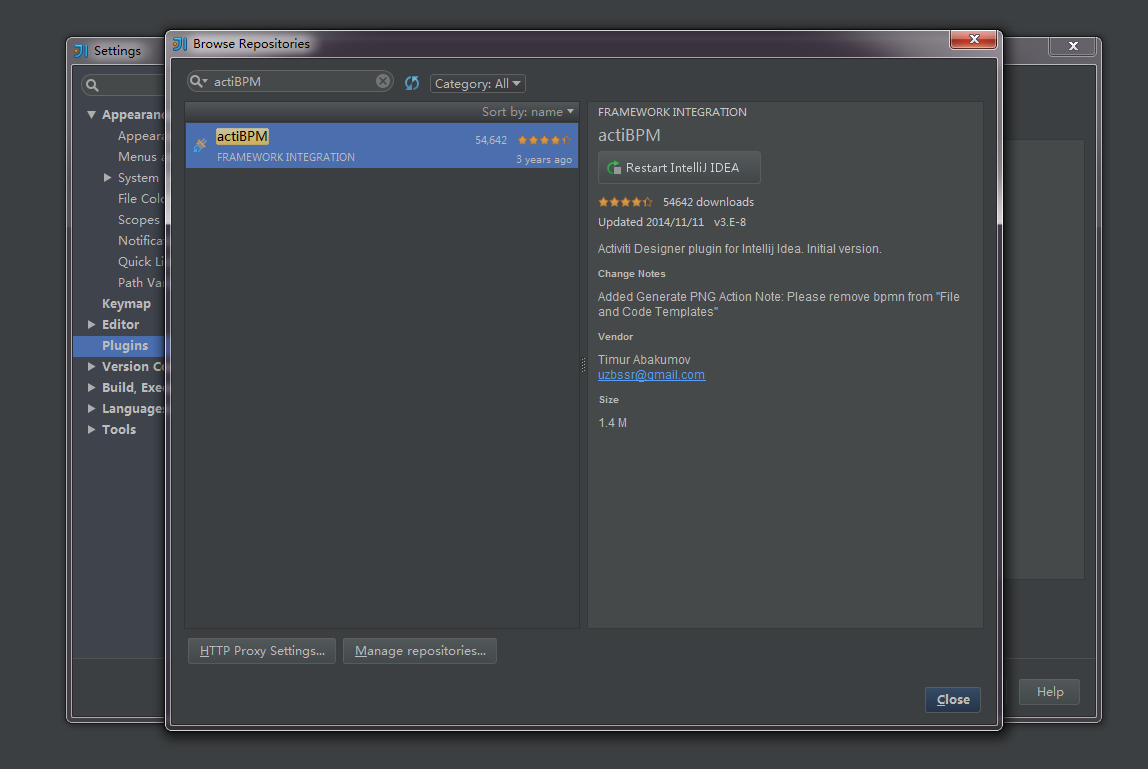
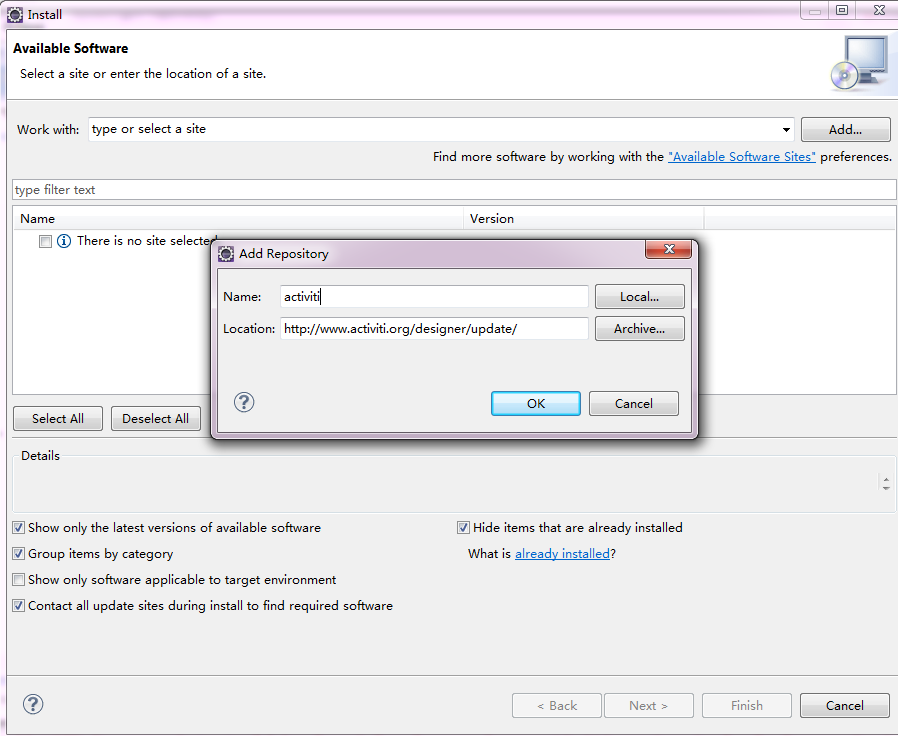
# 一、环境的搭建

IDEA 安装activiti插件:

## 1）在IDEA中ctrl+alt+s打开setting--🡪找到Pluging

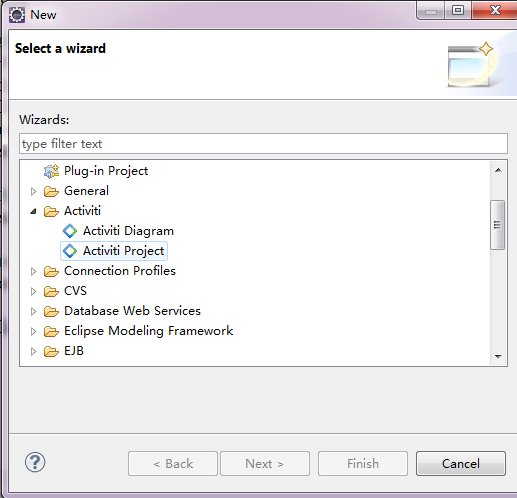


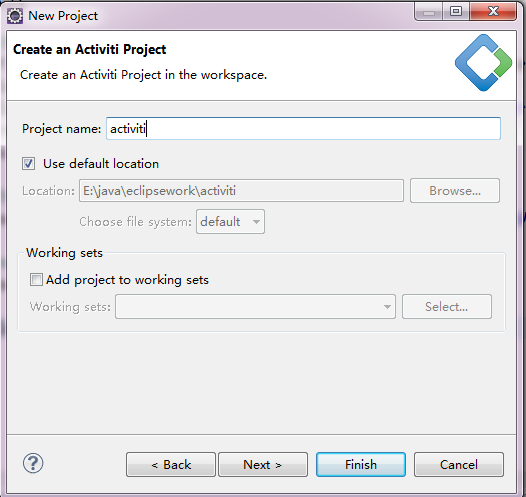
## 2）Eclipse添加插件：



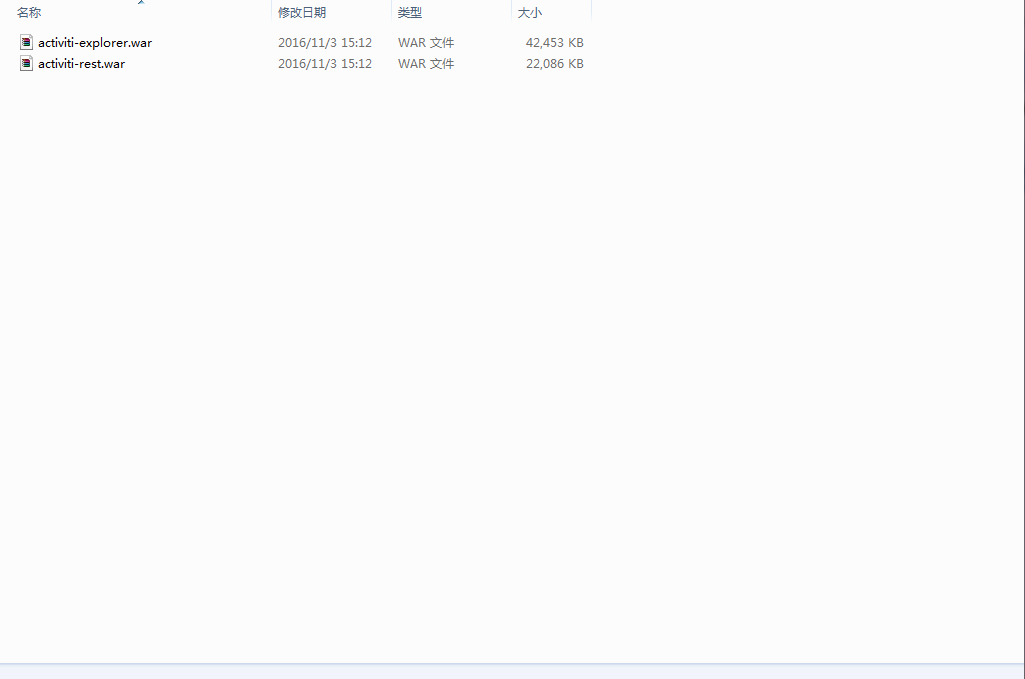
<http://www.activiti.org/designer/update/>

装完插件之后在eclipse中创建activiti项目：





从activiti5的文件中找到activiti-rest.war:

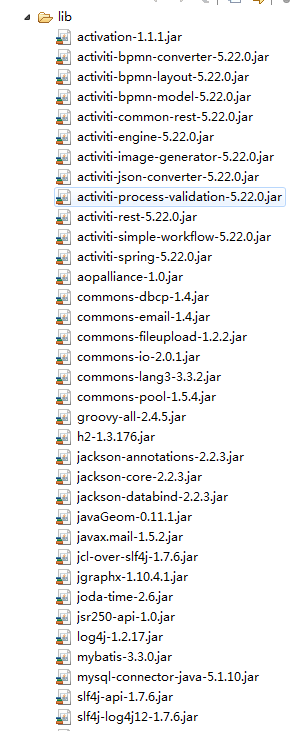


### 解压出出来，找到里面的lib文件，里面就有项目所需要的jar：



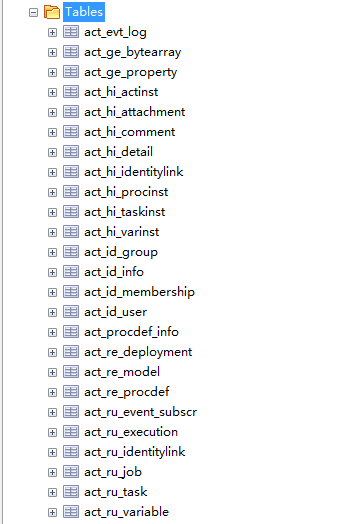
注意：这里面没有mysql的驱动包，所以驱动包自己另外导入

### 导入jar到eclipse的lib目录下：



### 工作流的表结构：

由于版本的不同这里创建了25张表：



**ACT\_RE\_\*: 'RE'表示repository。** 这个前缀的表包含了流程定义和流程静态资源 （图片，规则，等等）。

**ACT\_RU\_\*: 'RU'表示runtime**。 这些运行时的表，包含流程实例，任务，变量，异步任务，等运行中的数据。 Activiti只在流程实例执行过程中保存这些数据， 在流程结束时就会删除这些记录。 这样运行时表可以一直很小速度很快。

**ACT\_ID\_\*: 'ID'表示identity。** 这些表包含身份信息，比如用户，组等等。

**ACT\_HI\_\*: 'HI'表示history。** 这些表包含历史数据，比如历史流程实例， 变量，任务等等。

**ACT\_GE\_\*: 通用数据**， 用于不同场景下，如存放资源文件。

#### 资源库流程规则表

#### 1) act\_re\_deployment 部署信息表

#### 2) act\_re\_model  流程设计模型部署表

#### 3) act\_re\_procdef  流程定义数据表

#### 

#### 运行时数据库表

#### 1) act\_ru\_execution运行时流程执行实例表

#### 2) act\_ru\_identitylink运行时流程人员表，主要存储任务节点与参与者的相关信息

#### 3) act\_ru\_task运行时任务节点表

#### 4) act\_ru\_variable运行时流程变量数据表

#### 

#### 历史数据库表

1) act\_hi\_actinst 历史节点表

2) act\_hi\_attachment历史附件表

3) act\_hi\_comment历史意见表

4) act\_hi\_identitylink历史流程人员表

5) act\_hi\_detail历史详情表，提供历史变量的查询

6) act\_hi\_procinst历史流程实例表

7) act\_hi\_taskinst历史任务实例表

8) act\_hi\_varinst历史变量表

#### 组织机构表

1) act\_id\_group用户组信息表

2) act\_id\_info用户扩展信息表

3) act\_id\_membership用户与用户组对应信息表

4) act\_id\_user用户信息表

这四张表很常见，基本的组织机构管理，关于用户认证方面建议还是自己开发一套，组件自带的功能太简单，使用中有很多需求难以满足

#### 通用数据表

1) act\_ge\_bytearray二进制数据表

2) act\_ge\_property属性数据表存储整个流程引擎级别的数据,初始化表结构时，会默认插入三条记录，

# 二、工作流HelloWorld项目

## (1).工作流的创建

### 1)使用java方式来创建activiti的数据库文件：自己本地有要创建好一个对应的数据库

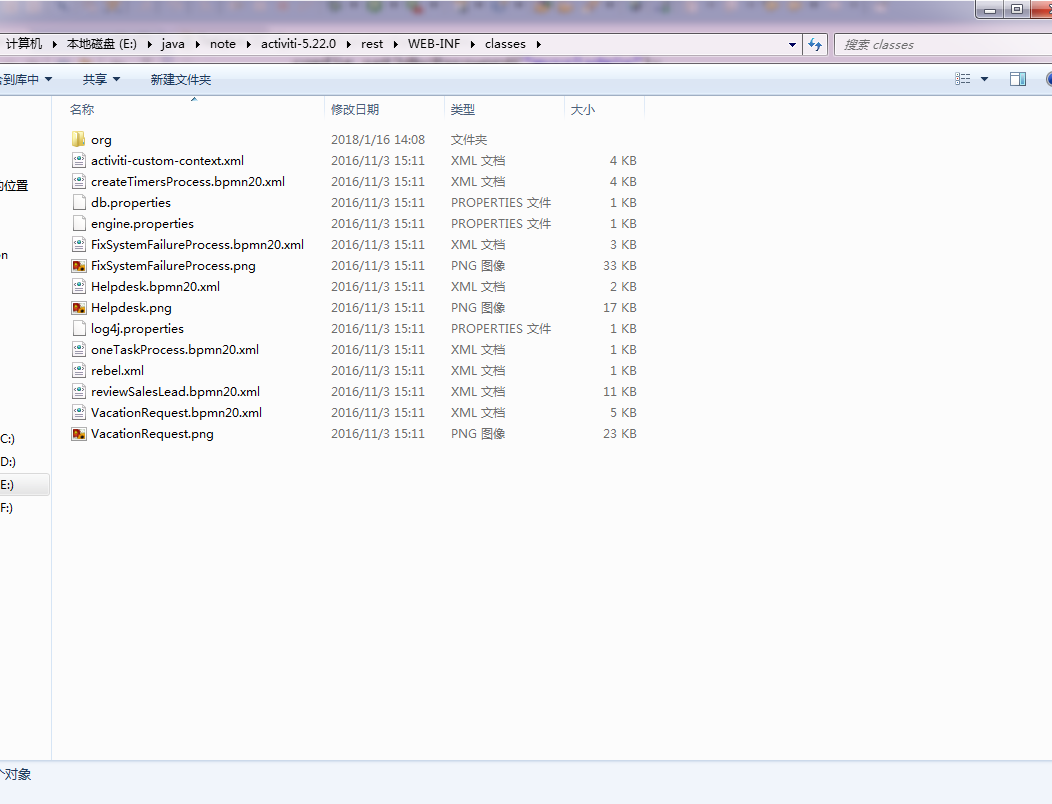


代码：

|  |
| --- |
| **public** **class** Test {  @org.junit.Test  **public** **void** TestJava(){    //使用java方式配置数据库  ProcessEngineConfiguration config = ProcessEngineConfiguration.*createStandaloneProcessEngineConfiguration*();  config.setJdbcDriver("com.mysql.jdbc.Driver");  config.setJdbcUrl("jdbc:mysql://localhost:3306/activiti?useUnicode=true&characterEncoding=UTF8");  config.setJdbcUsername("root");  config.setJdbcPassword("mysqladmin");  //配置如果没有表就自动创建  config.setDatabaseSchemaUpdate(ProcessEngineConfiguration.*DB\_SCHEMA\_UPDATE\_TRUE*);  //创建ProcessEngine  ProcessEngine process = config.buildProcessEngine();    System.*out*.println("创建成功："+process);  }  } |

## 2)使用自定义的xml的方式来创建：

从刚刚的activiti-rest.war解压出来的文件下，可以找到对应的配置文件：



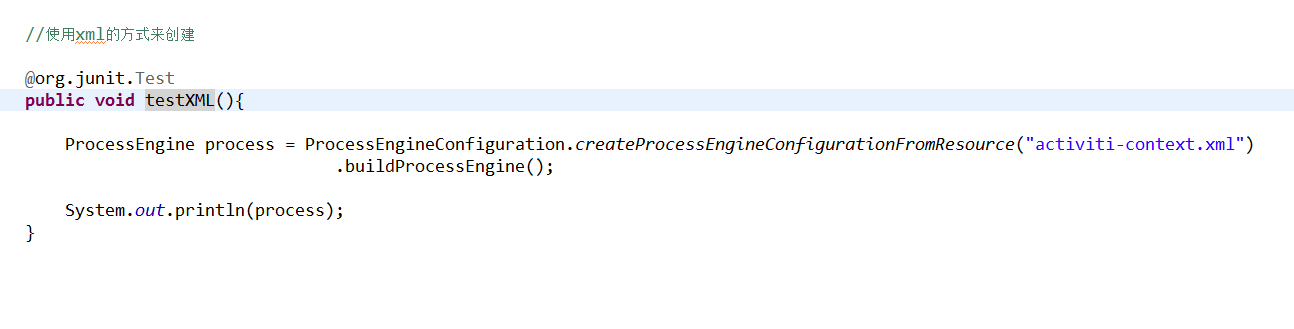
Activiti-content.xml的简单配置:



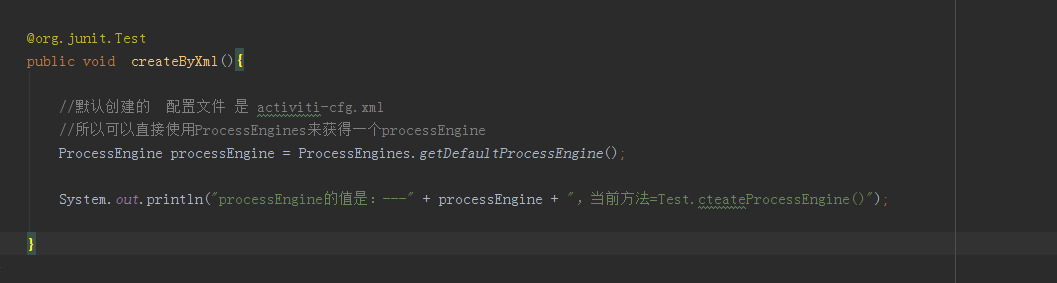
### 代码：

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <beans xmlns=*"http://www.springframework.org/schema/beans"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns:context=*"http://www.springframework.org/schema/context"* xmlns:tx=*"http://www.springframework.org/schema/tx"*  xmlns:jee=*"http://www.springframework.org/schema/jee"* xmlns:aop=*"http://www.springframework.org/schema/aop"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.0.xsd*  *http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.0.xsd*  *http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-jee-3.0.xsd*  *http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-3.0.xsd"*>  <!--  //使用java方式配置数据库  ProcessEngineConfiguration config = ProcessEngineConfiguration.createStandaloneProcessEngineConfiguration();  config.setJdbcDriver("com.mysql.jdbc.Driver");  config.setJdbcUrl("jdbc:mysql://localhost:3306/activiti?useUnicode=true&characterEncoding=UTF8");  config.setJdbcUsername("root");  config.setJdbcPassword("mysqladmin");    //配置如果没有表就自动创建  config.setDatabaseSchemaUpdate(ProcessEngineConfiguration.DB\_SCHEMA\_UPDATE\_TRUE);    //创建ProcessEngine  ProcessEngine process = config.buildProcessEngine();    System.out.println("创建成功："+process);    -->  <bean name=*"processEngineConfiguration"* class=*"org.activiti.engine.impl.cfg.StandaloneProcessEngineConfiguration"*>  <property name=*"jdbcDriver"* value=*"com.mysql.jdbc.Driver"*/>  <property name=*"jdbcUrl"* value=*"jdbc:mysql://localhost:3306/activiti"*/>  <property name=*"jdbcUsername"* value=*"root"*/>  <property name=*"jdbcPassword"* value=*"mysqladmin"*/>  <property name=*"databaseSchemaUpdate"* value=*"true"*/>  </bean>  </beans> |

## 测试：



### 3)使用activiti框架默认加载的activiti-cfg.xml来获得一个ProcessEngine 对象：



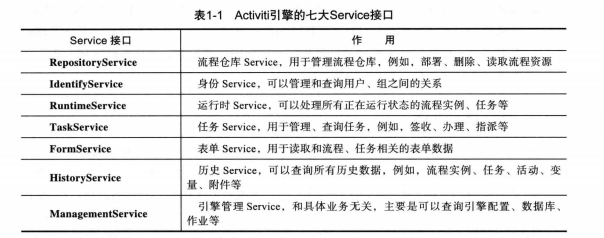
|  |
| --- |
| @org.junit.Test public void createByXml(){   //默认创建的 配置文件 是 activiti-cfg.xml  //所以可以直接使用ProcessEngines来获得一个processEngine  ProcessEngine processEngine = ProcessEngines.*getDefaultProcessEngine*();   System.*out*.println("processEngine的值是：---" + processEngine + "，当前方法=Test.cteateProcessEngine()");  } |

### 注意配置文件的名字一定要叫：activiti-cfg.xml

## (2).操作工作流

## 使用activiti的八大service来操作工作流：介绍

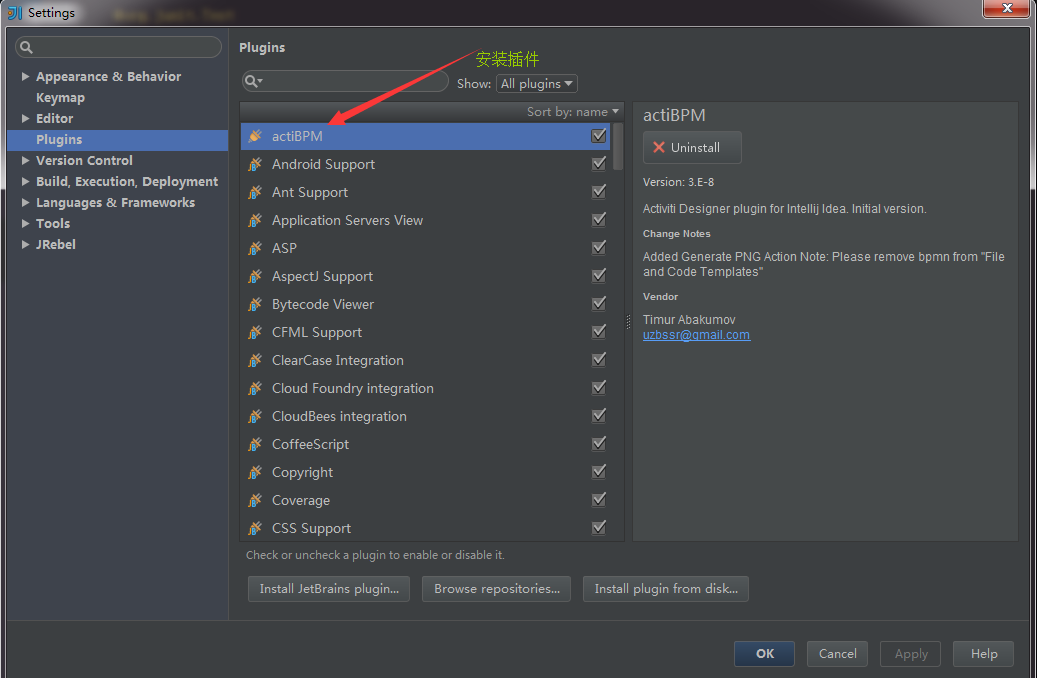
|  |
| --- |
| org.activiti.engine.DynamicBpmnService; // org.activiti.engine.FormService; //与表单相关的 org.activiti.engine.HistoryService; //操作历史的工作流 org.activiti.engine.IdentityService; //身份认证的 org.activiti.engine.ManagementService;//总得管理器 org.activiti.engine.RepositoryService;//定义,部署相关的 org.activiti.engine.RuntimeService;//与正在执行流程实例，对象相关的 org.activiti.engine.TaskService; //与流程任务相关的 |

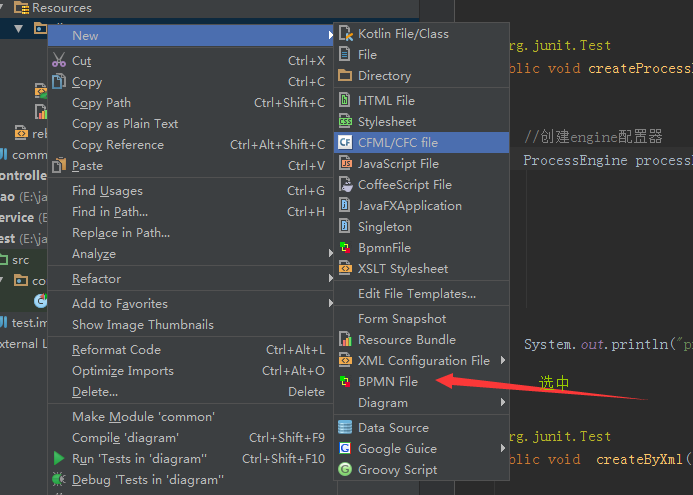


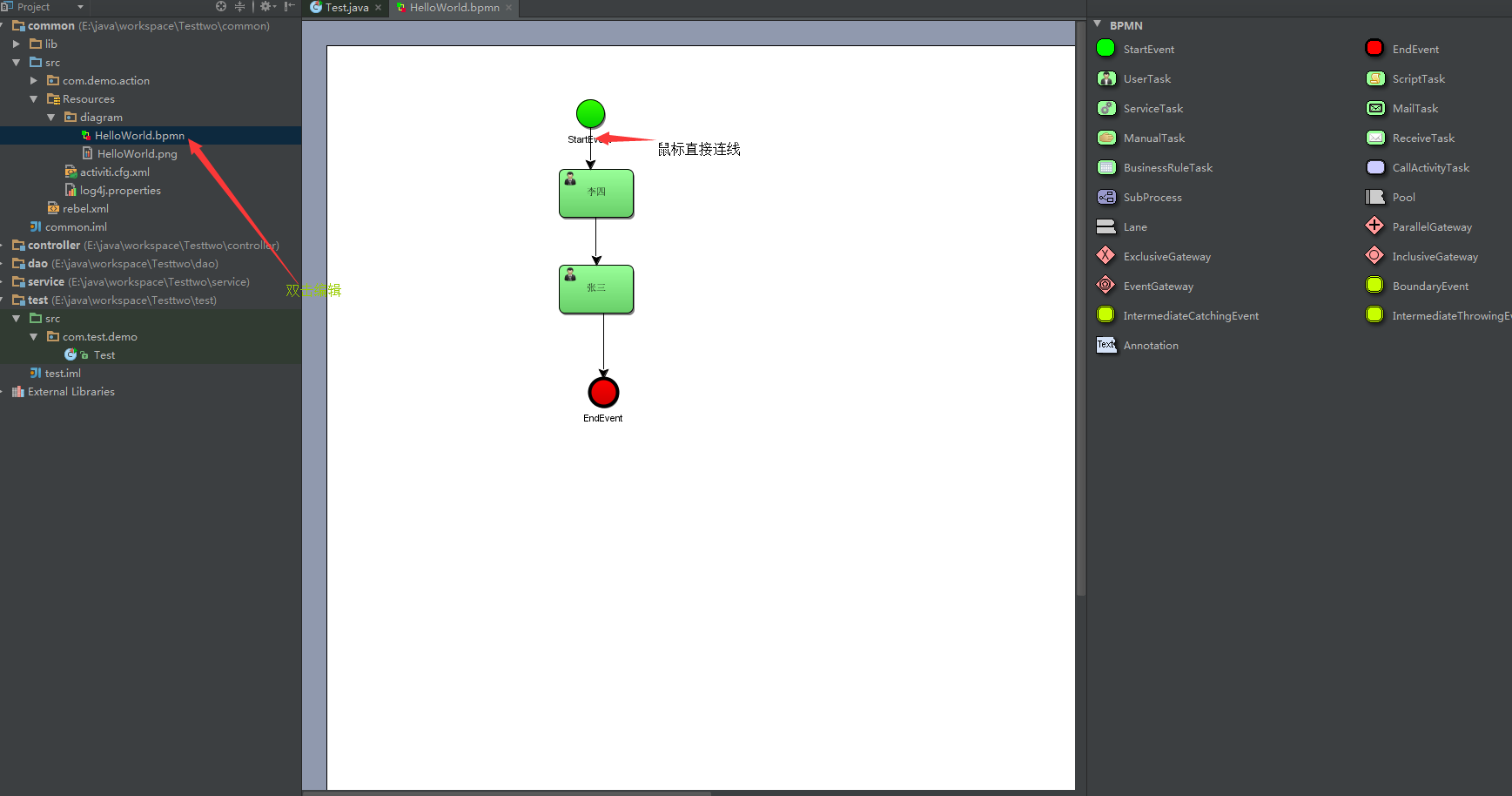
摘自activiti实战

1. 操作工作流首先需要有bpmn定义流程的文件：

在idea安装actiBPM插件：之后会重新启动

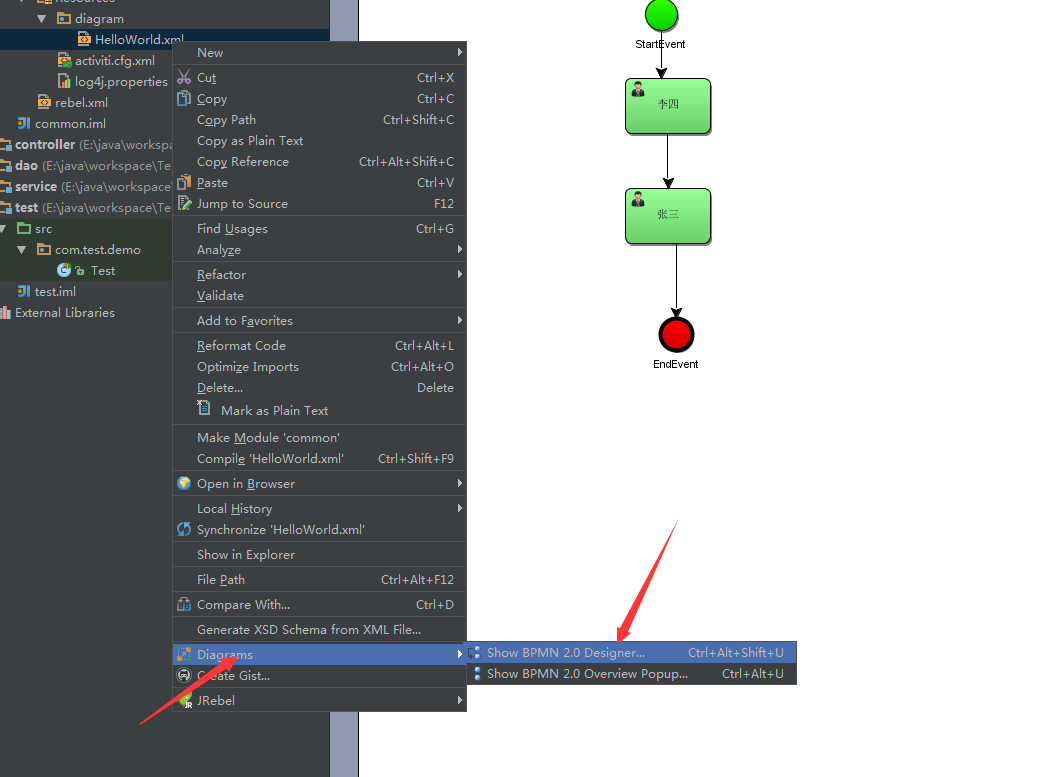


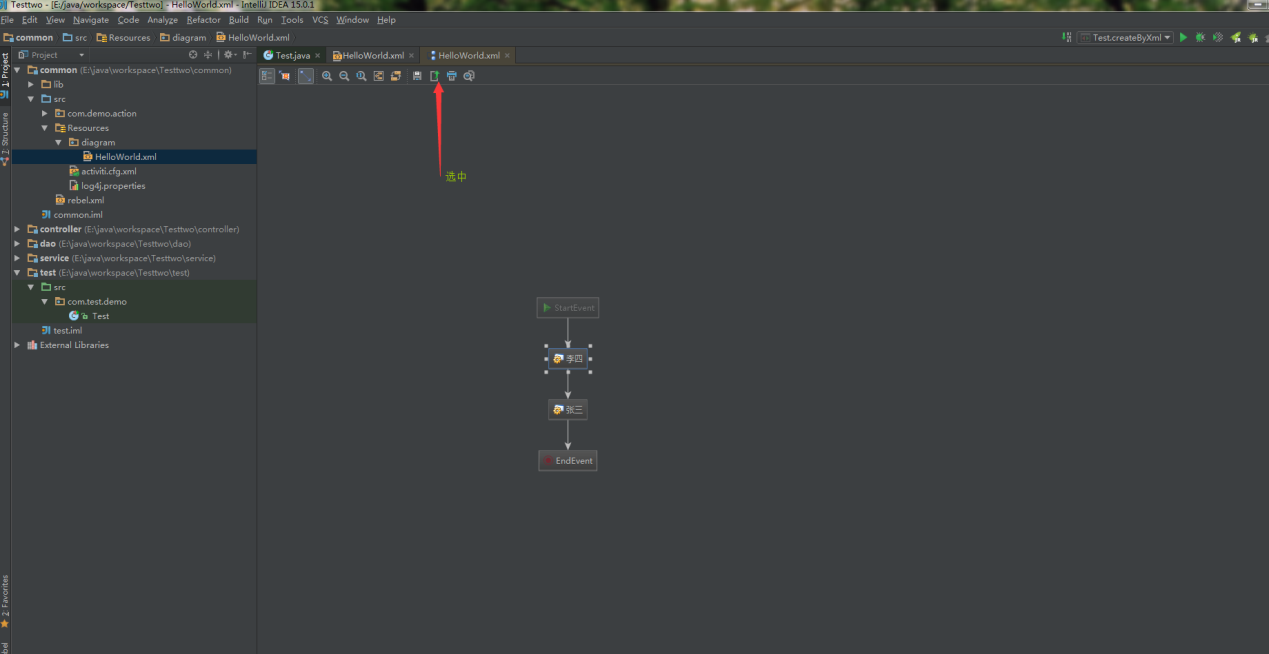


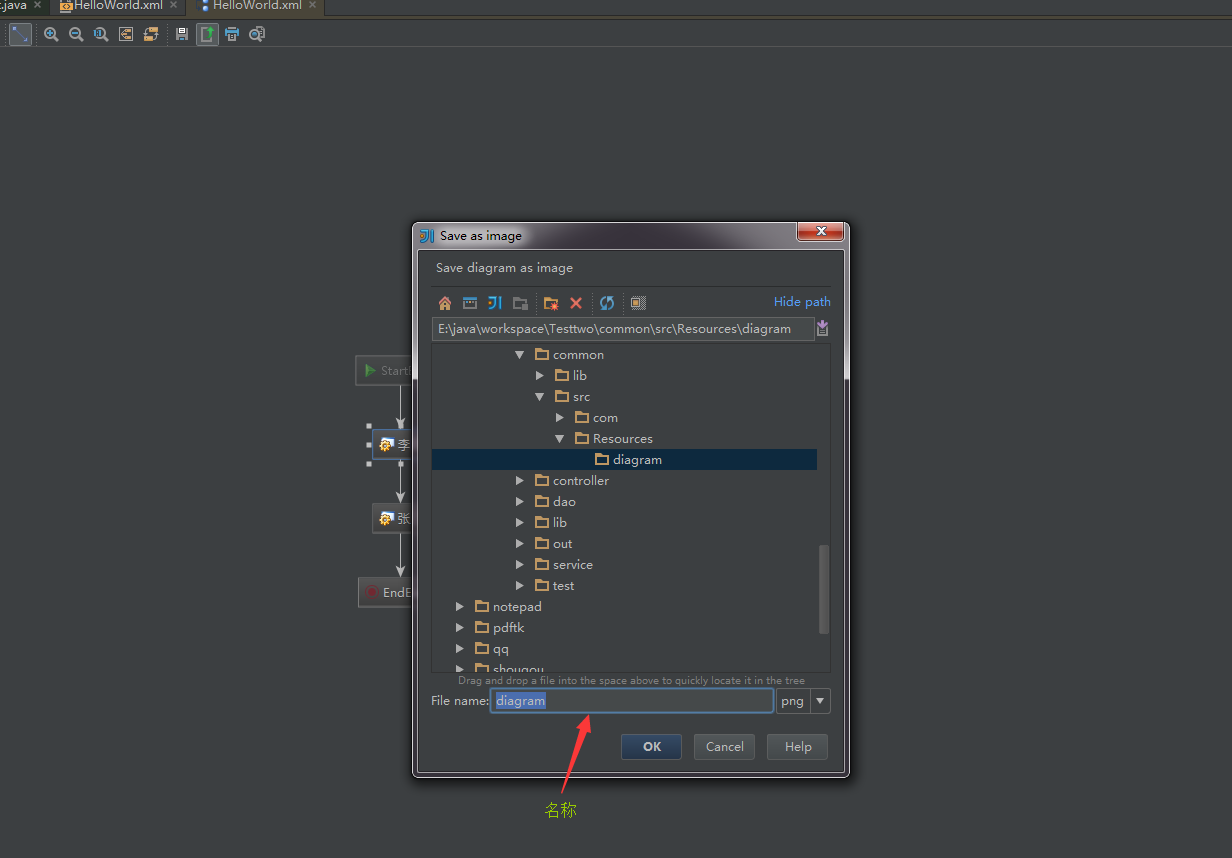


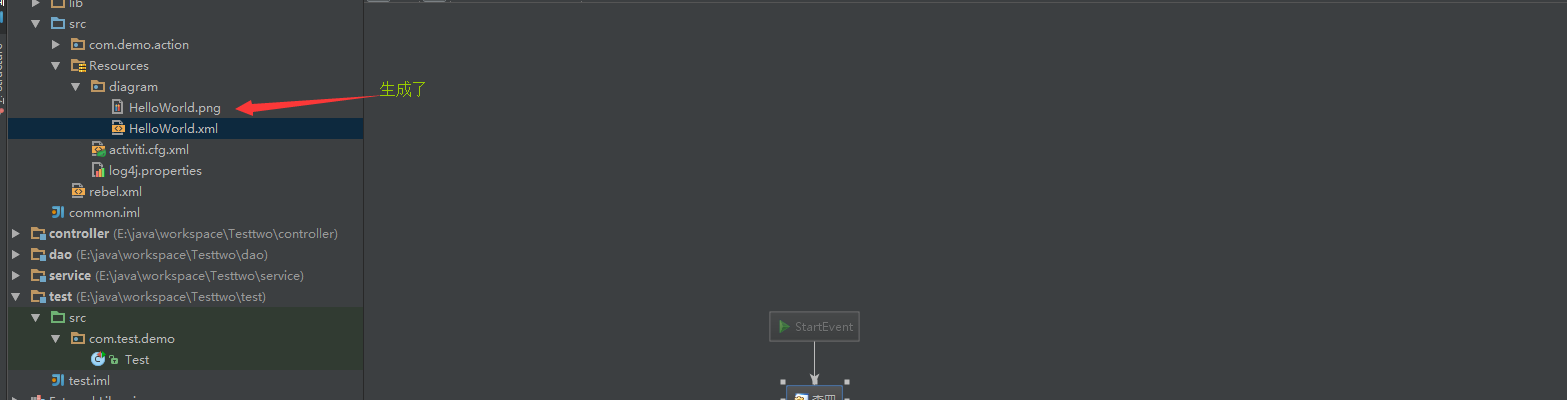
#### 问题：但是保存之后没有.png图片

解决：把HelloWorld.bpmn文件重新命名为HelloWorld.xml,然后安装下面操作：





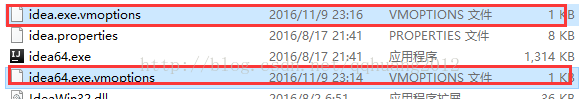


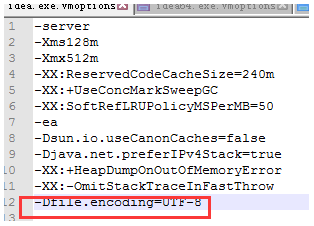


解决里面中文乱码：

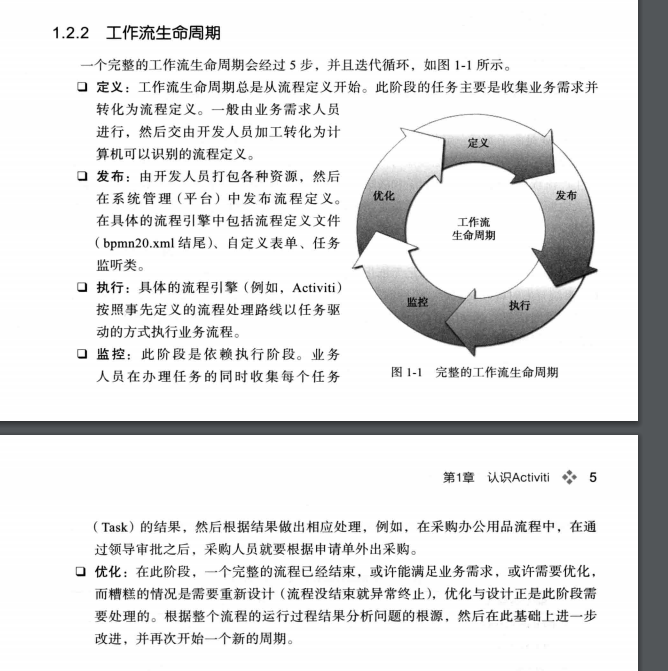
解决此问题，需要修改idea源字符集，修改方法如下：

找到idea安装目录bin目录下如下图所示两个文件，用编辑器打开，在文件末尾添加 -Dfile.encoding=UTF-8 ，然后重启idea，再打开流程图就会发现中文已经可以正常显示了。



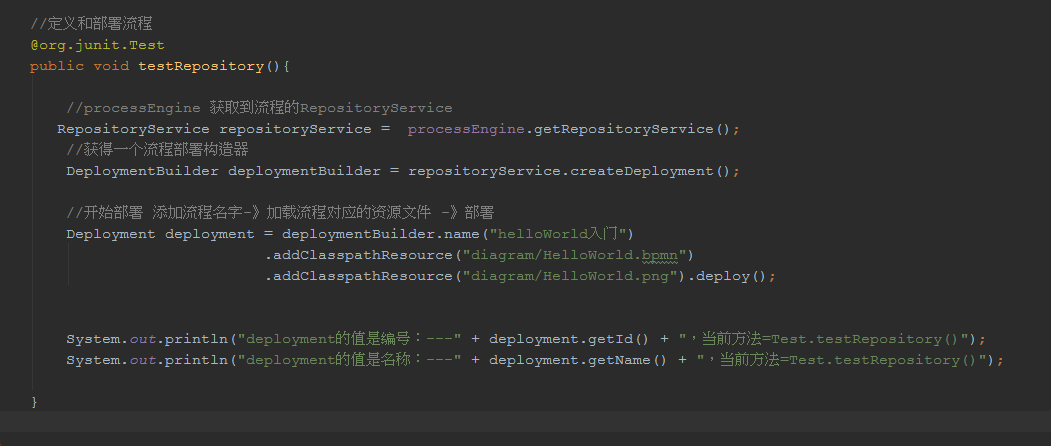


## 工作流的生命周期：



### 一、定义和部署工作流使用 RepositoryService：

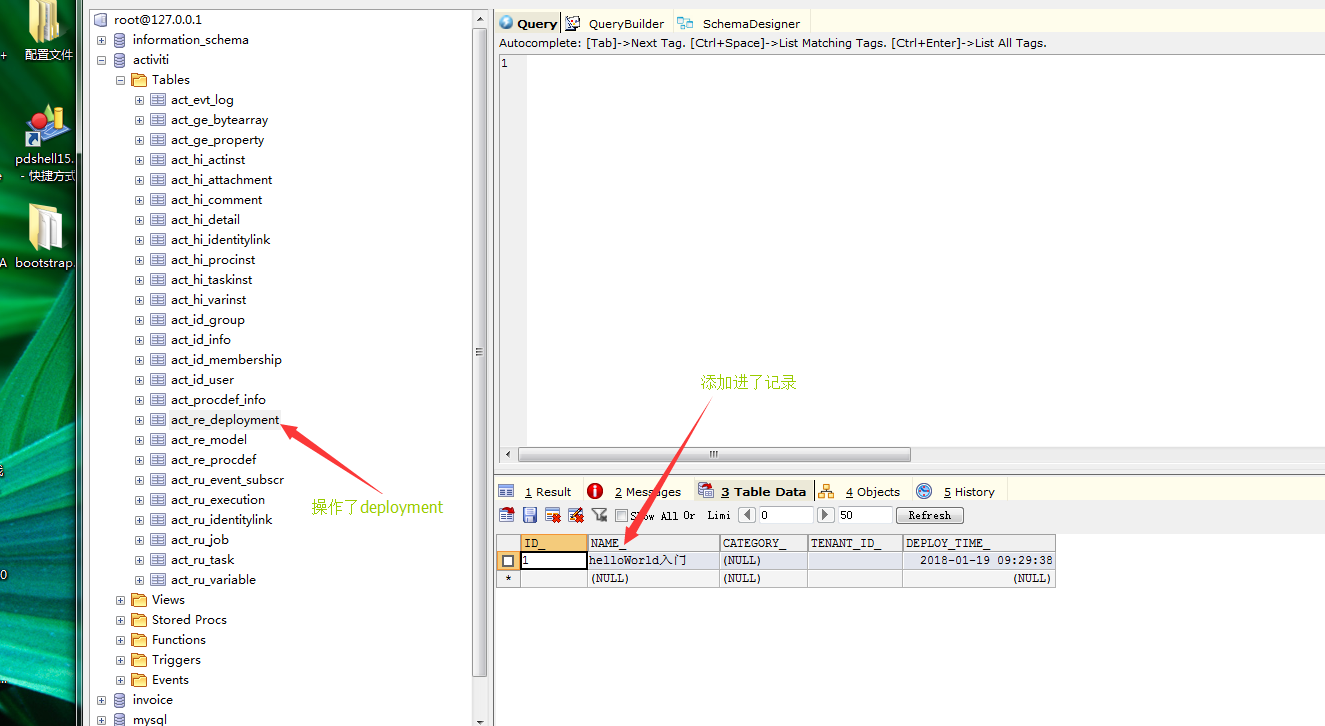
步骤：获取到部署服务(RepositoryService)-》获得一个流程部署构造器（DeploymentBuilder）-》添加流程名字-》加载流程对应的资源文件 -》部署

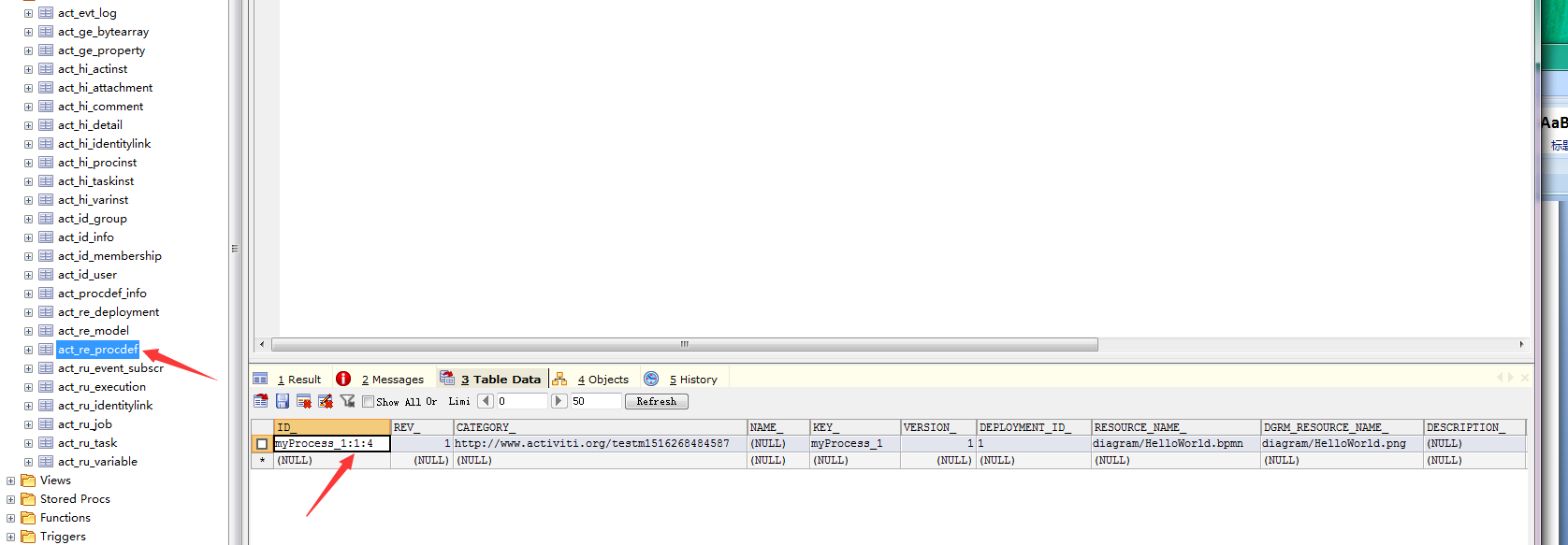


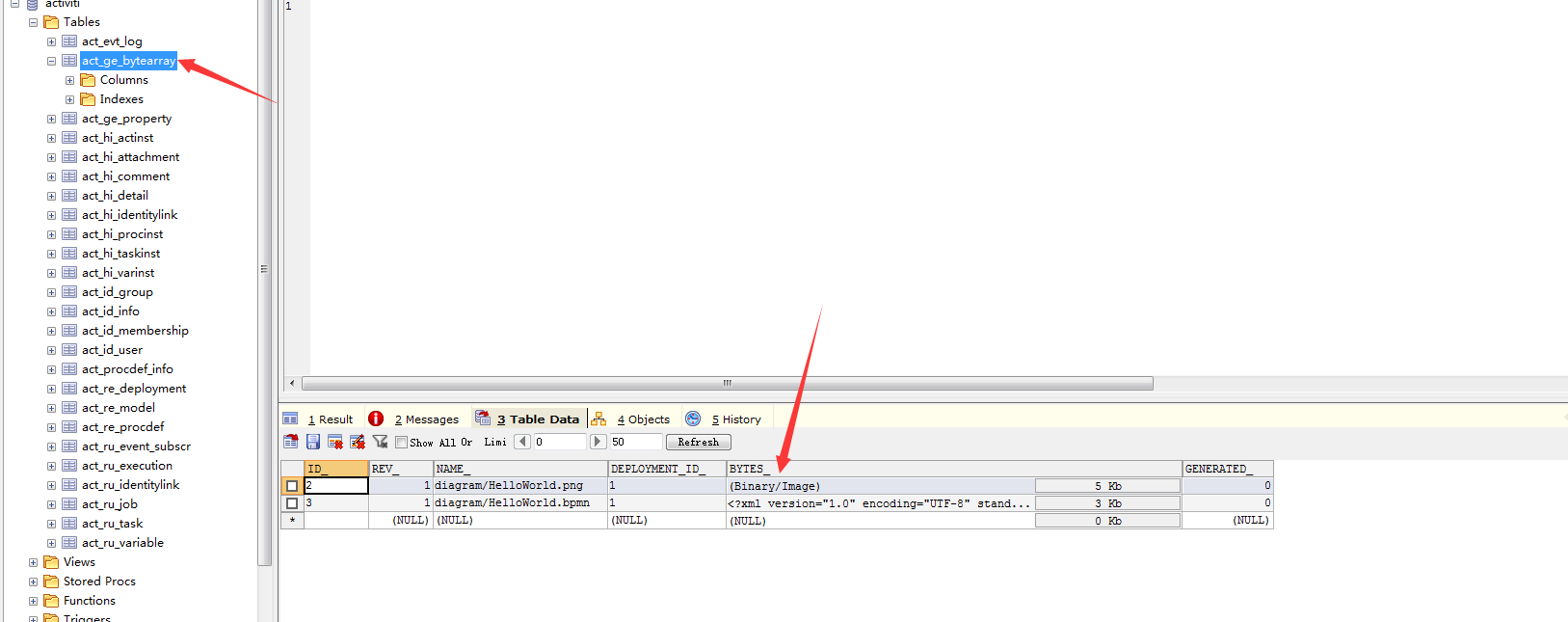
### 代码如下：

|  |
| --- |
| //定义和部署流程 @org.junit.Test public void testRepository(){   //processEngine 获取到流程的RepositoryService  RepositoryService repositoryService = processEngine.getRepositoryService();  //获得一个流程部署构造器  DeploymentBuilder deploymentBuilder = repositoryService.createDeployment();   //开始部署 添加流程名字-》加载流程对应的资源文件 -》部署  Deployment deployment = deploymentBuilder.name("helloWorld入门")  .addClasspathResource("diagram/HelloWorld.bpmn")  .addClasspathResource("diagram/HelloWorld.png").deploy();  System.*out*.println("deployment的值是编号：---" + deployment.getId() + "，当前方法=Test.testRepository()");  System.*out*.println("deployment的值是名称：---" + deployment.getName() + "，当前方法=Test.testRepository()");  } |

### 执行后查看数据库：在执行定义部署之后操作了act\_re\_deployment 、act\_re\_procdef和act\_ge\_bytearray



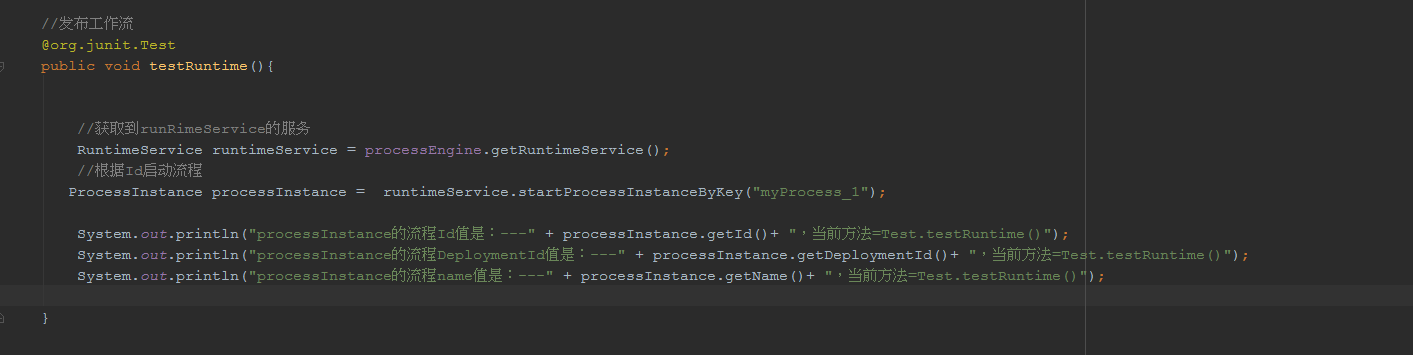






三个表之间的关系

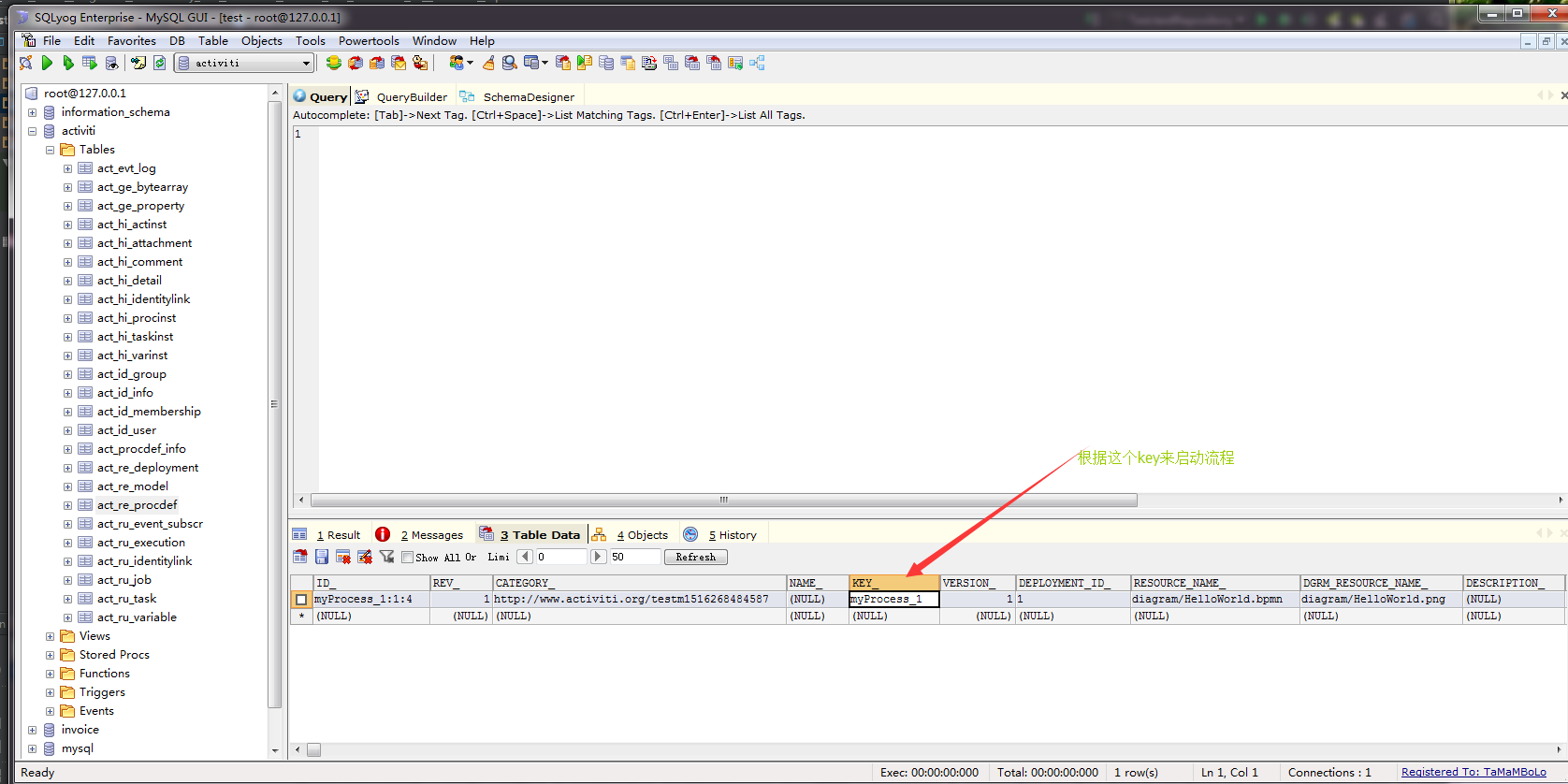
### 发布工作流：



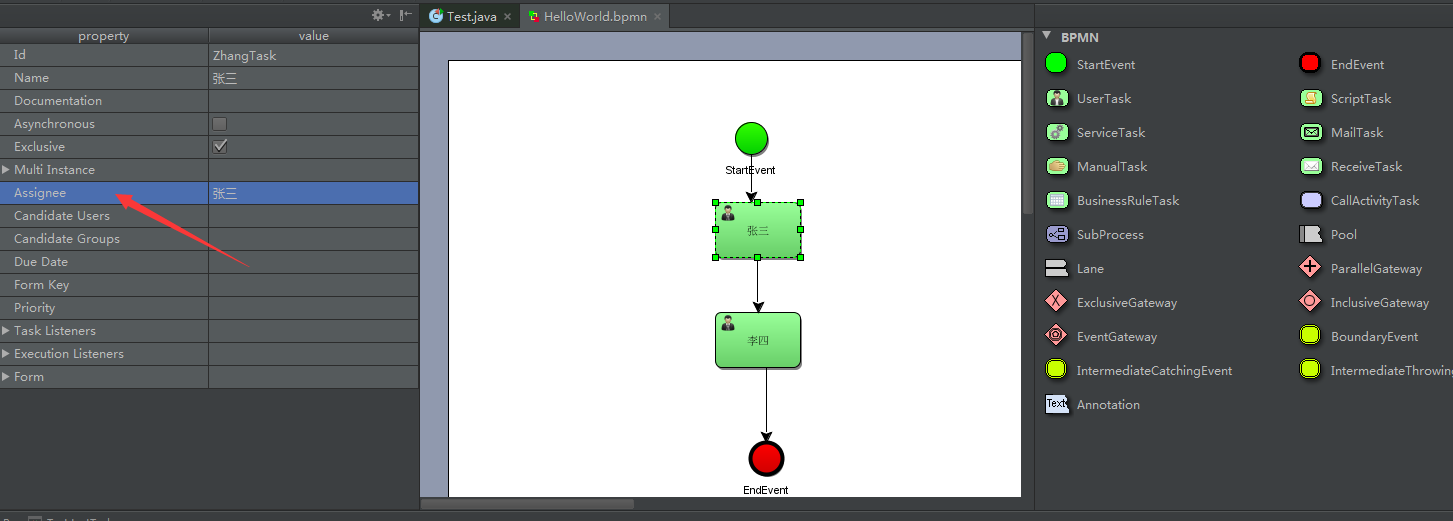
代码：

|  |
| --- |
| //发布工作流 @org.junit.Test public void testRuntime(){    //获取到runRimeService的服务  RuntimeService runtimeService = processEngine.getRuntimeService();  //根据Id启动流程  ProcessInstance processInstance = runtimeService.startProcessInstanceByKey("myProcess\_1");   System.*out*.println("processInstance的流程Id值是：---" + processInstance.getId()+ "，当前方法=Test.testRuntime()");  System.*out*.println("processInstance的流程DeploymentId值是：---" + processInstance.getDeploymentId()+ "，当前方法=Test.testRuntime()");  System.*out*.println("processInstance的流程name值是：---" + processInstance.getName()+ "，当前方法=Test.testRuntime()");  } |

数据库中key的位置在act\_re\_procdef表中：



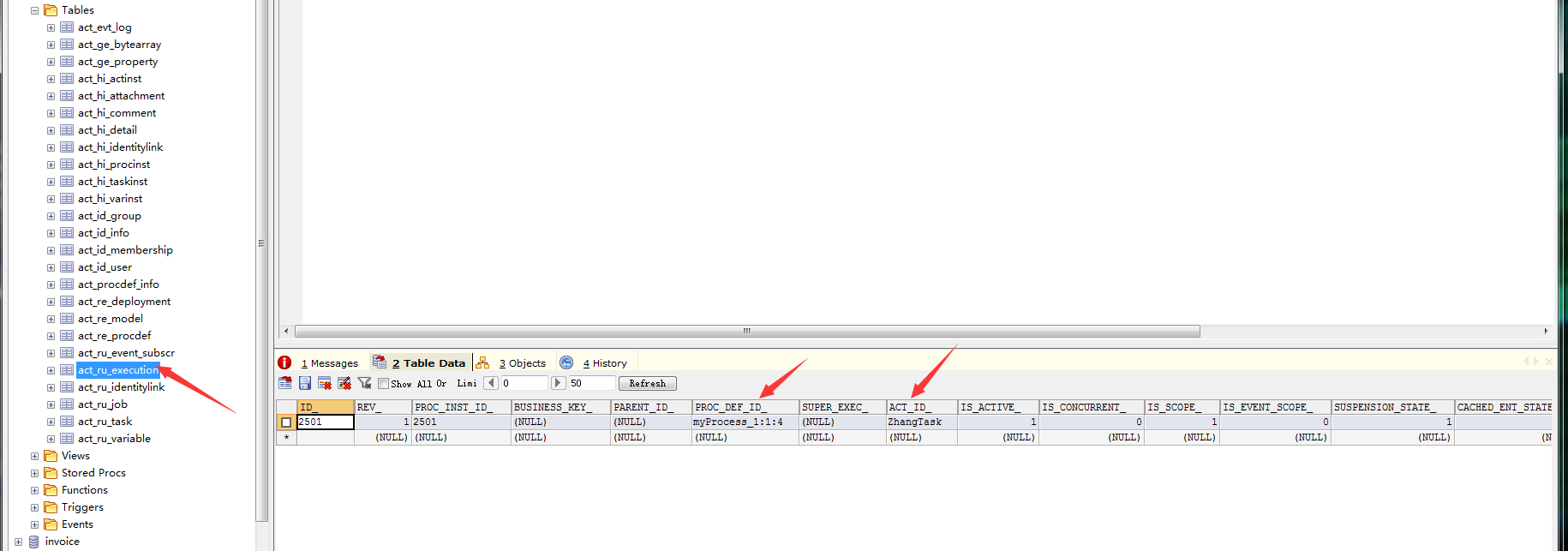
Bpm图中：



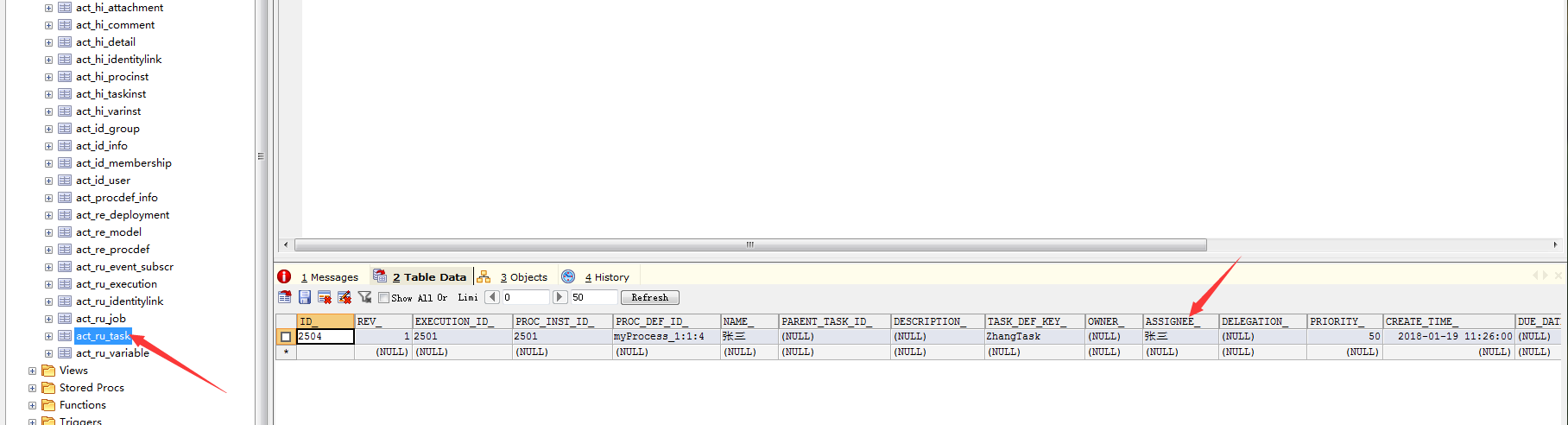
### 查看数据库：

操作到的表是：act\_ru\_execution(启动任务表)，act\_ru\_task(运行的任务表)、act\_hi\_actinst 历史节点表、act\_hi\_procinst历史流程实例表、act\_hi\_taskinst历史任务实例表

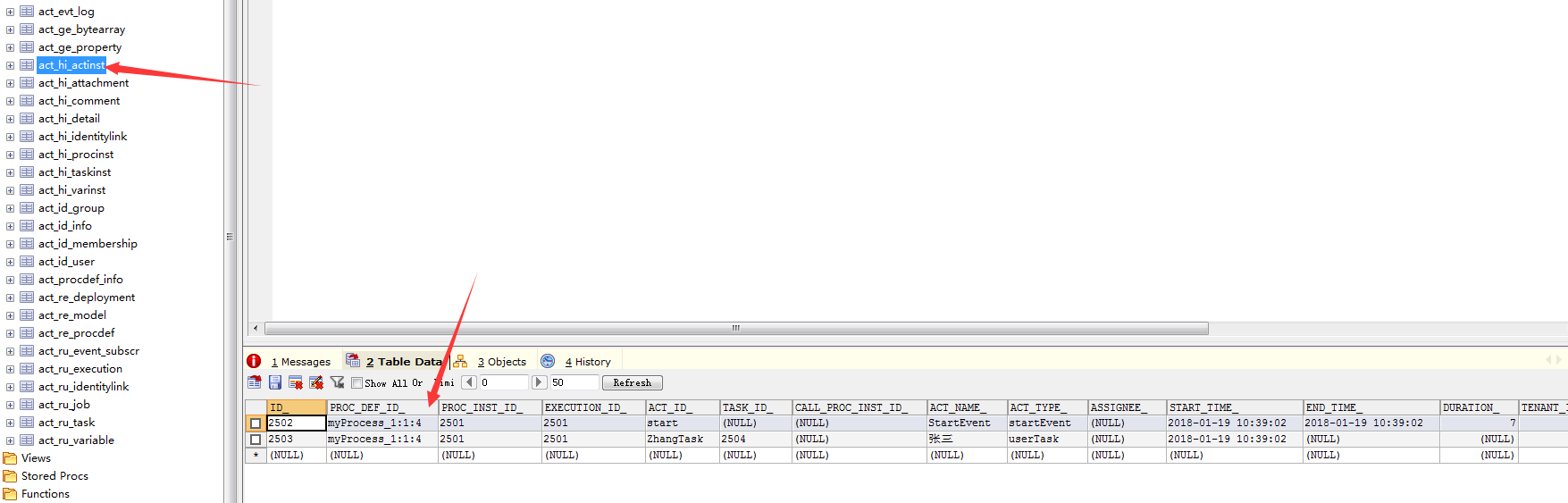
act\_ru\_executio



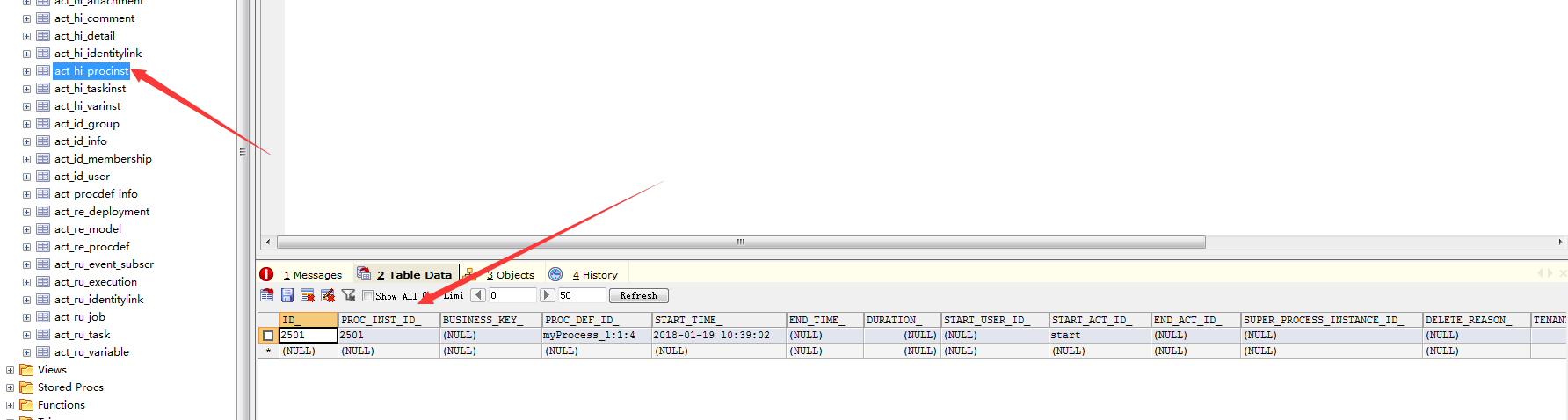
act\_ru\_task



历史节点表：

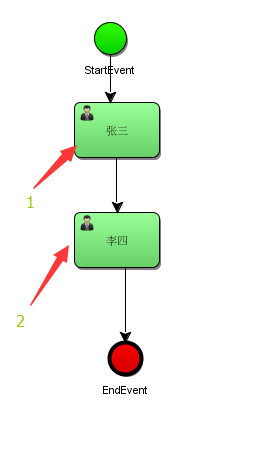


历史流程表：

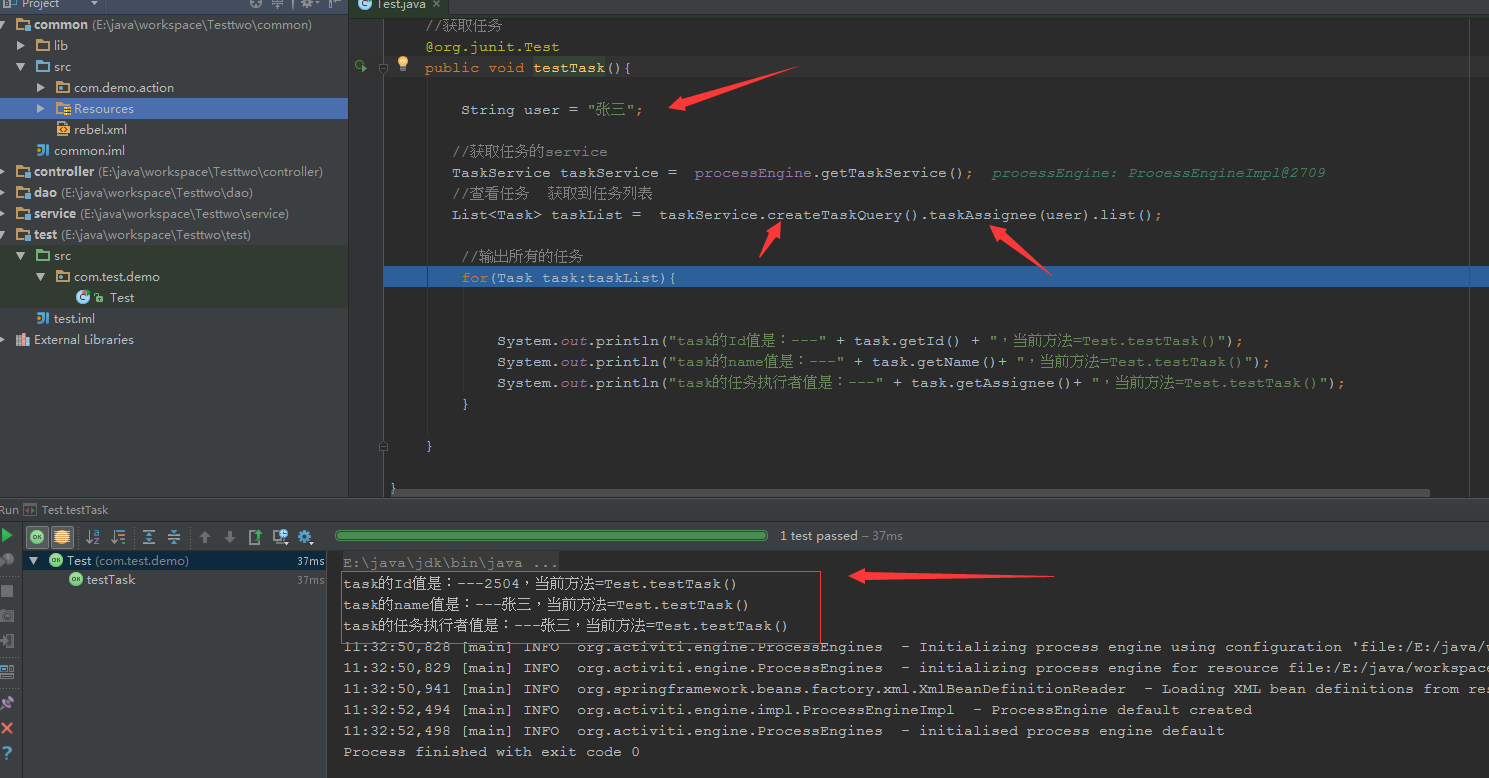


## 执行任务

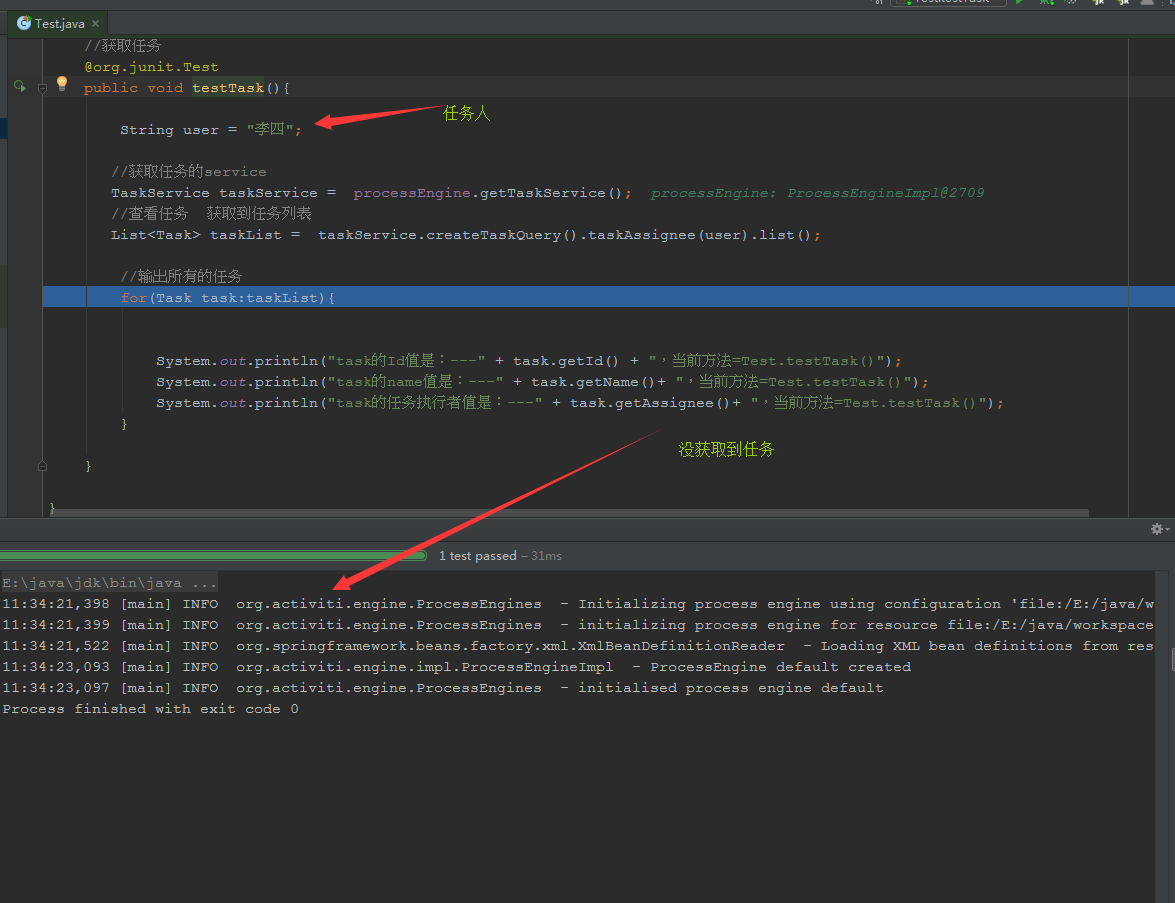
测试任务的执行：



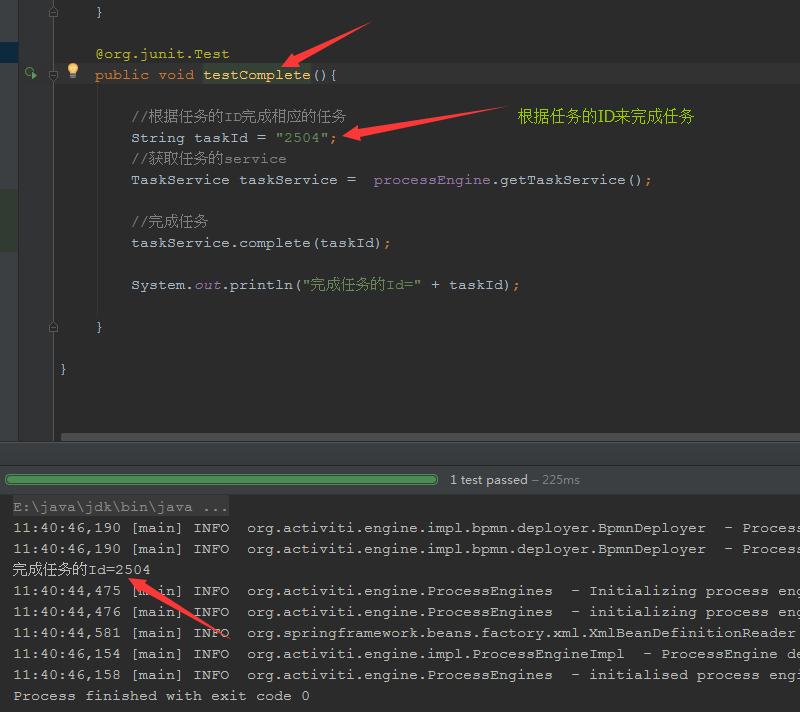
1）张三获取到任务



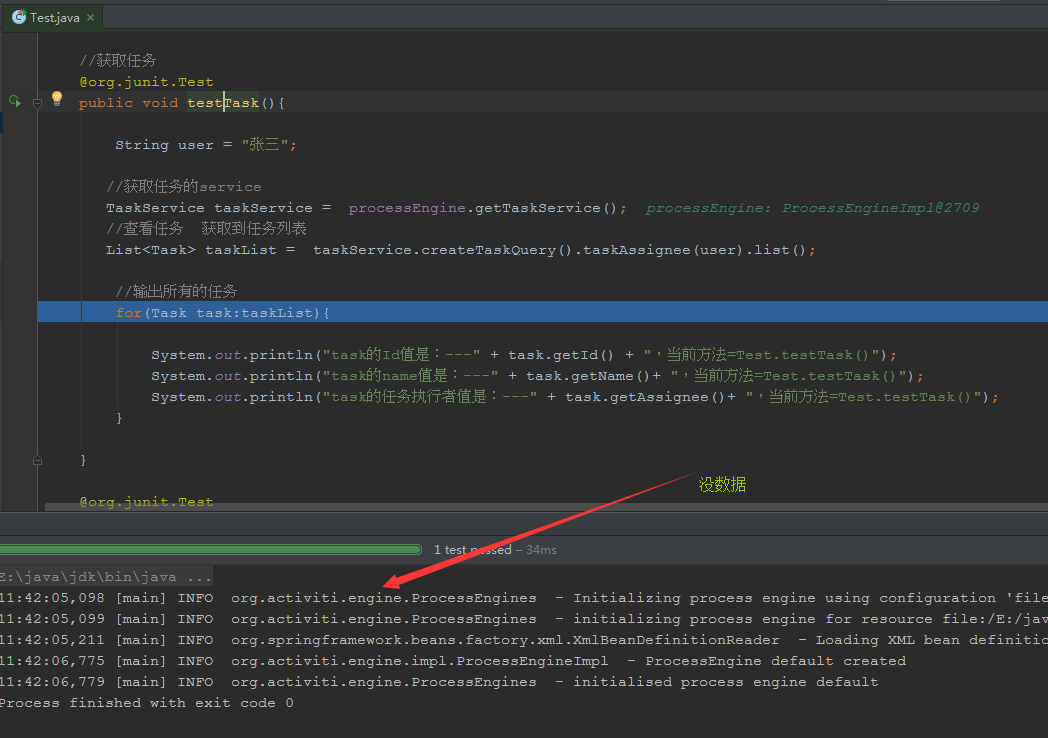
李四获取任务：由于张三还没有完成任务，所以流程还没有到李四那里，所以李四不会有任务



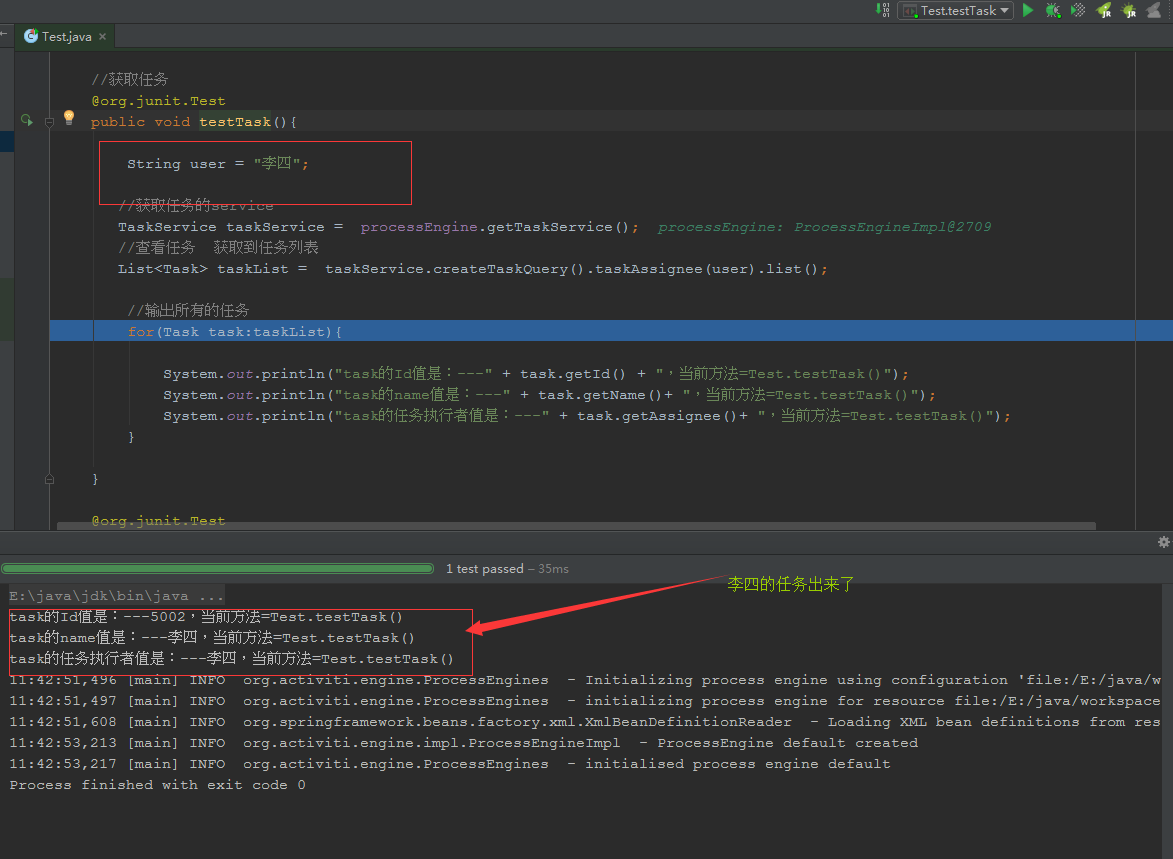
2）张三完成了任务:



重新查看张三的任务：完成任务后对应的任务没有了，对于他来说他的流程完成了

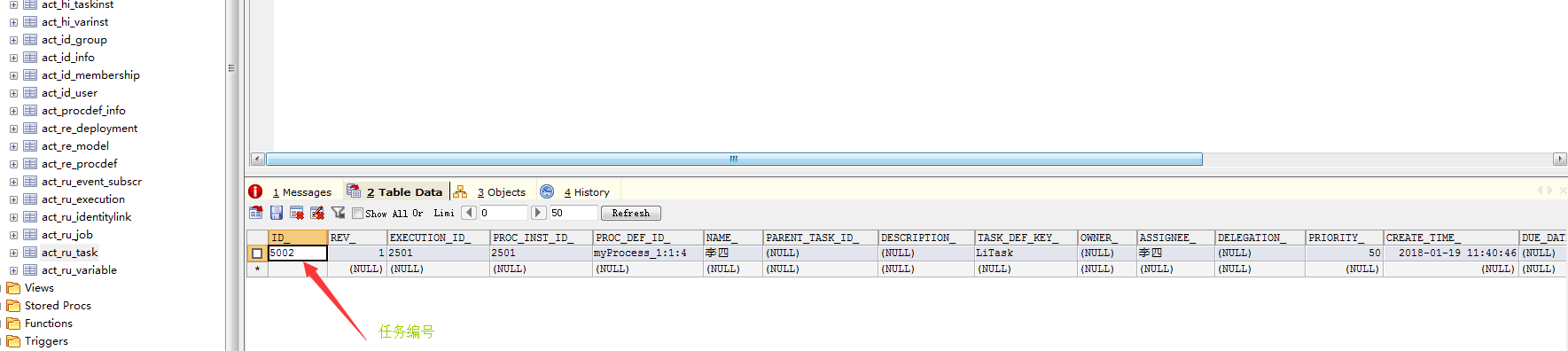


查看李四的任务：由于张三任务完成了接下来流程到了李四那里

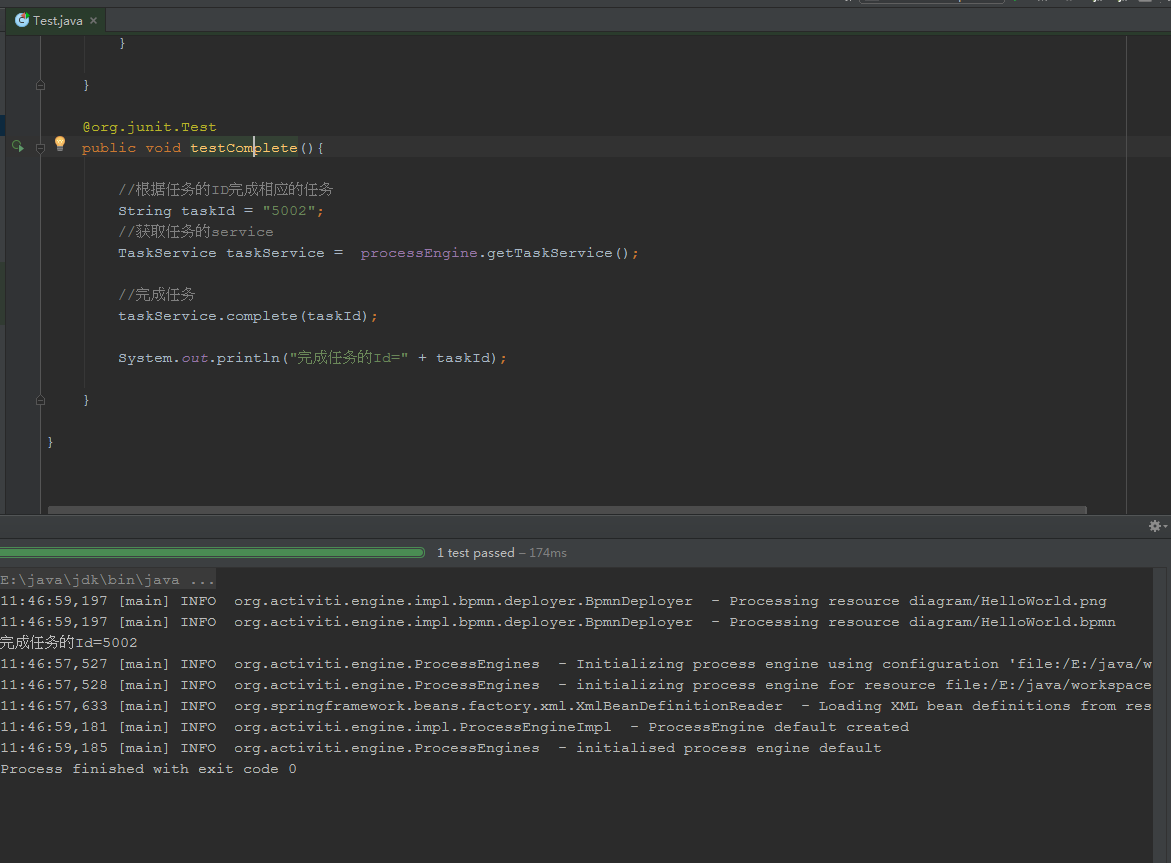


李四完成任务：

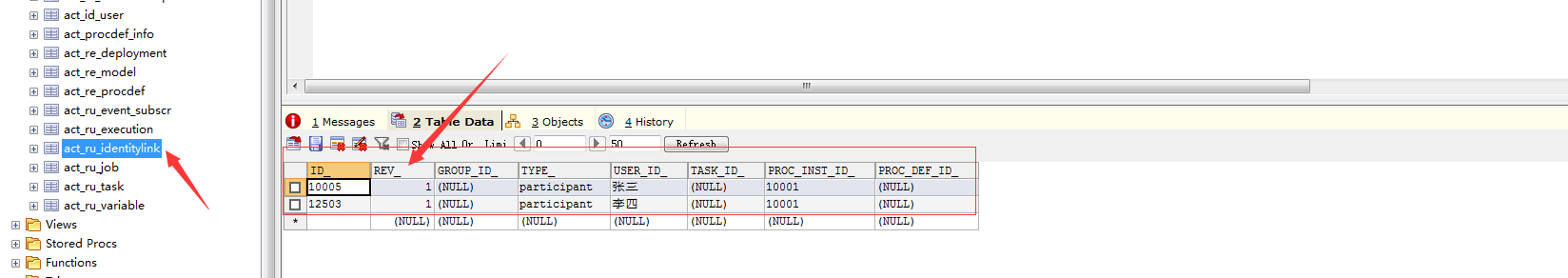
当前运行的任务表act\_ru\_task：查看李四任务的ID



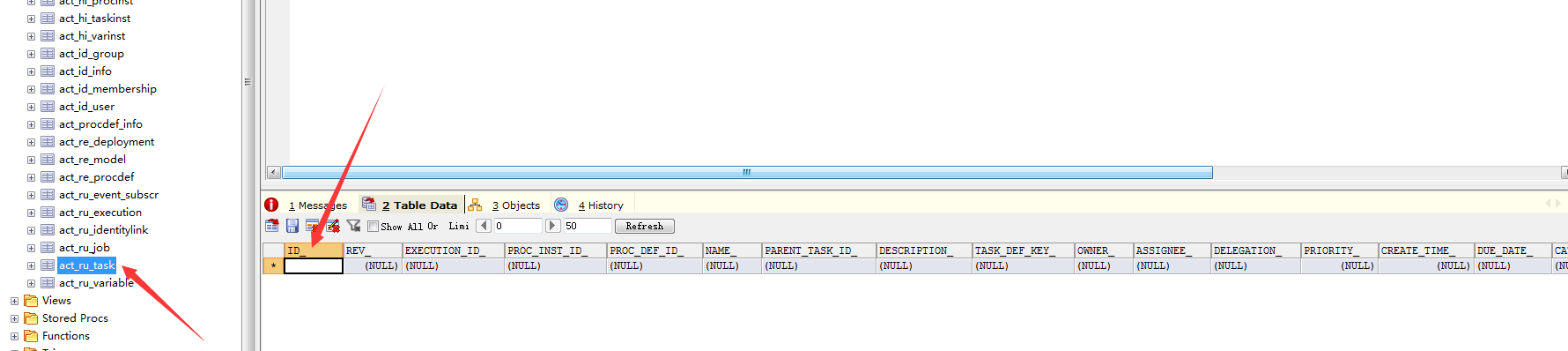
完成任务：



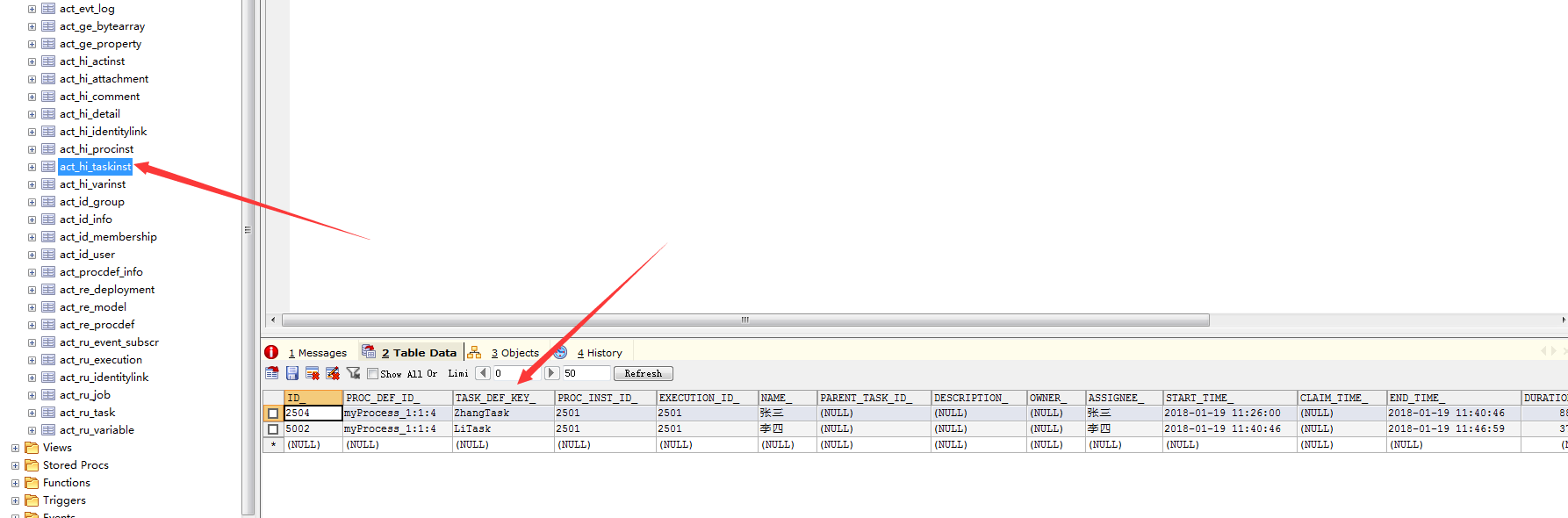
在任务执行期间，act\_ru\_identitylink表会关联两个user，当任务完成该表数据自动清空



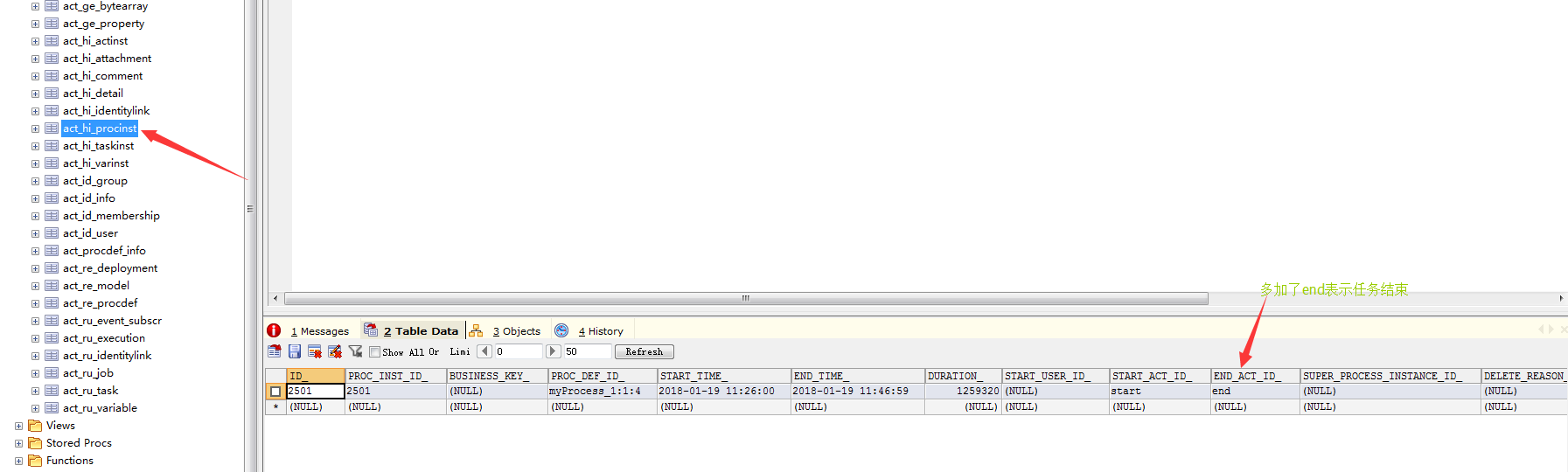
重新查看act\_ru\_task表的数据：此刻没有正在运行的任务，表示刚刚的任务流程已经完成



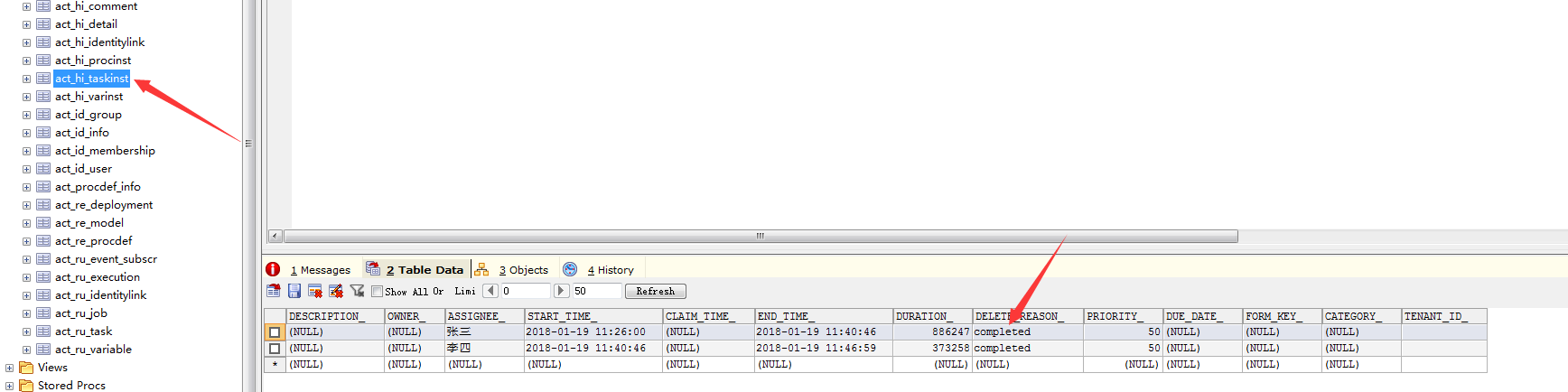
act\_hi\_taskinst记录任务的历史信息



act\_hi\_procinst历史流程表，添加了一个’end’表示一个流程的结束



历史节点表：添加了completed



代码：

|  |
| --- |
| //获取任务 @org.junit.Test public void testTask(){   String user = "李四"; //任务执行人   //获取任务的service  TaskService taskService = processEngine.getTaskService();  //查看任务 获取到任务列表  List<Task> taskList = taskService.createTaskQuery().taskAssignee(user).list();   //输出所有的任务  for(Task task:taskList){   System.*out*.println("task的Id值是：---" + task.getId() + "，当前方法=Test.testTask()");  System.*out*.println("task的name值是：---" + task.getName()+ "，当前方法=Test.testTask()");  System.*out*.println("task的任务执行者值是：---" + task.getAssignee()+ "，当前方法=Test.testTask()");  }  }  @org.junit.Test public void testComplete(){   //根据任务的ID完成相应的任务  String taskId = "12502"; //任务编号，根据你的taskId不同自己改变  //获取任务的service  TaskService taskService = processEngine.getTaskService();   //完成任务  taskService.complete(taskId);   System.*out*.println("完成任务的Id=" + taskId);  } |

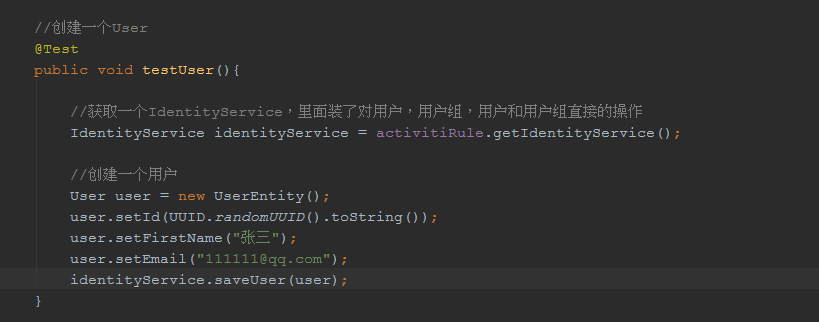
## 操作工作流的act\_id\_user, act\_id\_group，act\_id\_membership(两者关系表)

这里使用了org.activiti.engine.test.ActivitiRule 下面的测试函数来获取对应的service，其中如果要使用该类，配置文件默认要为activiti-cfg.xml

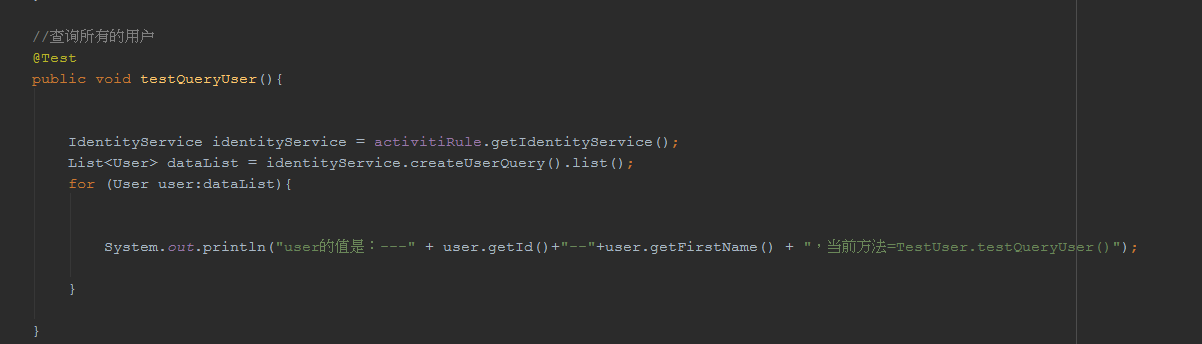
|  |
| --- |
| @Rule public ActivitiRule activitiRule = new ActivitiRule(); |



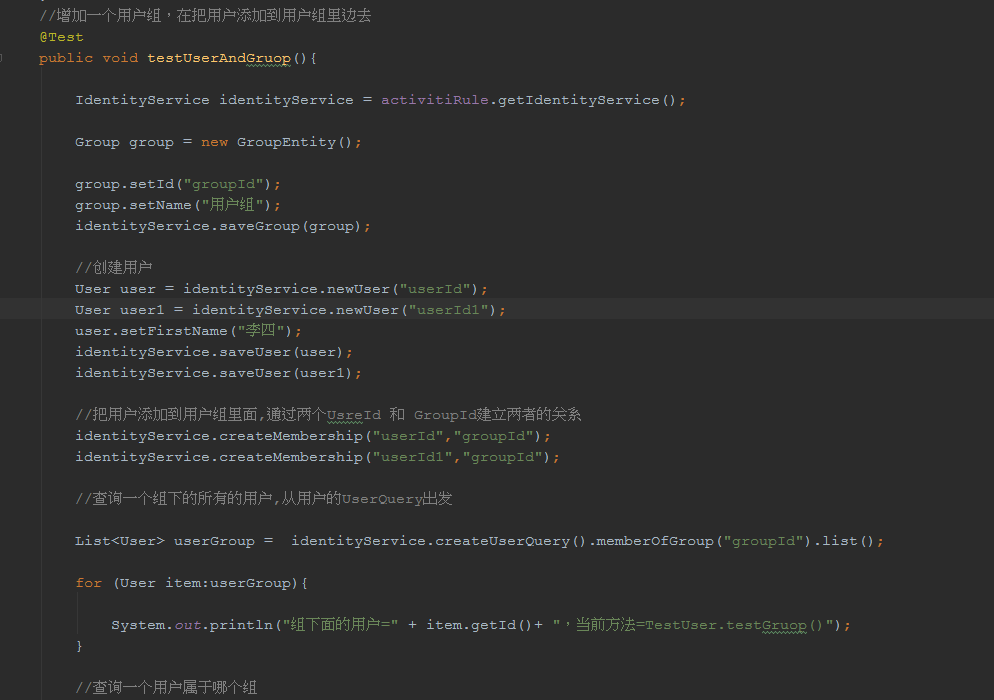
### 创建一个User

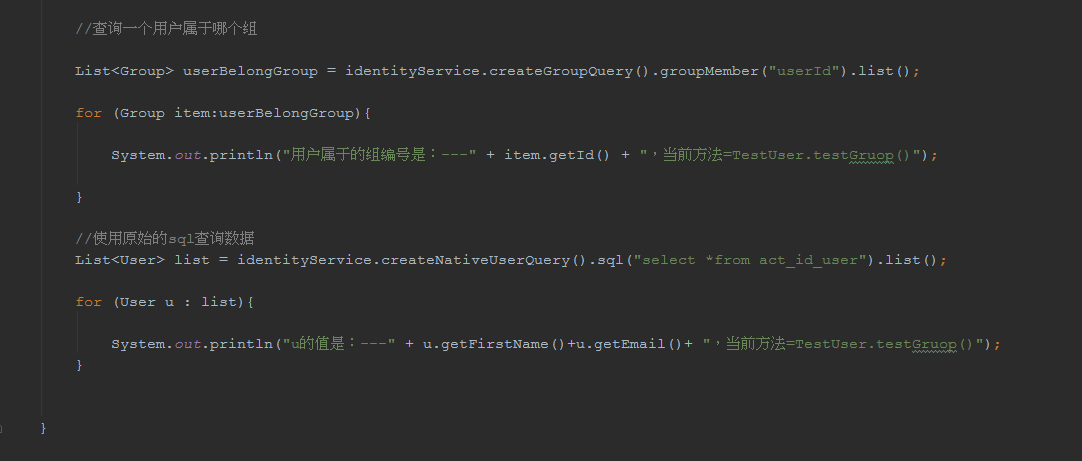


### 查询User

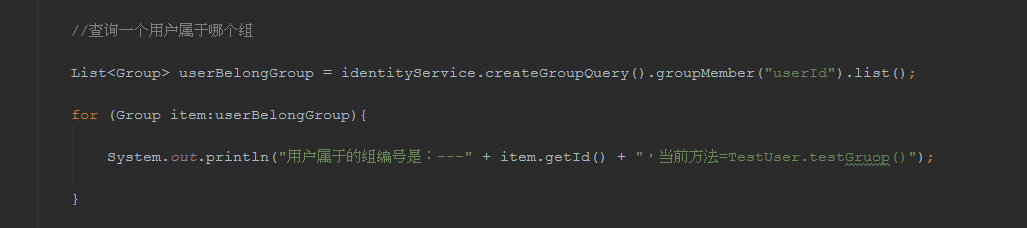


### 增加一个用户组，在把用户添加到用户组里边去

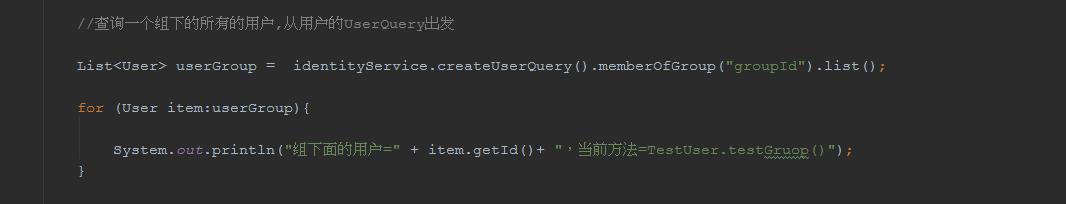




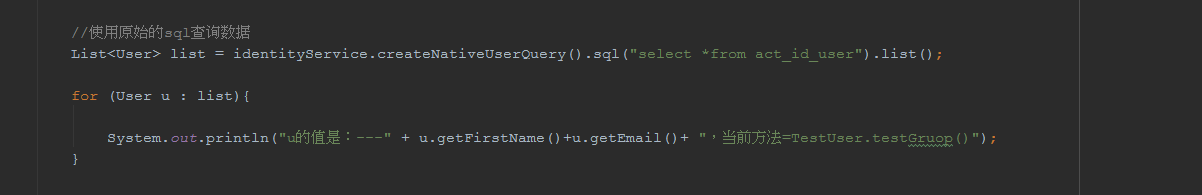
### 这里查询用户属于哪个组的要从group那里创建查询在使用userId进行筛选



### 查询一个组下的所有用户，结果返回的是List<User>所以要从User的查询出发



### 使用原始的sql语句来获取数据



完整代码如下：

|  |
| --- |
| */\*\*  \* Created by @author 王钟鑫  \*  \** ***@create*** *2018/1/20.  \*/* public class TestUser {   //获取到activiti测试类的ActivitiRule对象，可以来获得对应的Service  @Rule  public ActivitiRule activitiRule = new ActivitiRule();  //创建一个User  @Test  public void testUser(){  //获取一个IdentityService，里面装了对用户，用户组，用户和用户组直接的操作  IdentityService identityService = activitiRule.getIdentityService();  //创建一个用户  User user = new UserEntity();  user.setId(UUID.*randomUUID*().toString());  user.setFirstName("张三");  user.setEmail("111111@qq.com");  identityService.saveUser(user);  }   //查询所有的用户  @Test  public void testQueryUser(){    IdentityService identityService = activitiRule.getIdentityService();  List<User> dataList = identityService.createUserQuery().list();  for (User user:dataList){    System.*out*.println("user的值是：---" + user.getId()+"--"+user.getFirstName() + "，当前方法=TestUser.testQueryUser()");    }   }  //增加一个用户组，在把用户添加到用户组里边去  @Test  public void testUserAndGruop(){   IdentityService identityService = activitiRule.getIdentityService();   Group group = new GroupEntity();   group.setId("groupId");  group.setName("用户组");  identityService.saveGroup(group);   //创建用户  User user = identityService.newUser("userId");  User user1 = identityService.newUser("userId1");  user.setFirstName("李四");  identityService.saveUser(user);  identityService.saveUser(user1);   //把用户添加到用户组里面,通过两个UsreId 和 GroupId建立两者的关系  identityService.createMembership("userId","groupId");  identityService.createMembership("userId1","groupId");   //查询一个组下的所有的用户,从用户的UserQuery出发   List<User> userGroup = identityService.createUserQuery().memberOfGroup("groupId").list();   for (User item:userGroup){   System.*out*.println("组下面的用户=" + item.getId()+ "，当前方法=TestUser.testGruop()");  }  //查询一个用户属于哪个组   List<Group> userBelongGroup = identityService.createGroupQuery().groupMember("userId").list();   for (Group item:userBelongGroup){   System.*out*.println("用户属于的组编号是：---" + item.getId() + "，当前方法=TestUser.testGruop()");   }   //使用原始的sql查询数据  List<User> list = identityService.createNativeUserQuery().sql("select \*from act\_id\_user").list();   for (User u : list){   System.*out*.println("u的值是：---" + u.getFirstName()+u.getEmail()+ "，当前方法=TestUser.testGruop()");  }  }   //释放对象  @After  public void destory(){  if(activitiRule != null)  {  IdentityService identityService = activitiRule.getIdentityService();   identityService.deleteUser("userId");  identityService.deleteUser("userId1");  identityService.deleteGroup("groupId");  activitiRule = null;  }  } } |