环境要求

> 版本要求
>
> - TOMCAT 9.0以上版本
> - MAVEN 3.6.0以上版本
> - JDK 1.8.0 (建议再安装个11版本)
>
> 其他工具
>
> - IDEA **没有版本要求，能用就行**
> - Navicat premium 版本 （能连接Oracle数据库）
> - Oracle
>
> 建议笔记工具
>
> - Notepad++
> - Typora

> WIFI
>
> **WIFI名：jinqiu**
>
> **密码：GingK00#28o8&1o**

前提需要

- **Oracle数据导入**

> 从远程库中拉取数据到本地
>
> Oracle导出远程数据库到本地ℕ*acos的博客-CSDN博客*oracle远程导出数据到本地

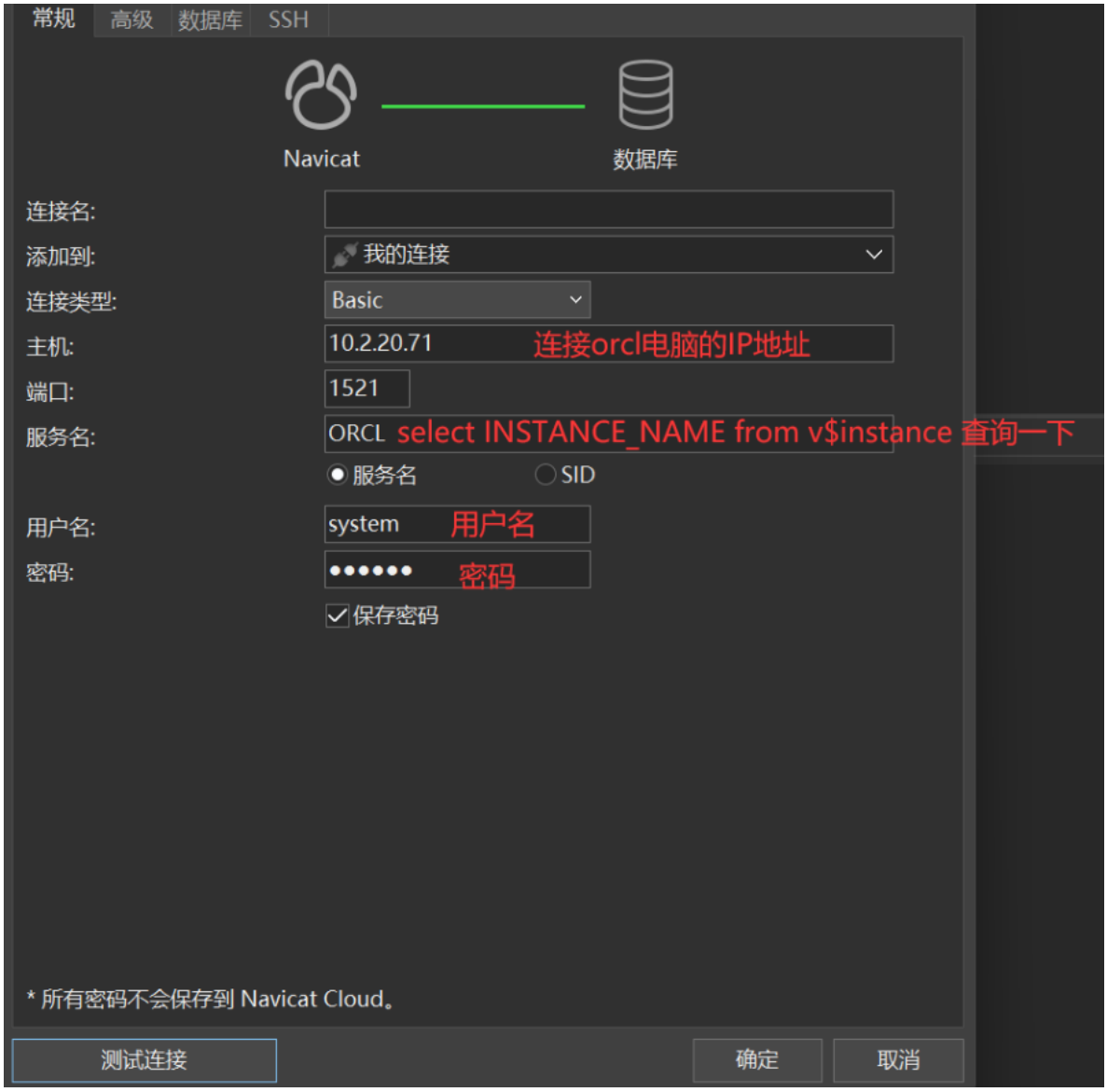
```
exp east3_bnpp/east3_bnpp@10.1.3.20:1521/oradb file=E:/ORCL_DB/GingKoo_DB.dmp
```

数据文件

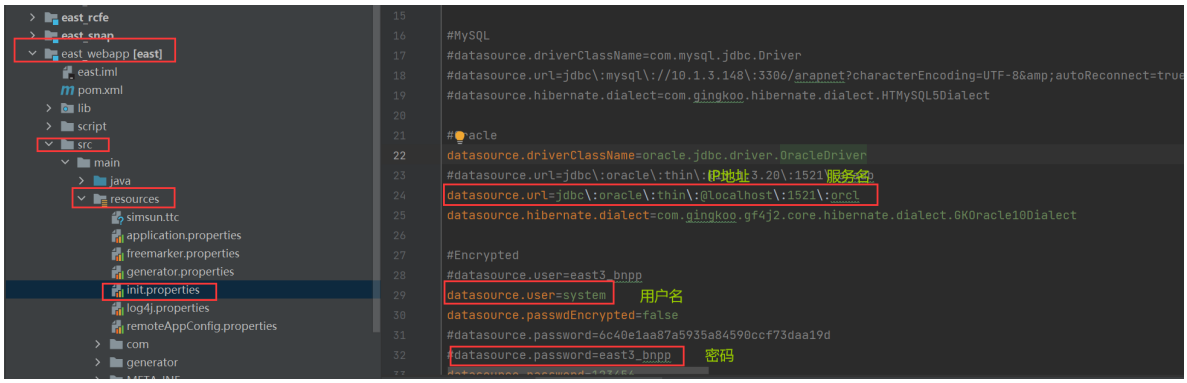
> 服务名查询语句 获取服务名
>
> ```
> select INSTANCE_NAME from v$instance
> ```

```
imp system(用户名)/123456(密码)@localhost:1521/orcl(服务名)
file=d:/Oracle/GingKooTest_db/GingKoo_db.dmp
(GingKoo_db.dmp的下载地址) full=y
```

- **NaviCat连接Oracle数据库**



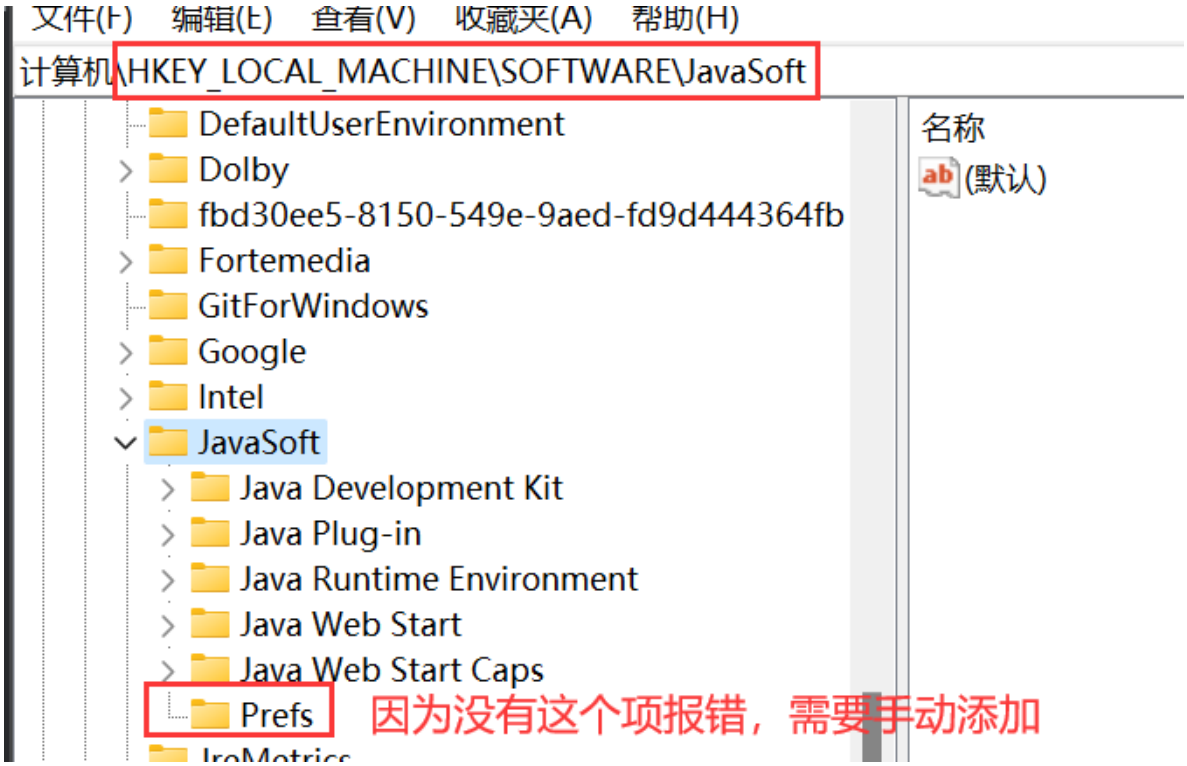
项目启动时，更改项目Oracle数据库的连接地址



- **成功启动East项目**

> 若启动时，报错如下时。需要对本机的注册表进行修改

```
org.springframework.web.context.ContextLoader 2022-02-15 16:02:30,293-- ERROR --
Context initialization failed
org.springframework.beans.factory.UnsatisfiedDependencyException: Error creating
bean with name 'eastModuleInfoProvider': Unsatisfied dependency expressed
through field 'home'; nested exception is
org.springframework.beans.factory.BeanDefinitionStoreException: Cannot access
specified node path [application.web.home:]; nested exception is
java.util.prefs.BackingStoreException: Could not open windows registry node
Software\JavaSoft\Prefs at root 0x80000002.
```

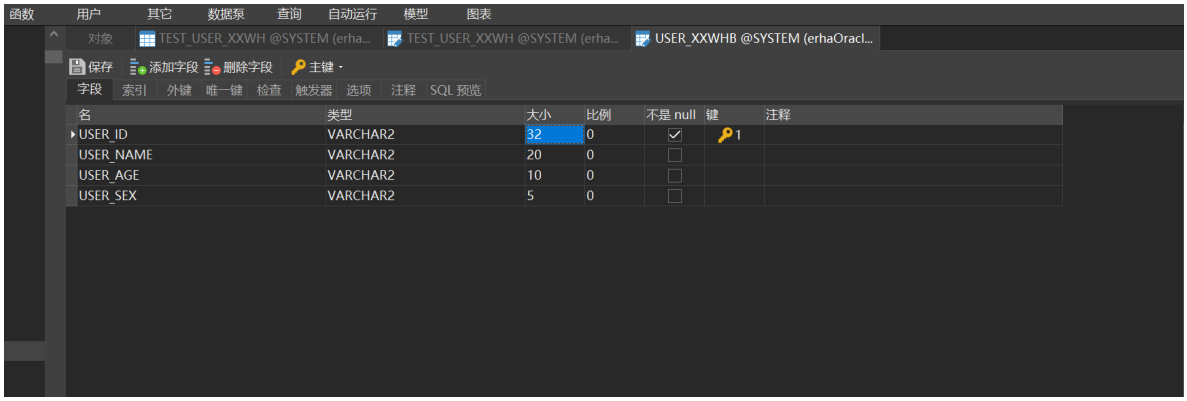


> 找到对应路径：HKEY_LOCAL_MACHINE\SOFTWARE\JavaSoft
>
> 在JavaSoft文件夹下，添加一个项Prefs

页面导入开发流程

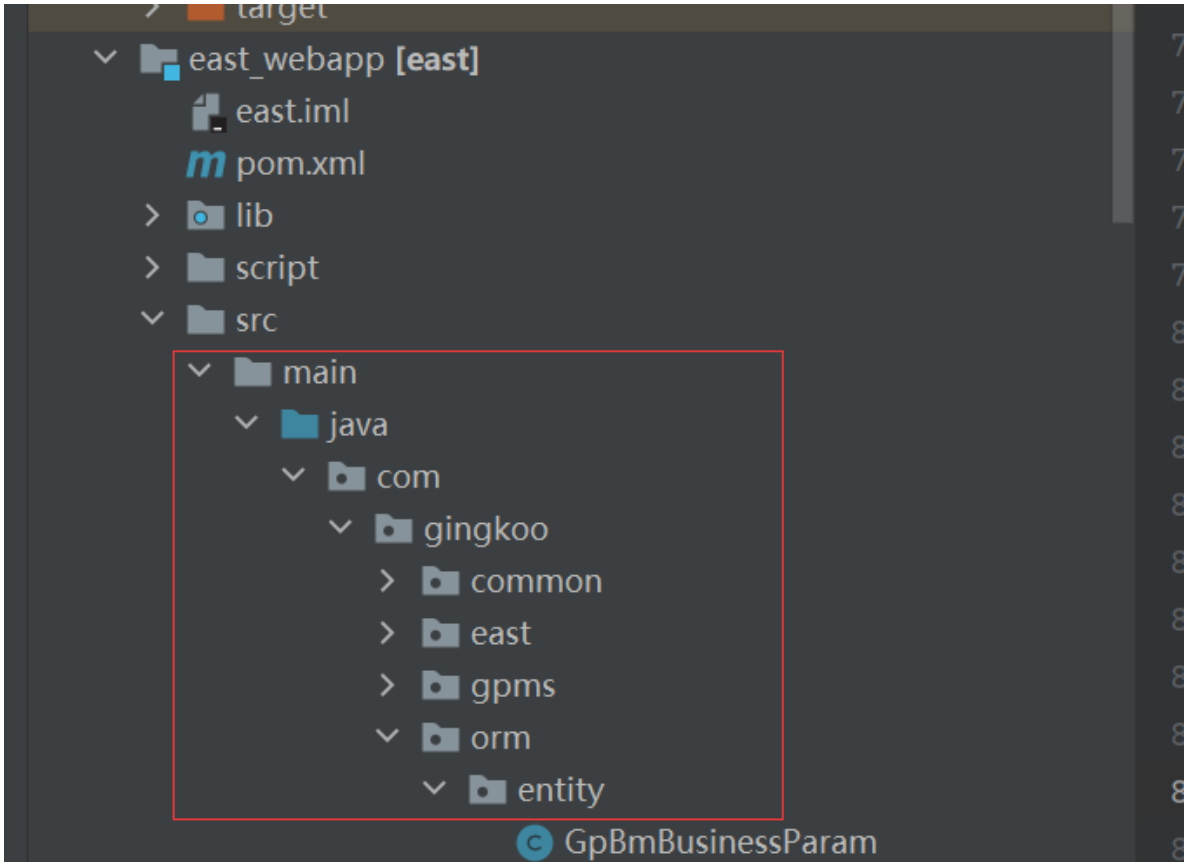
1、创建数据表

> 根据提供的表信息，将其在数据库中完成表的创建



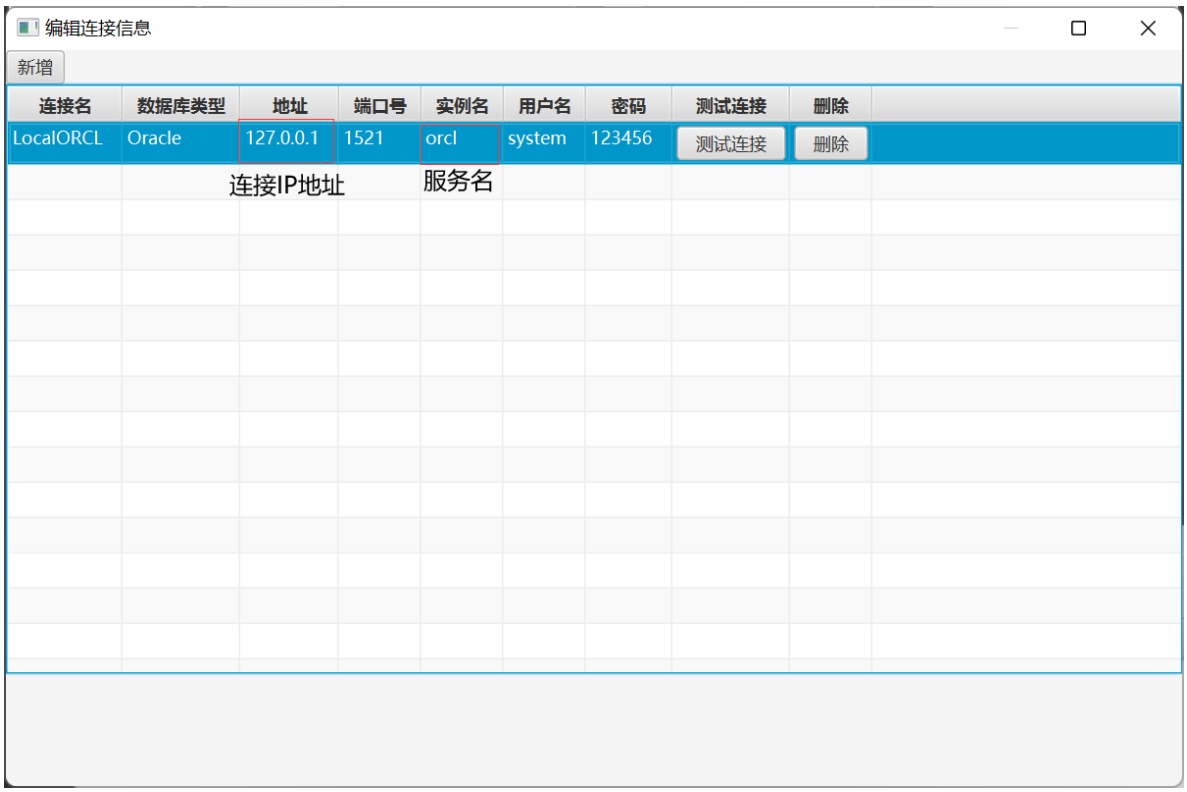
2、创建java entity

> 创建数据库中表对应的实体类，存放位置如下

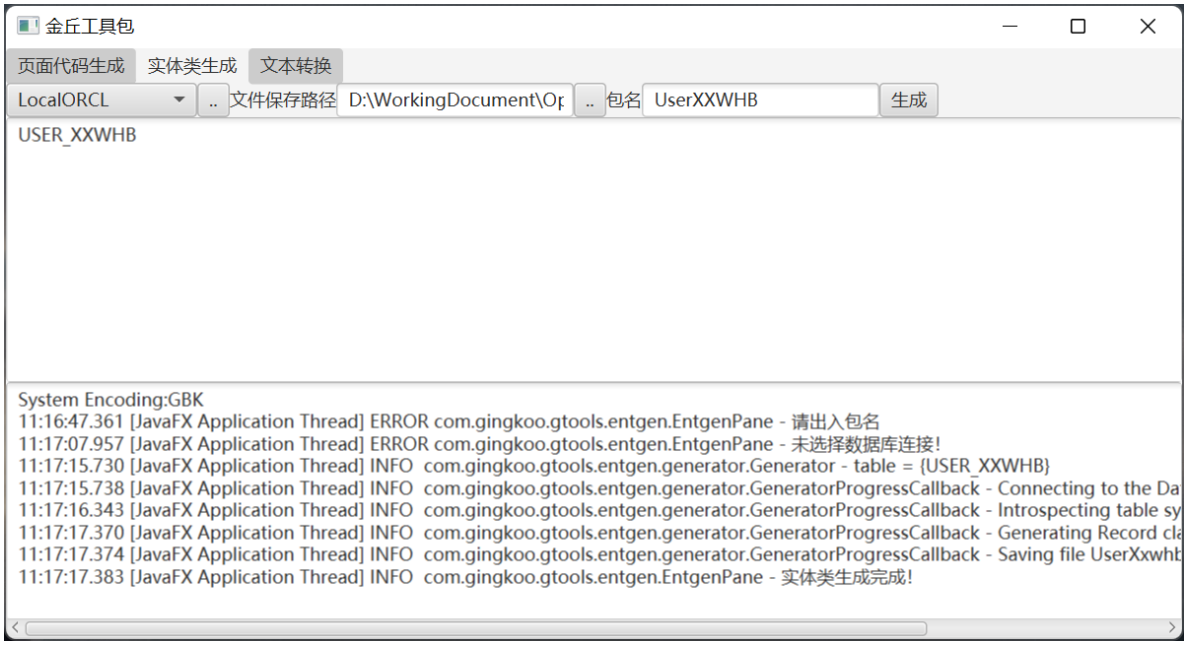


使用工具生成entity实体类

2.1、创建连接



2.2、生成实体类

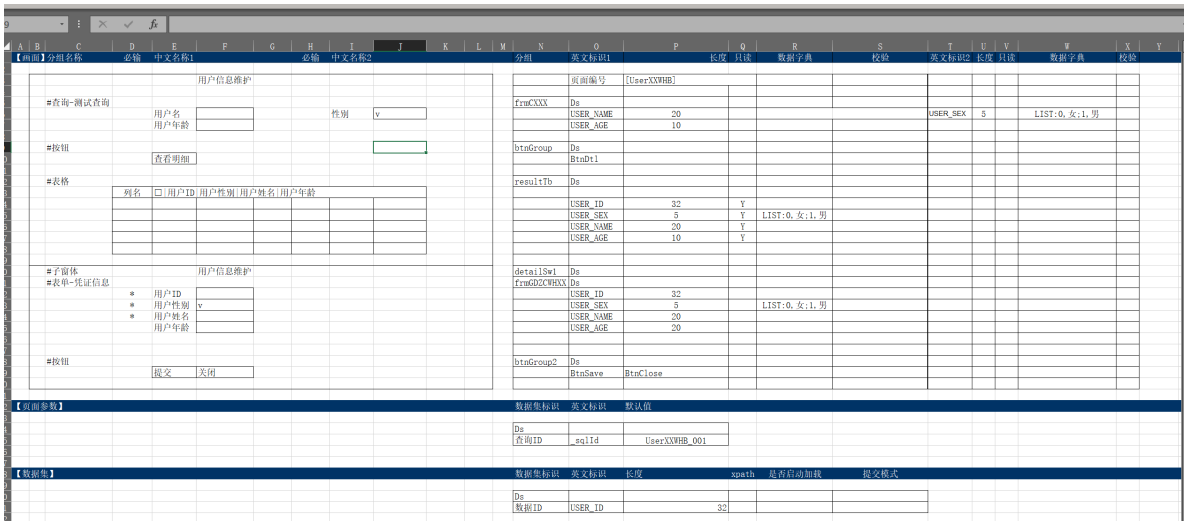


3、设计excle表格生成页面文件

> 写excle表格生成页面 ftl \ js.ftl \ xml，将文件放入制定的文件当中

创建Excle表格页面规范

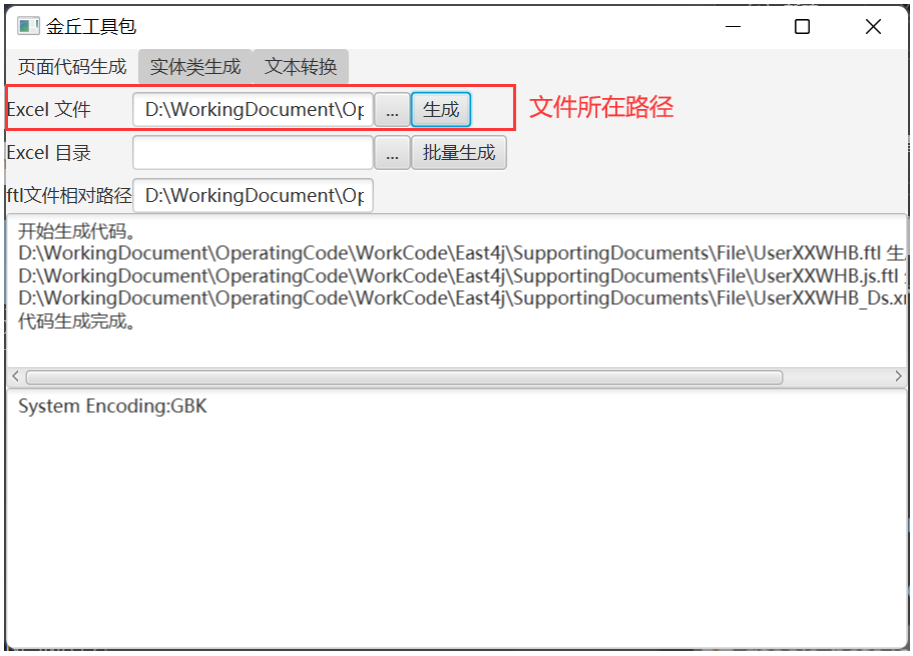
> 页面的详细操作流程教学，请参考 《GF4J开发手册.docx》

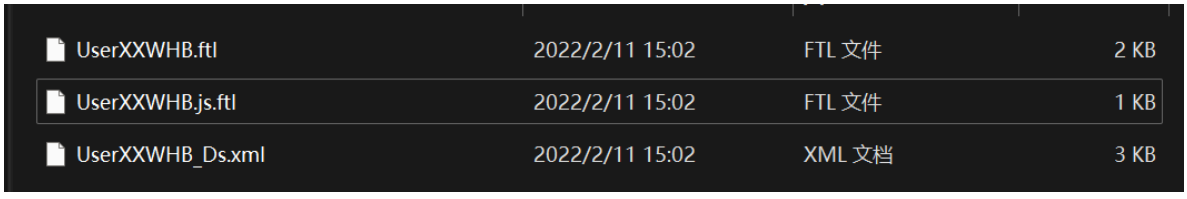


生成对应的ftl/js.ftl/xml文件

> 使用的生成工具如下

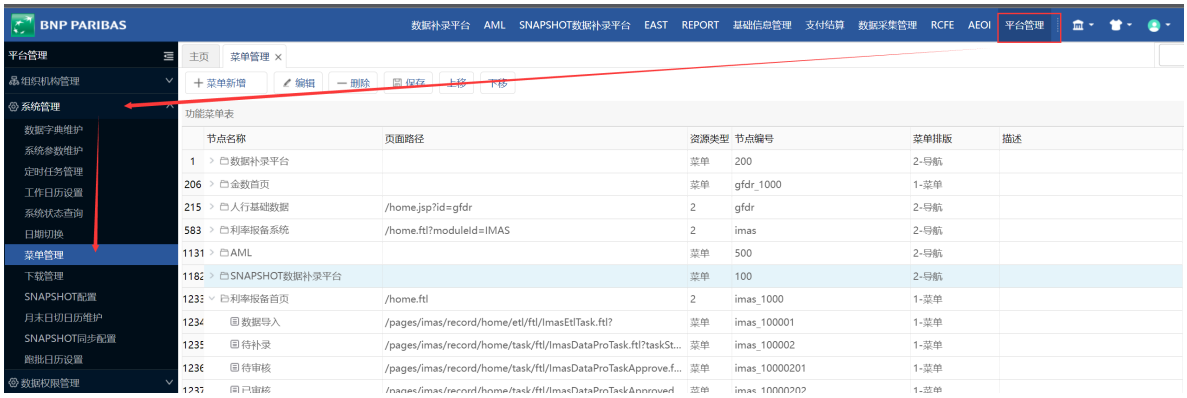
导入Excel文件



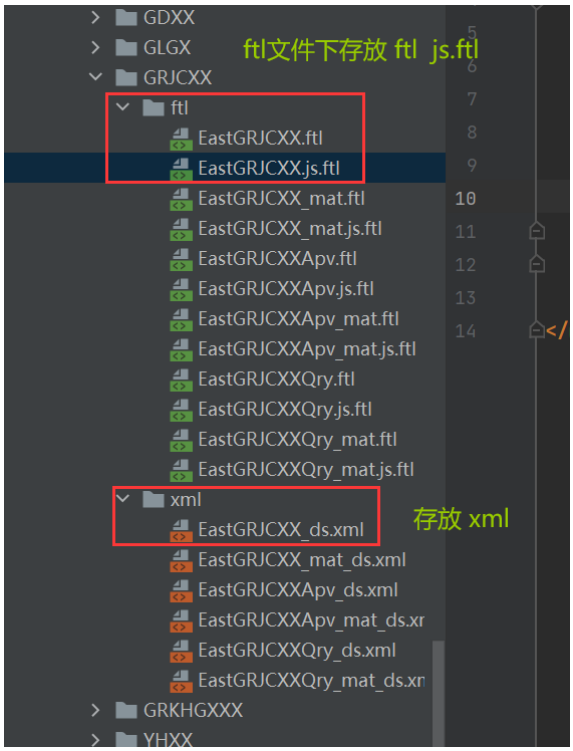


将生成的三个文件放到对应的目录下

> 如何查看你所导入目录的项目目录:
>
> 在平台管理中菜单管理，查看相邻页面的页面路径，从而找到页面存放的具体路径



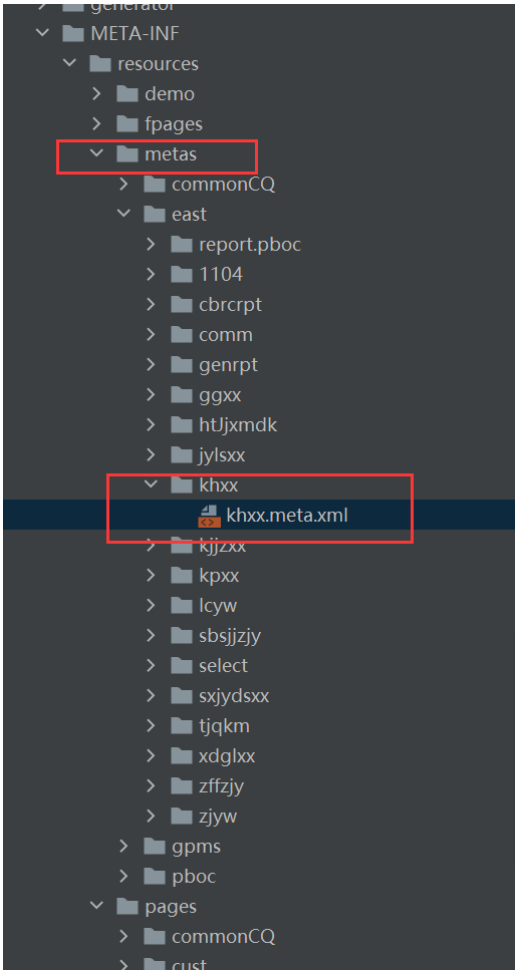
存放ftl \ js.ftl \ xml 文件



4、xml 文件导入系统

> 将ftl / js.ftl / xml 文件放置pages文件夹完毕后。再将xml文件配置到对应的meta文件下，meta文件的配置规则是

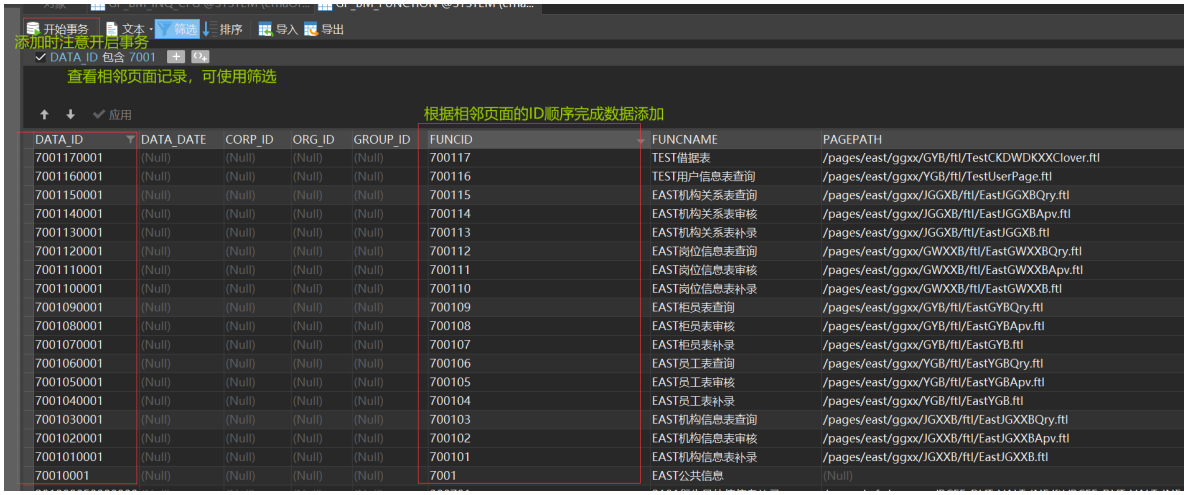
```
<Query id="xml页面名称不带后缀名"
path="/pages/east/ggxx/YGB/xml/EastYGBQry_ds.xml页面从page目录后的路径名称" />
```



5、ftl 文件导入系统

ftl 文件添加到对应的GP_BM_FUNCTION 表中（可参考菜单管理进行添加）-- **NAVICAT界面开启事务添加**

> xml文件导入系统后，要将页面对应的ftl文件同样导入到项目中，在数据库中找到 **GP_BM_FUNCTION** 表，在系统中平台管理中菜单管理**查看相邻页面的节点编号**，根据节点编号在数据表中对应**FINCID字段** 筛选，将相邻的页面在表中筛选出来，再根据相邻页面的DATA_ID、FINCID等字段的设置规范，将新页面导入到数据表中。详细如下图

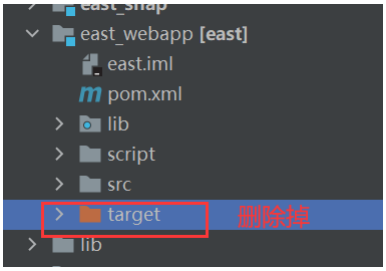
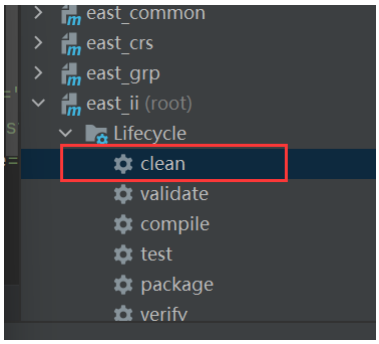
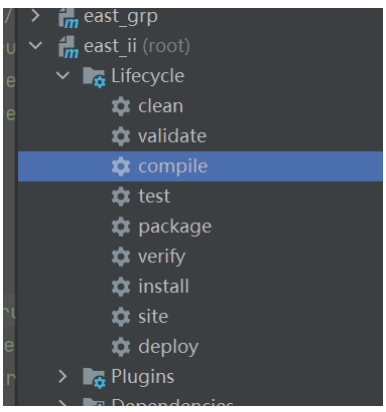


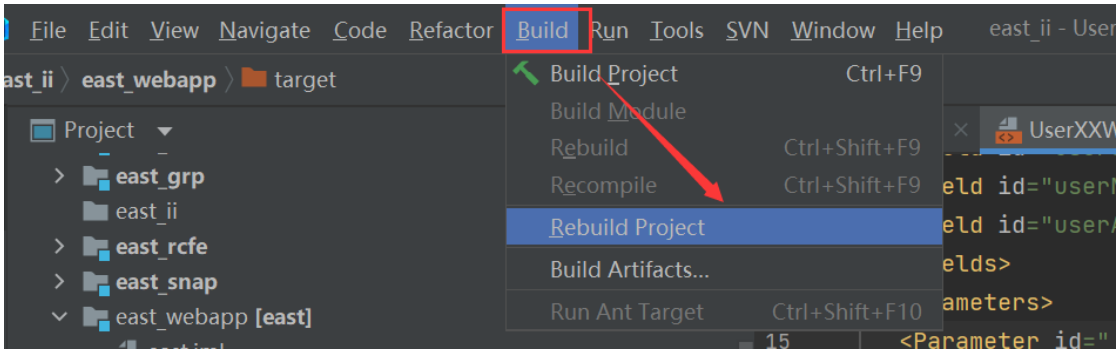
6、重启项目（容易报错）

> 重启项目，查看菜单管理是否将页面添加成功

前面步骤操作正常：

1、项目启动可以正常启动

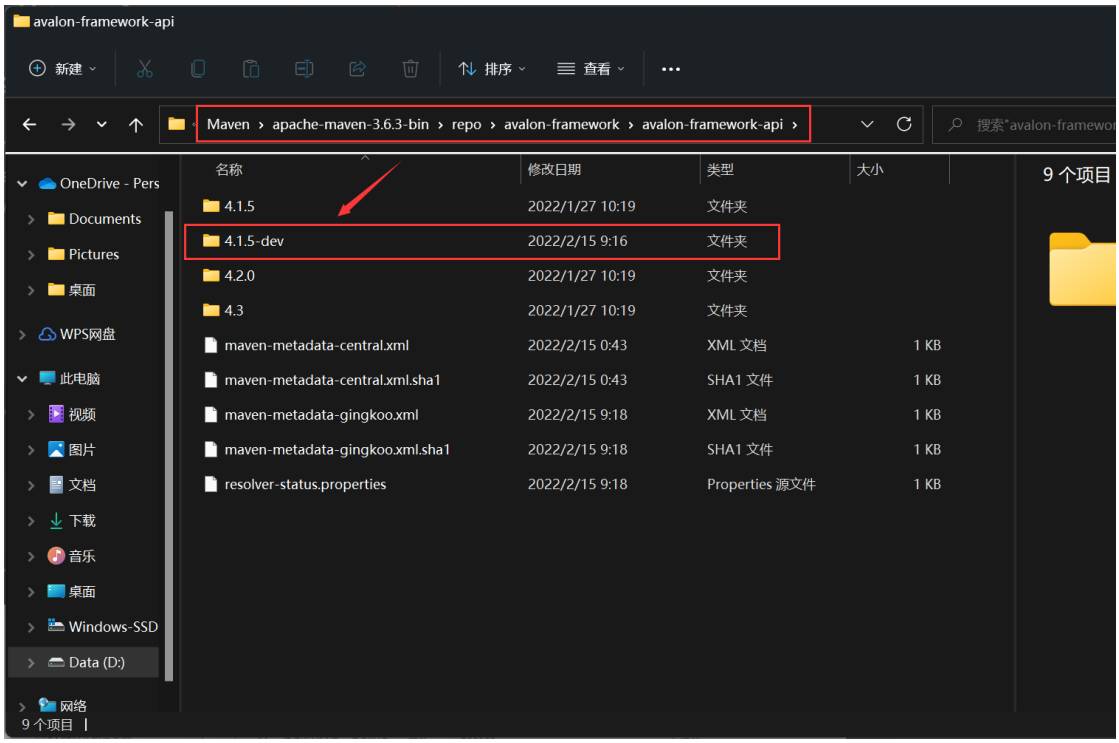
> 若出现报错信息：**找不到添加页面的xml文件**
>
> ### 1.1、删除原有编译文件
>
> 
>
>
> 
>
>
> ### 1.2、项目重新部署
>
> 
>



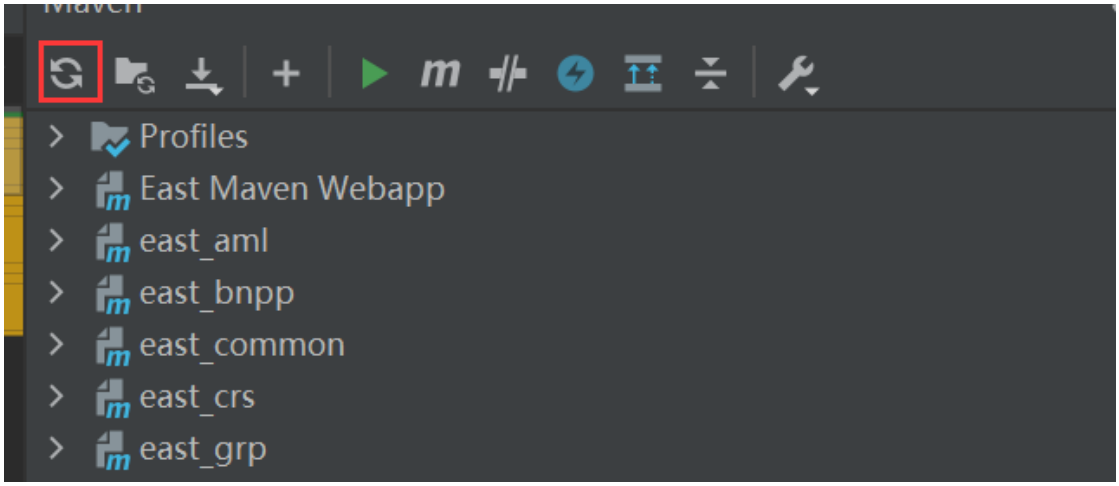
出现报错：如下错误可忽略

```
Cannot resolve Failure to transfer avalon-framework:avalon-framework-api:pom:4.1.5-dev from http://10.1.2.32/nexus/repository/public was cached in the local repository
```

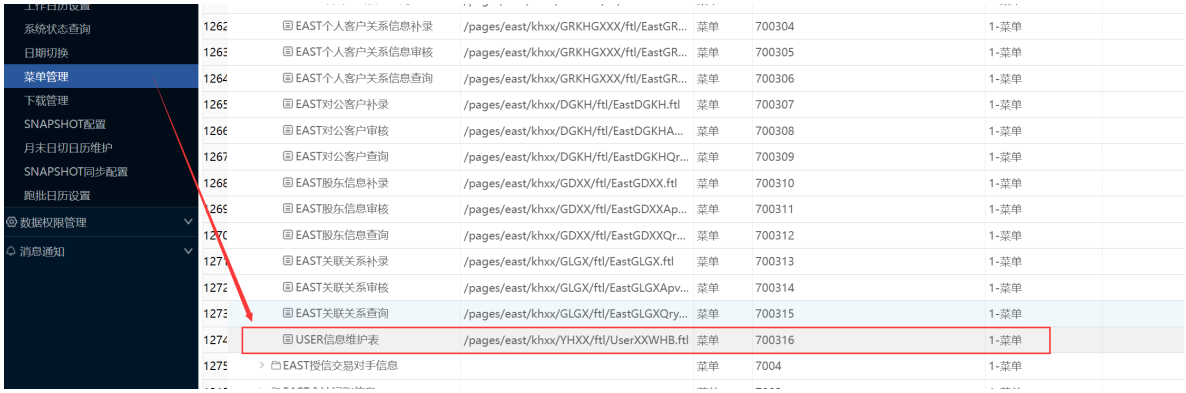
1.3.1、当出现这个错误时，打开Maven资源库，找**4.1.5-dev 文件，将其删除掉**



1.3.2、**pom.xml上右键 maven--Reimport** 就可以重新导入

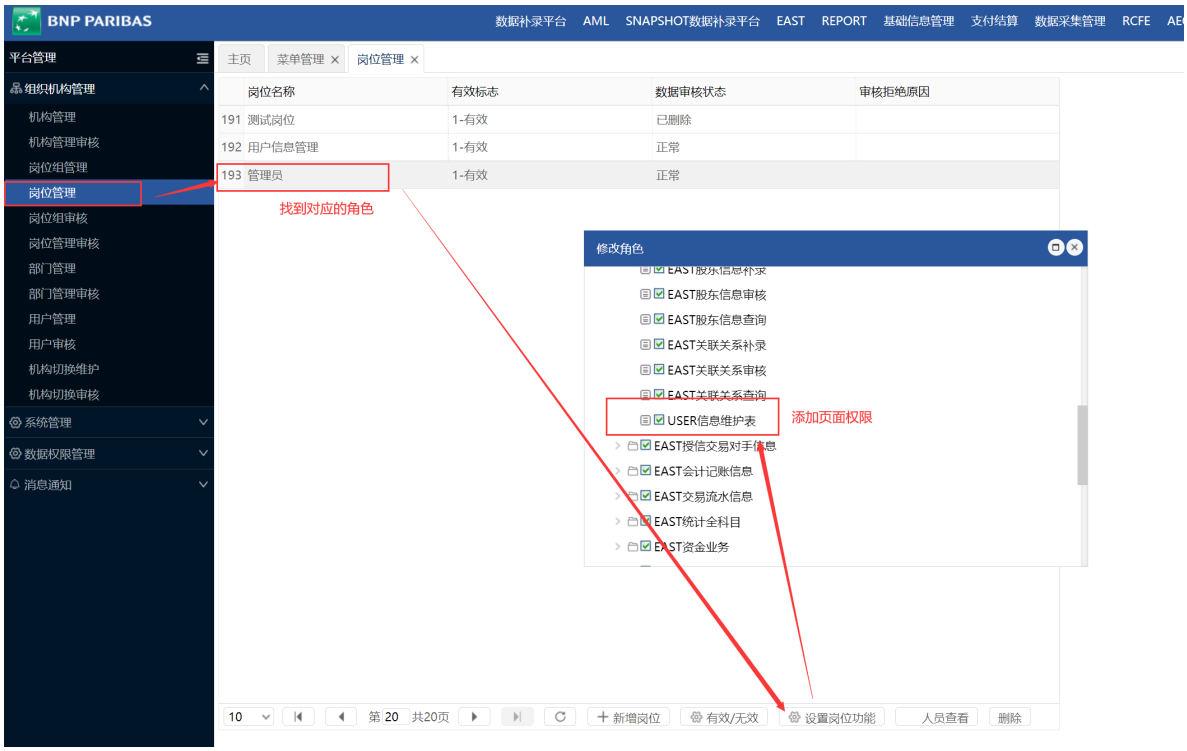


2、导入成功:在菜单管理中，出现导入的页面

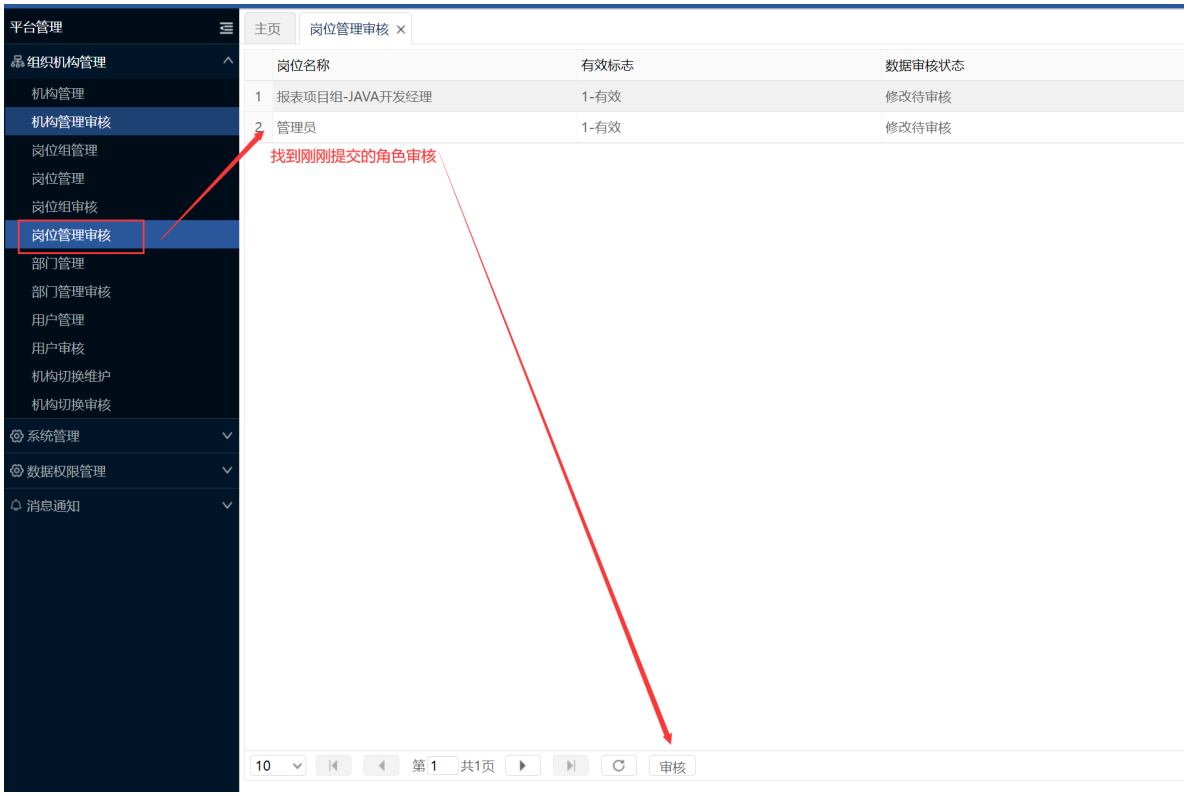


7、角色设置页面权限

> 岗位管理 --(选择岗位名称)--> 设置岗位功能 -->添加功能页面 --> 提交审核



> 更换用户登录 --> 岗位管理审核 --> 审核通过

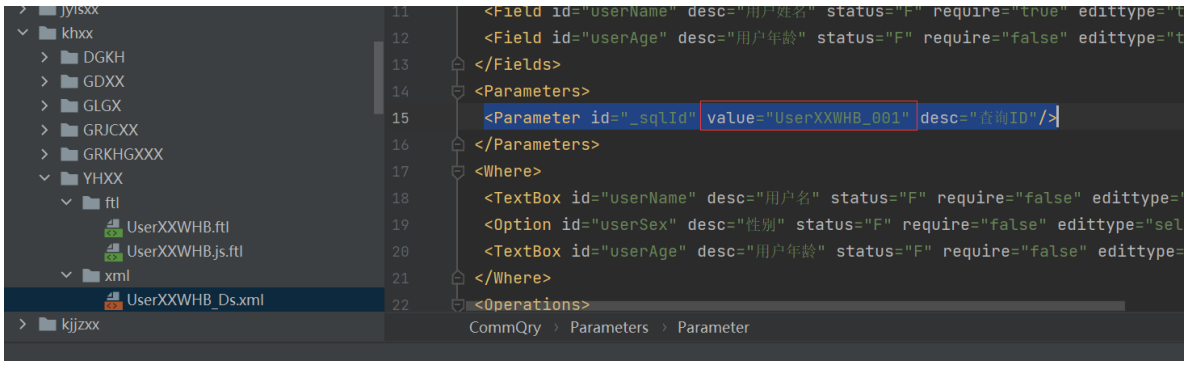


8、页面添加成功

页面实现增删改查

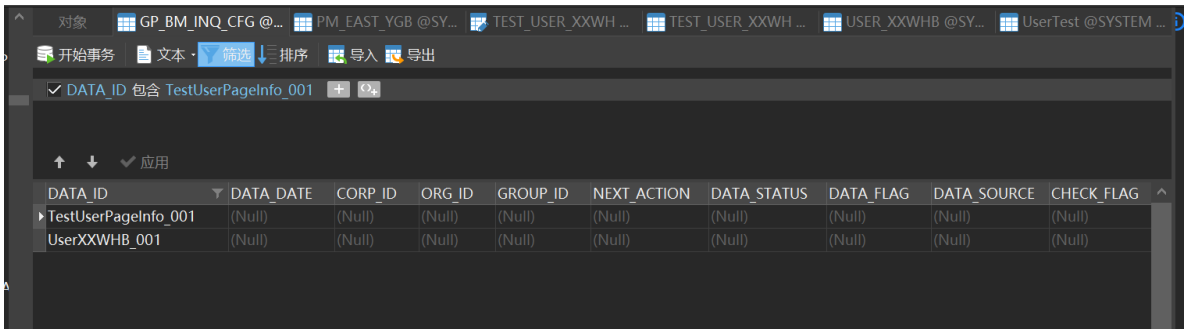
1、实现通用查询

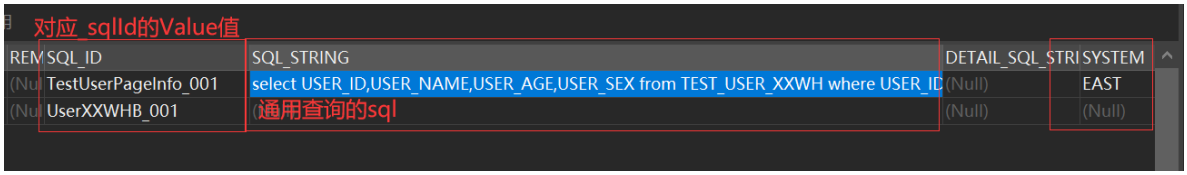
> 上述步骤中，页面导入成功后，查看xml文件中通用查询的_sqlId 设置的值



对应在数据库表GP_BM_INQ_CFG 中添加通用查询语句

> DATA_ID对应xml中_sqlId 的Value值，REMSQL_ID同理

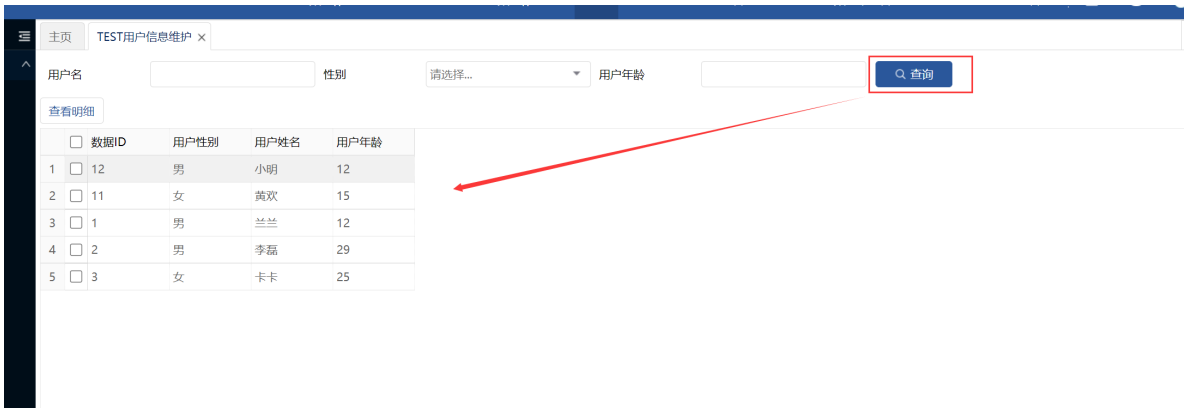




基本sql的格式

```
select USER_ID,USER_NAME,USER_AGE,USER_SEX from TEST_USER_XXWH
```

重启项目后在页面点击查询，即可显示出数据

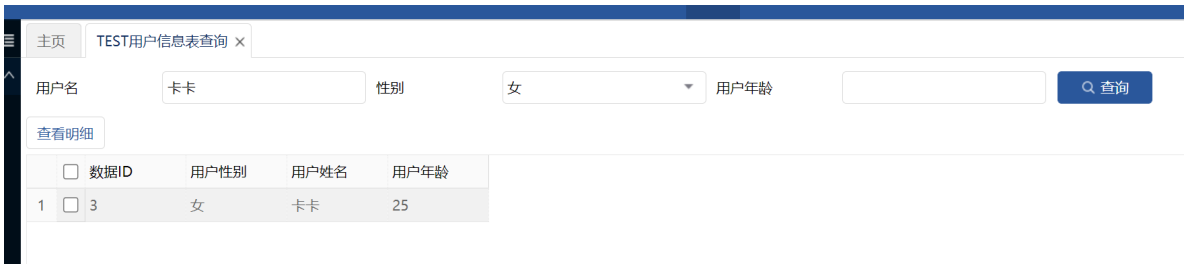


添加查询条件

```
select USER_ID,USER_NAME,USER_AGE,USER_SEX from TEST_USER_XXWH where
USER_NAME=:"userName" and USER_AGE=:"userAge" and USER_SEX=:"userSex"
```

这里的查询条件要与xml文件中的查询条件相一致

查询效果



2、分页面实现增删改

> **注意：**
>
> **此操作为了了解ftl /js.ftl /xml 文件使用的操作流程。实际操作可直接使用第 3 步 不划分子页面，直接调用子窗体实现增删改。**

2.1熟悉页面文件

2.1.1、ftl页面文件

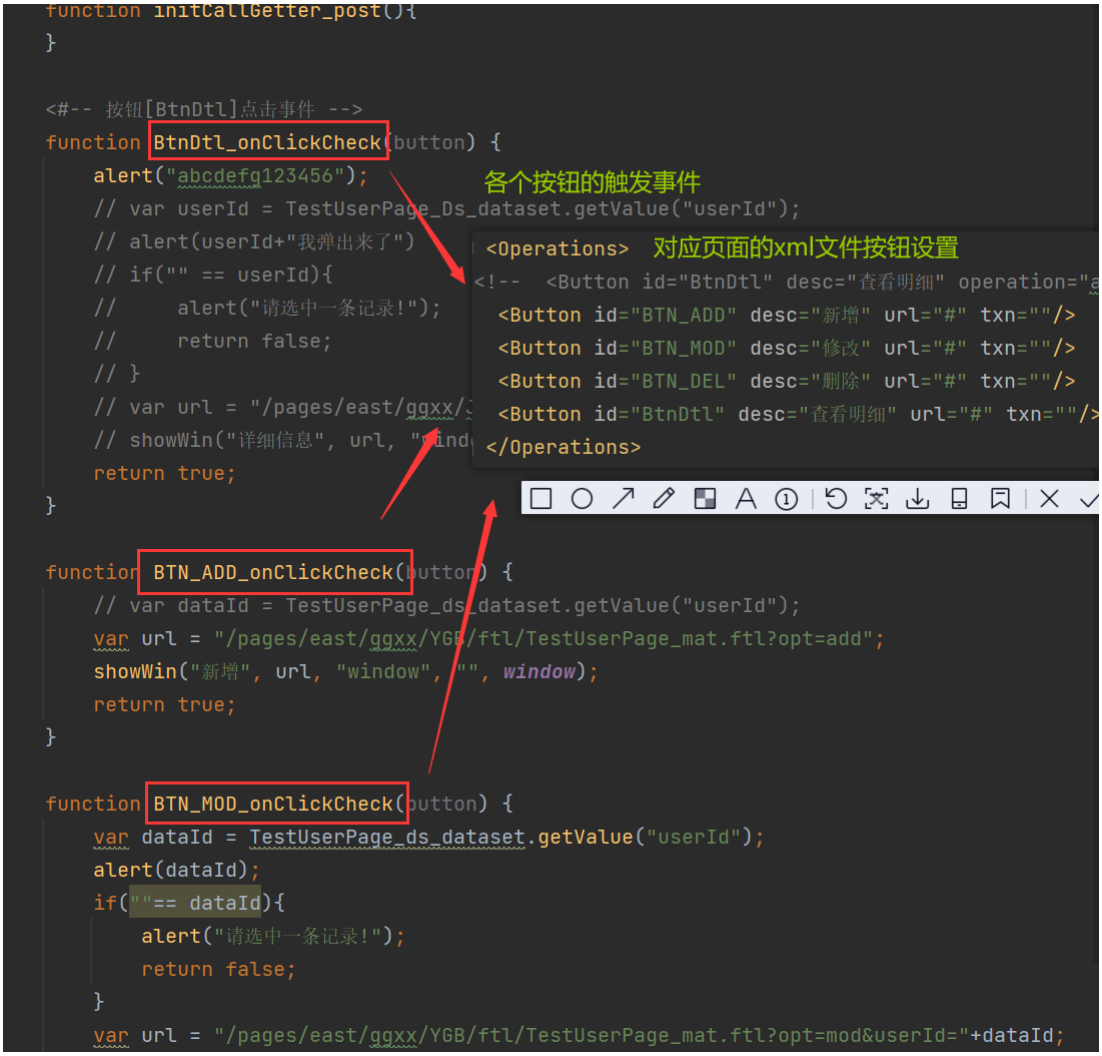
2.1.2、页面xml文件

```
<?xml version="1.0" encoding="UTF-8"?>

<CommQry title="凭证类型维护" navigate="" type="call" interface="true" pagesize="20" txnlogflag="true" async="true" databusid="
 <Include id="BankParam"/>
 <Fields>      数据库字段对应
     <Field id="dataId" desc="数据ID" status="F" primary="true" readonly="false" type="DATA_ID" size="32" xpath="/DATA_ID" ti
     <Field id="dataVersion" desc="数据版本" status="F" edittype="text" datatype="string" size="10" xpath="/DATA_VERSION" view
     <Field id="select" desc="选择" edittype="checkbox" readonly="false" type="select"/>
     <Field id="userId" desc="用户ID" status="F" primary="true" readonly="false" type="USER_ID" size="32" xpath="/USER_ID" ti
     <Field id="userSex" desc="用户性别" status="F" require="true" edittype="select" datatype="string" primary="true" readonly
     <Field id="userName" desc="用户姓名" status="F" require="true" edittype="text" datatype="string" primary="true" readonly=
     <Field id="userAge" desc="用户年龄" status="F" require="false" edittype="text" datatype="string" primary="true" readonly=
 </Fields>
 <Parameters>            _sqlId对应数据库中通用查询的绑定
     <Parameter id="_sqlId" value="TestUserPageInfo_001" desc="查询ID"/>
 </Parameters>
 <Where>
  <TextBox id="userName" desc="用户名" status="F" require="false" edittype="text" datatype="string" primary="true" readonly="
  <Option id="userSex" desc="性别" status="F" require="false" edittype="select" datatype="string" primary="true" readonly="fa
  <TextBox id="userAge" desc="用户年龄" status="F" require="false" edittype="text" datatype="string" primary="true" readonly="
 </Where>                          页面数据的筛选条件
 <Operations>
<!--  <Button id="BtnDtl" desc="查看明细" operation="asysubmit" updateClass="com.gingkoo.east.action.TestUserAction" txn=""/>-
  <Button id="BTN_ADD" desc="新增" url="#" txn=""/>
  <Button id="BTN_MOD" desc="修改" url="#" txn=""/>
  <Button id="BTN_DEL" desc="删除" url="#" txn=""/>
  <Button id="BtnDtl" desc="查看明细" url="#" txn=""/>
 </Operations>
</CommQry>
```

按钮目标路径绑定，url#及在js文件中，完成目标路径设置

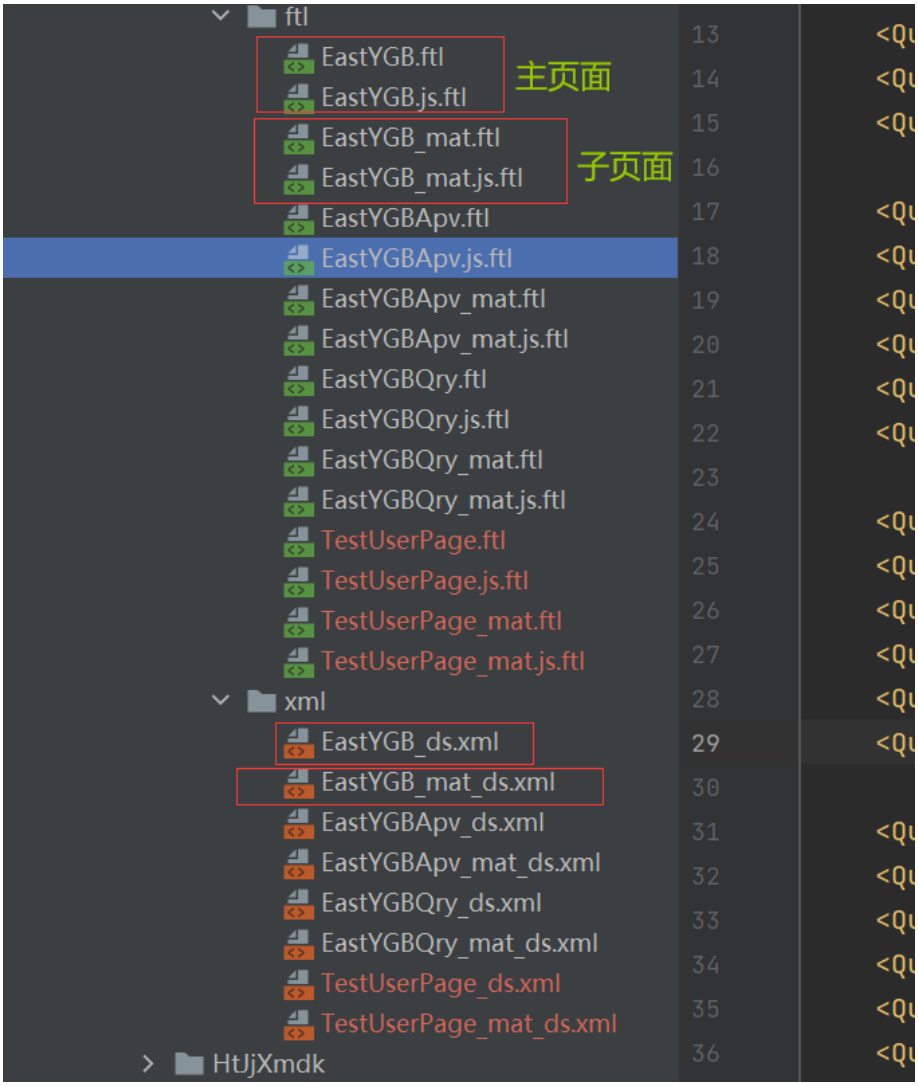
2.1.3、js.ftl页面文件

> 其他事件可参考 《GF4J开发手册.docx》，其中可见详解
>
> 编写时，注意按钮触发事件的按钮名称对应，**按钮名_onClickCheck** 按钮名及xml文件中 **标签的id**



2.2创建子窗口页面

> 创建子页面的页面文件命名要求,参考原有文件的命名规范进行命名



原主页面样式

> 观察主窗体中子窗口标签。目标将子窗口标签<@CommonQueryMacro.FloatWindow>独立出来,变成也页面标签<@CommonQueryMacro.CommonQuery>

原主页面修改过后

> 子页面提取完成后,**主页面可将子窗体标签删去,其他不发生改变.**
>
> **若要加按钮**,可自主添加**<@CommonQueryMacro.Button id="BtnXXX"/>标签**

```
<#import "/templets/commonQuery/CommonQueryTagMacro.ftl" as CommonQueryMacro>
<@CommonQueryMacro.page title="凭证类型维护">
    <div>
        <@CommonQueryMacro.CommonQuery id="TestUserPage_ds" init="false" submitMode="current" navigate="false">
            <@CommonQueryMacro.Interface id="frmCXXX" label="测试查询" colNm=4 labelwidth="100" />
            <@CommonQueryMacro.Button id="BtnDtl"/>
            <@CommonQueryMacro.Button id="BTN_ADD"/>
            <@CommonQueryMacro.Button id="BTN_MOD"/>
            <@CommonQueryMacro.Button id="BTN_DEL"/>
            <@CommonQueryMacro.DataTable id="resultTb" fieldStr="select,userId,userSex,userName,userAge" height="500" pagination="
        </@CommonQueryMacro.CommonQuery>
    </div>
    <#include "TestUserPage.js.ftl"/>
</@CommonQueryMacro.page>
```

> 页面对应xml文件添加按钮配置,**注意id对应**

```
<Operations>
<!--   <Button id="BtnDtl" desc="查看明细" operation="asysubmit" updateClass="com.gingkoo.east.act
 <Button id="BTN_ADD" desc="新增" url="#" txn=""/>
 <Button id="BTN_MOD" desc="修改" url="#" txn=""/>
 <Button id="BTN_DEL" desc="删除" url="#" txn=""/>
 <Button id="BtnDtl" desc="查看明细" url="#" txn=""/>
</Operations>
```

页面按钮配置

子页面

> <@CommonQueryMacro.FloatWindow>标签整体复制到
>
> 下,再将其修改为<@CommonQueryMacro.CommonQuery>
>
> **更改之前的id为子页面对应的xml文件名**,xml文件创建后续说明,**原有的其他属性删除.更改与主页面相同的属性,init属性改为true**

```
<#import "/templets/commonQuery/CommonQueryTagMacro.ftl" as CommonQueryMacro>
<@CommonQueryMacro.page title="详细">
    <div>
        <@CommonQueryMacro.CommonQuery id="TestUserPage_mat_ds" init="true" submitMode="current" navigate="false">
            <@CommonQueryMacro.Group id="frmGDZCWHXX" label="凭证信息" fieldStr="userId,userSex,userName,userAge" colNm=2/>
            <@CommonQueryMacro.Button id="BtnSave"/>
            <@CommonQueryMacro.Button id="BtnClose"/>
        </@CommonQueryMacro.CommonQuery>
    </div>
    <#include "TestUserPage_mat.js.ftl"/>
</@CommonQueryMacro.page>
```

子页面对应xml文件

原有属性删除，复制主页面该标签的后续属性

注意将init的属性值改为true

引入子页面的js.ftl事件

子页面对应事件js.xml

> 事件与子页面按钮对应,注意按钮名称要与页面写的Button标签的id对应

```
<script language="javascript">
    <#-- 所有dataset刷新完数据后触发，调用此方法 -->
    function initCallGetter_post(){
    }

    <#-- 按钮[BtnSave]点击事件 -->
    function BtnSave_onClickCheck(button) {
        alert("我点击了提交事件");
        return true;
    }

    <#-- 按钮[BtnClose]点击事件 -->
    function BtnClose_onClickCheck(button) {
        alert("我点击了关闭事件");
        closeWin();
        return true;
    }

</script>
```

子页面xml文件

> 可复制主页面的xml文件做修改,标签删除,其他标签内容根据情况做修改

```
<CommQry title="凭证信息维护" navigate="" type="call" interface="true" pagesize="20" txnlogflag="true" async="true" databusid="" transdataactionurl="/trans/TransDa
    <Include id="BankParam"/>
    <Fields>
        <!--显示字段配置-->  字段配置，可复制原有主页面的字段配置，删除select 字段即可，其他可不变。 注意 <parameters>设置的请求参数字段要在此处做配置
        <Field id="dataVersion" desc="数据版本" status="F" edittype="text" datatype="string" size="10" xpath="/DATA_VERSION" viewField="false"/>
        <Field id="userId" desc="用户ID" status="F" primary="true" readonly="false" type="USER_ID" size="32" xpath="/USER_ID" tip="用户ID"/>
        <Field id="dataId" desc="DATA_ID" status="F" primary="true" readonly="false" type="DATA_ID" size="32" xpath="/DATA_ID" tip="DATA_ID"/>
        <Field id="userSex" desc="用户性别" status="F" require="true" edittype="select" datatype="string" primary="true" readonly="false" type="USER_SEX" size="5"
        <Field id="userName" desc="用户姓名" status="F" require="true" edittype="text" datatype="string" primary="true" readonly="false" type="USER_NAME" size="20
        <Field id="userAge" desc="用户年龄" status="F" require="false" edittype="text" datatype="string" primary="true" readonly="false" type="USER_AGE" size="20"
    </Fields>
    <!--请求参数配置-->
    <Parameters>
        <Parameter id="_sqlId" value="TestUserPageInfo_002" desc="查询ID"/>  _sqlId设置数据库INQ_CFG表的sql语句，查询条件可用url请求参数传入
        <!--对应请求参数字段-->
        <Parameter id="userId" value="" desc="userId"/> 设置url请求参数
    </Parameters>
    <Operations>
        <!--  <Button id="BtnDtl" desc="查看明细" operation="asysubmit" updateClass="com.gingkoo.east.action.TestUserAction" txn=""/>-->
        <Button id="BtnSave" desc="提交" operation="asysubmit" updateClass="com.gingkoo.east.action.TestUserXXWHBAction" txn=""/>          按钮配置
        <Button id="BtnClose" desc="关闭" url="#" txn=""/>
    </Operations>
</CommQry>
                                                    设置该按钮事件通过后，跳转的action类路径
```

2.3将子页面添加到系统mate.xml文件中

> 将子页面的xml文件配置到系统的meta.xml文件中

```
    <Query id="EastJGGXB_ds" path="/pages/east/ggxx/JGGXB/xml/EastJGGXB_ds.xml" />
    <Query id="EastJGGXB_mat_ds" path="/pages/east/ggxx/JGGXB/xml/EastJGGXB_mat_ds.xml" />
    <Query id="EastJGGXBApv_ds" path="/pages/east/ggxx/JGGXB/xml/EastJGGXBApv_ds.xml" />
    <Query id="EastJGGXBApv_mat_ds" path="/pages/east/ggxx/JGGXB/xml/EastJGGXBApv_mat_ds.xml" />
    <Query id="EastJGGXBQry_ds" path="/pages/east/ggxx/JGGXB/xml/EastJGGXBQry_ds.xml" />
    <Query id="EastJGGXBQry_mat_ds" path="/pages/east/ggxx/JGGXB/xml/EastJGGXBQry_mat_ds.xml" />

    <Query id="TestUserPage_ds" path="/pages/east/ggxx/YGB/xml/TestUserPage_ds.xml"/>
    <Query id="TestUserPage_mat_ds" path="/pages/east/ggxx/YGB/xml/TestUserPage_mat_ds.xml"/>
    <Query id="TestCKDWDKXXClover_Ds" path="/pages/east/ggxx/GYB/xml/TestCKDWDKXXClover_Ds.xml"/>
</META>
```

2.4数据库中添加子页面sql查询语句

> 子页面的初始化数据配置在INQ_CFG表中的sql语句所需要的参数，要与xml文件中标签中设置的参数相互对应。
>
> 参数设置是要与数据库中表中字段相互对应的。遵循大小驼峰规则

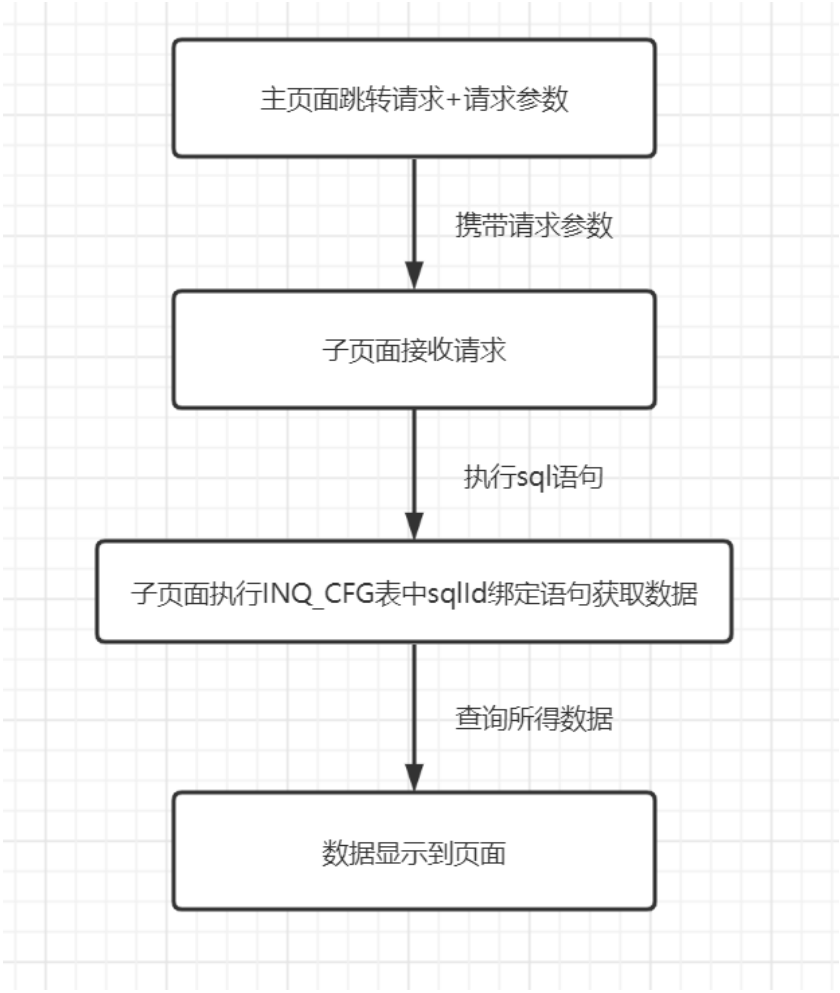
2.5主页面按钮事件编写

> 这里只对更改按钮做举例

```
function BTN_MOD_onClickCheck(button) {
    //获取页面域中,被选中记录的userId
    var dataId = TestUserPage_ds_dataset.getValue("userId");
    alert(dataId);
    //判断记录是否为空
    if(""== dataId){
        alert("请选中一条记录!");
        //return false 不允许跳转
        return false;
    }
    //编写跳转地址的url,填写子页面的ftl路径,只需要pages后的路径,"?"后填写请求所携带的参数,opt为请求的类型:
    //mod : 修改
    //del : 删除
```

```
    //add : 修改
    //dtl : 明细
    var url = "/pages/east/ggxx/YGB/ftl/TestUserPage_mat.ftl?
opt=mod&userId="+dataId;
    //页面跳转方法  参数1为页面开头,后两个参数不变
    showWin("修改", url, "window", "", window);
    //return true 允许跳转
    return true;
}
```

执行流程



2.6子页面按钮事件

> 子页面的提交按钮,是要将数据提交的后台做数据处理,因此在配置xml文件中,设置BtnSave按钮跳转的Action 类,return true 就可跳转到所设置的action前端交互类中.
>
> 事件中,在跳转之前可完成一些参数添加 或者校验

```
xml文件名_dataset.setParameter("key","value");//向页面请求域中添加参数
```

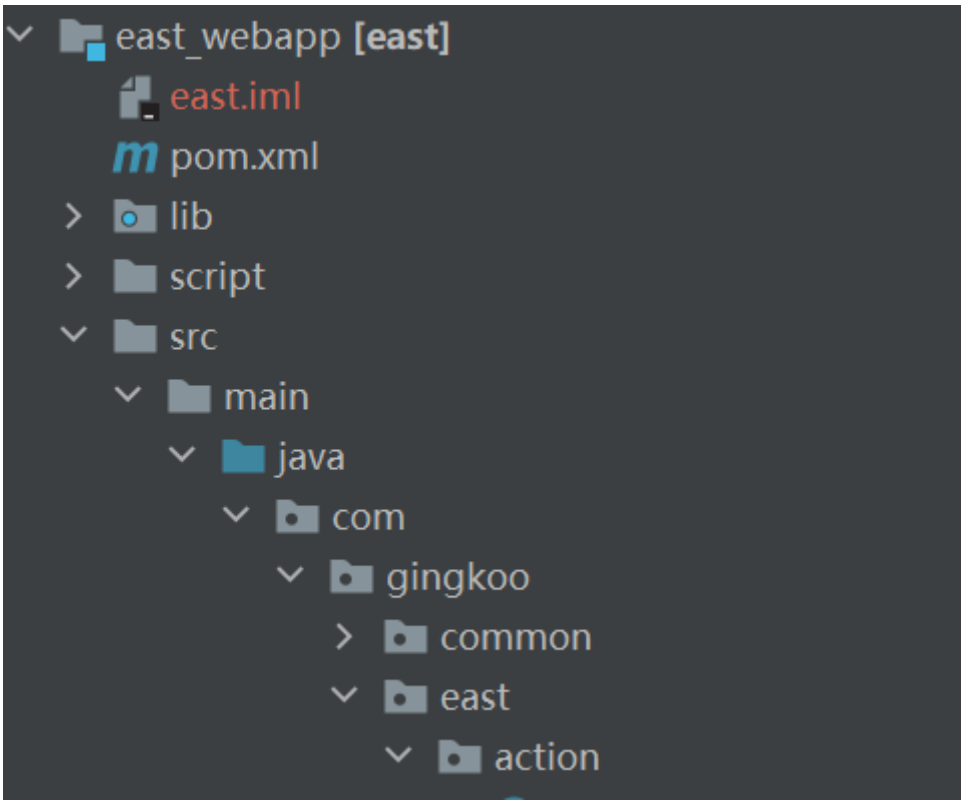
```
<#-- 所有dataset刷新完数据后触发，调用此方法 -->
function initCallGetter_post(){
}

<#-- 按钮[BtnSave]点击事件 -->
function BtnSave_onClickCheck(button) {
    alert("我点击了提交事件");
    return true; return true 允许跳转
}

<#-- 按钮[BtnClose]点击事件 -->
function BtnClose_onClickCheck(button) {
    alert("我点击了关闭事件");
    closeWin();
    return true;
}

<Operations>
    <!--    <Button id="BtnDtl" desc="查看明细" operation="asysubmit" updateClass="com.gingkoo.east.action.TestUserAction" txn=""/>-->
    <Button id="BtnSave" desc="提交" operation="asysubmit" updateClass="com.gingkoo.east.action.TestUserXXWHBAction" txn=""/>
    <Button id="BtnClose" desc="关闭" url="#" txn=""/>
</Operations>
</CommQry>
```

子页面提交事件

关闭子页面

2.7Action类编写

添加路径



> 创建类 **继承WebAlterAction实现saveOrUpdate方法**

```
package com.gingkoo.east.action;

import com.gingkoo.common.query.web.action.base.WebAlterAction;
import com.gingkoo.common.validator.service.CommonValidteServeice;
import com.gingkoo.east.service.TestUserXXWHBService;
import com.gingkoo.gf4j2.core.sys.excp.AppException;
import com.gingkoo.gf4j2.core.util.UuidHelper;
import com.gingkoo.gf4j2.framework.entity.global.GlobalInfo;
import com.gingkoo.gf4j2.framework.entity.result.MultiUpdateResultBean;
import com.gingkoo.gf4j2.framework.entity.result.UpdateResultBean;
import com.gingkoo.gf4j2.framework.entity.result.UpdateReturnBean;
import com.gingkoo.gf4j2.framework.service.base.OPCaller;
```

```
import com.gingkoo.gf4j2.framework.service.base.ServiceContext;
import com.gingkoo.orm.entity.TestUser;
import org.springframework.beans.factory.annotation.Autowired;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.util.Map;

public class TestUserXXWHBAction extends WebAlterAction {

    @Override
    public UpdateReturnBean saveOrUpdate(MultiUpdateResultBean
multiUpdateResultBean, HttpServletRequest httpServletRequest,
HttpServletResponse httpServletResponse) throws AppException {
        //日志打印
        logger.info("-------TestUserXXWHBAction start---------");
        //创建页面回传类,作为该方法的返回值
        UpdateReturnBean updateReturnBean = new UpdateReturnBean();
         //获取页面请求域
        UpdateResultBean updateResultBean =
multiUpdateResultBean.getUpdateResultBeanByID("TestUserPage_mat_ds");
        //创建与service类交互的参数域类
        ServiceContext oc = new ServiceContext();

        //获取页面请求参数  ---- opt:类型参数  在主页面按钮事件中设置
        String opt = updateResultBean.getParameter("opt");
        if (updateResultBean.hasNext()) {
            //获取所有页面参数
            Map<String, String> map = updateResultBean.next();
            //将页面请求转换至 实体类 中
            TestUser testUser = new TestUser();
            //实体类转换方法
            mapToObject(testUser, map);

            //向与service交互的参数域中添加参数
            oc.setAttribute("Bean", testUser);
            oc.setAttribute("cmd", opt);
            oc.setAttribute("IN_PARAM_1", multiUpdateResultBean);
            //调用service类,oc参数域作为参数传入
            OPCaller.call(TestUserXXWHBService.ID, oc);
            /*
                两种调用方式
                1\运行时类调用
                OPCaller.call(TestUserXXWHBService.class, oc);
                2\spring容器中id调用  上述使用
                OPCaller.call("testUserXXWHBService", oc);
            */
        }
        //日志打印
        logger.info("-------TestUserXXWHBAction end---------");
        return updateReturnBean;
    }
}
```

2.8Service类编写

```java
package com.gingkoo.east.service;

import java.util.Enumeration;
import java.util.List;
import java.util.UUID;

import com.gingkoo.gf4j2.core.util.UuidHelper;
import com.gingkoo.gf4j2.framework.config.database.MyHibernateTemplateImpl;
import com.gingkoo.orm.entity.TestUser;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.hibernate.SessionFactory;
import org.hibernate.impl.SessionFactoryImpl;
import org.junit.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Component;

import com.gingkoo.gf4j2.framework.config.database.base.MyHibernateTemplate;
import com.gingkoo.gf4j2.framework.excp.CommonException;
import com.gingkoo.gf4j2.framework.service.base.BaseService;
import com.gingkoo.gf4j2.framework.service.base.ServiceContext;
import org.springframework.stereotype.Service;

import javax.annotation.Resource;


@Service
public class TestUserXXWHBService extends BaseService {
    private static final Log logger =
LogFactory.getLog(TestUserXXWHBService.class);
    public static final String ID = "testUserXXWHBService"; // spring容器中该类名
称,静态属性方便action类直接调用


    private final MyHibernateTemplate template;

    public TestUserXXWHBService(MyHibernateTemplate template) {
        this.template = template;
    }

    /**
     * execute 调用前处理
     */
    @Override
    public void afterProc(ServiceContext paramServiceContext) throws
CommonException {
    }

    /**
     * execute 调用后处理
     */
    @Override
    public void beforeProc(ServiceContext paramServiceContext) throws
CommonException {
    }

    @Override
```

```
    public void execute(ServiceContext serviceContext) throws CommonException {
        //从参数域中获取请求类型
        String cmd = (String) serviceContext.getAttribute("cmd");
        //获取实体类对象
        TestUser test_user = (TestUser) serviceContext.getAttribute("Bean");
        //根据请求类型,执行不同操作
        if("add".equals(cmd)){
            //生成一个DATA_ID
            test_user.setDataId(UuidHelper.getCleanUpperUuid().substring(0,18));
            template.save(test_user);
        }else if("mod".equals(cmd)){
            template.update(test_user);
        }else if("del".equals(cmd)){
            template.delete(test_user);
        }
    }
}
```

3、使用页面子窗口实现增删改

直接调用子窗口

> 后续的操作是使用手动划分页面，方便理解页面标签使用的，也可以不使用下方分页面的做法。
> 若ftl中包含子窗体，可直接使用如下方法直接调用子窗体。

- FloatWindow 子窗体

floatWindow 的对象名称为"subwindow_"+id.

方法：

方法名称	返回值	参数	描述
close()	void		关闭弹出窗
show()	void		打开弹出窗

事件

事件名称	返回值	参数	描述
beforeShow	boolean	floatwindow	打开窗口之前执行
beforeHide	boolean	floatwindow	在隐藏窗口前执行 返回 true，则允许隐藏，否则不能隐藏
beforeClose	boolean	floatwindow	在关闭窗口前执行 返回 true，则允许关闭，否则不能关闭
afterHide	void	floatwindow	在隐藏窗口后执行
afterClose	void	floatwindow	在关闭窗口后执行

注意：事件函数的调用方式为 id+"_floatWindow_"+事件名。

小窗口的调用：

```
<@CommonQueryMacro.FloatWindow id="detailSw1" label="信息单" width="100%" resize="true" defaultZoom="normal" minimize="false" ma
    <@CommonQueryMacro.Group id="frmGDZCWHXX"label="详细信息"fieldStr="nbjgh,jkrzjdm,jkrzjlx,dkye"colNm=2/>
    <@CommonQueryMacro.Button id="BtnSave"/>
    <@CommonQueryMacro.Button id="BtnClose"/>
</@CommonQueryMacro.FloatWindow>
</@CommonQueryMacro.CommonQuery>
```

```
subwindow_子窗体id（如上所示）
//窗口弹出
subwindow_detailSw1.show();
//窗口关闭
subwindow_detailSw1.close();
```

3.1、按钮绑定控制类

> 确定那个按钮绑定控制类，在页面的xml文件中完成绑定，书写格式如下
>
> ```
> <Button id="BtnSave" desc="提交" operation="asysubmit"
> updateClass="com.gingkoo.east.action.TestCKDWDKXXWZAction" txn=""/>
> ```

```
</Where>
<Operations>
 <Button id="BtnDtl" desc="查看明细" url="#" txn=""/>
 <Button id="BtnAdd" desc="新增" url="#" txn=""/>
 <Button id="BtnMod" desc="修改" url="#" txn=""/>
 <Button id="BtnDle" desc="删除" url="#" txn=""/>
 <Button id="BtnSave" desc="提交" operation="asysubmit" updateClass="com.gingkoo.east.action.TestCKDWDKXXWZAction" txn=""/>
 <Button id="BtnClose" desc="关闭" url="#" txn=""/>
</Operations>
</CommQry>
```

3.2、事件跳转

> xml文件中绑定完毕后，按钮事件方法中。
>
> return true 允许进行跳转
>
> return false 不允许跳转
>
> 后续有需求的话，可以在return 之前进行一些数据的判断审核之类的操作。

```
<#-- ??t[BtnSave]????¼? -->
function BtnSave_onClickCheck(button) {

    return true;
}
```

除此之外，另外介绍两种操作，方便后续完成批量操作使用

3.2.1、获取被选中记录条数

```
function getSelectedRecord(){
    //获取第一条记录
    var record = AML_TB_CST_PERSApv_ds_dataset.getFirstRecord();
    //设置记录条数
    var selectNum = 0;
    //record判断是否为空
    while(record){
        //查看该记录是否被选中
        if(record.getValue("select")==true){
            selectNum = selectNum+1;
        }
```

```
            //到下一条记录
            record = record.getNextRecord();
        }
        return selectNum;
    }
```

3.2.2、获取第一个被选中记录

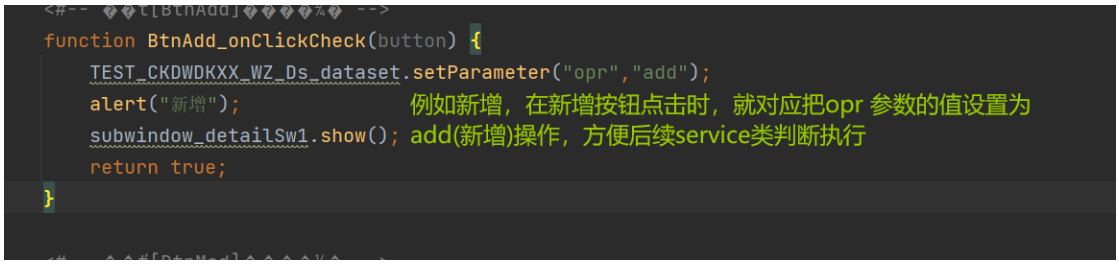
```
    function getRecord(){
        //获取第一条记录
        var record = AML_TB_CST_PERSApv_ds_dataset.getFirstRecord();
        //record判断是否为空
        while(record){
            //查看该记录是否被选中
            if(record.getValue("select")==true){
                //若被选中返回选中的记录
                return record;
            }
            //获取下一条记录
            record = record.getNextRecord();
        }
        return null;
    }
```

3.3、Actioon控制类接收请求

> 注意 控制类中，通过页面请求域获取的参数都是提前在事件中 **xml名_dataset.setParameter("参数名","参数值");**，的方式已经设置好的。后才能在控制类中通过页面请求域的**getParameter("参数名")**的方法调用。
>
> **参数 opt / opr：该请求的请求类型，是add(新增) 、mod(修改)还是 del(删除)**

> 建议请求类型的设置都在对应的按钮事件触发时就完成设置

```
//页面请求域固定格式：   xml文件名 + _dataset
//获取域中数据
var value = TEST_CKDWDKXX_WZ_Ds_dataset.getValue("ckid");
//添加数据
TEST_CKDWDKXX_WZ_Ds_dataset.setParameter("opr","add");
```




```
package com.gingkoo.east.action;

import com.gingkoo.common.query.web.action.base.WebAlterAction;
import com.gingkoo.common.validator.service.CommonValidteServeice;
import com.gingkoo.east.service.TestUserXXWHBService;
import com.gingkoo.gf4j2.core.sys.excp.AppException;
```

```java
import com.gingkoo.gf4j2.core.util.UuidHelper;
import com.gingkoo.gf4j2.framework.entity.global.GlobalInfo;
import com.gingkoo.gf4j2.framework.entity.result.MultiUpdateResultBean;
import com.gingkoo.gf4j2.framework.entity.result.UpdateResultBean;
import com.gingkoo.gf4j2.framework.entity.result.UpdateReturnBean;
import com.gingkoo.gf4j2.framework.service.base.OPCaller;
import com.gingkoo.gf4j2.framework.service.base.ServiceContext;
import com.gingkoo.orm.entity.TestUser;
import org.springframework.beans.factory.annotation.Autowired;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.util.Map;

public class TestUserXXWHBAction extends WebAlterAction {

    @Override
    public UpdateReturnBean saveOrUpdate(MultiUpdateResultBean
multiUpdateResultBean, HttpServletRequest httpServletRequest,
HttpServletResponse httpServletResponse) throws AppException {
        //日志打印
        logger.info("-------TestUserXXWHBAction start---------");
        //创建页面回传类,作为该方法的返回值
        UpdateReturnBean updateReturnBean = new UpdateReturnBean();
         //获取页面请求域
        UpdateResultBean updateResultBean =
multiUpdateResultBean.getUpdateResultBeanByID("TestUserPage_mat_ds");
        //创建与service类交互的参数域类
        ServiceContext oc = new ServiceContext();

        //获取页面请求参数  ---- opt:类型参数  在主页面按钮事件中设置
        String opt = updateResultBean.getParameter("opt");
        if (updateResultBean.hasNext()) {
            //获取所有页面参数
            Map<String, String> map = updateResultBean.next();
            //将页面请求转换至 实体类 中
            TestUser testUser = new TestUser();
            //实体类转换方法
            mapToObject(testUser, map);

            //向与service交互的参数域中添加参数
            oc.setAttribute("Bean", testUser);
            oc.setAttribute("cmd", opt);
            oc.setAttribute("IN_PARAM_1", multiUpdateResultBean);
            //调用service类,oc参数域作为参数传入
            OPCaller.call(TestUserXXWHBService.ID, oc);
            /*
                两种调用方式
                1\运行时类调用
                OPCaller.call(TestUserXXWHBService.class, oc);
                2\spring容器中id调用  上述使用
                OPCaller.call("testUserXXWHBService", oc);
            */
        }
        //日志打印
        logger.info("-------TestUserXXWHBAction end---------");
        return updateReturnBean;
    }
```

```
}
```

> 注意调用service 类时不要直接调用或属性注入的方式进行调用。需要通过方法
>
> ```
> OPCaller.call(TestUserXXWHBService.ID, oc);
> ```
>
> 原因：service类中需要 MyHibernateTemplate模板需要进行初始化，直接调用是不能使用的。

3.4、service类数据

> 注意

```
package com.gingkoo.east.service;

import java.util.Enumeration;
import java.util.List;
import java.util.UUID;

import com.gingkoo.gf4j2.core.util.UuidHelper;
import com.gingkoo.gf4j2.framework.config.database.MyHibernateTemplateImpl;
import com.gingkoo.orm.entity.TestUser;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.hibernate.SessionFactory;
import org.hibernate.impl.SessionFactoryImpl;
import org.junit.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Component;

import com.gingkoo.gf4j2.framework.config.database.base.MyHibernateTemplate;
import com.gingkoo.gf4j2.framework.excp.CommonException;
import com.gingkoo.gf4j2.framework.service.base.BaseService;
import com.gingkoo.gf4j2.framework.service.base.ServiceContext;
import org.springframework.stereotype.Service;

import javax.annotation.Resource;


@Service
public class TestUserXXWHBService extends BaseService {
    private static final Log logger =
LogFactory.getLog(TestUserXXWHBService.class);
    public static final String ID = "testUserXXWHBService"; // spring容器中该类名
称,静态属性方便action类直接调用


    private final MyHibernateTemplate template;

    public TestUserXXWHBService(MyHibernateTemplate template) {
        this.template = template;
    }

    /**
     * execute 调用前处理
     */
```

```
    @Override
    public void afterProc(ServiceContext paramServiceContext) throws
CommonException {
    }

    /**
     * execute 调用后处理
     */
    @Override
    public void beforeProc(ServiceContext paramServiceContext) throws
CommonException {
    }

    @Override
    public void execute(ServiceContext serviceContext) throws CommonException {
        //从参数域中获取请求类型
        String cmd = (String) serviceContext.getAttribute("cmd");
        //获取实体类对象
        TestUser test_user = (TestUser) serviceContext.getAttribute("Bean");
        //根据请求类型,执行不同操作
        if("add".equals(cmd)){
            //生成一个DATA_ID
            test_user.setDataId(UuidHelper.getCleanUpperUuid().substring(0,18));
            template.save(test_user);
        }else if("mod".equals(cmd)){
            template.update(test_user);
        }else if("del".equals(cmd)){
            template.delete(test_user);
        }
    }
}
```

！注意点

SQL语句关键子必须小写

> SqlParserImpl类中，用来拼接sql语句参数时，用来分隔sql语句内容的默认时用小写的关键字信息，大写的关键字容易报错
>
> 详细代码可见：SqlParserImpl

```
            s1 = s.substring(s.indexOf(":\"") + 2, s.length());
            whereprefix = false;
            int i0 = s0.lastIndexOf( str: " where ");
            i = s0.lastIndexOf( str: " and ");
            j = s0.lastIndexOf( str: " or ");
            i = s0.lastIndexOf( str: "(");
            int i4 = s0.lastIndexOf( str: ")");
```