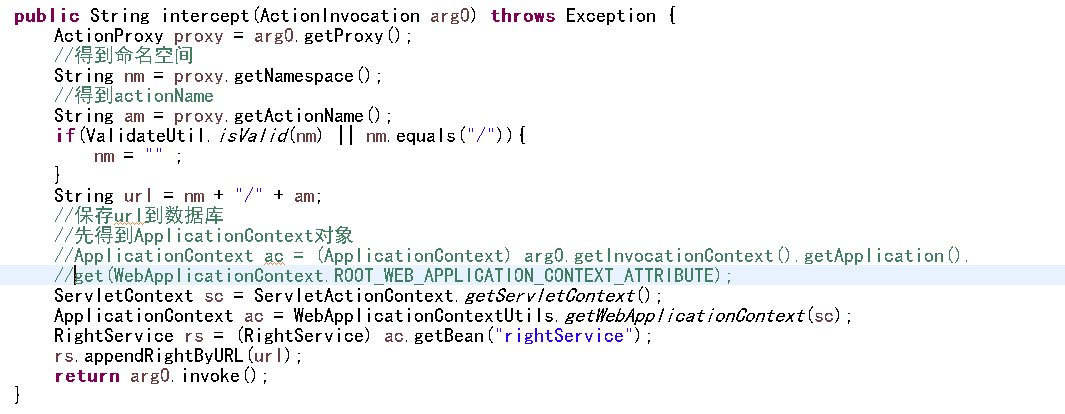
#### 通过拦截器获取所有权限

在某些情况下，我们的url会不希望别人知道，所以url只是一堆无用的东西，这时通过url无法获取，并且还存在一点不好就是，当我们添加新的权限时，必须要在运行一下获取权限的方法，很麻烦，这时我们可以考虑用拦截器获取所有的拦截器，当我们添加新的权限时，通过拦截器会自动捕获所要访问的权限

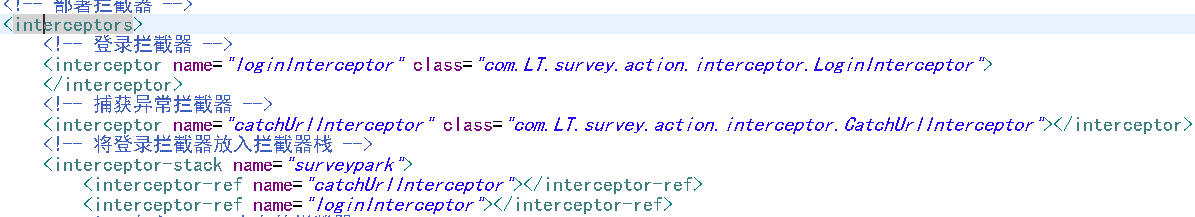
1. 定义拦截器



1. 编写拦截器的方法



1. 部署拦截器



#### 批量修改权限

1. 将要修改的权限在jsp页面内name改为对应的格式



2.创建批量修改权限的方法

### 角色管理

#### 角色列表

1.导航栏-->角色管理

2.RoleAction.findAllRoles()

public String findAllRoles(){

this.allRoles = roleService.findAllEntities();

return "roleListPage" ;

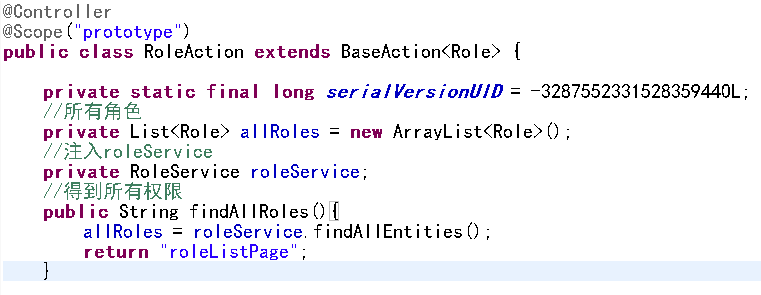
}

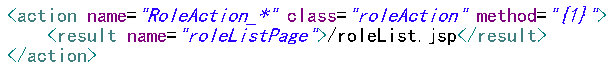
3.struts.xml

<action name="RoleAction\_\*" class="roleAction" method="{1}">

<result name="roleListPage">/roleList.jsp</result>

</action>





#### 添加角色

--------------

1.角色列表-->添加角色

2.RoleAction.toAddRolePage();

public String toAddRolePage(){

this.noOwnRights = rightService.findAllEntities();

return "addRolePage" ;

}

3.struts.xml

4.跳转到/editRole.jsp

<s:form>

<s:textfield name="roleName|roleValue|roleDesc" />

....

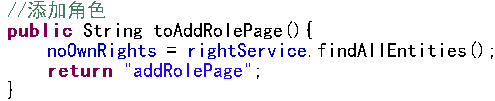
<!-- 左侧列表:Role.rights集合绑定---角色拥有的权限列表 -->

<s:select id="left" list="rights" listKey="id" listValue="rightName" multiple="true"/>

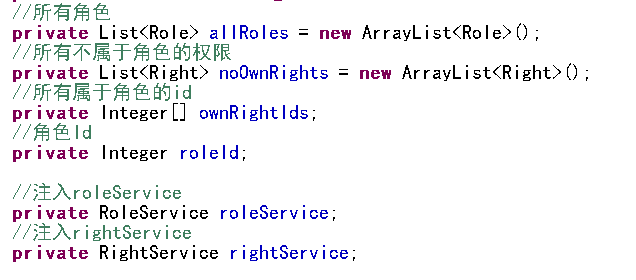
<!-- 右侧列表:同RoleAction.noOwnRights集合绑定---角色没有的权限列表 -->

<s:select id="right" list="noOwnRights" listKey="id" listValue="rightName" multiple="true"/>

</s:form>



#### 保存/更新角色



--------添加角色时，要从所有的权限中选择相应的权限添加给要保存的角色

1.添加页面-->提交

2.RoleAction.saveOrUpdateRole();

public String saveorUpdateRole(){

model + ownRightIds(Integer[])

roleService.saveOrUpdateRole(model,ownRightIds);

public void saveOrUpdateRole(Role r ,Integer[] ids){

if(!ids有效?){

r.getRights().clear();

}

else{

//

List<Right> list = rightService.findRightsInRange(ids);

r.setRights(list);

}

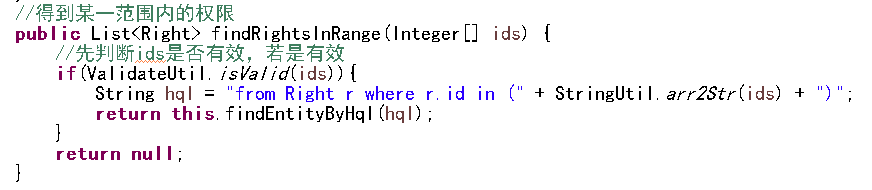
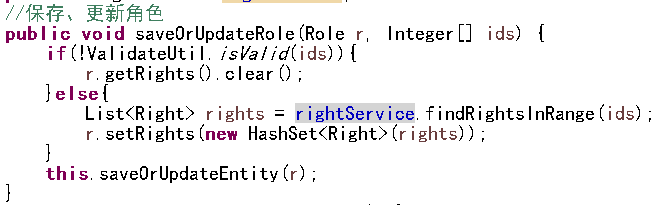
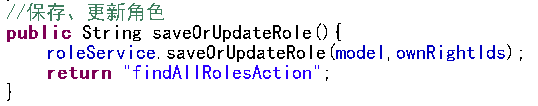
this.saveOrUpdateEntity(r);

}

return "findAllRolesAction" ;

}

3.struts.xml



#### 修改权限

对于修改权限，主要是对于到达编辑页面后对于角色已有的权限和没有的权限的回显，

对于已有的权限取出， 需要注意懒加载问题，由于一旦有关角色都要取出权限，所以这里选择关闭懒加载

对于没有的权限，这里用的是一个not in 将已有集合权限通过一个方法，将其转换成由id和‘,’分割的字符串

--------------------

1.角色列表-->修改

2.RoleAction.editRole()

public String editRole(){

roleId

this.model = roleService.getEntity(roleId);

this.noOwnRights = rightService.findRightsNotInRange(model.getRights());

public List<Right> findRightsNotInRange(Set<Right> rights){

if(!rights有效?){

retutn this.findAllEntities();

}

else{

String hql = "from Right r where r.id not in ("+extractRightIds(rights)+")" ;

return this.findEntitiesByHQL(hql);

}

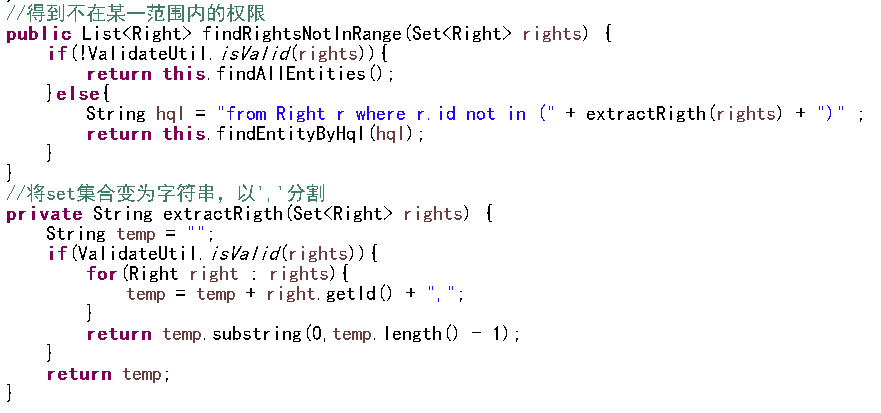
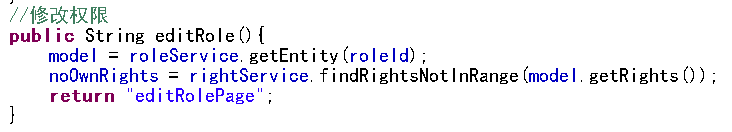
}

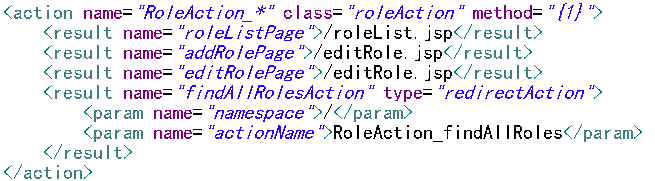
return "editRolePage" ;

}

3.struts.xml

<result name="editRolePage">/editRole.jsp</result>





#### 删除角色

-----------------

1.角色列表-->删除

2.RoleAction.deleteRole();

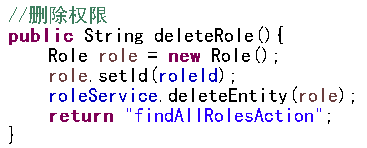
public String deleteRole(){

roleId

roleService.deleteEntity(new Role(roleId));

return "findAllRolesAction" ;

}



### 用户授权

#### 用户授权

到达用户授权页面，获取所有用户即可

------------------

1.导航栏-->用户授权

2.UserAuthorizeAction.findAllUsers()

public String findAllUsers(){

this.allUsers = userService.findAllEntities();

return "userAuthorizeListPage" ;

}

3.struts.xml

<action name="UserAuthorizeAction\_\*" class="userAuthorizeAction" method="{1}">

<result name="userAuthorizeList">/userAuthorizeList.jsp</result>

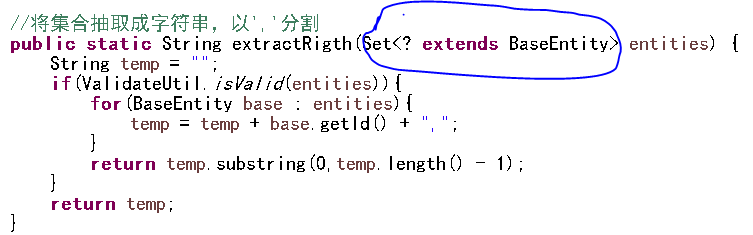
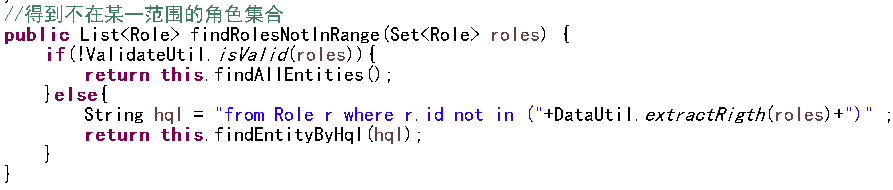
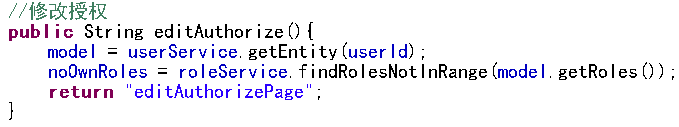
</action>

1. 跳转到/.jsp



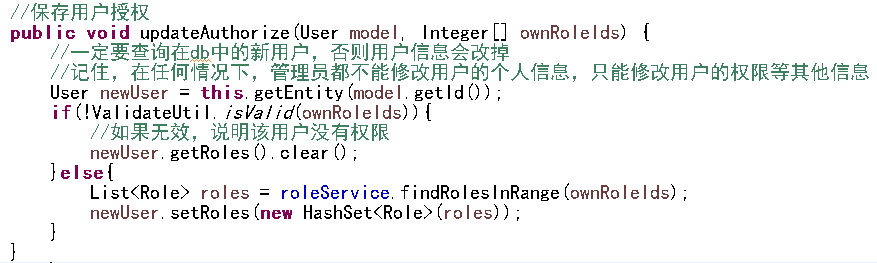
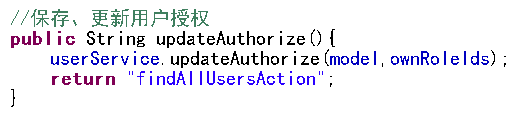
#### 修改授权

到达修改授权页面，和角色管理中的修改权限类似，由于都需要用到一个将集合抽取成字符串，那么可以这样将其封装成一个工具类，但是问题出现了集合中的泛型是不一样的，如果写Object，那么每次都必须判断，很麻烦，为了解决这个问题，将所有的实体向上抽取，形成一个BaseEntity实体类,让所有实体类继承该类,然后采用Ser<? Extends BaseEntity>这种形式，便可解决



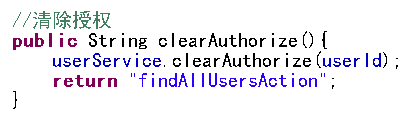
#### 保存、更新授权

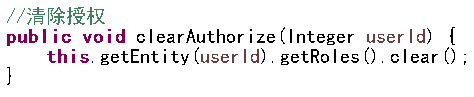
在保存和更新授权时，要一直记住，无论在任何情况下，管理员都不可以修改用户的个人信息，只能修改权限，因此一定要取出新的user来进行权限的修改



#### 清除授权

得到user，将它的roles全部清空







### 权限过滤(粗粒度)

权限控制分为粗细两种粒度:

-------------------------

粗粒度:拦截器,安全的.

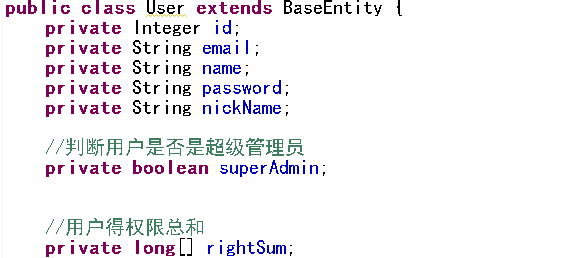
细粒度:使用标签覆盖实现.

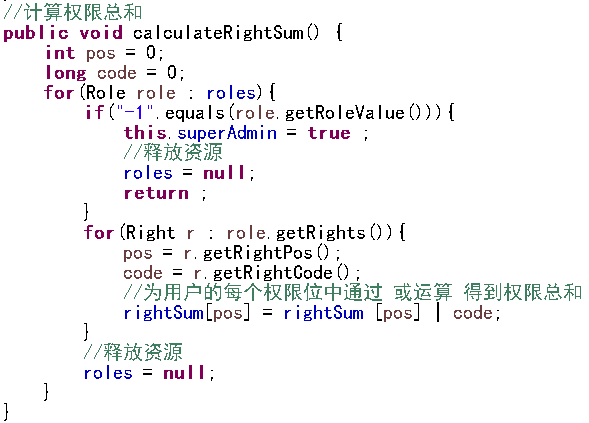
从此步开始便开始进行粗粒度权限的过滤，在进行权限过滤之前需要做些准备工作

#### 权限总和

在用户登录验证时，计算出该用户的权限总和，用来进行过滤的前提。首先在user中设置权限总和数组rightSum，及计算权限总和的方法，在loginAction中的登录验证时计算

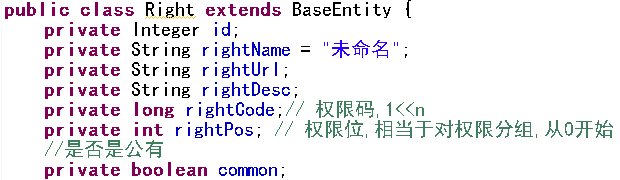




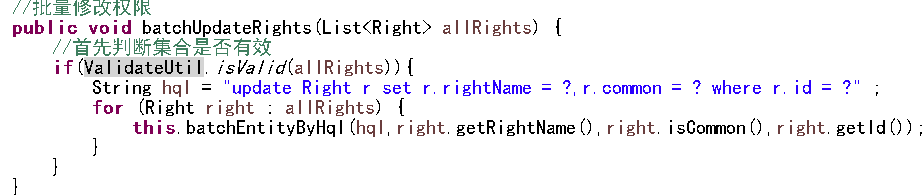


#### 公共资源

用户在访问网站时，看是否访问的是公共资源，如果是公共资源，那么用户不用登陆也可，不是公共资源在进行其他判断。具体实现是在资源实体中添加一个字段common，用以标识是不是公共资源，然后通过批量修改资源时，可以修改是否是公共资源





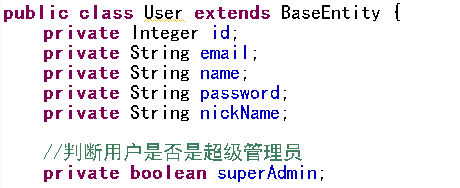


#### 超级管理员

在权限过滤时，如果该用户拥有超级管理员的角色，那么一切皆可访问

具体实现是在user中添加superAdmin字段用来标识是否具有超级管理员权限

然后重写计算权限总和方法，因为若是超级管理员，那么便不需要计算权限总和



#### 权限过滤

带有超级管理员的权限过滤流程，将登陆拦截器进行修改

------------------------------

1.用户点击链接--->拦截中

2.url --查询--> Right

3.对Right对象进行判断

if(公共资源?){

//放行

}

else{

if(登陆?){

if(超级管理员?){

//放行

}

else{

if(有权限?){

//放行

}

else{

//错误页

}

}

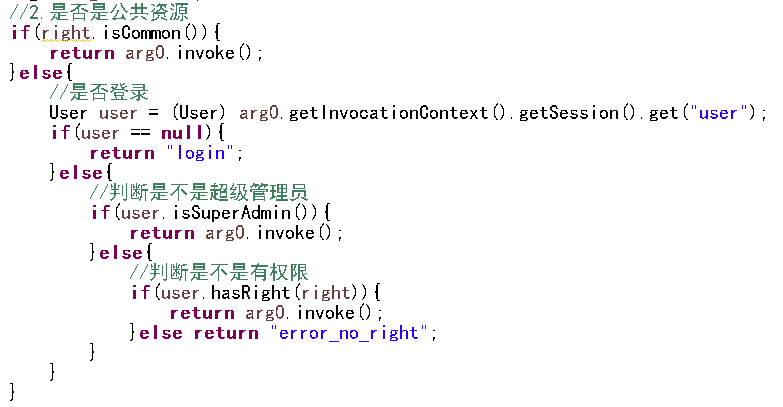
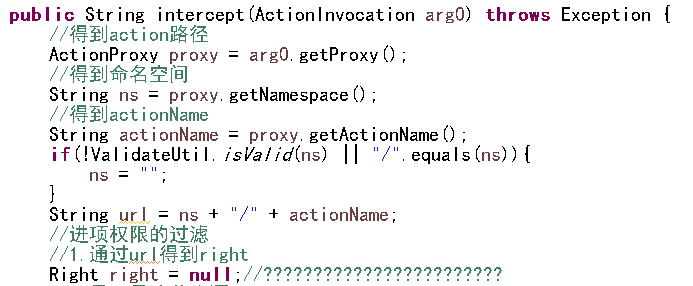
}

else{

//去登陆页面

}

}

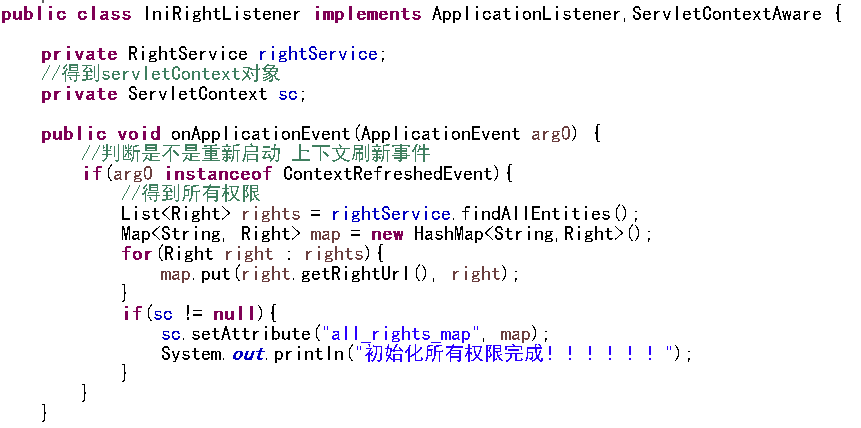


#### 初始化所有权限

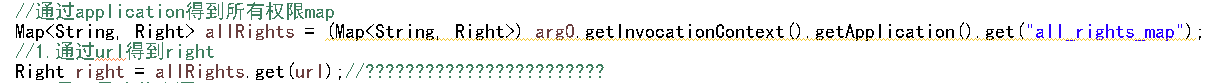
由于在每一次调用action时都会启用拦截器，如果在拦截器中查询所有的权限使用的是连接数据库的方法，会使得系统性能极大地下降，为了解决这种情况，可以想办法当启动系统时就一次性把所有权限都初始化，放入application中，然后从application中取出对应权限。

在application中有一个方法publishEvent，它的作用就是发布事件。对应的是在spring中有一个监听器ApplicationListener，可以监听到application所发布的事件，只要在对应的事件中获得所有权限，并放入application即可。

使用spring监听器实现容器初始化完成后,立刻将所有权限查询出来并放置到application.



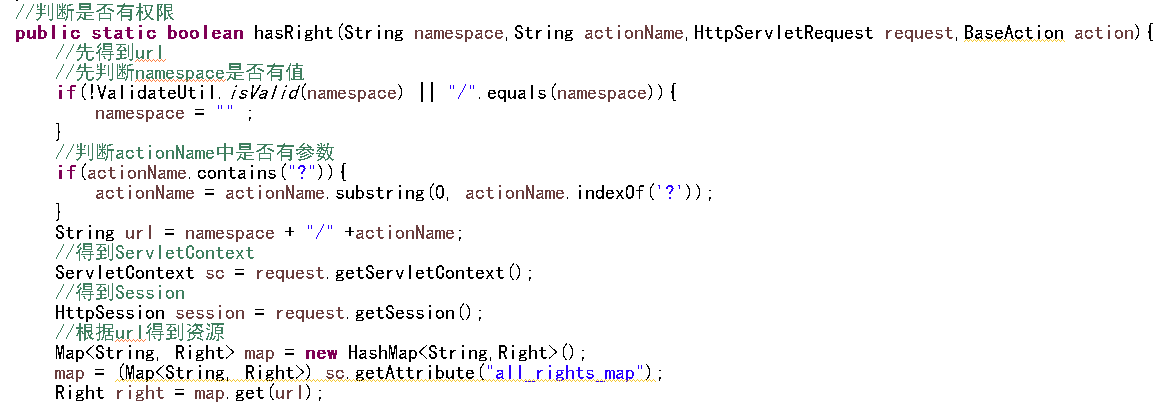
然后拦截器获得权限从application中



### 权限过滤(细粒度)

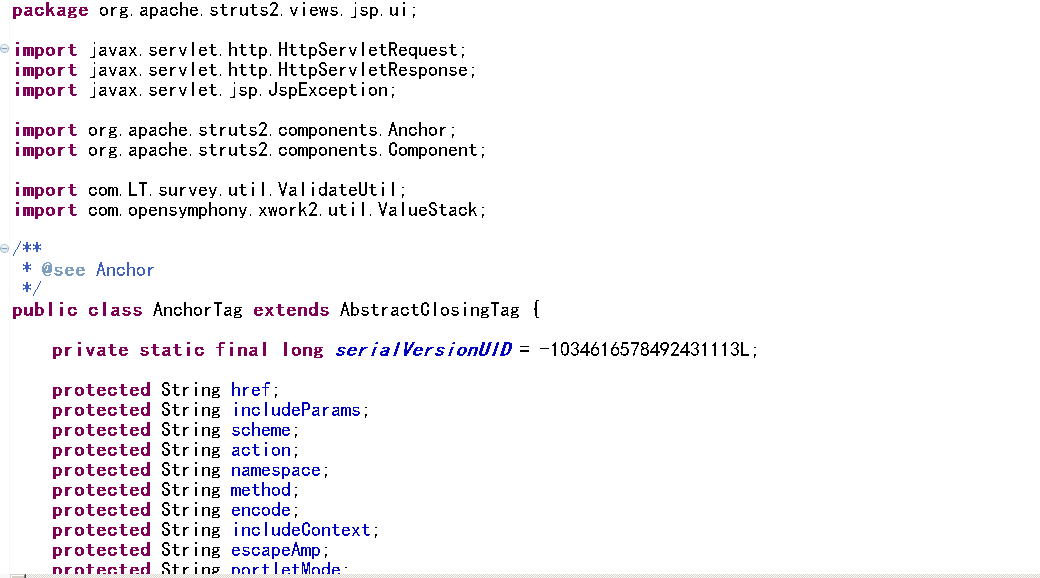
要实现细粒度的权限过滤，由于我们使用的标签是struts2的标签，所以我们要重写struts2的标签控制类，只要在其中加上权限的控制，判断是否有权限，如果有，标签显现出来，没有则不显现

1. 将判断权限的方法向上抽取成一个工具类



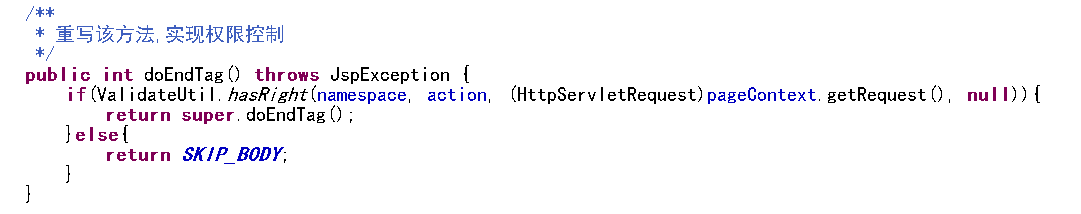


1. 覆盖struts2中原有的标签类

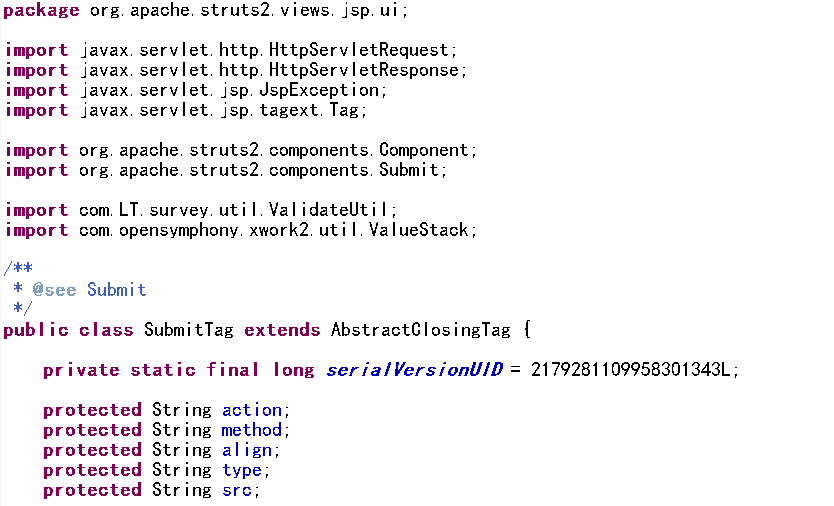


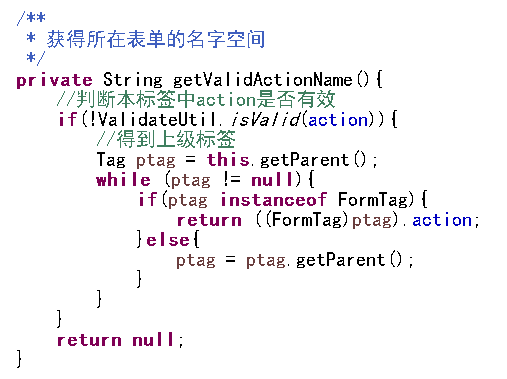
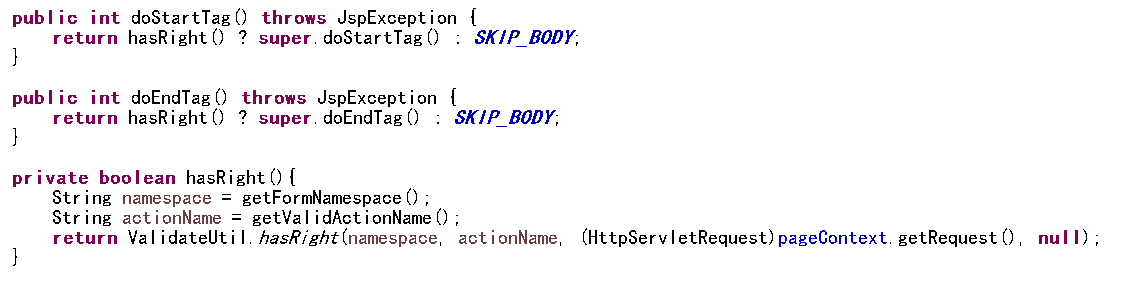
.......

类中重写doEndTag方法



1. 覆盖struts2中提交按钮类





### 日志管理(记录日志)

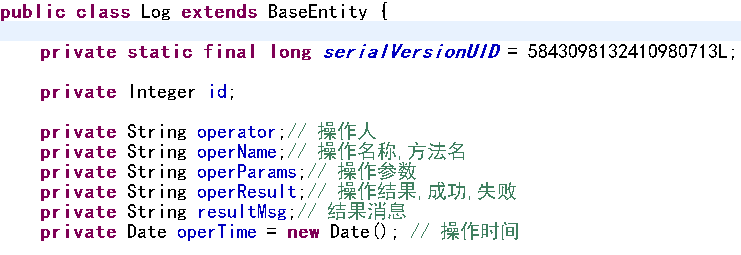
使用spring中的aop来完成日志的管理

aop: 前置 后置 环绕 引入 异常

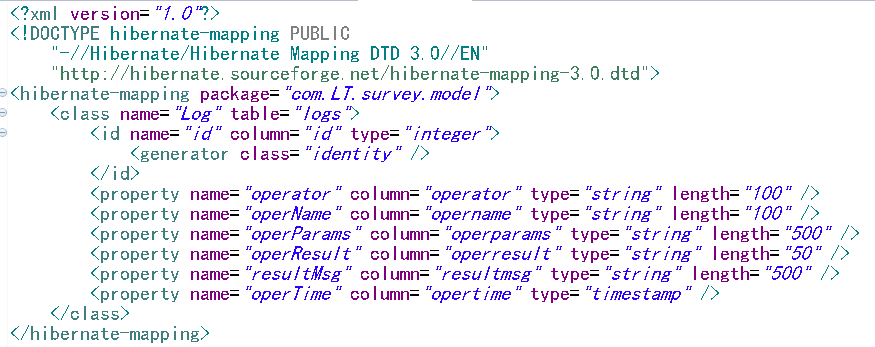
我们用 环绕 进行日志的管理

#### 准备工作

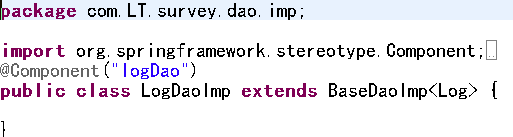
1. 添加log日志类

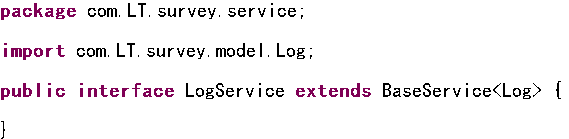


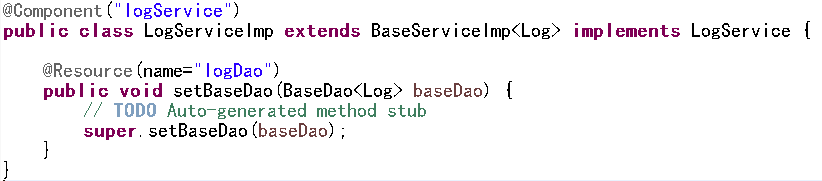
1. 建立映射文件



1. 建立对应的dao实现类和service及实现类





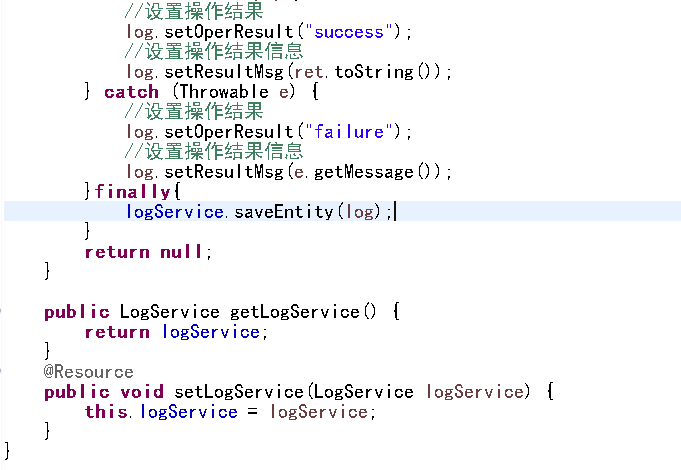


#### 建立日志记录仪

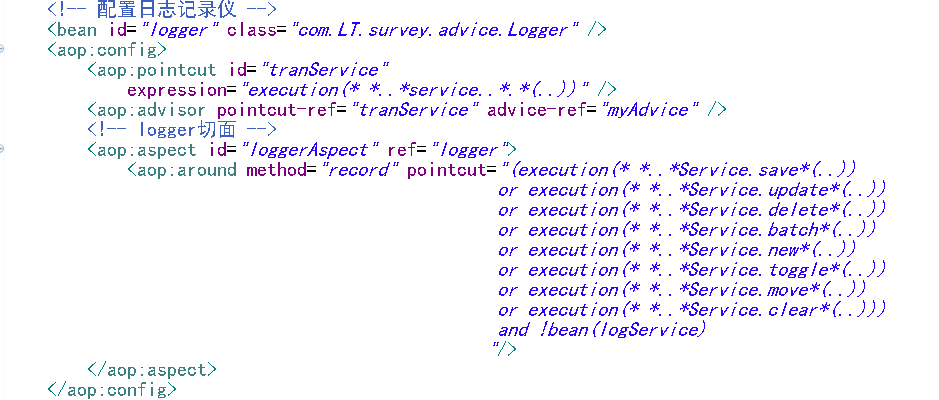
此次日志管理应用的是aop利用它的pojos + xml来开发环绕通知

Execution(\* \*.. \*Service .save\* (..))

匹配返回值 任何包及其子包 以Service结尾的类 以save开始的方法 参数不限



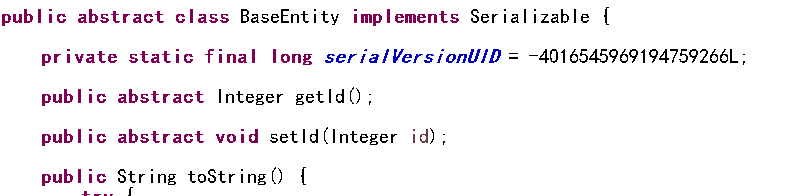
#### 配置文件编写

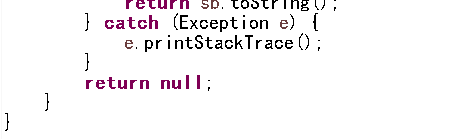
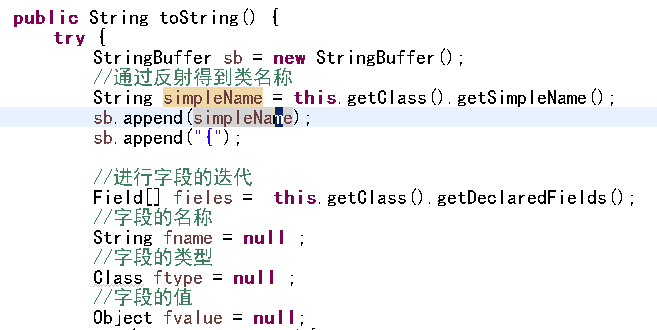


#### 重写BaseEntity的toString方法

为了让存入数据库中的日志数据能够更加容易看懂，需要重写BaseEntity的toString方法

存入数据库的格式如下 User{id:1,email:xupccc@hotmail.com,}





### 日志管理(查看日志)

#### 页面查看

1.aop环绕通知

2.导航栏-->日志管理

3.LogAction.findAllLogs()

public String findAllLogs(){

this.allLogs = logService.findAllEntities();

return "logListPage" ;

}

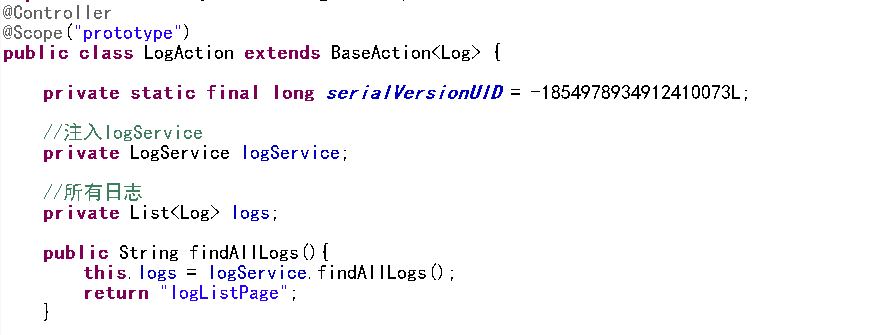
4.struts.xml

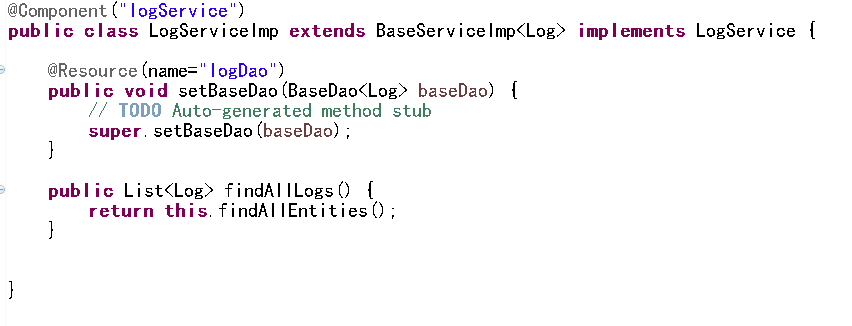
<action name="LogAction\_\*" class="logAction" method="{1}">

<param name="logListPage">/logList.jsp</param>

</action>

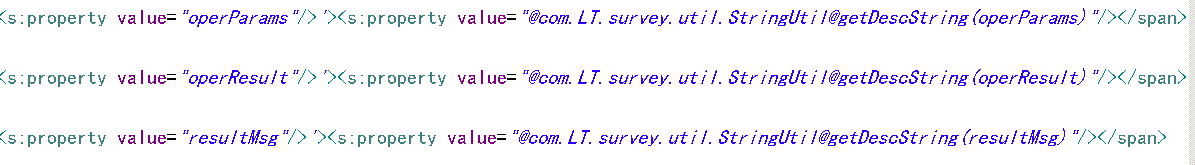
1. 跳转到/logList.jsp







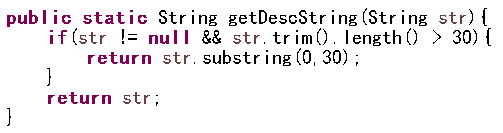
对于参数和返回结果比较长的，先在工具类StringUtil中定义一个静态方法对结果进行处理，jsp页面中在调用对应的静态方法，调用过程是



由于struts2中对于静态方法的调用默认是禁止的，需要手动打开



对应的静态处理方法



#### 石英调度动态创建日志表

对于 日志 来说，由于日志的产生会很多，如果只有一个表，会很不方便，造成大数据

为了解决这种问题，一般有两种方法

1.分表

动态表:logs\_2013\_9 logs\_2013\_10 调度

1. 分库

使用spring的quartz(石英)调度,动态生成日志表

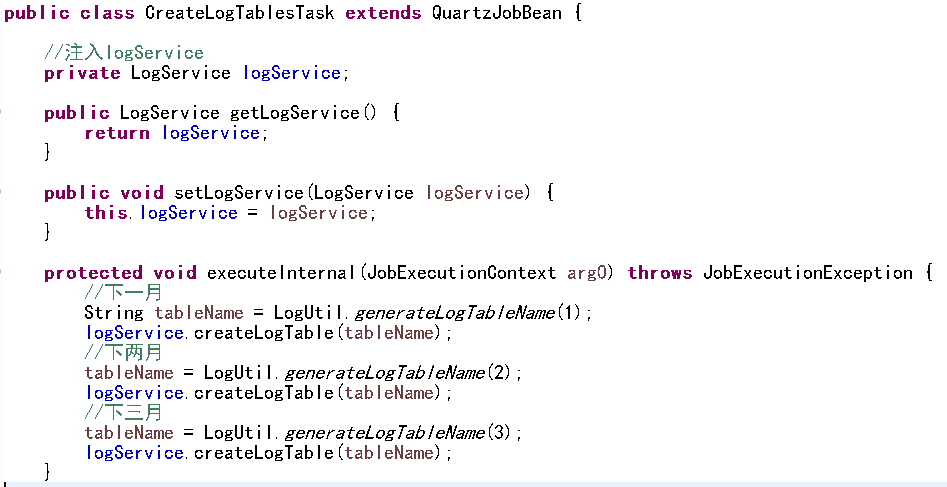
---------------------------------------

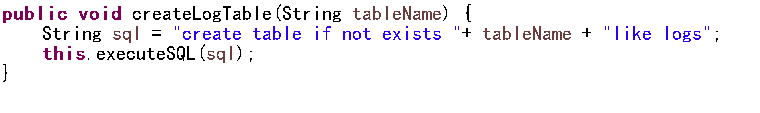
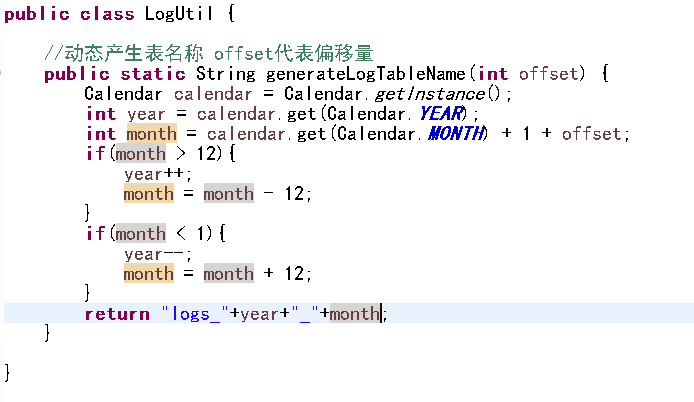
1.引入quartz类库

com.springsource.org.quartz-1.6.2.jar

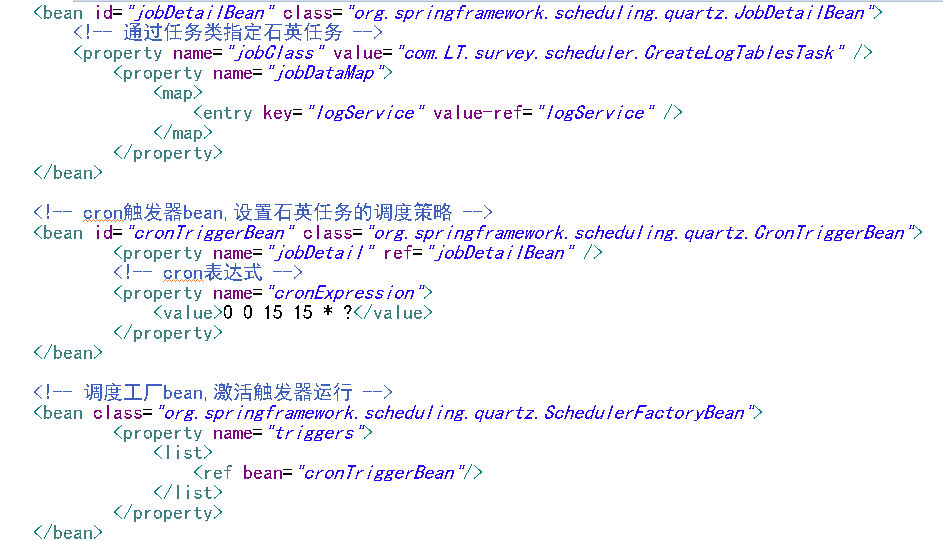
1. 创建动态生成日志表的任务.

为了不容易出现问题，我们每个月一次生成后三个月的表

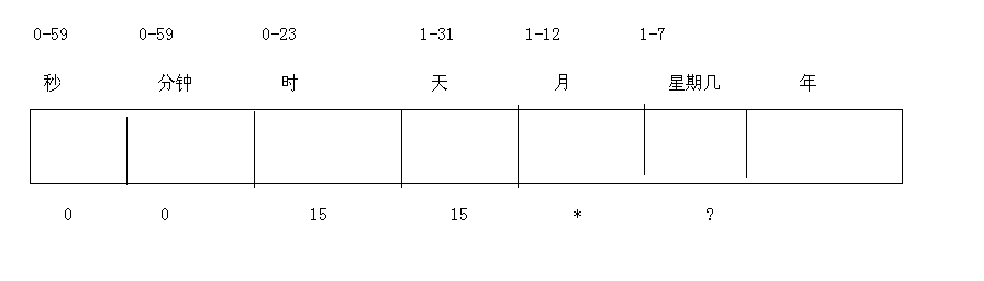




#### [配置石英任务---schedules.xml]

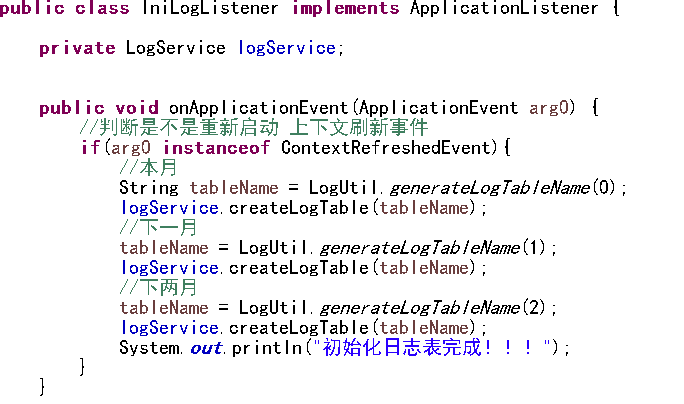


其中触发器表达式中意思



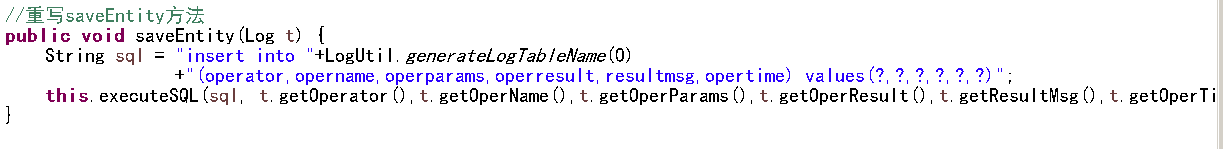
#### 初始化日志表

为了保证当系统启动时，已经存在日志表，不会存在无表可用的情况，直接在系统启动时初始化日志表完成



#### 保存日志到对应表

我们保存日志时，应该把日志保存到对应的表中，因此我们要重写logService中的saveEntity方法

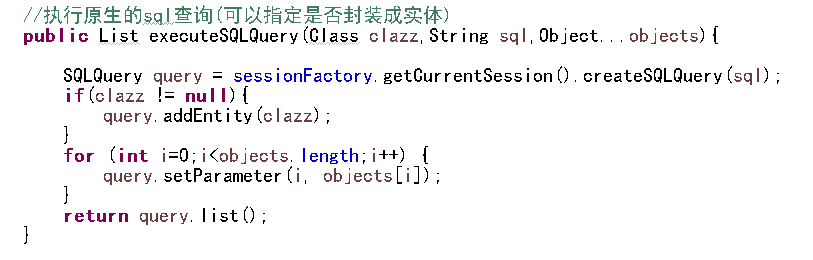
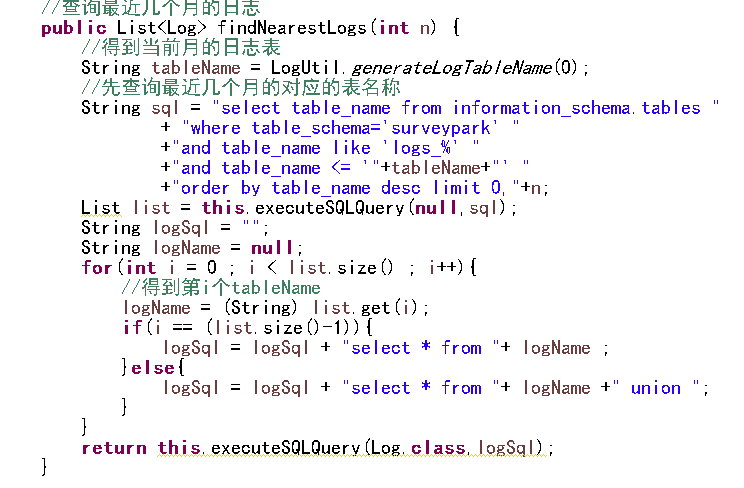


### 日志查询

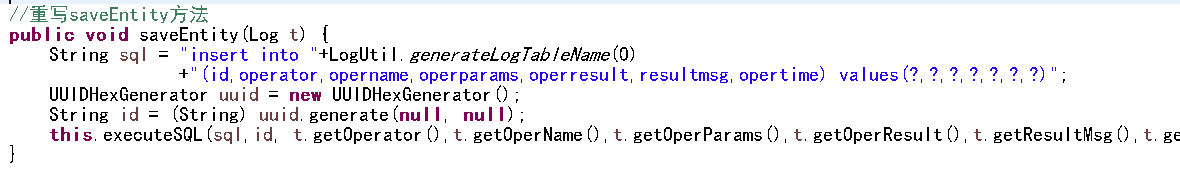
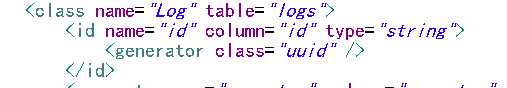
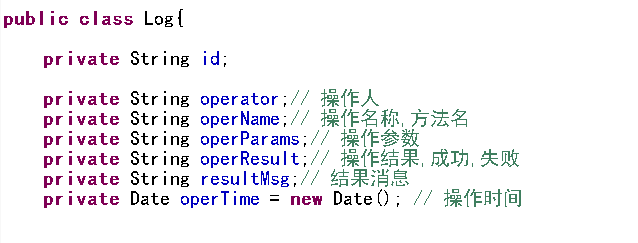
从主页面点击日志管理，查看的不是全部，而是最近3个月的日志，这就需要重新改一下LogAction，而且查询最近三个月的日志表，有可能日志表并不存在，这就要用到

查询最近月份的日志表,从当前月份开始往前推算

通过元数据(information\_schema数据库)信息查询表是否存在.，最后用union联合表查询,由于这是用sql语句操作，从数据库查询出来的内容hibernate不会将其封装成一个对象，故应该自己在dao层中执行sql语句查询时封装



当这写完之后，会发现出现一个问题，就是同一月的会出现好几次，这是因为，我们的日志表都是用的自增主键，而hibernate查询时，是根据主键查询，如果两张表的某两个记录的主键一样，它就会自动用查到的第一个主键赋值，为了解决这种问题，我们用uuid主键



### 答案分布式数据库存储

由于答案有很多的数据，如果所有答案都存在一个数据库一张表里，那将会特别影响系统的性能，对于这种 ’大数据’ 问题，之前提到过，可以使用 分表 分库 这里我们使用分库

分库

水平分库

将数据库整体结构复制，形成两个数据库或者更多数据库，它们的特点是结构相同，可能会跨库查询，但是要尽量避免跨库查询

竖直分库

将一个系统按模块分数据库，不同的模块在不同的数据库，特点是数据库之间的结构不相同，不会跨库查询

这里的答案存储，我们采用两个数据库，一个主数据库，一个从数据库，从数据库中只有一个answer表，表结构和主数据库的answer相同，当调查的id是 奇数 时，答案存入主数据库，id是 偶数 时，答案存入从数据库。(接近于竖直)

#### 创建从数据库及其answer表

数据库 Surveypark\_1 表 answer

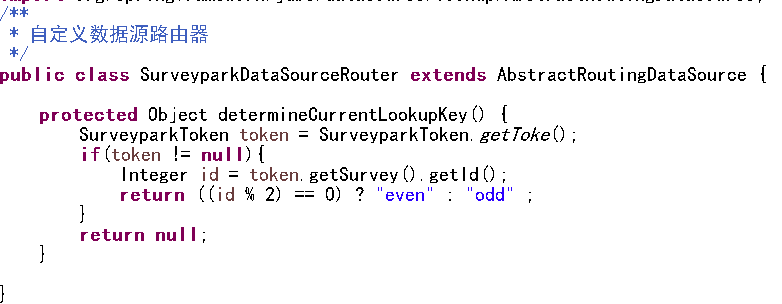
#### 2.配置数据源

在beans.xml中配置两个数据源，采用自定义数据源路由器的方法，进行具体的分库

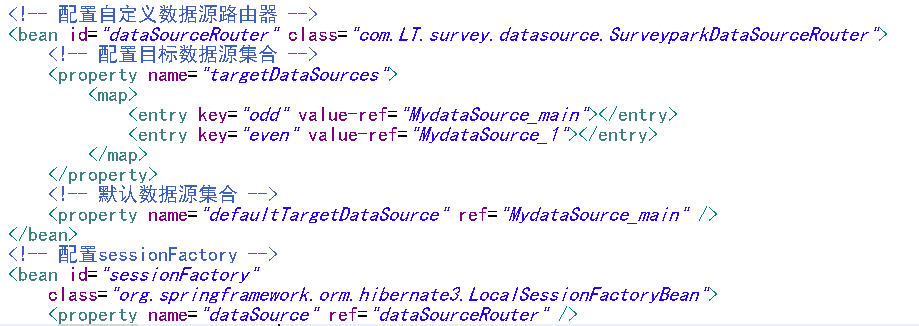


#### 自定义数据源路由器并配置

在spring中已经定义了数据源路由器abstractRoutingDatasource,我们需要自定义一个数据源路由器，通过继承abstractRoutingDatasource，在其中实现分库

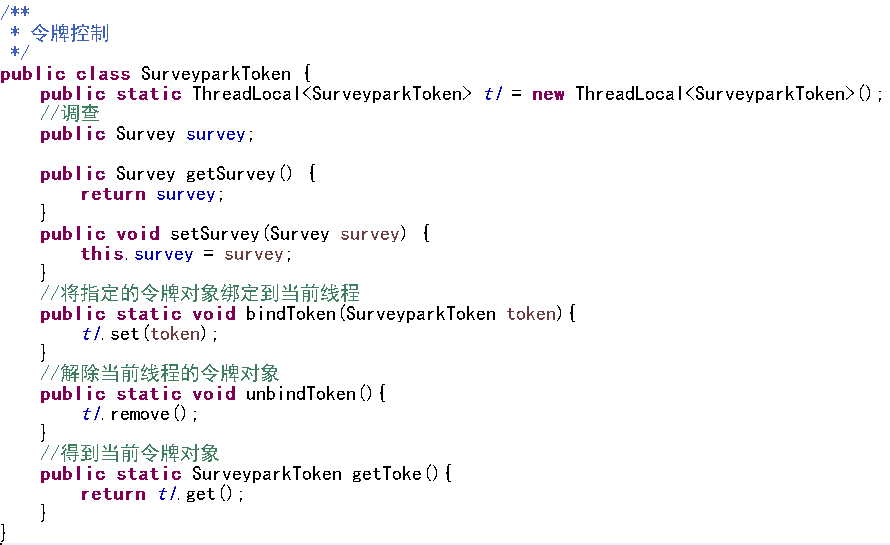


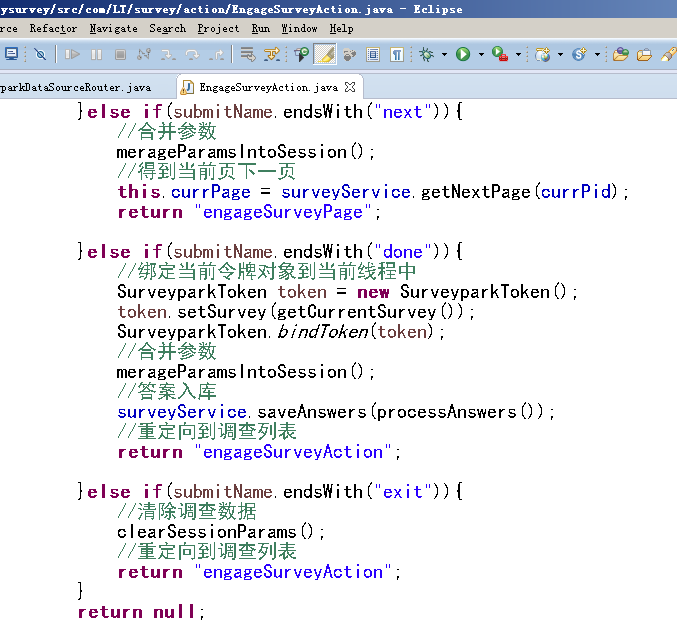
在beans.xml中配置自定义数据源路由器,并修改LocalSessionFactoryBean

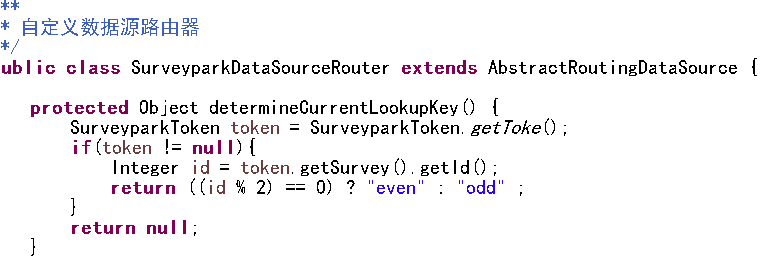


#### ThreadLocal技术实现token机制

实现分库管理时还存在一个问题，就是线程不安全，比如当一个用户参与调查时是 偶数 然而在未写入时 另一个用户参与调查是 奇数，刚好改写该值，就会出现问题，因此我们采用ssh中的ThreadLocal技术实现token机制，将对应的事件绑定到对应的线程上，实现线程安全

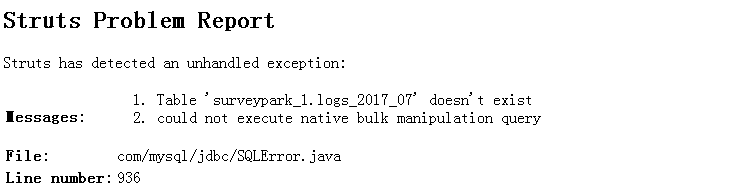




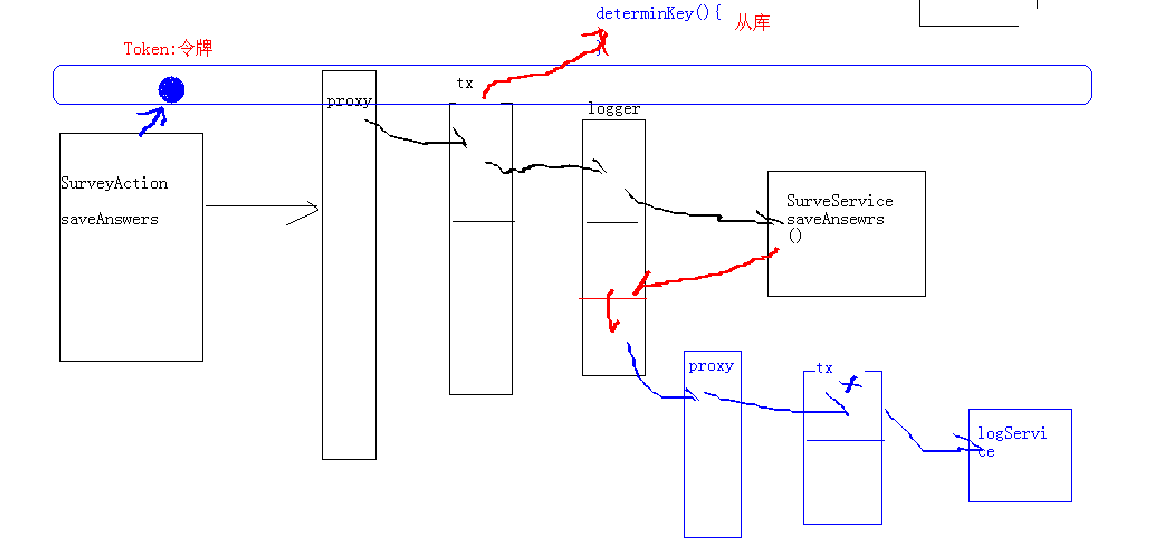


#### 问题解决-日志

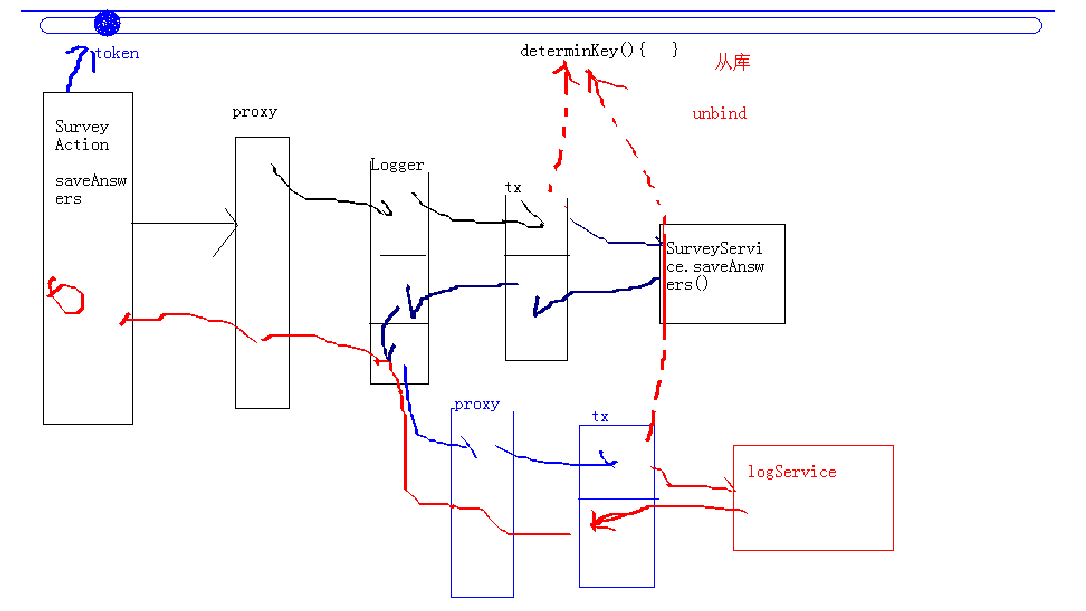
完成上述操作后，进行测试时，发现出现了一个问题



这种问题出现的原因是在保存答案的过程中已经开启了事务，且由于线程已经绑定了令牌，根据令牌对象检测出了要保存在 从库 中，当保存日志信息时，由于事务并没有关闭，所以此时默认保存日志也是在从库，然而从库并没有对应的日志表，所以出现了该问题，具体流程如下图

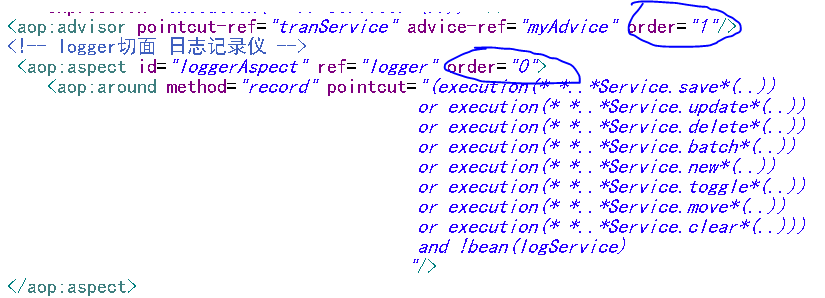


如果要解决这种问题，首先要做的是将事务开启和日志记录仪顺序相反，这样当日志保存时，事务已经关闭，可以重新检测线程中令牌，开启新的事务，然后解除令牌绑定，当第一次开启判断令牌后，就已经可以解除令牌，这样当保存日志的事务开始时，直接到达主库，如下图



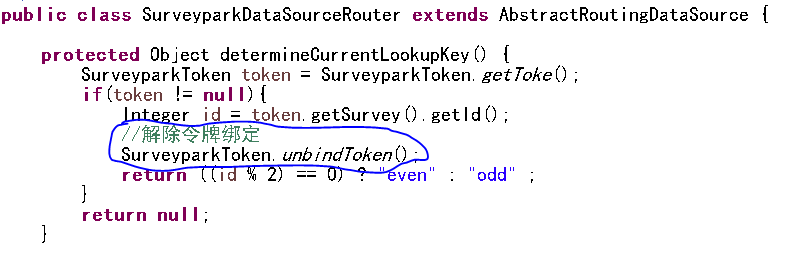
具体操作

1. beans.xml配置，修改事务和日志记录仪执行顺序



Order值越大，优先级越低

1. 解除绑定



### 使用缓存对service，以提高性能

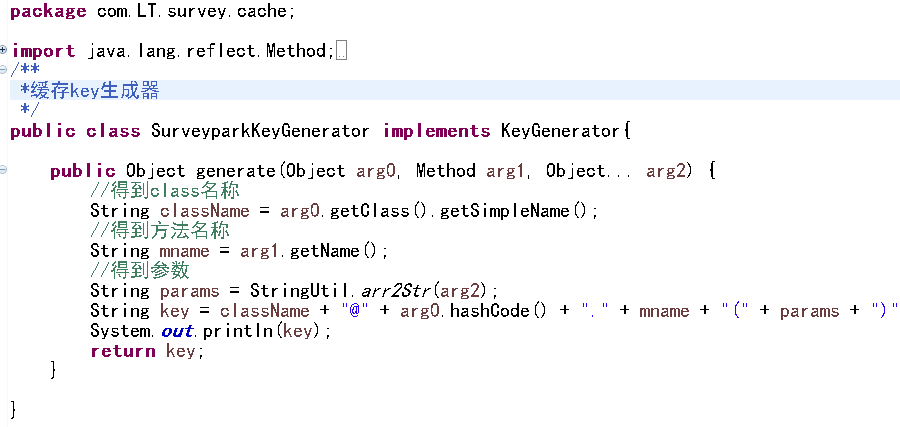
1.引入缓存供应商--ehcache.类库

com.springsource.net.sf.ehcache-1.6.2.jar

2.配置ehcache缓存的过期策略



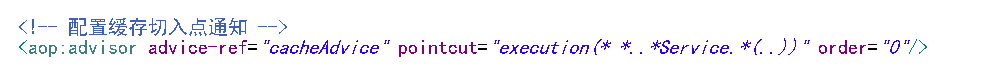
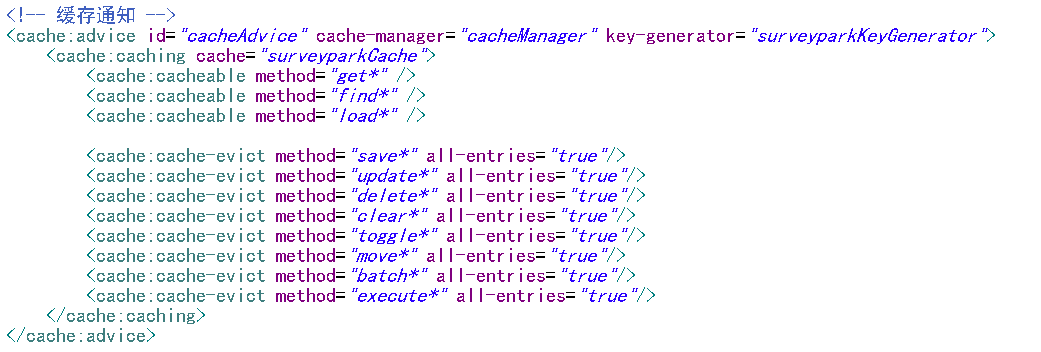
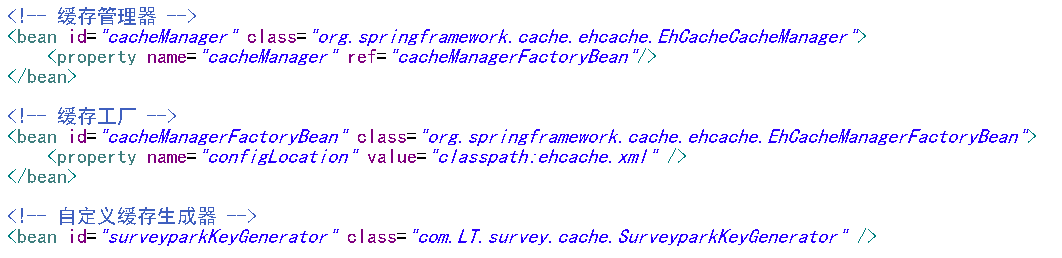
1. 自定义缓存key生成器



1. beans.xml中配置缓存管理 缓存工厂 自定义缓存key生成器 缓存建议 缓存切入点

读数据，写入缓存

写数据，清除缓存



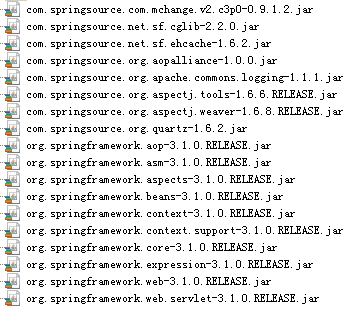
缓存的优先级相对于事务和日志是最高的

### 远程调用

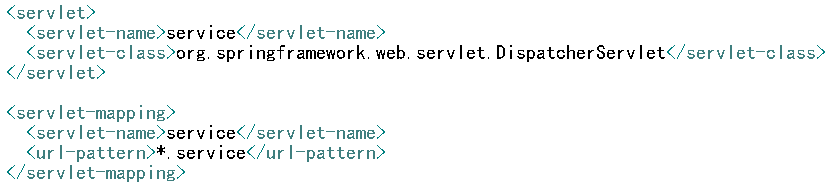
利用spring中的servlet中的Dispatcher来进行远程调用

**服务端**

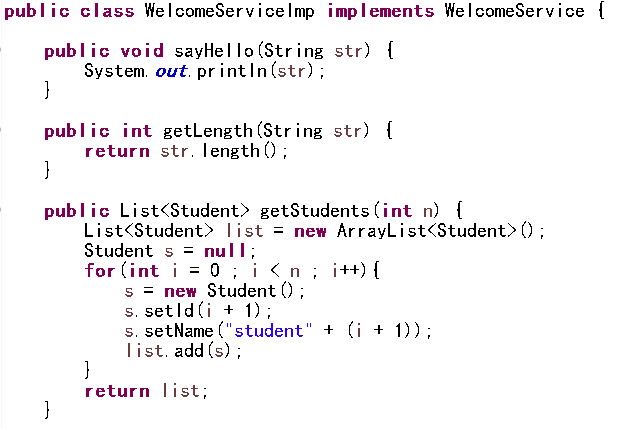
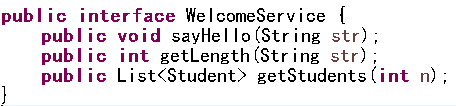
1. 导入jar包



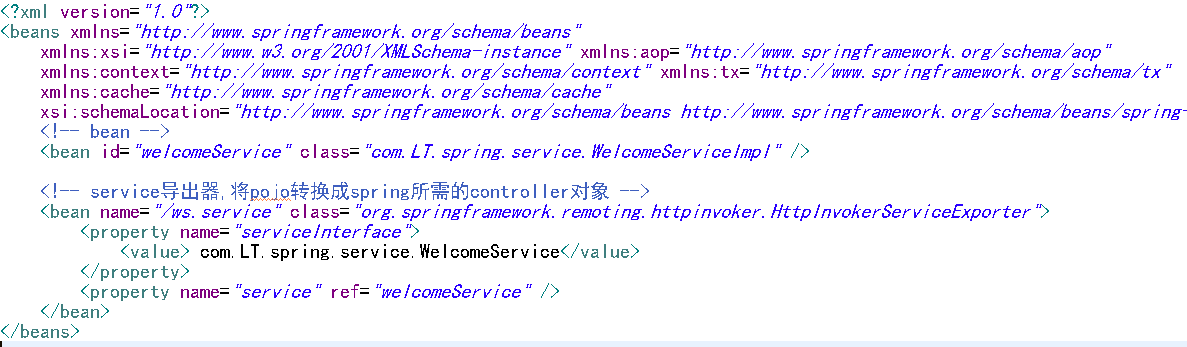
1. Web.xml中定义Dispatcher



1. 定义对应的接口及其实现类

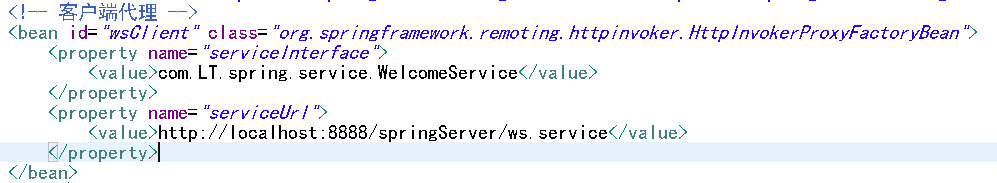


在远程调用时，实体类一定要实现serializable接口，并且在客户端和服务端的结构最好一样，尤其是***serialVersionUID*** 必须一样

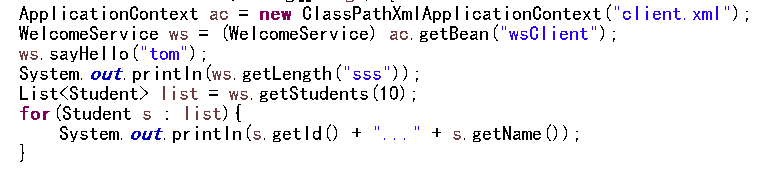
1. 配置service-servlet.xml  
   

**客户端**

1. 创建java project，导入jar包
2. 配置client.xml

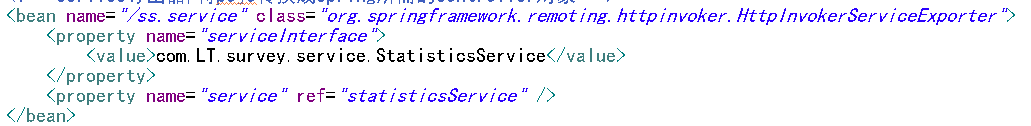
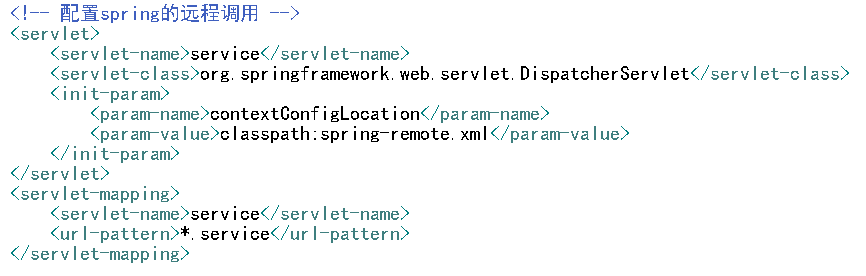


1. 编写测试代码



### Spring远程调用到系统-----统计服务

依据上面的步骤，将系统的统计服务加载到远程调用上



## 项目描述

-------------------------------

同时,系统还采用web service技术,将调查服务对外公开,可实现第三方系统的无缝整合.

系统通过对service进行缓存,能够大量降低数据库负载,从而提高程序性能.

技术要点:

1.系统采用ssh架构,便于维护和扩展.

2.系统wizard方案,供参与人员作答,避免用户反感,界面更加友好.

3.使用java seriliaze技术实现deeply copy,便于数据移植.

4.通过struts2 jfreechart plugin同jfreechat集成,生成各种图表.

5.使用poi导出excel文件,供决策人员参考.

6.采用bitwise技术实现权限控制,利用interceptor实现粗粒度控制以及采用tag replace实现细粒度控制.

采用二位运算能够是的权限运算更加快捷,而且极大地session存储量,有效利用服务器资源.

7.使用spring http invoker实现远程调用,和第三方基于java的应用实现更高效率的传输.

//8.系统中通过cxf webservice引擎,将统计服务对外公开,可实现第三方系统的无缝整合.

9.利用aop + scheduler采用动态表实现日志管理.

10.使用RoutingDatasource + token实现分库处理.