重复提交处理：

前台；

后台；

Struts2的令牌机制；

Springmvc token机制；

适用：

所有表单提交的操作，都需要重复提交处理；前台js，后台代码(包括，可能在存储过程中处理)；

//让当前的线程睡眠5秒钟，模拟网络延迟而导致表单重复提交的现象

Thread.*sleep*(5\*1000);

//////////////////////////////////////////////////////////////////////

前台：

1，js中处理，点击按钮后禁用按钮，防止再次点击；

1），按钮：

<input onclick="goto\_register()" type=*"button"* name=*"Submit1"* id=*"subbtn"* class=*"login\_btn"* value=*"提交注册"* />

$('#subbtn').attr("disabled","disabled");

恢复，

$('#subbtn').removeAttr("disabled");

或，

$("#btnVIPAppro").removeAttr("onclick");

$("#btnVIPAppro").attr("onclick","forsubmit();"); //恢复；

2），超链的：

<a id=*"registerHref"* href=*"javascript:void(0);"* onclick="goto\_register();">提交注册</a>

$('#registerHref').attr("onclick", "");

恢复，

$('#registerHref').attr("onclick", "goto\_register();");

后台：

只有有并发冲突的，才需要如此操作，没有的不需要；

1，发起债权转让的投标记录，不能重复发起； | 本质是修改添加，使用加锁方式；

方式1：sql语句加锁；

查询数据，加锁；---事务完成自动解锁语句。

<select id=*"lockCollectionRecordByTenderId"* parameterType=*"java.lang.Integer"* resultMap=*"com.cxdai.portal.invest.mapper.AccCommonCollectionMapper.accCommonCollectionResultMap"* >

SELECT \* FROM rocky\_b\_collectionrecord rbc WHERE rbc.TENDER\_ID = #{tenderId} and rbc.`ORDER`=#{transferBeginOrder} and rbc.STATUS=0 LIMIT 0,1 for update

</select>

// 锁住代收表中 第一条待还记录 的期数是不是原先期数 ，数据库加锁防止并发 转让操作，事务不结束，锁不解开

CommonCollectionVo conCollectionVo = bTransferMapper.lockCollectionRecordByTenderId(bTransferVo.getTenderId(), bTransferVo.getTransferBeginOrder().intValue());

// 判断期数 和 转让期数是不是同一期

**if** (conCollectionVo == **null**) {

**throw** **new** RuntimeException("该笔代收已经不存在,不能发起转让");

}

**if** (exists TransferRecordBytenderId(bTransferVo.getTenderId())) {

**throw** **new** RuntimeException("该投标已经被转让");

}

// 插入转让记录

bTransferMapper.insert(bTransferVo);

~

另：

存储过程中，同样对预修改的数据，先查询，添加for update；

方式2：悲观锁；代码中加锁；

hibernate的锁：

LockMode.UPGRADE ：利用数据库的for update子句加锁。

Hibernate的悲观锁，也是基于数据库的锁机制实现。 下面的代码实现了对查询记录的加锁：

1  String hqlStr  =   " from TUser as user where user.name=’Erica’ " ;   
2 Query query  =  session.createQuery(hqlStr);   
3 query.setLockMode( " user " ,LockMode.UPGRADE);  // 加锁    
4  List userList  =  query.list(); // 执行查询，



获取数据 query.setLockMode 对查询语句中特定别名所对应的记录进行加锁（我们为 TUser类指定了一个别名“user”），这里也就是对返回的所有user记录进行加锁。 观察运行期Hibernate生成的SQL语句：

1  select tuser0\_.id as id, tuser0\_.name as name, tuser0\_.group\_id as group\_id, tuser0\_.user\_type as user\_type, tuser0\_.sex as sex from t\_user tuser0\_ where (tuser0\_.name = ’Erica’ )  for  update



 这里Hibernate通过使用数据库的for update子句实现了****悲观锁****机制。

2，单纯添加的操作：

方式1： //存在userid相同，已有记录type=0,status=0，则不能再添加；

//代码中的先查询是否存在数据(查询userid,type,status)，没控制住；

mybatis的xml中，添加语句执行前，先判断是否有锁定语句，可以保证去除重复；

insert into rocky\_bankinfo\_log(USER\_ID, CARDNUM, TYPE, STATUS, ADD\_BY, ADD\_TIME, REMARK)

select \* from (

select 20884 as USER\_ID, '0' as CARDNUM, 0 as TYPE, 0 as STATUS, 0 as ADD\_BY,'2014-12-2','系统锁定'

) t

where not exists (select 1 from rocky\_bankinfo\_log c where c.USER\_ID = t.USER\_ID and c.TYPE = 0 and c.STATUS = 0);

~

<insert id=*"insert"* parameterType=*"com.cxdai.base.entity.BankinfoLog"* >

insert into rocky\_bankinfo\_log (USER\_ID, CARDNUM,

TYPE, STATUS, ADD\_BY, ADD\_TIME,

REMARK)

select \* from (

select #{userId,jdbcType=INTEGER} as USER\_ID, #{cardNum,jdbcType=VARCHAR} as CARDNUM,

#{type,jdbcType=TINYINT} as TYPE, #{status,jdbcType=TINYINT} as STATUS, #{addBy,jdbcType=INTEGER} as ADD\_BY,

#{addTime,jdbcType=TIMESTAMP} as ADD\_TIME,#{remark,jdbcType=VARCHAR} as REMARK

) t

where not exists (select 1 from rocky\_bankinfo\_log c where c.USER\_ID = t.USER\_ID and c.TYPE = 0 and c.STATUS = 0);

</insert>

方式2：session令牌方式；

·进入页面前，| 创建session，页面控件绑定值；

String token = System.*currentTimeMillis*()+**new** Random().nextInt()+"";

request.getSession().setAttribute("token", token);

mav.addObject("c\_token", token);

·页面，

<input type=*"hidden"* id=*"c\_token"* name=*"c\_token"* value=*"*${c\_token}*"*>

·Js，

$.ajax({

url : "${path}/bankinfo/lockBankCard.do",

data : {'c\_token' : c\_token

·后台添加操作，

//锁定银行卡新增功能 | 比较页面传递的参数和session中的值，

**public** MessageBox lockBankCard(HttpServletRequest request,HttpSession session) {

String c\_token = request.getParameter("c\_token");

String s\_token = (String) request.getSession().getAttribute("token");

**try** {

**if** (c\_token!=**null** && s\_token!=**null**) {

**if**(c\_token.equals(s\_token)){

msg = bankInfoService.insertBankCardLock(shiroUser.getUserId());

**if** (msg.equals("success")) {

**return** MessageBox.*build*("1", "锁定成功.");

} **else** {

**return** MessageBox.*build*("0", msg);

}

}

**else** {

**return** MessageBox.*build*("0", "不能重复提交");

}

}**else**{

**return** MessageBox.*build*("0", "锁定失败..");

}

} **catch** (Exception e) {

logger.error("锁定银行卡操作失败.", e);

msg = "锁定失败";

e.printStackTrace();

}**finally**{

request.getSession().removeAttribute("token");

}

再优化：

一个用户只有一个session，当session存在时，不可在添加；

就可以确保不能同时点击了；

~session令牌方式，可以解决；---只是或者还需要其它一些写法之类；

实例：

债权转让撤销：---防止重复：

在存储过程中，

需要修改状态的语句，加锁；并：再判断一次状态正确；不正确的，跳出；

for update的作用域：

spring配置service事务的，应该是在service执行完后，自动释放；

存储过程中的，应该也是在所在的service执行完后，自动释放；

~

在mysql中的select \* for update是写锁，当指明索引时是行锁，否则是表锁。

所以，加锁前，语句必须有索引的where id 条件；

索引不明确的(>、< 等符号)，加表锁；

~

表锁，锁定语句太多，容易造成其它地方的错误，不使用；

///////////////////////////////////