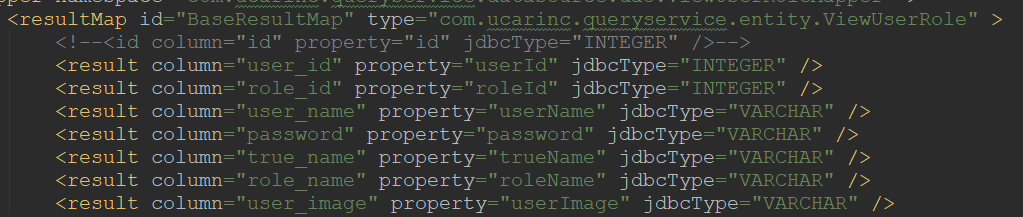
# SSM问题笔记日常记录

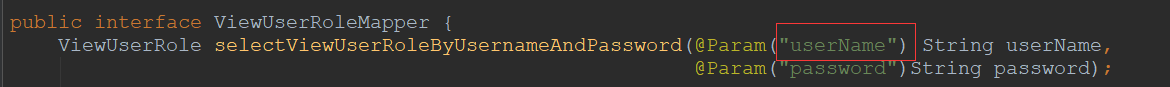
## 一、MyBatis知识点

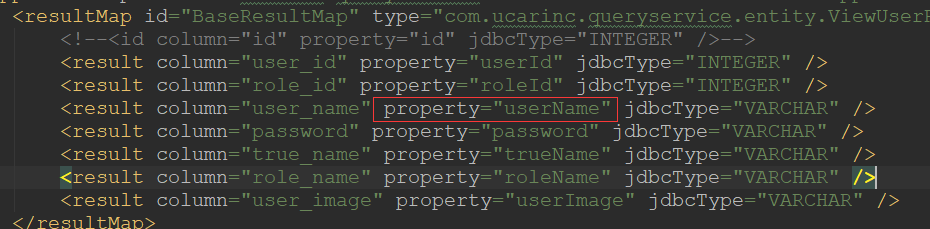
1.视图操作与一般的表一样处理，在xml中少了<id/>映射。

在设计视图时一定要全面考虑将来需要用到的数据，以免重复劳动。（确定所需列数）



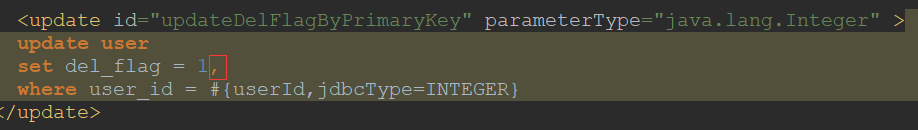
2.在多参数下，注意要将mapper接口使用“@Param(参数)”注解，并且参数要与映射xml文件中的参数名相同。（mapper多参数问题）



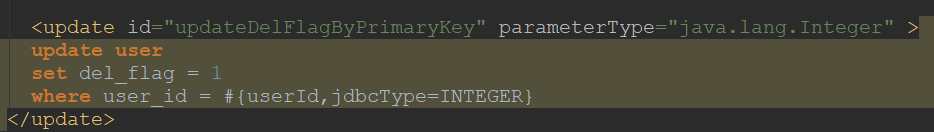


3.xml配置时，注意细节格式，此处如图所示无逗号。

错误：



正确：



4.模糊查询时，其xml配置为：

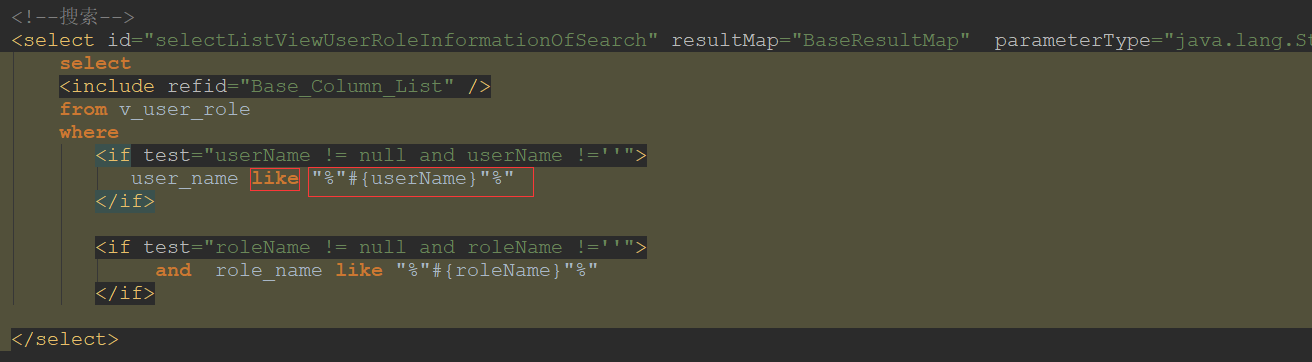
<if test="userName != null and userName !=''">

user\_name like ‘%’#{userName}‘%’

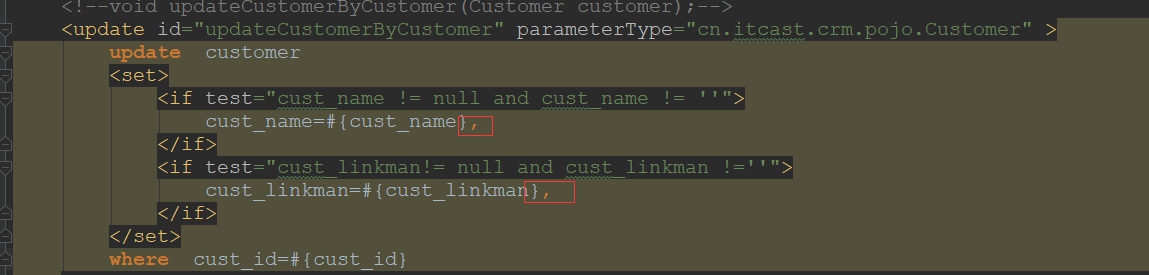
或者user\_name like ‘%#{userName}%’

</if>

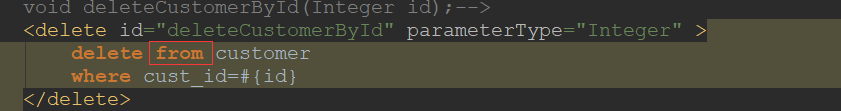
如图所示：



5.书写更新操作时，<set>标签内没有写逗号



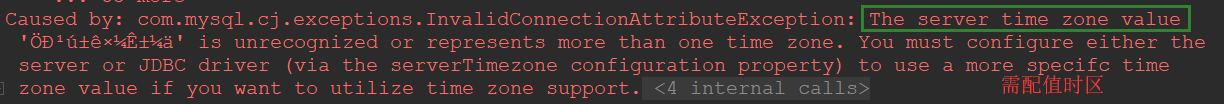
6.书写delete操作是，必须有from



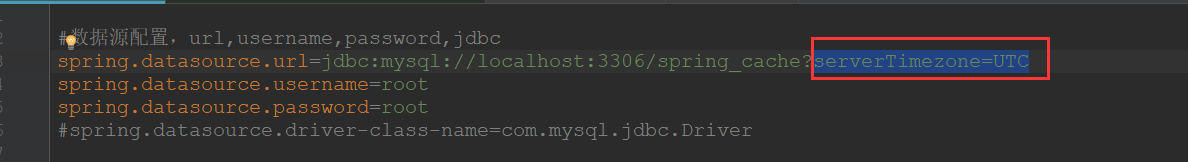
7.数据源配置问题

1）时区问题，在数据源url后添加serverTimezone=UTC

问题：



解决：



8.delete后忘了加from

delete from employee where id=#{id}

9、精讲#{}和${}的区别是什么？

经常碰到这样的面试题目：#{}和${}的区别是什么？

正确的答案是：#{}是**预编译处理**，${}是字符串替换(占位符)。

（1）mybatis在处理#{}时，会将sql中的#{}替换为?号，调用PreparedStatement的set方法来赋值。

（2）mybatis在处理${}时，就是把${}替换成变量的值。

（3）使用#{}可以有效的防止SQL注入，提高系统安全性。原因在于：预编译机制。**预编译完成之后，SQL的结构已经固定，即便用户输入非法参数，也不会对SQL的结构产生影响，从而避免了潜在的安全风险。**

（4）预编译是提前对SQL语句进行预编译，而其后注入的参数将不会再进行SQL编译。我们知道，SQL注入是发生在编译的过程中，因为恶意注入了某些特殊字符，最后被编译成了恶意的执行操作。而预编译机制则可以很好的防止SQL注入。

最后，补充一点：

$符号一般用来当作占位符，常使用Linux脚本的人应该对此有更深的体会吧。例如：$1，$2等等表示输入参数的占位符。知道了这点就能很容易区分$和#，从而不容易记错了。

## 二、iBaits

### 1. Dao层queryForList()

当查询List<Model>时，返回结果集合“resultClass=Model”,Dao层调用用queryForList()。

### 2.操作数据库时注意时间问题，默认为00.00.00 00:00映射到结果时会报错。

原因：空时间在映射到结果时会出现转换错误

解决方法：测试时，在数据库里修改00.00.00 00:00为其他时间，不要空就行。

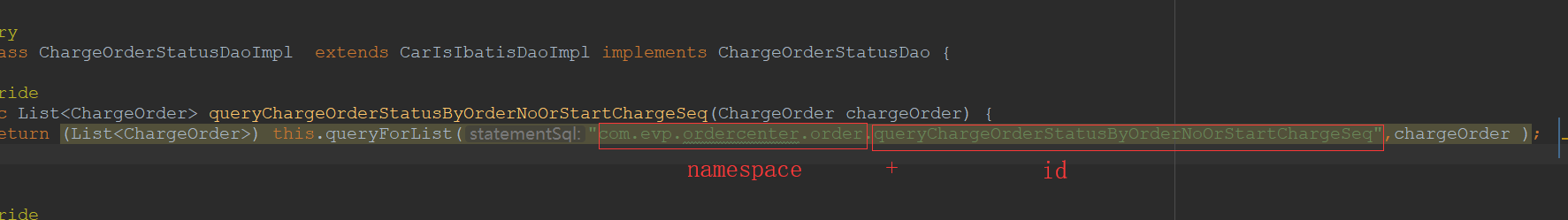
### 3.Dao接口调用对应的sql中的id问题。

StatementSql语句=namespace+id

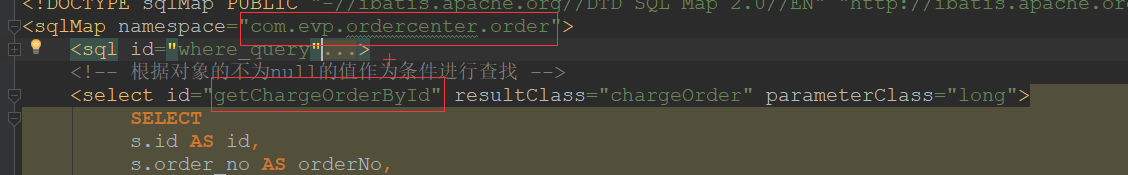
例如：

update("com.evp.ordercenter.order.updateChargeOrderStatusByOrderNoOrStartChargeSeq", record);

参考理解：图一图二



图一：DaoImpl层



图二：sql.xml

### 4. dynamic动态sql查询

方式一：定义引用**dynamic**的方式

<!—1.定义动态sql-->

<sql id="where\_query">  
 <dynamic prepend="where">  
 <isNotEmpty property="id" prepend="and">  
 s.id = #id#  
 </isNotEmpty>  
 <isNotEmpty property="orderNo" prepend="and">  
 s.order\_no = #orderNo#  
 </isNotEmpty>  
 <isNotEmpty property="startChargeSeq" prepend="and">  
 s.start\_charge\_seq = #startChargeSeq#  
 </isNotEmpty>

</dynamic>

</sql>

<!—2.动态sql查询-->  
<select id="queryChargeOrder" resultClass="chargeOrder" parameterClass="chargeOrder">  
 SELECT  
 s.id AS id,  
 s.order\_no AS orderNo,  
 s.start\_charge\_seq AS startChargeSeq,  
 s.order\_status AS orderStatus,  
 s.plate\_id AS plateId,  
 s.user\_id AS userId,  
 s.operator\_id AS operatorId,  
 s.city\_id AS cityId,  
 s.equipment\_id AS equipmentId,  
 s.city\_name AS cityName,  
 s.station\_id AS stationId,  
 s.connector\_id AS connectorId,  
 s.place\_time AS placeTime,  
 s.start\_fail\_reason AS startFailReason,  
 s.stop\_reason AS stopReason,  
 s.current\_quantity AS currentQuantity,  
 s.before\_quantity AS beforeQuantity,  
 s.after\_quantity AS afterQuantity,  
 s.change\_quantity AS changeQuantity,  
 s.charge\_fee AS chargeFee,  
 s.service\_fee AS serviceFee,  
 s.reduce\_fee AS reduceFee,  
 s.total\_fee AS totalFee,  
 s.pay\_status AS payStatus,  
 s.charge\_start\_time AS chargeStartTime,  
 s.charge\_end\_time AS chargeEndTime,  
 s.scan\_no AS scanNo,  
 s.create\_time AS createTime,  
 s.modify\_time AS modifyTime  
 FROM t\_charge\_order s  
 <include refid="where\_query"/><!—引用-->  
</select>

方式二：将dynamic直接写入语句的方式

<select id="queryChargeOrder" resultClass="chargeOrder" parameterClass="chargeOrder">  
 SELECT  
 s.id AS id,  
 s.order\_no AS orderNo,  
 s.start\_charge\_seq AS startChargeSeq,  
 s.order\_status AS orderStatus,  
 s.plate\_id AS plateId,  
 s.user\_id AS userId,  
 s.operator\_id AS operatorId,  
 s.city\_id AS cityId,  
 s.equipment\_id AS equipmentId,  
 s.city\_name AS cityName,  
 s.station\_id AS stationId,  
 s.connector\_id AS connectorId,  
 s.place\_time AS placeTime,  
 s.start\_fail\_reason AS startFailReason,  
 s.stop\_reason AS stopReason,  
 s.current\_quantity AS currentQuantity,  
 s.before\_quantity AS beforeQuantity,  
 s.after\_quantity AS afterQuantity,  
 s.change\_quantity AS changeQuantity,  
 s.charge\_fee AS chargeFee,  
 s.service\_fee AS serviceFee,  
 s.reduce\_fee AS reduceFee,  
 s.total\_fee AS totalFee,  
 s.pay\_status AS payStatus,  
 s.charge\_start\_time AS chargeStartTime,  
 s.charge\_end\_time AS chargeEndTime,  
 s.scan\_no AS scanNo,  
 s.create\_time AS createTime,  
 s.modify\_time AS modifyTime  
 FROM t\_charge\_order s  
 <dynamic prepend="where">  
 <isNotEmpty property="orderNo" prepend="and">  
 s.order\_no = #orderNo#  
 </isNotEmpty>  
 <isNotEmpty property="startChargeSeq" prepend="and">  
 s.start\_charge\_seq = #startChargeSeq#  
 </isNotEmpty>  
 </dynamic>

</select>

### 5. dynamic动态sql更新

<!--通过本地或第三方订单编号更新订单-->  
 <update id="updateChargeOrderStatusByOrderNoOrStartChargeSeq" parameterClass="chargeOrder">  
 update t\_charge\_order

🡨-set语句—>  
<dynamic prepend="set">  
 <isNotEmpty property="id" prepend=",">  
 id = #id#  
 </isNotEmpty>  
 <isNotEmpty property="orderNo" prepend=",">  
 order\_no = #orderNo#  
 </isNotEmpty>  
 <isNotEmpty property="startChargeSeq" prepend=",">  
 start\_charge\_seq = #startChargeSeq#  
 </isNotEmpty>  
 <isNotEmpty property="orderStatus" prepend=",">  
 order\_status = #orderStatus#  
 </isNotEmpty>  
 <isNotEmpty property="plateId" prepend=",">  
 plate\_id = #plateId#  
 </isNotEmpty>  
 <isNotEmpty property="userId" prepend=",">  
 user\_id = #userId#  
 </isNotEmpty>  
 <isNotEmpty property="operatorId" prepend=",">  
 operator\_id = #operatorId#  
 </isNotEmpty>  
 <isNotEmpty property="cityId" prepend=",">  
 city\_id = #cityId#  
 </isNotEmpty>  
 <isNotEmpty property="equipmentId" prepend=",">  
 equipment\_id = #equipmentId#  
 </isNotEmpty>  
 <isNotEmpty property="cityName" prepend=",">  
 city\_name = #cityName#  
 </isNotEmpty>  
 <isNotEmpty property="stationId" prepend=",">  
 station\_id = #stationId#  
 </isNotEmpty>  
 <isNotEmpty property="connectorId" prepend=",">  
 connector\_id = #connectorId#  
 </isNotEmpty>  
 <isNotEmpty property="placeTime" prepend=",">  
 place\_time = #placeTime#  
 </isNotEmpty>  
 <isNotEmpty property="startFailReason" prepend=",">  
 start\_fail\_reason = #startFailReason#  
 </isNotEmpty>  
 <isNotEmpty property="stopReason" prepend=",">  
 stop\_reason = #stopReason#  
 </isNotEmpty>  
 <isNotEmpty property="currentQuantity" prepend=",">  
 current\_quantity = #currentQuantity#  
 </isNotEmpty>  
 <isNotEmpty property="beforeQuantity" prepend=",">  
 before\_quantity = #beforeQuantity#  
 </isNotEmpty>  
 <isNotEmpty property="afterQuantity" prepend=",">  
 after\_quantity = #afterQuantity#  
 </isNotEmpty>  
 <isNotEmpty property="changeQuantity" prepend=",">  
 change\_quantity = #changeQuantity#  
 </isNotEmpty>  
 <isNotEmpty property="chargeFee" prepend=",">  
 charge\_fee = #chargeFee#  
 </isNotEmpty>  
 <isNotEmpty property="serviceFee" prepend=",">  
 service\_fee = #serviceFee#  
 </isNotEmpty>  
 <isNotEmpty property="reduceFee" prepend=",">  
 reduce\_fee = #reduceFee#  
 </isNotEmpty>  
 <isNotEmpty property="totalFee" prepend=",">  
 total\_fee = #totalFee#  
 </isNotEmpty>  
 <isNotEmpty property="payStatus" prepend=",">  
 pay\_status = #payStatus#  
 </isNotEmpty>  
 <isNotEmpty property="chargeStartTime" prepend=",">  
 charge\_start\_time = #chargeStartTime#  
 </isNotEmpty>  
 <isNotEmpty property="chargeEndTime" prepend=",">  
 charge\_end\_time = #chargeEndTime#  
 </isNotEmpty>  
 <isNotEmpty property="scanNo" prepend=",">  
 scan\_no = #scanNo#  
 </isNotEmpty>  
 <isNotEmpty property="createTime" prepend=",">  
 create\_time = #createTime#  
 </isNotEmpty>  
 <isNotEmpty property="modifyTime" prepend=",">  
 modify\_time = #modifyTime#  
 </isNotEmpty>  
</dynamic>

🡨-where语句—>  
<dynamic prepend="where">  
 <isNotEmpty property="orderNo" prepend="and">  
 order\_no = #orderNo#  
 </isNotEmpty>  
 <isNotEmpty property="startChargeSeq" prepend="and">  
 start\_charge\_seq = #startChargeSeq#  
 </isNotEmpty>  
</dynamic>  
 </update>

### 6、批量查询:in+iterate

需求：根据多个证件号码和平台类型查多个用户

实现：

1. 须将参数封装为dto

LicenceNumbersDto{

private List<String> licenseNumbers; // '证件号码集合',

private Integer platform;//平台类型

}

1. Xml写法:

易错点：conjunction属性要加逗号；property属性为需要遍历的集合；有时候根据入参封装得不同，导致加上property属性可能会报错

<!-- 根据多个用户licenceNumber+平台类型

查询询用户是否存在-->

<select id="findUserListByLicenceNumbers" resultClass="userBase" parameterClass="licenceNumbersDto">  
 select  
 <include refid="Base\_Column\_List"/>  
 FROM  
 t\_base\_user as baseUser  
 where baseUser.platform=#platform# and baseUser.license\_number in  
 <iterate property="licenseNumbers" conjunction ="," open="(" close=")">  
 #licenseNumbers[]#  
 </iterate>  
</select>

### 7、批量查询

**需求**：

根据用户名、手机号、平台类型来查询用户信息，用户名、手机号、平台类型可有传入多组；用户名和平台类型为必填字段，手机号可能为null。

**实现方式一**（**失败）**：iterate+<isNotEmpty>

1. 失败原因

由于手机号可能为空，在iterate嵌套<isNotEmpty>时，<isNotEmpty>中使用的property属性失败。

报错如下：

1. Error getting ordinal list from JavaBean. Cause java.lang.StringIndexOutOfBoundsException: String index out of range: -1

其写法如：

1. <!--根据用户名、平台、手机号查询用户信息-->
2. **<select** id="findUserListByNameMobile" resultClass="userBase" parameterClass="java.util.List"**>**
3. select
4. **<include** refid="Base\_Column\_List"**/>**
5. FROM
6. t\_base\_user  as baseUser
7. where
8. **<iterate**   conjunction="or" **>**  //此处不能使用property="list"
9. baseUser.name = #list[].name# and
10. baseUser.platform\_type = #list[].platformType#
11. <!—此处写入property会报错，解析不了list[].mobile；如果不使用，手机号为null的行数据查不到，导致数据不正确 -->
12. **<isNotEmpty**  property="list[].mobile" prepend="and"**>**
13. baseUser.mobile = #list[].mobile#
14. **</isNotEmpty>**
15. **</iterate>**
16. **</select>**

2) Dto写法

1. private String name;// '用户/企业名称',
2. private String mobile; // '手机号',
3. private Integer platformType;//平台(1:xxx、2:xx),

**实现方式二（成功）**iterate+字符串拼接instr()

解决难点：

无法在iterate使用<isNotEmpty>中property属性的问题

So:

如何在iterate中判断手机号为空时只用用户名和平台类型来查询。

1)封装dto

1. **public** **class** JointStrDto **implements** Serializable {
2. **private** **static** **final** **long** serialVersionUID = -xxxx;
3. /\*\*
4. \* 此字符串由以下三个字段拼接而成，用'\_'连接
5. \*      private String name;// '用户/企业名称',
6. \*     private String mobile; // '手机号',
7. \*     private Integer platformType;//平台(1:xxx、2:xx),
8. \*
9. \*     其格式如：
10. \*          name\_platformType\_mobile
11. \*/
12. **private**  String  jointStr;
14. **public** String getJointStr() {
15. **return** jointStr;
16. }
18. **public** **void** setJointStr(String jointStr) {
19. **this**.jointStr = jointStr;
20. }

2)Dao层

1. **public**  List<UserBase>  findUserListByNameMobile(List<JointStrDto> list){
2. **return** (List<UserBase>) **super**.queryForList(**this**.getStatement("findUserListByNameMobile"), list);
4. }

3)Mapper.xml

1. <!--根据用户名、平台、手机号查询用户信息-->
2. **<select** id="findUserListByNameMobile" resultClass="userBase" parameterClass="java.util.List"**>**
3. select
4. **<include** refid="Base\_Column\_List"**/>**
5. FROM
6. t\_base\_user  as baseUser
7. where
8. **<iterate**   conjunction="or" **>**
9. (
10. instr(#list[].jointStr#, name)**>**0 and
11. instr(#list[].jointStr#, platform\_type)**>**0 and
12. <!—此处假如case when判断语句，null需用单引号包裹，否则查不出mobile为空的数据 -->
13. instr(#list[].jointStr#,(case when mobile is null then 'null' else mobile end))**>**0
14. )
15. **</iterate>**
16. **</select>**

4)简要分析

a、将用户名、平台类型、手机号拼接成字符串进行封装在JointStrDto中，入参为(List<JointStrDto> list)。

b、instr(str, columnName)表示：此行列名对应的数据中是否被str包含。换句话说，假如instr(‘周星驰\_2\_187999999’, name),假如此行数据字段为name的内容为‘周星驰’，则返回大于0，否则不满足。

c、 关于 case when语句

1. <!—此处case when表达意思：当mobile字段数据为null时，结果为'null'字符串，否则为字段对应的数据-->
2. <!-由于instr()中第二个参数要与第一个参数（字符串）比较，则第二个参数也必须为字符串，所以null需用单引号包裹-->
3. instr(#list[].jointStr#,(case when mobile is null then 'null' else mobile end))**>**0

### 8、[ibatis中使用like模糊查询](https://www.cnblogs.com/gaojing/archive/2012/11/22/2844932.html)

无效的方法：

select \* from table1 where name like '%#name#%'

 两种有效的方法：

1) 使用$代替#。此种方法就是去掉了类型检查，使用字符串连接，不过可能会有sql注入风险。

select \* from table1 where name like '%$name$%'

 2) 使用连接符。不过不同的数据库中方式不同。

mysql：

select \* from table1 where name like concat('%', #name#, '%')

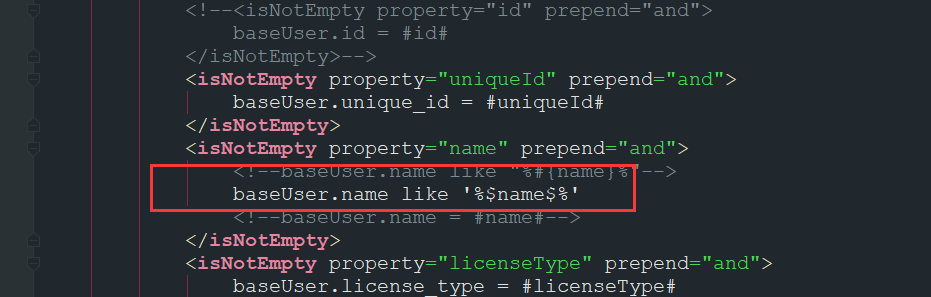
 oracle:

select \* from table1 where name like '%' || #name# || '%'

 sql server:

select \* from table1 where name like '%' + #name# + '%'

如图所示，亲测有效



9、mybatis批量更新

<!--修改 根据dkid -->

<**update** id="updateDkInfoDkid" parameterType="java.util.List">  
 <**foreach** item="item" collection="list"  
 open="" separator=";" close="">  
 update dk\_info  
 <**set**>  
 <**if** test="item.userId != null">  
 user\_id=#{item.userId },  
 </**if**>  
 <**if** test="item.userPhone != null">  
 user\_phone=#{item.userPhone },  
 </**if**>  
 <**if** test="item.vin != null">  
 vin=#{ item.vin },  
 </**if**>  
 <**if** test="item.dkType != null">  
 dk\_type=#{ item.dkType },  
 </**if**>  
 <**if** test="item.dkdNo != null">  
 dkd\_no=#{ item.dkdNo },  
 </**if**>  
 <**if** test="item.dkId != null">  
 dk\_id=#{ item.dkId },  
 </**if**>  
 <**if** test="item.dkData != null">  
 dk\_data=#{ item.dkData },  
 </**if**>  
 <**if** test="item.dkStatus != null">  
 dk\_status=#{ item.dkStatus },  
 </**if**>  
 <**if** test="item.dkStartTime != null">  
 dk\_start\_time=#{item.dkStartTime },  
 </**if**>  
 <**if** test="item.dkEndTime != null">  
 dk\_end\_time=#{ item.dkEndTime },  
 </**if**>  
 <**if** test="item.dkPermissions != null">  
 dk\_permissions=#{item.dkPermissions },  
 </**if**>  
 <**if** test="item.createTime != null">  
 create\_time=#{item.createTime },  
 </**if**>  
 <**if** test="item.updateTime != null">  
 update\_time=#{item.updateTime }  
 </**if**>  
 </**set**>  
 where dk\_id =#{item.dkId}  
 </**foreach**>  
  
</**update**>

1）collection="list" 此处只能写list

2）<if test="dkInfo.userId != null">

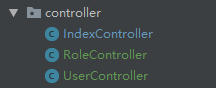
user\_id=#{dkInfo.userId },

</if>

这两处都要写dkInfo

## 三、springMVC

1.同样的功能（此处使用的是登录方法）写在IndexController下可以，但是写在UserController中不行？

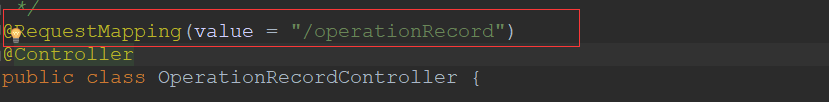


2.访问不到contolller，在配置上可能有以下2中原因

1）在web.xml中没有配置前端控制器DispatcherServlet类（在servlet标签中）

2）在spingMCV.xml中没有配置扫描组件<context:component-scan>或注解驱动<mvc:annotation-driven>

3.编写controller时，为了防止和他人的方法重名，在注解上，除了对方法注解外，需对类注解



4.关于获取不到前端参数问题，在注解上可能有以下几种方式解决。

1）注解@RequestBody，在参数处



1. 注解@Parameter()

5.serviceImpl忘了implements service接口

6.serviceImpl类忘了注解@Service

7. bean重复

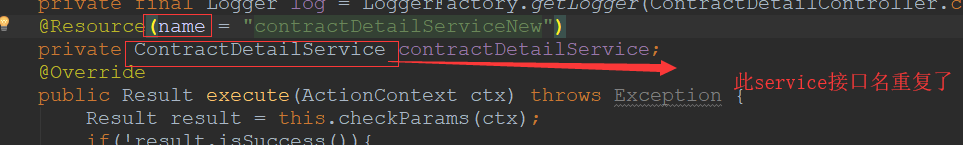
问题：在书写service时，接口service类名、service注解重复。在引用时会报错bean冲突，接口名重复。

原因：由于在引用时是使用接口引入，所以如果接口名相同会造成spring容器不知道去找哪个实现类。

解决：

1、@service("xxxNew")

2、如果不修改接口名，须在引入时说明：@Resource(name="xxxNew")



## 四、spring

1、bean重复：

问题：在书写service时，接口service类名和serviceImpl注解重复。在启动时会报错bean冲突，接口名重复。 ​

原因：由于在引用时是使用接口引入，所以如果接口名相同会造成spring容器不知道去找哪个实现类。 ​

解决：

1. @service("xxxNew");
2. 如果不修改接口名，须在引入时说明：@Resource(name="xxxNew"

2、在引入Bean存在错误时，

如： ​

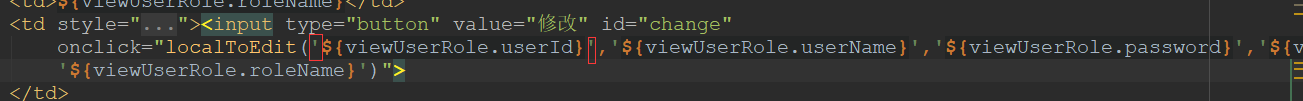
@Autowired ​

private UserDao xxx; ​

xxx最好写成userDao，否则可能存在导入错误

## 五.JSP&EL&JS&JQuery

1.JS函数传参又多个参数时，需要使用单引号处理。

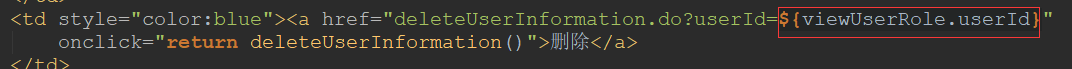


2从前端通过“.do”来加后缀传参为什么不能传对象，而只能穿具体的值

错误：传对象



正确：传值



问题：如果后端直接想获取对象，前端该如何处理？？springMVC直接可以？还是必须在form表单中写上对应的id与name？

3.关于<c:if> 标签tset=“”书写规则，已经错了好几次(谨记)。

正确：



错误：



4.关于EL表达式取值时一定注意与实体entity一致(易错)。

5．使用Jquery时需注意以下几点。

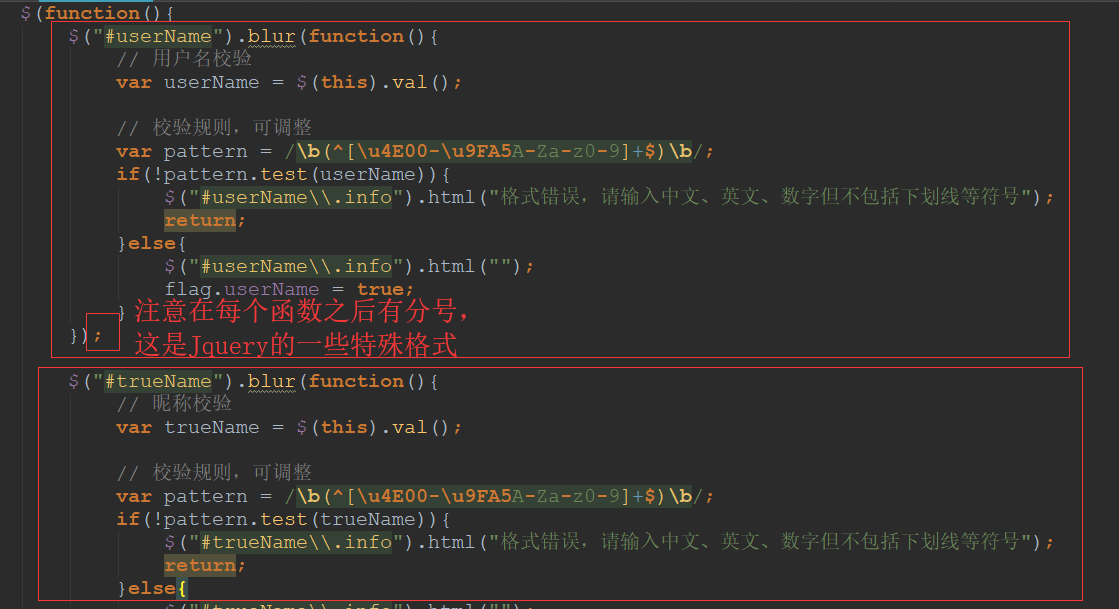
（1）包的引入 注意地址



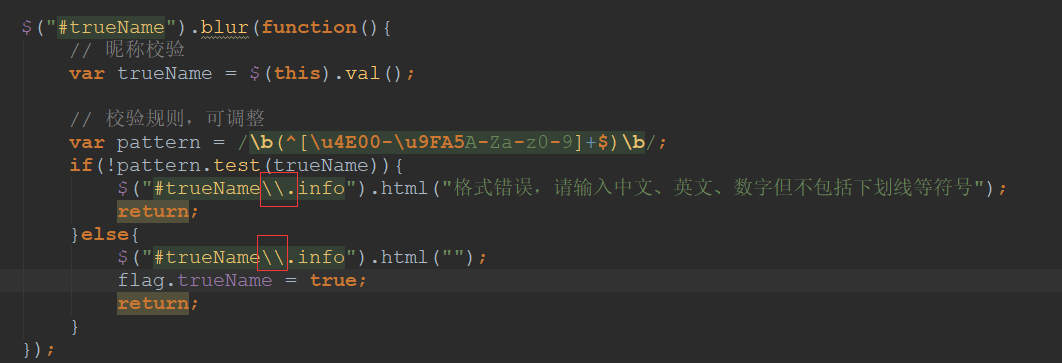
（2）$(function(){});

末尾有分号

（3）每个小的函数之后都包含有分号

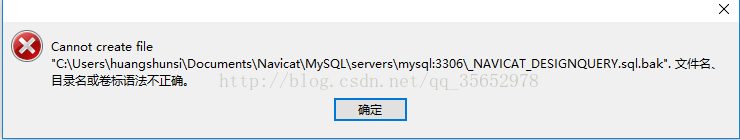


（4）如下所示，不知为何需要在id间加入“\\”才能成功,猜测可能是因为id中包含逗号，需要“\\”来作转义符。



## 五.数据库

1. Navicat 连接MySQL数据不能新建查询，提示文件名、目录名或卷标语法不正确。



原因：

出现该问题的原因的是数据库连接名出现了：分号，而Windows不允许文件名中包含分号，导致Navicat不能创建该目录；

解决方法：

删除原来的连接，新建一个，并且连接名只能包括字母数字下划线等等Windows允许的文件名即可

2.操作数据库时注意时间问题，默认为00.00.00 00:00映射到结果时会报错。

原因：空时间在映射到结果时会出现转换错误

解决方法：测试时，在数据库里修改00.00.00 00:00为其他时间，不要空就行。

3、字段设计

**基本：**

1) 在创建字段并且不设置默认值的时候，mysql都会默认给字段加上默认值null

2) \*\***NULL表示“此字段值为空”，而”表示“此字段暂未设置值”**\*\*

**实际设计**：

1. 如果字段为int 或者varchar类型，那么最好设置字段类型为 not null 并且设置default 0 或者default ''
2. 因为在业务中使用innoDB多一些，所以尽量还是用varchar（）好一些
3. varchar（N）的N取值范围逻辑如下：

4.0版本以下，varchar(50)，指的是**50字节**，如果存放UTF8汉字时，只能存16个（每个中文3字节）

5.0版本以上，varchar(50)，指的是**50字符**，无论存放的是数字、字母还是UTF8中文（每个中文3字节），都可以存放50个

N取值范围为65535，超过了自动转换为text类型被强转成text类型

1. 如果字段为text,则既不需要设置not null,也不需要手动设置default 的值
2. 关于text字段不能有默认值的问题，这个只针对于手动增加的default属性，如果创建字段不设置default的话，mysql会默认加上一个默认值Null

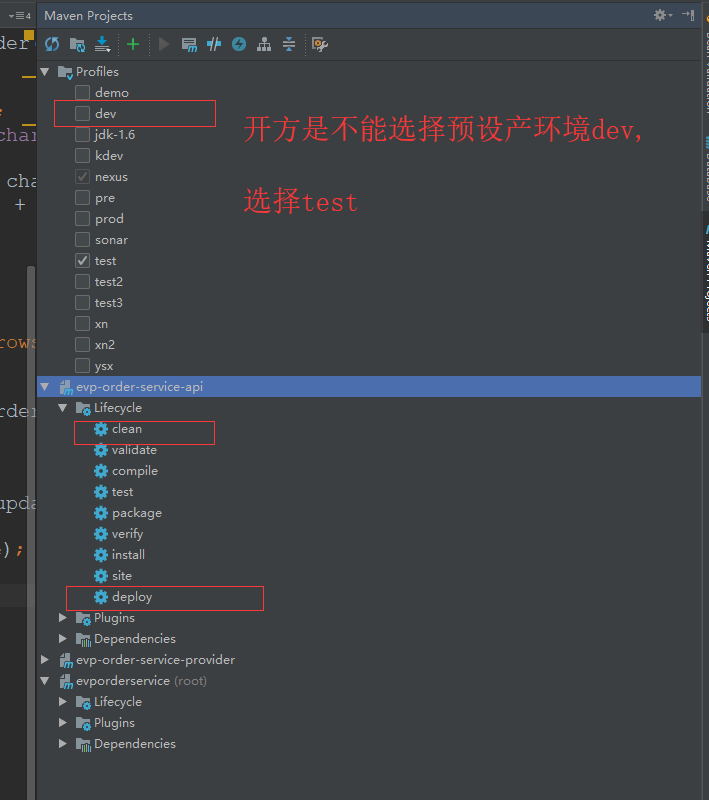
4、mysql修改表结构(alter table)，多列/多字段

1. -**add**支持多列，change/**drop**需要在每列前添加关键字，逗号隔开，'column'可有可无
3. --添加多列
4. **alter** **table** test **add** (c1 **char**(1),c2 **char**(1));   --正确，add支持多列
5. **alter** **table** test **add** **column** (c1 **char**(1),c2 **char**(1));    --正确
6. **alter** **table** test **add** c1 **char**(1),**add** c2 **char**(1);     --正确
8. --修改多列
9. **alter** **table** test change c1 c3 **char**(1),change c2 c4 **char**(1);     --正确
10. **alter** **table** test change **column** c1 c3 **char**(1),change **column** c2 c4 **char**(1);       --正确
11. --name关键字作为字段名，重命名需要加反引号(`)
12. **alter** **table** table\_name change `**name**` field\_name **varchar**(50);
14. **alter** **table** test change (c1 c3 **char**(1),c2 c4 **char**(1));      --错误
16. --删除多列
17. **alter** **table** test **drop** c1,**drop** c2;   --正确
18. **alter** **table** test **drop** **column** c1,**drop** **column** c2;     --正确
20. **alter** **table** test **drop** c1,c2;    --错误
21. **alter** **table** test **drop** (c1,c2);  --错误
22. ---------------------

## 六、Maven

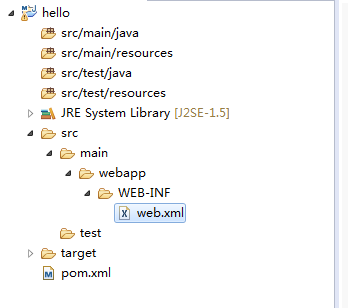
1. 当重新部署环境后，需要clean与deloy

在开发时，选择test环境，而不选dev预生产环境



2.处理红色叉号

**手动**在webapp文件夹下创建一个WEB-INF文件夹，在里面放一个web.xml文件



3、处理编译版本

在pom.xml中添加如下代码

<build>

<!-- 配置了很多插件 -->

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.5.1</version>

<configuration>

<source>1.7</source>

<target>1.7</target>

<encoding>UTF-8</encoding>

</configuration>

</plugin>

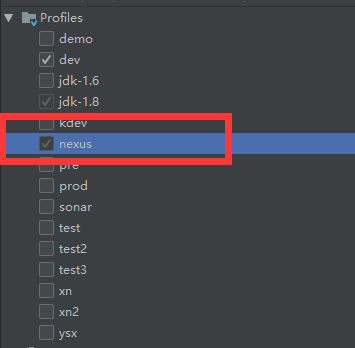
</plugins>

</build>

4、测试不通接口？配置文件没有转义，出现：${maven.xxx.xxx}

maven管理的配置文件需要打包才能

5、拉取包失败：记得勾选nexus，去私服去拉包，否则可能只会拉取本地的。



## 七、git版本控制

1.提交代码时出现手动merge情况，导致自己和别人的代码缺失？

原因：git提交时与他人发生冲突

解决：git前先更新(pull)，再执行commit、push

2. 工作区处于冲突状态，无法在进行提交操作。

解决办法有两个:

一是使用git reset放弃合并操作;

二是打开冲突的文件，进行手工编辑解决冲突的部分，之后再进行提交。

## 八、远程service层

1．入参出参校验问题，如何全面校验？

*\* <p>1. 检测参数是否异常，如果异常，则返回错误并打印日志</p>*

*\* <p>2. 组装查询参数</p>  
\* <p>3. 检测订单是否存在，如果不存在，则返回错误并打印日志</p>  
\* <p>4. 判断订单状态是否合法，如果合法则返回合法的订单状态，如果不合法则返回错误并打印日志</p>  
\*/*

具体代码示例：

@Override

public Result<ChargeOrderOrderStatusReDTO> queryChargeOrderStatus(ChargeOrderOrderStatusDTO dto) {  
 //1.检查参数是否正常，如果异常，则返回错误并答打印日志  
 logger.info("ChargeOrderOrderStatusDTO"+dto);  
 if(dto==null){  
 logger.error(*CHAGRE\_COMMON\_PARAM\_ERROR*.getMsg());  
 return Result.*getServiceError*(*CHAGRE\_COMMON\_PARAM\_ERROR*.getMsg(), *CHAGRE\_COMMON\_PARAM\_ERROR*.getCode());  
 }  
  
 //2.组装查询参数，如果订单编号不存在(注意：此处只要存在本地或第三方订单任意一个即可)，则返回错误信息  
 List<ChargeOrder> chargeOrderList=null;  
 if((dto.getOrderNo() != null&& !"".equals(dto.getOrderNo()))||(dto.getStartChargeSeq() != null&& !"".equals(dto.getStartChargeSeq()))){  
 ChargeOrder chargeOrder=new ChargeOrder();  
 if(dto.getOrderNo() != null&& !"".equals(dto.getOrderNo())){  
 chargeOrder.setOrderNo(dto.getOrderNo());  
 }  
 if(dto.getStartChargeSeq() != null&& !"".equals(dto.getStartChargeSeq())){  
 chargeOrder.setStartChargeSeq(dto.getStartChargeSeq());  
 }  
 chargeOrderList = chargeOrderStatusService.queryChargeOrderStatus(chargeOrder);  
 }  
  
 //3.检查订单是否存在，如果不存在，则返回错误日志并打印  
 if(chargeOrderList==null || chargeOrderList.isEmpty()){  
 logger.error(*CHARGE\_ORDER\_NOT\_EXIST*.getMsg(), dto);  
 return Result.*getBusinessException*(*CHARGE\_ORDER\_NOT\_EXIST*.getMsg(), *CHARGE\_ORDER\_NOT\_EXIST*.getCode());  
 }  
  
 //4.判断订单状态是否合法，如果合法则返回合法的订单状态，如果不合法则返回错误并打印日志  
 ChargeOrderOrderStatusReDTO reDTO=new ChargeOrderOrderStatusReDTO();  
 for(OrderStatusEnum status : OrderStatusEnum.*values*()){  
 if(chargeOrderList.get(0).getOrderStatus().equals(status.getCode())){  
 reDTO.setOrderStatus(status.getCode());  
 reDTO.setDescribe(status.getDescribe());  
 break;  
 }  
 }  
 if(reDTO.getOrderStatus()==null){  
 logger.error(*CHARGE\_STATUS\_NOT\_EXIST*.getMsg(), reDTO);  
 return Result.*getBusinessException*(*CHARGE\_STATUS\_NOT\_EXIST*.getMsg(), *CHARGE\_ORDER\_NOT\_EXIST*.getCode());  
 }  
  
 return Result.*getSuccessResult*(reDTO);  
}

## 九、综合

1.编写功能时，大致总结流程如下：

Jsp——controller(确定功能名称)——xml——mapper接口——service接口、实现——controller

2.每次写一个接口都要写一套(本地service+dao+mapper.xml),特别是有些类似功能代码重复性更高，从而降低开发效率？

原因：本地servie复用性不高

解决：在service层提高入参出参类型(对象)；另外，在ibatis中利用dynamic动态语句。

3.在文件上传时出现的问题：

（一）不能简单有效地获取普通表单项。

在文件上传并不太难，难的是表单中除了提交文件外还要提交其他信息，可能导致获取前端数据的困难，以及如何保持一次请求(调用commonsMultipartResolver.resolveMultipart(httpRequest); 方法时，原request会消亡！)；

当表单<form>标签加入enctype="multipart/form-data"后，不能使用request.getParametre()传值

（二）编写代码的混乱

发现编码十分困难。

后端：既要单独书写文件上传代码，也要书写controller获取其他前端数据，

前端：考虑前端如何写才能让后端

（三）解决方法

直接使用ajax处理解决这些复杂的问题

$.ajax({

id: '',

url : '${\_ctx}/index/addUserInformation.do',//1.do的调用

type: 'post',

data : $('#addFrom').serialize(),//2.将数据直接系列化封装（)

dataType : 'json',

success : function () {

//3.处理返回数据

}

})

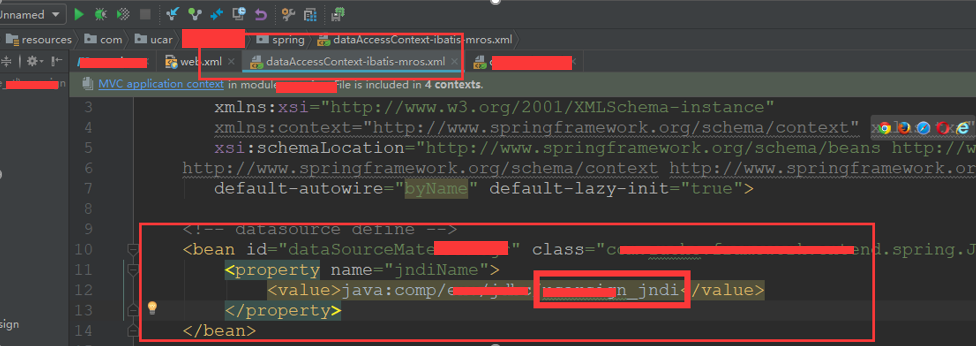
## 十、项目启动

1.jndi配置(数据源配置)

Apache-tomcat下的context.xml

Name值；url值





2、tomcat服务启动的问题

关于定位服务启动的问题， tomcat 有个localhost 日志，关于 listern fitler 等容器启动过程中的错误信息都会输出到 localhost 里面，所以 如果发现tomcat 没启动成功 第一个想到看这个日志。

其实，我们的应用都是 单部署 一个 应用，这个localhost 可以默认输出到 catalina 日志最好这个需要运维上面统一

3、启动服务一直处在等待状态

**提示**：Artifact xxx:war exploded: Artifact is being deployed, please wait...

**原因**：自动编译失败，有些配置文件没有完成编译。

**解决办法**：尝试手动编译（多次可能是)

Clean>Install/package

## 十一、日志与异常

1. 关于日志和异常：在前期编写的时候出现的问题包括有：有些地方重复地校验参数、日志输出太笼统该详细到错误点、异常抛出太详细该笼统(参数错误、用户创建等模块性错误)。
2. 日志输出
3. 日志级别使用error
4. 日志描述直接**手动描述**：功能+错误类型+错误详细原因，不用自己定义的错误枚举。

如：logger.error("创建用户参数错误：用户名为空");

1. 异常输出
2. 统一错误码Code+详细错误定义Message

如：

logger.error("创建用户参数错误：用户类型错误");

//参数错误码+‘用户类型错误’ BusinessException.throwMessageWithCode(CodeEnum.PARAM\_CHECK\_ERROR.getCode().toString(),CodeEnum.USER\_CHECKPARAM\_CATEGORY\_NONE\_ERROR.getMessage());}

## 十二、测试

1、测试修改代码时，配置文件等有时候不会自动编译，常出现文件找不到、qlStatement不存在等错误。开启热部署或及时手动编译。

2、在进行junit test时，@IbatisAliasConfig(alias = "xxx")会失效，如果要使用别名，需单独配置。

如：<sqlMap><typeAlias alias="xxx" type="com.xxx.xxx.user.beans.xxx" /></sqlMap>

## 十二、编码规范

后端

1. 在书写代码时，一个方法不宜太长，可以抽取方法、优化结构。
2. 如果参数验证比较多，可以单独写一个方法。
3. 开发中基本都时协同开发，开发时，则先做自己的逻辑，如果需要等待使用他人模块则留白**todo**
4. 开发时应当遵守循环渐进的编码方式：搭建逻辑框架、填补参数验证、填补业务逻辑。并且适当抽取方法。

前端后端交互系统

1. 提前设计好需要的页面、接口，并统一后端入参出参、前端请求获取等参数格式
2. 在实际开发时，前后端的开发顺序可根据难度来来做评估。一般来说，做了简要的前端界面后，便可进行后端接口的开发，再返回前端进行显示控制。