项目第二天

# 复习：

环境问题总结：

## Oracle环境：

Tns文件：

在**oracle服务端**或oracle client

通过oci.dll

Oracle服务器

可以在本机也可以在虚拟机

电脑

Pl/sql工具

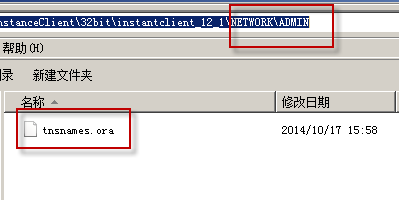
Pl/sql是32位，只能通过32位的oracle client连接

电脑是32位：

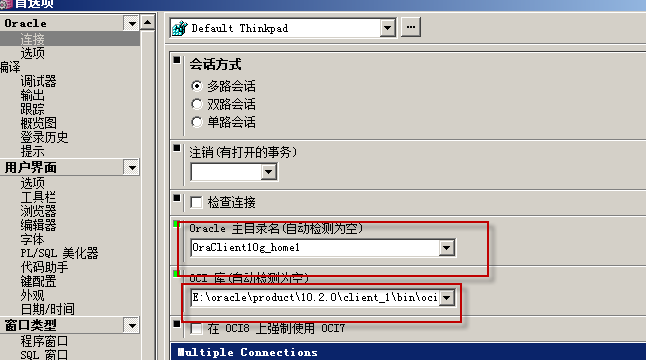
如果oracle服务端安装在本机，只需要在电脑上安装Pl/sql即可连接。

如果 oracle服务端没有在本机，需要在自己电脑上安装oracle client软件（安装版本10201\_client\_win32.zip，免安装解压即可使用 instantclient\_12\_1\_32bit.rar）

免安装oracle client，创建以下目录，目录中有tns文件 ：



在pl/sql上配置，找到tns文件和oci.dll



电脑64位：

**安装pl/sql时，安装目录有(x86)，注意(x86)去掉，pl/sql所在安装目录不能有括号。**

参考：



需要安装32位的oracle client软件，pl/sql通过32位client连接服务端。

## Maven问题

仓库（jar包）找不到，就是仓库地址没有配置对。

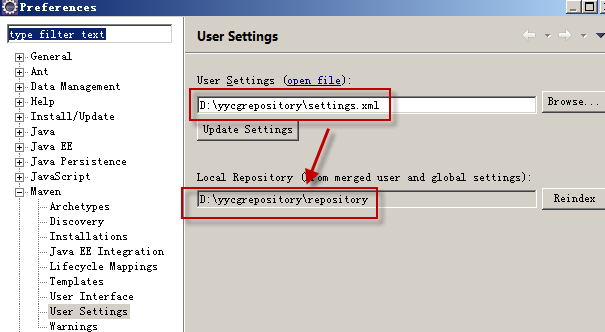
在命令行运行mvn命令

maven先找用户setting.xml（在C:\Users\Thinkpad\.m2），再找maven安装目录（在环境变量MAVEN\_HOME） 下的conf/setting.xml

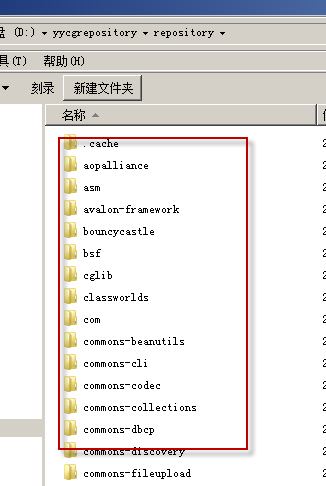
建议用户setting.xml删除，在命令执行只会找全局setting.xml(MAVEN\_HOME 下的conf/setting.xml)

在eclipse下开发，

Eclipse下配置maven环境：



本地仓库样子：



## 内容回顾

介绍这个系统：

第一方面：系统概述

本系统是一个市级医药采购系统，通过本系统监管单位上网监管，医疗单位上网采购，供货商上网销售，实现交易透明化、网络化、自动化。

第二方面：系统业务流程

核心业务流程是药品采购，医院创建一个采购单，卫生院审核采购单，供货商去受理采购单，按照采购单内容去发货，医院将药品入库。

用户角色：

卫生局：

卫生院：

卫生室：

供货商：

系统管理员：

系统架构：

采用maven进行构建。

技术架构模块：springmybatis

将spring3.1.4和mybatis3.2.3的jar添加在pom.xml中，使用maven的依赖管理。

如果要开发，只会在本工程。

主工程模块yycgproject：

包括base：系统基础功能（系统管理）

Business系统业务功能（药品目录 、采购单管理，）

在pom.xml添加插件，生成war包。

工具模块：yycgutil

包括系统所用到工具类

聚合模块：yycgparent

负责对各各子模块进行整合(聚合)，

自动生成war包。

对系统进行集成测试要聚合所有子模块，生成一个war包，将war包拷贝到web服务器上进行测试。

学习阶段，使用maven，用本地仓库。

在公司开发，使用maven私服（局域网一台服务器，有仓库）

# 教学计划

对系统进行封装：

用户管理模块的数据模型（掌握）：

数据库表结构及表与表关系。

用户查询列表：(**掌握**)

使用jquery easyui的datagrid数据列表。

Datagrid要求服务端返回json格式的数据。

用户添加(掌握)：

采用jquery提供一个ajax 的form提交的组件。

实现ajax提交采用原始form提交，方便开发。

本系统提交统一采用：提交的key/value数据（使用ajax 的form提交的组件），服务端响应json数据（使用ResponseBody）(重点掌握)。

高级知识：开发统一异常处理器（架构师实现）。

掌握统一异常处理器统程

学习一些系统工具类（掌握）

# 用户管理模型

## 模型分析

业务是什么？业务就是用户需求。

用户角色：卫生局、卫生院、卫生室、供货商、系统管理员

市卫生局（单位）

记录管理区域

卫生院（单位）

记录管理区域

卫生室（单位）

记录所属区域

供货商（单位）

卫生院（单位）

行政管理关系，根据行政区域划分

卫生室（单位）

行政管理关系

卫生院（单位）

**指定**供货商供应多个区域

供货区域表：

记录供货商供货区域

供货商id（外键）

区域id

外键

用户：

登陆系统进行业务操作。

实体分析：

系统用户表SYSUSER：

记录系统中所有用户

Id：主键

USERID 账号

USERNAME 名称

GROUPID：用户类别 0:系统管理员,1：卫生局 2:卫生院 3：卫生室 4:供货商

SYSID：系统用户所属的单位

根据GROUPID判断，所对应的单位

监督单位表USERJD：

Id：主键

MC: 单位名称：

DQ：管理区域

医院单位表USERYY：

Id：主键

MC: 单位名称：

DQ：所属区域

供货商单位表USERGYS：

Id：主键

Mc：单位名称：

基础表：

行政区域表BSS\_SYS\_AREA：

市（一级）、县/乡/镇(二级)、村(三级)

树型结构。

区域id规则：XX.XX.XX.

使用规则方便：

比如查询某个镇的区域：

select \* from BSS\_SYS\_AREA t where areaid like '1.1.%' and t.arealevel='3'

管理关系：

市卫生局管理本市所有卫生院

卫生院管理本辖区的卫生室

通过在监督单位 和医院单位表中添加DQ（地区）字段，实现行政管理关系。

比如：查询某镇下的医院

select \* from USERYY t where t.dq like '1.1.%'

供货关系：

指定供货商供货的区域

供货商供货区表USERGYSAREA：

USERGYSID：供货商id（外键，指向供货商单位表）

AREAID：区域id（外键，指定区域表BSS\_SYS\_AREA）

注意：业务表(比如采购单表)，要和单位表关联，记录某单位所采购的药品，而不是记录某个用户所采购药品。

用户登陆系统如何用户所属的单位？

根据用户账号查询用户表，得到sysid(单位id)

## 表结构

监督单位表USERJD

**记录卫生院和卫生院的单位信息，它两个管理地区不同。**

Id：主键

Mc:单位名称

DQ：管理地区

卫生室 ：useryy

记录所有卫生室

Id：主键

Mc:单位名称

DQ：所属地区

通过DQ的规则（XX.XX.XX.）建立行政关系

供货商单位表:usergys

Id：主键

Mc:单位名称

供货商供货区域表：

记录供货商所供货的区域（二级区域乡/镇）

Usergysid：供货商id

Areaid：区域id

外键：Usergysid

系统用户：sysuser

记录系统中所有用户信息

Id：

Userid：账号

Username：名称

Groupid：用户类型0:系统管理员,1：卫生局 2:卫生院 3：卫生室 4:供货商

Sysid：所属单位id（系统管理是空）

Sysid(非外键)

Sysid(非外键)

Sysid(非外键)

区域表：BSS\_SYS\_AREA

外键：areaid

# 用户查询列表

关于原型：

美工主要做CSS（样式），将某一类页面做一个样式页面(html)，程序员在表现层，将html（静态页面）改为动态jsp。

常见javaEE系统页面原型：

首页框架：由UI的人员实现

表单：程序员主要实现表单

查询列表：程序员主要实现查询列表

**如果美工没有提供静态页面，就可以上网参考现成的页面，加以修改，创新实现！！！！**

## 需求

根据自定义查询，查询用户信息



内容包括：

账号

名称

用户类型

所属单位

状态

用户添加：



## 分析开发

Dao:

使用mybatis查询用户列表。

是使用逆向生成代码，还是自定义？

需要多表查询，需要自定义mapper

* Sql语句：

**主查询表**：SYSUSER

**关联查询表**：userjd（监督单位表），useryy（医院表）

如果主查询表的字段通过关联查询表能查到（且只能查询一条），可以使用内链接

如果用户是系统管理员，就不能从单位表查询到，所以要外链接

*---子查询方法*

select \* from (

select SYSUSER.id,

SYSUSER.userid,

SYSUSER.username,

SYSUSER.groupid,

SYSUSER.sysid,

decode(SYSUSER.Groupid,

'1',

(select mc from userjd where id = sysuser.sysid),

'2',

(select mc from userjd where id = sysuser.sysid),

'3',

(select mc from useryy where id = sysuser.sysid),

'4',

(select mc from usergys where id = sysuser.sysid)

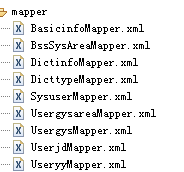
) sysmc

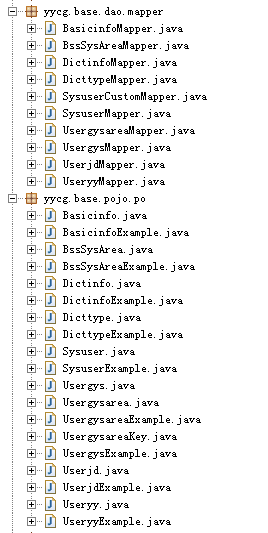
from SYSUSER

)sysuser

where sysuser.sysmc like '%卫生室%'

将用户管理模块的所有表逆向工程。





* 自定义mapper.xml：

自定义的mapper命名规则：XXXMapperCustom.xml

在yycg\base\pojo\vo：存储自定义pojo类

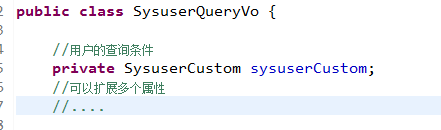
查询结果集统一采用resultType，需要扩展一个类

扩展类,命名规则：XXXXCustom.java ，用于提交信息、查询条件，原来生成po类不够用要使用扩展

通常自定义扩展类需要**继承**查询列较多的po类。

parameterType 输入参数类 统一采用包装类。

XXXQueryVo.java 用于页面向action传递参数，将参数传到mybatis。



SysuserMapperCustom.xml

<!-- 用户查询条件 通常情况，为了提高 sql片段可重用性，按单表创建sql片段 -->

<sql id=*"query\_sysuser\_where"*>

<if test=*"sysuserCustom!=null"*>

<if test=*"sysuserCustom.userid!=null and sysuserCustom.userid!=''"*>

and sysuser.userid = #{sysuserCustom.userid}

</if>

<if test=*"sysuserCustom.username!=null and sysuserCustom.username!=''"*>

and sysuser.username = #{sysuserCustom.username}

</if>

<if test=*"sysuserCustom.sysmc!=null and sysuserCustom.sysmc!=''"*>

and sysuser.sysmc like '%${sysuserCustom.sysmc}%'

</if>

<if test=*"sysuserCustom.groupid!=null and sysuserCustom.groupid!=''"*>

and sysuser.groupid = #{sysuserCustom.groupid}

</if>

</if>

</sql>

<select id=*"findSysuserList"* parameterType=*"yycg.base.pojo.vo.SysuserQueryVo"*

resultType=*"yycg.base.pojo.vo.SysuserCustom"*>

select \* from (

select SYSUSER.id,

SYSUSER.userid,

SYSUSER.username,

SYSUSER.groupid,

SYSUSER.sysid,

decode(SYSUSER.Groupid,

'1',

(select mc

from userjd where id = sysuser.sysid),

'2',

(select mc from userjd where

id = sysuser.sysid),

'3',

(select mc from useryy where id =

sysuser.sysid),

'4',

(select mc from usergys where id = sysuser.sysid)

)

sysmc

from SYSUSER

)sysuser

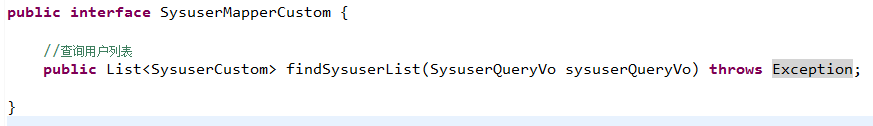
<where>

<include refid=*"query\_sysuser\_where"* />

</where>

</select>

mapper.java接口：



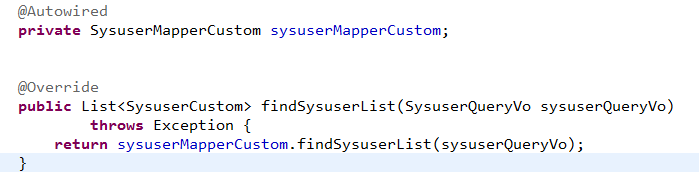
service

接口功能：根据条件查询用户列表

接口参数：查询条件，接口参数尽量细化。

建议：Service接口参数尽量使用简单类型，不要使用Map集合类型。

注意：service开发系统核心



action

在页面上使用jquery easyui的datagrid显示数据列表。

数据查询列表，action就需要创建两个方法：

* 显示查询页面
* Datagrid加载json结果集：

如何将java对象转成datagrid要求json结果集：

{

"total":239,

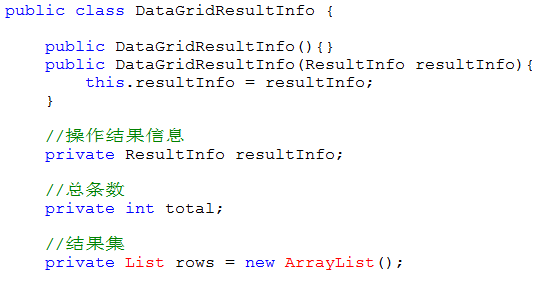
"rows":[

]

}

使用系统统一的一个结果类：**DataGridResultInfo**

此类有两个属性total和rows(list集合)



**DataGridResultInfo经过@responseBody注解自动转json，格式就是：**

**Java对象的属性作为json的key，对list集合转json后就成了数组。**

{

"total":239,

"rows":[

]

}

@responseBody内部使用两个jar包转json



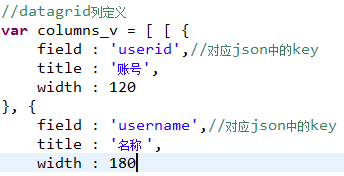
页面

通过jquery easyui的datagrid加载数据列表。

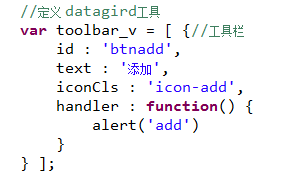
第0步：确定页面布局

根据美工提供html静态页面，进行编写。

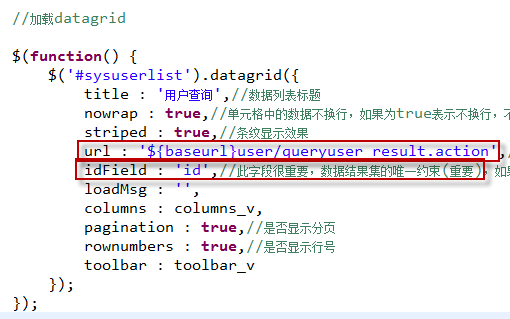
第一步：定义datagrid的列



第二步：定义datagrid的工具栏



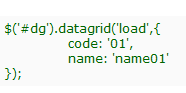
第三步：加载datagrid



第四步：编写查询方法

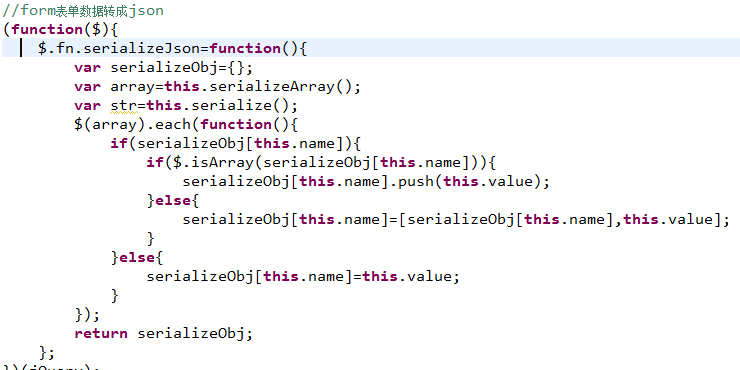
点击查询按钮执行js方法

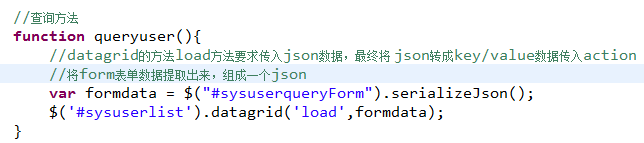
调用datagrid的load加载datagrid数据



datagrid的load方法需要传递参数类型是json（查询条件）

为了方便组织json查询条件，使用custom.box.main.js 将form表单数据提取出来，自动组成一个json





## 数据列表的分页

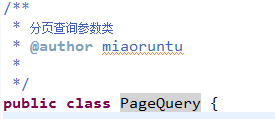
使用oracle的分页实现。

dao

修改SysuserMapperCustom.xml中findSysuserList添加分页支持。

Oracle分页需要起始和结束下标，需要根据当前页码、每页显示数量、总条数（符合查询条件记录数）

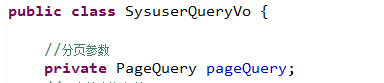
使用PageQuery.java计算起始和结束的下标：



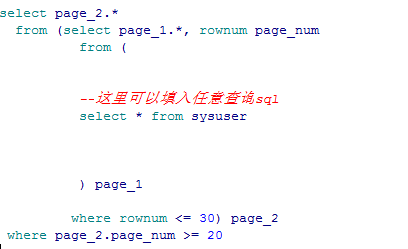
使用方法：

New构造pageQuery对象，调用setPageParams方法，传入总条数、每页显示数量、当前页码，该对象中的属性PageQuery\_start和PageQuery\_end计算出了起始和结束的下标。

将PageQuery对象传入mybatis，将此对象作为包装对象的属性。



Oracle分页sql：



分页mapper.xml

分页头和分页的尾。

<!-- 用户查询 -->

<select id=*"findSysuserList"* parameterType=*"yycg.base.pojo.vo.SysuserQueryVo"*

resultType=*"yycg.base.pojo.vo.SysuserCustom"*>

<!-- 分页头 -->

<if test=*"pageQuery!=null"*>

select page\_2.\*

from (select page\_1.\*, rownum page\_num

from (

</if>

select \* from (

select SYSUSER.id,

SYSUSER.userid,

SYSUSER.username,

SYSUSER.groupid,

sysuser.USERSTATE,

SYSUSER.sysid,

decode(SYSUSER.Groupid,

'1',

(select mc

from userjd where

id = sysuser.sysid),

'2',

(select mc from userjd where

id =

sysuser.sysid),

'3',

(select mc from useryy where id =

sysuser.sysid),

'4',

(select mc from usergys where id = sysuser.sysid)

)

sysmc

from

SYSUSER

)sysuser

<where>

<include refid=*"query\_sysuser\_where"* />

</where>

<!-- 分页尾 -->

<if test=*"pageQuery!=null"*>

) page\_1

<![CDATA[

where rownum <= ${pageQuery.PageQuery\_end}) page\_2

where page\_2.page\_num >= ${pageQuery.PageQuery\_start}

]]>

</if>

</select>

总条数：符合条件记录数

在mapper.xml中创建一个select，返回总条数

<!-- 查询列表的总记录数 -->

<select id=*"findSysuserCount"* parameterType=*"yycg.base.pojo.vo.SysuserQueryVo"*

resultType=*"int"*>

select count(\*) from (

select SYSUSER.id,

SYSUSER.userid,

SYSUSER.username,

SYSUSER.groupid,

sysuser.USERSTATE,

SYSUSER.sysid,

decode(SYSUSER.Groupid,

'1',

(select mc

from userjd where

id = sysuser.sysid),

'2',

(select mc from userjd where

id =

sysuser.sysid),

'3',

(select mc from useryy where id =

sysuser.sysid),

'4',

(select mc from usergys where id = sysuser.sysid)

)

sysmc

from

SYSUSER

)sysuser

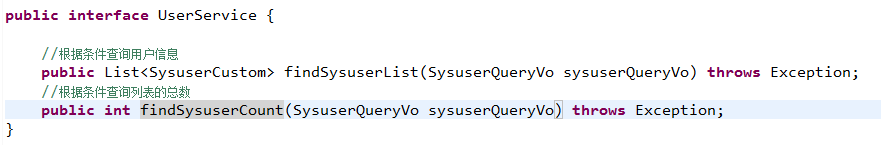
<where>

<include refid=*"query\_sysuser\_where"* />

</where>

</select>

service



action

调用service执行分页查询，准备PageQuery对象数据（包括起始和结束下标）。

先调用service查询出总数。

Datagrid加载数据列表时，固定向action方法传两个分页参数：



当前页码和每页显示个数

Action分页查询代码：

//用户查询页面的结果集

//最终DataGridResultInfo通过@ResponseBody将java对象转成json

@RequestMapping("/queryuser\_result")

**public** @ResponseBody DataGridResultInfo queryuser\_result(

SysuserQueryVo sysuserQueryVo,

**int** page,//页码

**int** rows//每页显示个数

)**throws** Exception{

//非空校验

sysuserQueryVo = sysuserQueryVo!=**null**?sysuserQueryVo:**new** SysuserQueryVo();

//查询列表的总数

**int** total = userService.findSysuserCount(sysuserQueryVo);

PageQuery pageQuery = **new** PageQuery();

pageQuery.setPageParams(total, rows, page);

sysuserQueryVo.setPageQuery(pageQuery);

//分页查询，向sysuserQueryVo中传入pageQuery

List<SysuserCustom> list = userService.findSysuserList(sysuserQueryVo);

DataGridResultInfo dataGridResultInfo = **new** DataGridResultInfo();

//填充 total

dataGridResultInfo.setTotal(total);

//填充 rows

dataGridResultInfo.setRows(list);

**return** dataGridResultInfo;

}

# 用户添加

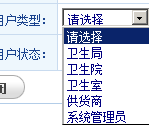
## 需求



添加用户：

用户账号(输入)、用户名称(输入)、用户密码(输入)、用户类型(下拉)、用户单位名称(输入)、用户状态（单选）

用户类型：



前置条件(操作约束)：

用户账号不允许重复

根据用户类型，输入单位名称必须存在对应的单位表

后置条件(数据库操作)：

向sysuser表插入一条记录

## 分析开发

dao

向sysuser表插入一条记录

使用逆向工程生成的mapper实现

service

接口功能：添加系统用户

操作参数：用户信息，使用SysuserCustom

接口约束：

用户账号不允许重复

根据用户类型，输入单位名称必须存在对应的单位表

代码：

注意在service进行数据合法性校验。

遇到异常要抛出异常信息。

**public** **void** insertSysuser(SysuserCustom sysuserCustom) **throws** Exception {

//参数校验

//通用的参数合法校验，非空校验，长度校验

//...使用一些工具类来完成

//数据业务合法性校验

//账号唯一性校验，查询数据库校验出来

//思路：根据用户账号查询sysuser表，如果查询到说明 账号重复

Sysuser sysuser = **this**.findSysuserByUserid(sysuserCustom.getUserid());

**if**(sysuser!=**null**){

//账号重复

//抛出异常，可预知异常

**throw** **new** Exception("账号重复");

}

//根据用户类型，输入单位名称必须存在对应的单位表

String groupid = sysuserCustom.getGroupid();//用户类型

String sysmc = sysuserCustom.getSysmc();//单位名称

**if**(groupid.equals("1") || groupid.equals("2")){

//监督单位

//根据单位名称查询单位信息

Userjd userjd = **this**.findUserjdByMc(sysmc);

**if**(userjd==**null**){

//抛出异常，可预知异常

**throw** **new** Exception("单位名称输入错误");

}

}**else** **if**(groupid.equals("3")){

//卫生室

//根据单位名称查询单位信息

Useryy useryy = **this**.findUseryyByMc(sysmc);

**if**(useryy==**null**){

//抛出异常，可预知异常

**throw** **new** Exception("单位名称输入错误");

}

}**else** **if**(groupid.equals("4")){

//供货商

//根据单位名称查询单位信息

Usergys usergys = **this**.findUsergysByMc(sysmc);

**if**(usergys==**null**){

//抛出异常，可预知异常

**throw** **new** Exception("单位名称输入错误");

}

}

//设置主键

sysuserCustom.setId(UUIDBuild.*getUUID*());

sysuserMapper.insert(sysuserCustom);

}

action

添加用户，写两个方法：

进入添加页面方法

此方法准备数据在添加页面显示。

提交方法：

调用service类提交，将service返回的结果在页面展示，如果service抛出异常，在action中捕获异常。

Action方法中注意对异常处理，异常信息就是service抛出的异常。

//添加用户提交

//提交 结果转json输出到页面

//提交表单数据统一使用包装类

@RequestMapping("/addsysusersubmit")

**public** @ResponseBody Map<String,Object> addsysusersubmit(SysuserQueryVo sysuserQueryVo)**throws** Exception{

//提示用户信息

String message = "操作成功！！";

**int** type=0;//成功

**try** {

//调用service执行用户添加

userService.insertSysuser(sysuserQueryVo.getSysuserCustom());

} **catch** (Exception e) {

//输出异常信息

e.printStackTrace();

//对应异常信息进行解析

message = e.getMessage();

type=1;//失败

}

//将执行结果返回页面

Map<String, Object> result\_map = **new** HashMap<String, Object>();

result\_map.put("type", type);

result\_map.put("message", message);

**return** result\_map;

}

页面

* 用户添加页面

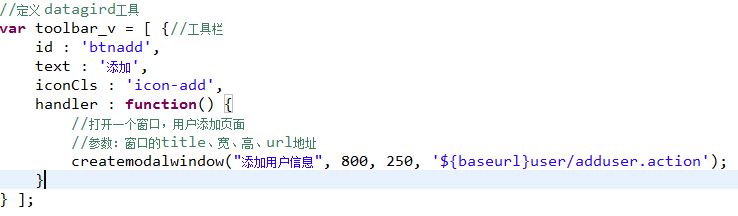
在用户列表页面中，点击“添加”，打开添加页面。

使用如下方法打开一个新窗口：

//参数：窗口的title、宽、高、url地址

createmodalwindow("添加用户信息", 800, 250, '${baseurl}user/adduser.action');

createmodalwindow:使用jquery easyui的window组件，里边拼接一个html串，串就是ifram，ifram的链接就是用户添加的url



* 用户添加提交

页面提交给服务端是key/value数据，服务端返回是json。

为什么服务要返回json？

Json作为接口交互通用格式越来越流行，在页面通过js解析json数据非常方便。

本系统采用post提交，使用jquery提供一个ajax的form提交组件。



封装一个js方法，用于提交form表单的数据，进行ajax提交，内部使用上边form提交的组件。

/\*

\*form提交(post方式)

\*

\*formId form Id

\*callbackfn 回调函数名(要求函数必须有参数且不能多与两个,一个参数时参数为响应文本,两个参数时第一个参数为响应文本)

\*param 回调函数参数(如果为null,那么调用一个参数的回调函数,否则调用两个参数的回调函数)

\*dataType：预期服务器返回的数据类型

\*/

**function** jquerySubByFId(formId,callbackFn,param,dataType){

**var** formObj = jQuery("#" + formId);

**var** options = {

dataType: ("undefined"!=dataType && **null**!=dataType)?dataType:"json",

success: **function**(responseText) {

**if**(param === **null**){

callbackFn(responseText);

}**else**{

callbackFn(responseText,param);

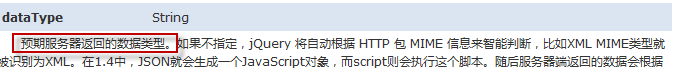
}

}

};

formObj.ajaxSubmit(options);

}



在用户添加页面上编写提交方法：

**function** sysusersave(){

//准备使用jquery 提供的ajax Form提交方式

//将form的id传入，方法自动将form中的数据组成成key/value数据，通过ajax提交，提交方法类型为form中定义的method，

//使用ajax form提交时，不用指定url，url就是form中定义的action

//此种方式和原始的post方式差不多，只不过使用了ajax方式

//第一个参数：form的id

//第二个参数：sysusersave\_callback是回调函数，sysusersave\_callback当成一个方法的指针

//第三个参数：传入的参数， 可以为空

//第四个参数：dataType预期服务器返回的数据类型,这里action返回json

//根据form的id找到该form的action地址

jquerySubByFId('userform',sysusersave\_callback,**null**,"json");

}

//编写提交回调方法

**function** sysusersave\_callback(data){

alert(data.message);

}

在回调方法中，使用jqueryeasyui的提示方法。

# 统一异常处理

## 统一异常结果类型

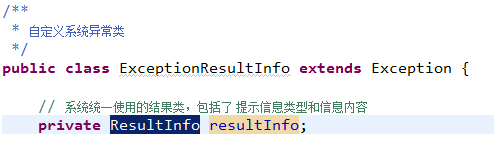
Java中进行异常处理：

一类是可预知的异常，程序员在编码时，主动抛出的异常，为了给用户操作提示，提前检查代码中可能存在异常。

通过开发中，采用自定义的异常类，每个异常类表示每一类异常信息。类需要继承Exception类。

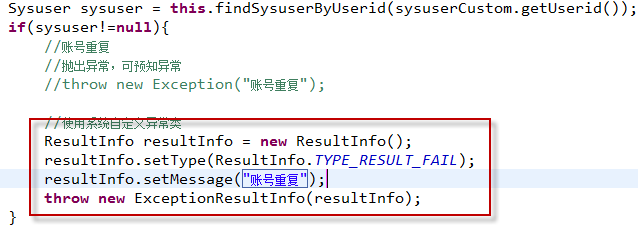
本系统采用统一异常类，提供一个属性标识异常类。

另一类是不可预知异常，就是runtimeException异常，通过提高代码编写质量来避免此类异常，也可通过后期测试 人员进行系统功能测试对runtime异常进行避免。



如何使用异常类？

在service中要抛出此异常类。

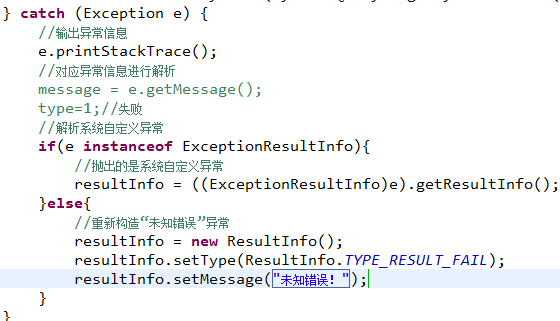


在action中捕获系统自定义异常，解析系统自定义异常。

解析系统自定义异常过程：

如果抛出的ExceptionResultInfo系统自定义异常，直接获取该异常信息即可。

如果不是抛出的ExceptionResultInfo系统自定义异常,重新构造一个“未知错误异常！”。

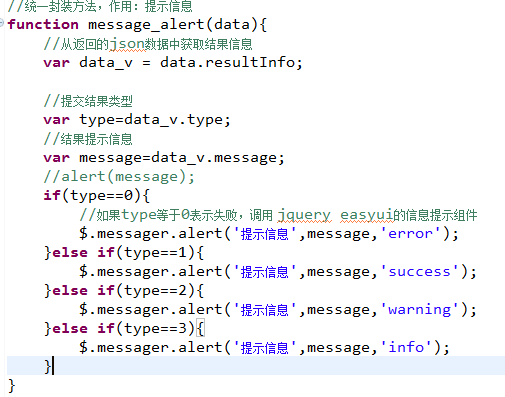


将异常信息或成功信息返回到页面:

统一使用类SubmitResultInfo，提交类的方法统一使用该类型作为返回值类型。



由于action中所有的提交类方法统一返回SubmitResultInfo，所以在页面要统一使用一个方法解析json结果。



修改回调函数：

**function** sysusersave\_callback(data){

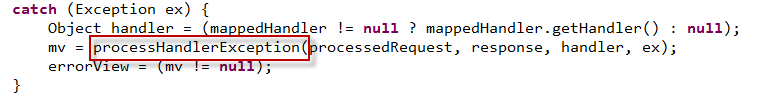
message\_alert(data);

｝

## 统一异常处理器

Springmvc提供统一处理器机制，springmvc的前端控制器在调用适配器，去调用action，过程中如果发生异常，前端控制器交给异常处理器进行异常处理。

前端控制器源代码：



自定义异常处理器流程(掌握)

对dao、service及action所抛出的异常进行统一处理，在action中就不需要添加try{}catch{}捕获的代码。

抛出异常

由统一异常处理器解析

如果是系统自定义异常则直接获取，如果不是构造一个未知错误异常

Action是否返回json

将异常处理转成json输出到页面

将异常处理在错误页面显示

返回json

### 自定义异常处理器编写

自定义全局异常处理器，实现HandlerExceptionResolver接口

**public** **class** ExceptionResolverCustom **implements** HandlerExceptionResolver {

// json转换器

// 将异常信息转json

**private** HttpMessageConverter<ExceptionResultInfo> jsonMessageConverter;

// 前端控制器调用此方法执行异常处理

// handler，执行的action类就包装了一个方法（对应url的方法）

@Override

**public** ModelAndView resolveException(HttpServletRequest request,

HttpServletResponse response, Object handler, Exception ex) {

// 输出 异常信息

ex.printStackTrace();

// 转成springmvc底层对象（就是对action方法的封装对象，只有一个方法）

HandlerMethod handlerMethod = (HandlerMethod) handler;

// 取出方法

Method method = handlerMethod.getMethod();

// 判断方法是否返回json

// 只要方法上有responsebody注解表示返回json

// 查询method是否有responsebody注解

ResponseBody responseBody = AnnotationUtils.*findAnnotation*(method,

ResponseBody.**class**);

**if** (responseBody != **null**) {

// 将异常信息转json输出

**return** **this**.resolveJsonException(request, response, handlerMethod,

ex);

}

// 这里说明action返回的是jsp页面

// 解析异常

ExceptionResultInfo exceptionResultInfo = resolveExceptionCustom(ex);

// 将异常信息在异常页面显示

request.setAttribute("exceptionResultInfo",

exceptionResultInfo.getResultInfo());

// 转向错误页面

ModelAndView modelAndView = **new** ModelAndView();

modelAndView.addObject("exceptionResultInfo",

exceptionResultInfo.getResultInfo());

modelAndView.setViewName("/base/error");// 逻辑视图名

**return** modelAndView;

}

// 异常信息解析方法

**private** ExceptionResultInfo resolveExceptionCustom(Exception ex) {

ResultInfo resultInfo = **null**;

**if** (ex **instanceof** ExceptionResultInfo) {

// 抛出的是系统自定义异常

resultInfo = ((ExceptionResultInfo) ex).getResultInfo();

} **else** {

// 重新构造“未知错误”异常

resultInfo = **new** ResultInfo();

resultInfo.setType(ResultInfo.*TYPE\_RESULT\_FAIL*);

resultInfo.setMessage("未知错误！");

}

**return** **new** ExceptionResultInfo(resultInfo);

}

// 将异常信息转json输出

**private** ModelAndView resolveJsonException(HttpServletRequest request,

HttpServletResponse response, Object handler, Exception ex) {

// 解析异常

ExceptionResultInfo exceptionResultInfo = resolveExceptionCustom(ex);

HttpOutputMessage outputMessage = **new** ServletServerHttpResponse(response);

**try** {

//将exceptionResultInfo对象转成json输出

jsonMessageConverter.write(exceptionResultInfo, MediaType.*APPLICATION\_JSON*, outputMessage);

} **catch** (HttpMessageNotWritableException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**return** **new** ModelAndView();

}

**public** HttpMessageConverter<ExceptionResultInfo> getJsonMessageConverter() {

**return** jsonMessageConverter;

}

**public** **void** setJsonMessageConverter(

HttpMessageConverter<ExceptionResultInfo> jsonMessageConverter) {

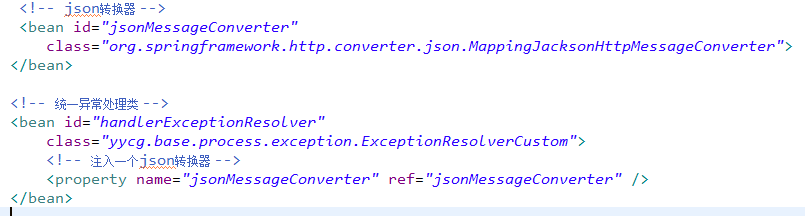
**this**.jsonMessageConverter = jsonMessageConverter;

}

}

统一异常处理器配置

在springmvc.xml配置统一异常处理器。



在web.xml中配置：

前端控制器知道全局异常处理器id为handlerExceptionResolver

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:spring/springmvc.xml</param-value>

</init-param>

<init-param>

<param-name>detectAllHandlerExceptionResolvers</param-name>

<param-value>false</param-value>

</init-param>

</servlet>

detectAllHandlerExceptionResolvers：

屏蔽自动注册异常处理器，固定使用bean的id为handlerExceptionResolver的异常处理器。

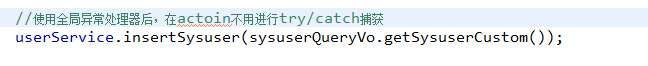
统一异常处理器使用

系统中所有异常由全局异常处理器处理。

Dao方法向外抛出系统自定义异常。

Service方法向外抛出系统自定义异常。

Action方法向外抛出系统自定义异常。



统一异常处理器测试

1. 在action的提交方法中发生异常，由全局异常处理器进行处理。

2、在action返回jsp方法中发生异常，由全局异常处理器进行处理。