Activity

# 市场需求

JBPM：jboss business process management，jboss开发的业务流程管理框架。

Activity：开源工作流引擎。

## 企业应用范围

OA系统的请假流程

电子政务

## 核心点分析

1. 数据
2. 角色
3. 传递参数

# 相关概念

工作流(Workflow)，就是“业务过程的部分或整体在计算机应用环境下的自动化”，它主要解决的是“使在多个参与者之间按照某种预定义的规则传递文档、信息或任务的过程自动进行，从而实现某个预期的业务目标，或者促使此目标的实现”。

工作流管理系统(Workflow Management System, WfMS)是一个软件系统，它完成工作量的定义和管理，并按照在系统中预先定义好的工作流规则进行工作流实例的执行。工作流管理系统不是企业的业务系统，而是为企业的业务系统的运行提供了一个软件的支撑环境。

工作流管理联盟(WfMC，Workflow Management Coalition)给出的关于工作流管理系统的定义是：工作流管理系统是一个软件系统，它通过执行经过计算的流程定义去支持一批专门设定的业务流程。工作流管理系统被用来定义、管理、和执行工作流程。

工作流管理系统的目标：管理工作的流程以确保工作在正确的时间被期望的人员所执行——在自动化进行的业务过程中插入人工的执行和干预。

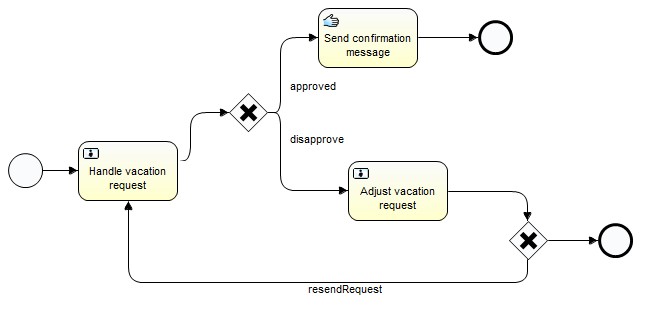
# Activity工作流介绍

## 常见工作流框架

OSWorkFlow、JBPM、activity（是对JBPM的升级，JBPM架构师来创建的）

Activiti5是由Alfresco软件在2010年5月17日发布的业务流程管理（BPM）框架，它是覆盖了业务流程管理、工作流、服务协作等领域的一个开源的、灵活的、易扩展的可执行流程语言框架。Activiti基于Apache许可的开源BPM平台，创始人Tom Baeyens是JBoss jBPM的项目架构师，它特色是提供了eclipse插件，开发人员可以通过插件直接绘画出业务

流程图。



### Activity工作流介绍

## 工作流引擎

ProcessEngine，是activti的核心类，可以去获取其他的一些服务（历史服务、仓库服务（描述多个流程）、任务服务、参与者服务）。

## BPMN

业务流程建模与标注。

就是类似于UML用例图的，那种业务流程设计图。

## 数据库

Activity的后台是有数据库支持的，所有的表都以ACT\_开头。 第二部分是表示表的用途的两个字母标识。 用途也和服务的API对应。

Activiti的后台是有数据库的支持，所有的表都以ACT\_开头。

第二部分是表示表的用途的两个字母标识。 用途也和服务的API对应

Activiti的工作流数据库有23张表

要保存流程定义

* + act\_ge\_\* ：通用表
  + act\_hi\_\* ：历史流程相关表
  + act\_re\_\* ：仓库表：保存流程定义
  + act\_ru\_\* ：保存流程运行相关的表
  + act\_id\_\* ：用户参与相关的表

|  |
| --- |
| * 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。 |

### 数据库流程表

* act\_re\_deployment 部署信息表
* act\_re\_model 流程设计模型部署表
* 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，

## 配置文件

### Activti.cfg.xml

Activiti核心配置文件，配置流程引擎创建工具的基本参数和数据库连接池参数。

定义数据库配置参数：

* **jdbcUrl**: 数据库的JDBC URL。
* **jdbcDriver**: 对应不同数据库类型的驱动。
* **jdbcUsername**: 连接数据库的用户名。
* **jdbcPassword**: 连接数据库的密码。

基于JDBC参数配置的数据库连接 会使用默认的**MyBatis**连接池。 下面的参数可以用来配置连接池（来自MyBatis参数）：

* **jdbcMaxActiveConnections**: 连接池中处于被使用状态的连接的最大值。默认为10。
* **jdbcMaxIdleConnections**: 连接池中处于空闲状态的连接的最大值。
* **jdbcMaxCheckoutTime**: 连接被取出使用的最长时间，超过时间会被强制回收。 默认为20000（20秒）。
* **jdbcMaxWaitTime**: 这是一个底层配置，让连接池可以在长时间无法获得连接时， 打印一条日志，并重新尝试获取一个连接。（避免因为错误配置导致沉默的操作失败）。 默认为20000（20秒）。

示例数据库配置：



也可以使用javax.sql.DataSource。 （比如，Apache Commons的DBCP）：



### Logging.properties

日志的配置文件，Activiti操作数据库的时候，整合的日志文件

# 环境搭建

## 软件环境

1) JDK1.6或者更高版本

2) 支持的数据库有：h2（框架默认使用的数据库）, mysql, oracle, postgres, mssql, db2等。

3) 支持activiti5运行的jar包

4) 开发环境为Eclipse3.7或者以上版本,myeclipse为8.6版本

## 相关资源下载

1) JDK可以到sun的官网下载

http://www.oracle.com/technetwork/java/javase/downloads/index.html

2) 数据库，例如：mysql可以在官网上下载。

http://www.mysql.com

3) activiti也可以到Activiti官方网站下载得到。

http://activiti.org/download.html

4) Eclipse3.7或者MyEclipse8.6也可以到相应的网站上获得下载。

## 安装流程设计器

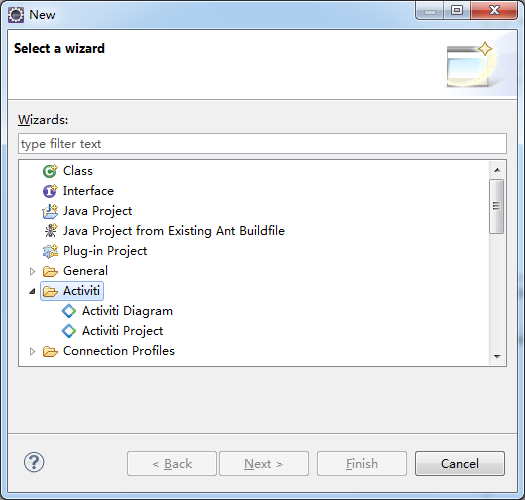
Bpmn：业务流程建模与标注。

每一个业务流程都需要一个bpmn文件来描述业务流程，另外可以提供一个图片来提供查看流程图的功能。

### 无网安装

在**没有网络**的情况下，安装流程设计器步骤如下：

1. 解压
2. 把压缩包中的内容放入eclipse根目录的dropins文件夹下
3. 重启eclipse，点击新建工程new->Other…打开面板，如果看到下图内容：

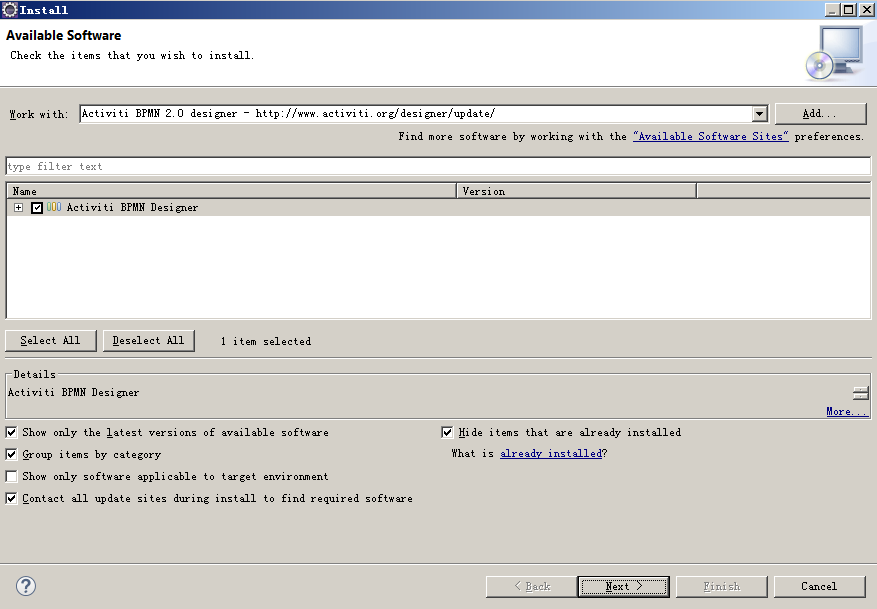


说明安装成功了

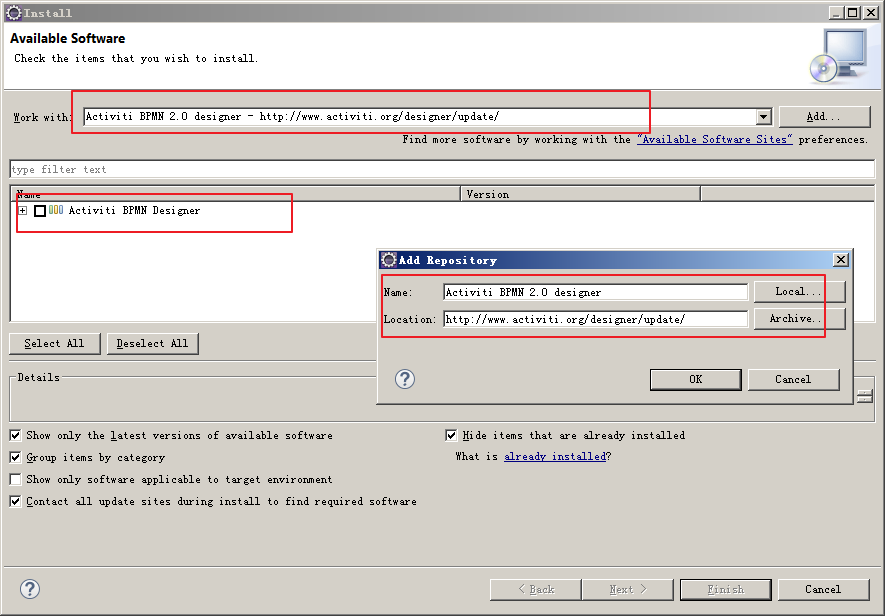
### 有网络安装

在有网络的情况下，安装流程设计器步骤如下：

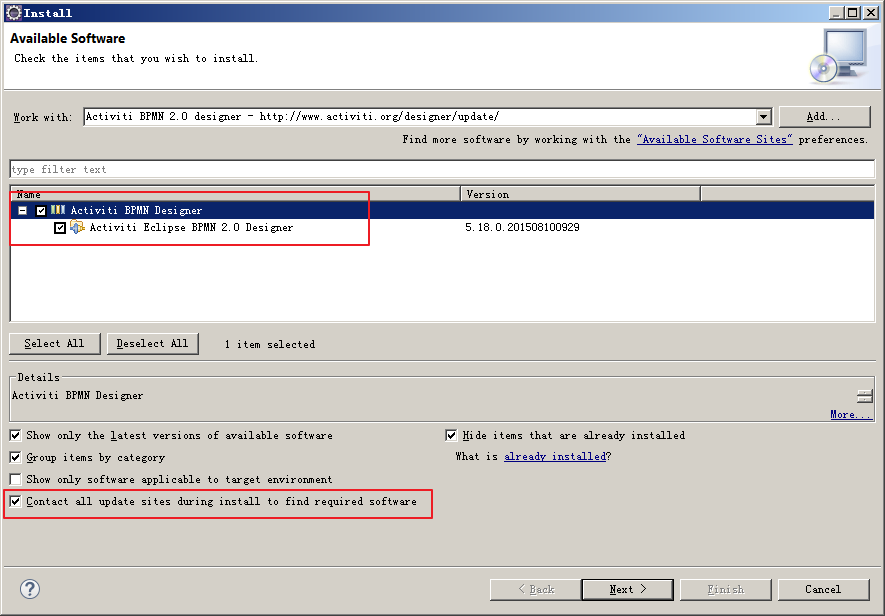
1. 打开 Help -> Install from site . 在如下面板中:



1. 点击add添加站点信息。

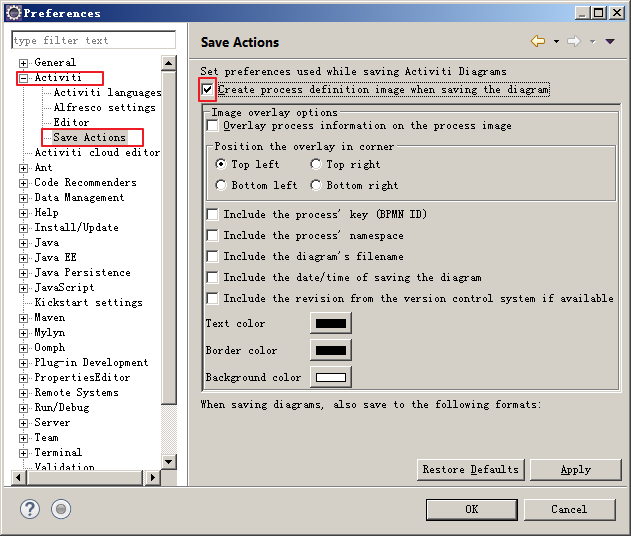


1. 选中所有打开部分



Next进行安装。

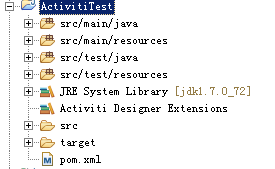
1. 配置在产生BPMN文件时，同时产生图片文件。



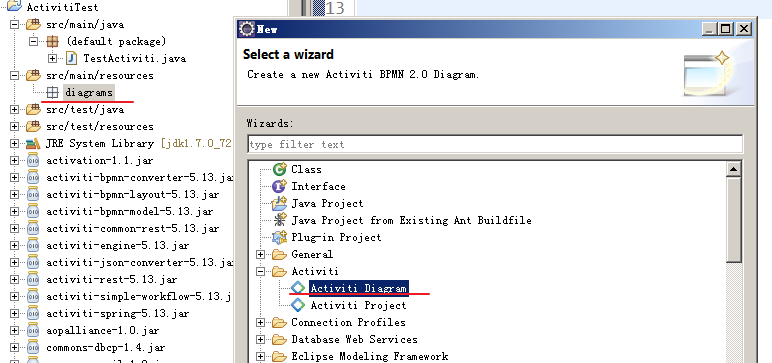
### 模拟activiti程序执行

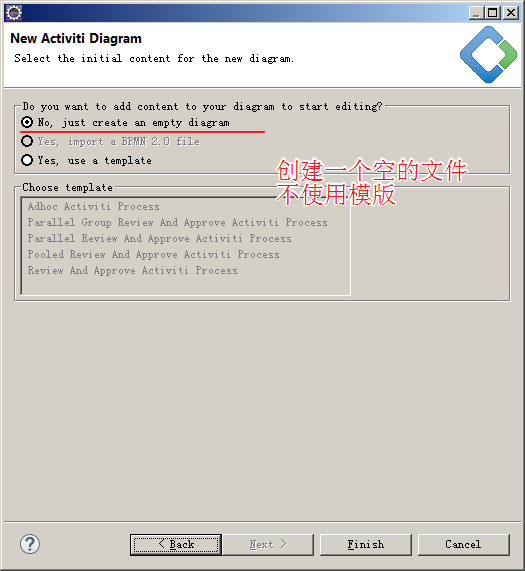
New-->other-->activiti project

新建项目出来之后，发现这是一个maven项目。

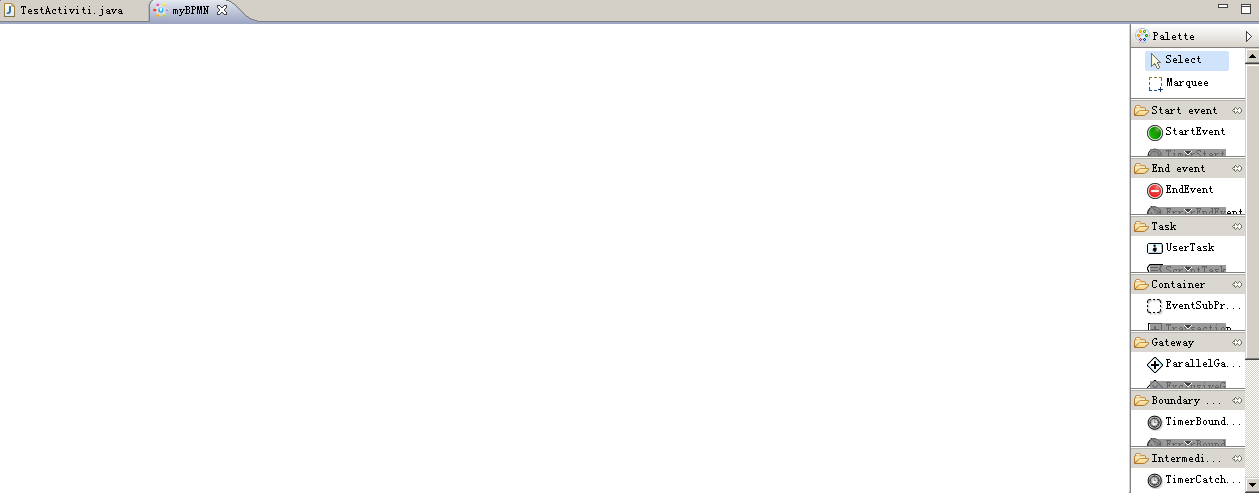


1. 导入jar包，并且build path。
2. 设置流程定义图bpmn

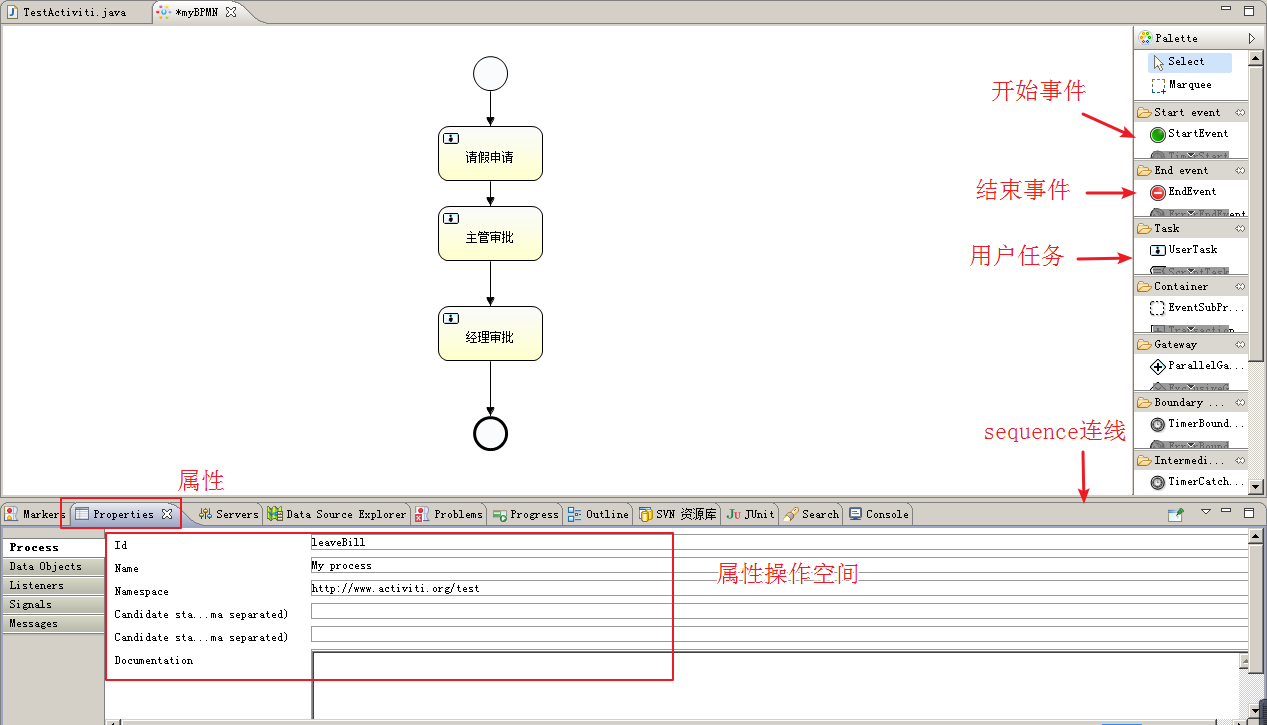




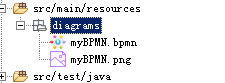
工作台：



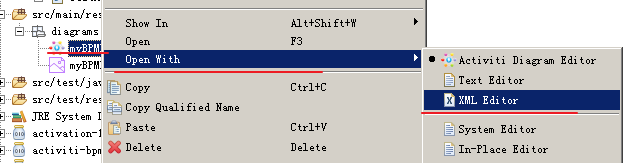
模拟请假的流程图：



保存之后，文件夹中会出现以下文件：



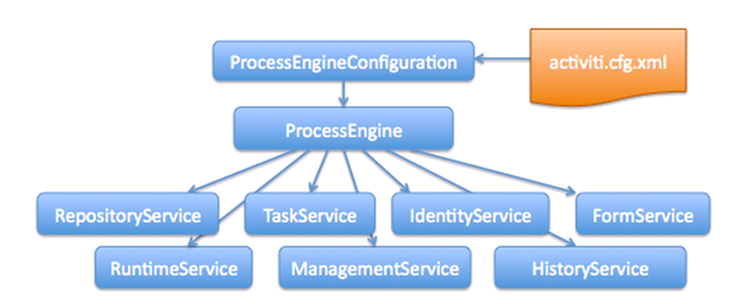
1. 以xml形式查看bpmn文件。





# 创建流程引擎及工作流数据表

1. 数据库及表可以自动创建，可以在配置数据库连接时，实现字段创建数据库和表。
2. 取得流程引擎(process engine)对象，先要取得processEngineConfiguration对象，再由该对象来构建process engine对象。

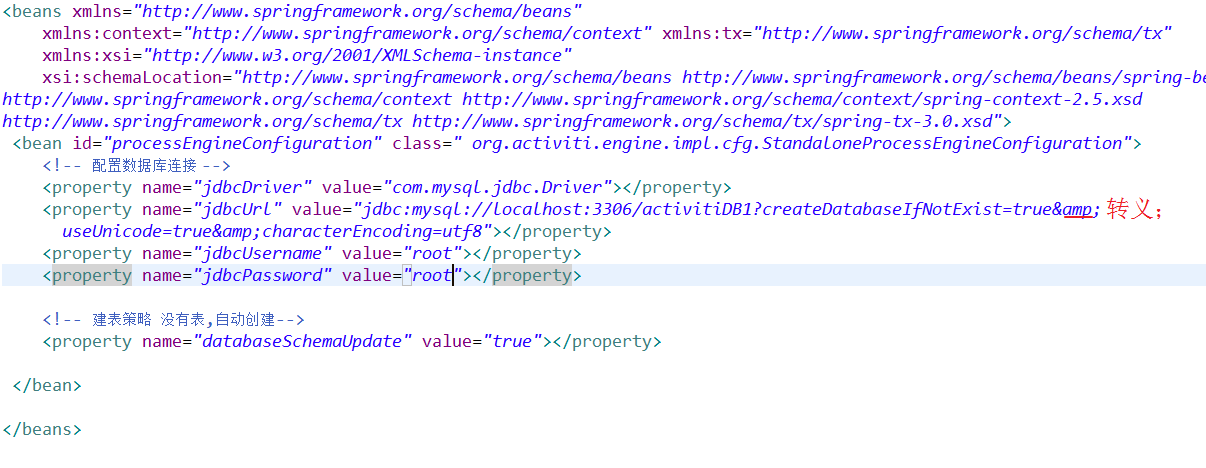


## 通过代码的形式

|  |
| --- |
| //取得流程引擎，且自动创建activiti涉及的数据库和表  @Test  **public** **void** createEngine(){  /\*\*  \* 1、通过代码形式创建  \* - 取得ProcessEngineConfiguration对象  \* - 设置数据库连接属性  \* - 设置创建表的策略（当没有表时，自动创建表）  \* \*/  ProcessEngineConfiguration pecf = ProcessEngineConfiguration.*createStandaloneProcessEngineConfiguration*();  pecf.setJdbcDriver("com.mysql.jdbc.Driver");  pecf.setJdbcUrl("jdbc:mysql://localhost:3306/activitiDB?createDatabaseIfNotExist=true&useUnicode=true&"  + "characterEncoding=utf8");  pecf.setJdbcUsername("root");  pecf.setJdbcPassword("root");  // pecf.DB\_SCHEMA\_UPDATE\_CREATE\_DROP：先删除，再创建  // pecf.DB\_SCHEMA\_UPDATE\_FALSE：不会自动创建表，没有表，则抛出异常  // pecf.DB\_SCHEMA\_UPDATE\_TRUE：假如没有表，则自动创建  pecf.setDatabaseSchemaUpdate("true");  ProcessEngine processEngine = pecf.buildProcessEngine();  System.***out***.println("创建引擎成功");  } |

## 通过配置的形式

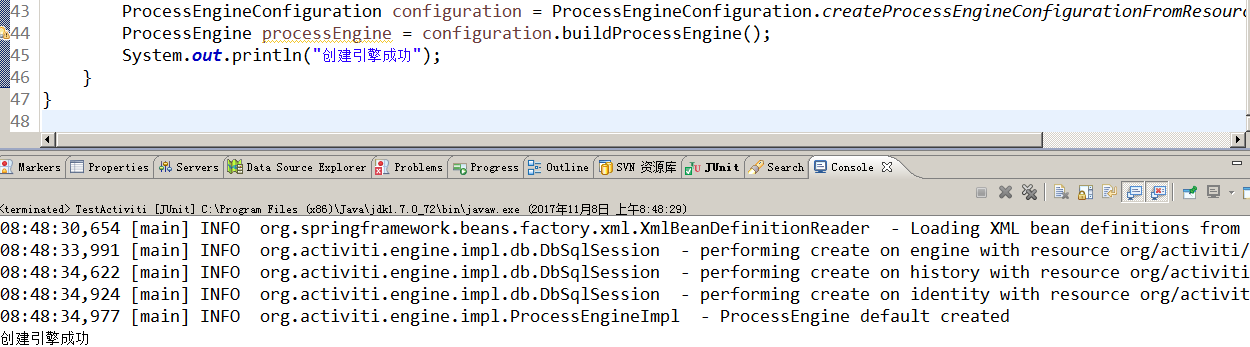
配置文件：



创建processEngine对象：

|  |
| --- |
| ProcessEngineConfiguration configuration = ProcessEngineConfiguration.*createProcessEngineConfigurationFromResource*("activiti.cfg.xml");  ProcessEngine processEngine = configuration.buildProcessEngine();  System.***out***.println("创建引擎成功"); |

创建成功：



配置文件的名字可以自定义。

## 通过流程引擎来自动加载

默认会加载路径下的activiti.cfg.xml

|  |
| --- |
| ProcessEngine engine = ProcessEngines.*getDefaultProcessEngine*();  System.***out***.println("创建引擎成功"); |

# 部署流程

可以通过ProcessEngine获取到所有的服务实例。

|  |
| --- |
| @Test//部署流程定义  **public** **void** deploy(){  //取得流程引擎对象  ProcessEngine engine = ProcessEngines.*getDefaultProcessEngine*();  //获取仓库服务，管理流程定义  Deployment deploy = engine.getRepositoryService().createDeployment()//创建一个部署构建器（返回DeploymentBuild对象）  .addClasspathResource("diagrams/myBPMN.bpmn")//从类路径中添加资源，一次只能添加一个资源  .addClasspathResource("diagrams/myBPMN.png")//从类路径中添加资源，一次只能添加一个资源  .name("请假流程定义")//设置部署的名称  .category("办公类别")//设置部署的类别  .deploy();  System.***out***.println(deploy.getId());  } |

**影响的表：**

* 1. act\_re\_procdef：流程定义表。
     1. Key：bpmn的id决定。
     2. Name：bpmn的name属性决定。
     3. 每执行一次act\_re\_procdef中就会多一条记录。包含版本号和deployment\_id等信息。
  2. act\_re\_deployment：部署表
     1. id：由act\_ge\_property的next\_dbid决定的。
  3. act\_ge\_property：通用属性表
     1. id生成策略 next.dbid 影响的是部署的id。

# 执行流程

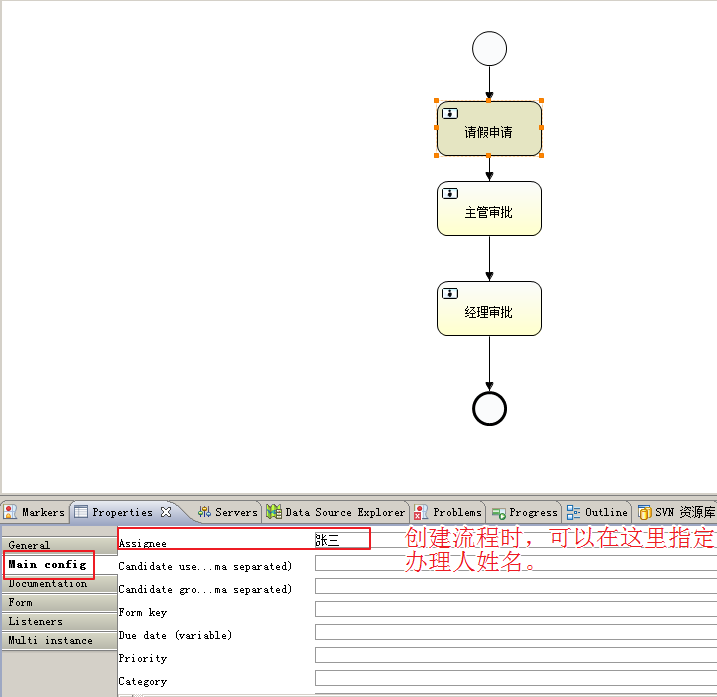
|  |
| --- |
| @Test  **public** **void** startProcess(){  String key = "leaveBill";  RuntimeService runtimeService = engine.getRuntimeService();  ProcessInstance instance = runtimeService.startProcessInstanceByKey(key);  System.***out***.println(instance.getId());  System.***out***.println(instance.getProcessDefinitionId());    } |

Act\_ru\_execution：表中的proc\_def\_id对应Act\_re\_procdef中的id。

Act\_id：表示的是流程走到那一步。

Is\_active：表示是否正在执行该任/流程。

# 查询任务



查询：

|  |
| --- |
| @Test  **public** **void** queryTask(){  //任务的办理人  String assignee = "张三";  //取得任务服务  TaskService taskService = engine.getTaskService();  //创建一个任务查询对象  TaskQuery query = taskService.createTaskQuery();  //根据办理人姓名来获取任务结果  List<Task> list = query.taskAssignee(assignee).list();  System.***out***.println(list.size());  **for** (Task task : list) {  System.***out***.println(task.getId());  System.***out***.println(task.getName());  System.***out***.println(task.getAssignee());  }  } |

**涉及到的表：**

Act\_ru\_task。

# 完成任务

|  |
| --- |
| @Test  **public** **void** taskCompile(){  String taskId = "504";  //根据任务id来完成任务  //id对应的是act\_ru\_task表中的id，完成任务后，id值、name、assigen会变化。  engine.getTaskService().complete(taskId);  System.***out***.println("任务完成");  } |

当所有的流程走完，task、execution表中将没有这条数据。

# 流程定义管理

涉及到的对象以及表：

对象：ProcessEngine、RepositoryService、Dployment、ProcessDefinition

表：

1. act\_re\_deployment：部署对象表
2. act\_re\_procdef：流程定义表
3. act\_ge\_bytearray：资源文件表
4. act\_ge\_property：主键生成策略表

## 部署流程定义

### 通过bpmn和png格式文件加载及部署

|  |
| --- |
| @Test  **public** **void** deploy(){  RepositoryService repositoryService = engine.getRepositoryService();  Deployment deploy = repositoryService.createDeployment()  .addClasspathResource("diagrams/buyBill.bpmn")  .addClasspathResource("diagrams/buyBill.png")  .name("采购流程")  .category("办公类别")  .deploy();  System.***out***.println(deploy.getId());  } |

### 通过zip格式加载及部署

Zip文件放在lib/classes文件夹中

|  |
| --- |
| @Test  **public** **void** deployByZIP(){  InputStream is = getClass().getClassLoader().getResourceAsStream("buy.zip");  RepositoryService repositoryService = engine.getRepositoryService();  Deployment deploy = repositoryService.createDeployment()  .addZipInputStream(**new** ZipInputStream(is))  .name("采购流程")  .category("办公类别")  .deploy();  System.***out***.println(deploy.getId());  } |

## 流程定义查询

|  |
| --- |
| //查看流程定义  @Test  **public** **void** queryProcessDefination(){  String processDefiKey="buyBill";//流程定义key  //获取流程定义列表  List<ProcessDefinition> list = engine.getRepositoryService().createProcessDefinitionQuery()  //查询 ，好比where  // .processDefinitionId(proDefiId) //流程定义id  // 流程定义id ： buyBill:2:704 组成 ： proDefikey（流程定义key）+version(版本)+自动生成id  .processDefinitionKey(processDefiKey)//流程定义key 由bpmn 的 process 的 id属性决定  // .processDefinitionName(name)//流程定义名称 由bpmn 的 process 的 name属性决定  // .processDefinitionVersion(version)//流程定义的版本  .latestVersion()//最新版本    //排序  .orderByProcessDefinitionVersion().desc()//按版本的降序排序    //结果  // .count()//统计结果  // .listPage(arg0, arg1)//分页查询  .list();      //遍历结果  **if**(list!=**null**&&list.size()>0){  **for**(ProcessDefinition temp:list){  System.***out***.print("流程定义的id: "+temp.getId());  System.***out***.print("流程定义的key: "+temp.getKey());  System.***out***.print("流程定义的版本: "+temp.getVersion());  System.***out***.print("流程定义部署的id: "+temp.getDeploymentId());  System.***out***.println("流程定义的名称: "+temp.getName());  }  }  } |

## 查看bpmn资源图片

|  |
| --- |
| //查看bpmn图片  @Test  **public** **void** viewImage(){  String deploymentId = "901";  String imageName = **null**;  List<String> list = engine.getRepositoryService().getDeploymentResourceNames(deploymentId);  **for** (String string : list) {  **if** (string.endsWith(".png")) {  System.***out***.println(string);  imageName = string;//获取资源路径名  }  }    //取得资源:arg0(部署id)、arg1(resourceName资源文件名)  InputStream inputStream = engine.getRepositoryService().getResourceAsStream(deploymentId, imageName);  // 把文件输出流写入到文件中  File file = **new** File("d:/" + imageName + ".jpg");  **try** {  FileUtils.*copyInputStreamToFile*(inputStream, file);//选择commons.io包下的fileUtils  } **catch** (IOException e) {  e.printStackTrace();  }  } |

## 删除流程定义

|  |
| --- |
| //删除流程定义  @Test  **public** **void** deleteProcess() {  //根据部署id来删除  String deploymentId = "1";  engine.getRepositoryService().deleteDeployment(deploymentId);  } |

# 流程实例、任务执行

**Execution 执行对象**

按流程定义的规则执行一次的过程.

对应的表：

act\_ru\_execution： 正在执行的信息

act\_hi\_procinst：已经执行完的历史流程实例信息

act\_hi\_actinst：存放历史所有完成的活动

没有分支就没有parentId。

**ProcessInstance 流程实例**

特指流程从开始到结束的那个最大的执行分支，一个执行的流程中，流程实例只有1个。

注意

（1）如果是单例流程，执行对象ID就是流程实例ID

（2）如果一个流程有分支和聚合，那么执行对象ID和流程实例ID就不相同

（3）一个流程中，流程实例只有1个，执行对象可以存在多个。

**Task 任务**

执行到某任务环节时生成的任务信息。

对应的表：

act\_ru\_task：正在执行的任务信息

act\_hi\_taskinst：已经执行完的历史任务信息

**启动：**

|  |
| --- |
| @Test  **public** **void** startProcess(){  RuntimeService service = engine.getRuntimeService();  ProcessInstance instance = service.startProcessInstanceByKey("buyBill");  System.***out***.println(instance.getId());  System.***out***.println(instance.getProcessInstanceId());  System.***out***.println(instance.getProcessDefinitionId());  } |

**查看：**

|  |
| --- |
| @Test  **public** **void** Query(){  TaskService service = engine.getTaskService();  TaskQuery query = service.createTaskQuery();  query.processInstanceId("1401");  List<Task> list = query.list();  **for** (Task task : list) {  System.***out***.println(task.getId());  }  } |

**办理：**

|  |
| --- |
| @Test  **public** **void** compileProcess(){  TaskService service = engine.getTaskService();  service.complete("1304");  } |

## 查询流程实例状态

判断流程是否结束。

|  |
| --- |
| @Test  **public** **void** getProcessInstanceState(){  RuntimeService service = engine.getRuntimeService();  ProcessInstance singleResult = service.createProcessInstanceQuery()  .processInstanceId("1301")  .singleResult();//返回的数据要么是单行，要么是空，其他情况报错。    **if** (singleResult != **null**) {  System.***out***.println("当前实例1301正在运行" + ",当前任务:" + singleResult.getActivityId());  }**else** {  System.***out***.println("当前实例1301已经结束");  }  } |

Act\_hi\_proinst：流程历史表。当流程没有结束时，记录中结束时间没有值，结束后，结束时间就有值了。

当整个流程结束之后，返回的pi就是null。

## 查看历史流程

涉及到的表：

act\_hi\_procinst（历史流程实例表）

duration：流程持续了多长时间。

act\_hi\_taskinst（历史流程任务表）

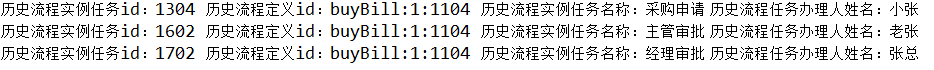
### 查询历史流程实例信息

|  |
| --- |
| @Test  **public** **void** queryHiProcessInstance(){  HistoryService service = engine.getHistoryService();  List<HistoricProcessInstance> list = service.createHistoricProcessInstanceQuery()  .list();    **if** (list != **null** && list.size() != 0) {  **for** (HistoricProcessInstance historicProcessInstance : list) {  System.***out***.print("历史流程实例id " + historicProcessInstance.getId());  System.***out***.print("流程开始时间 " + historicProcessInstance.getStartTime() +  "----" + historicProcessInstance.getEndTime());  System.***out***.println("流程定义id " + historicProcessInstance.getProcessDefinitionId());  }  }  } |

### 查看历史执行流程任务信息

查询某个流程，到底经历了哪些任务，根据流程实例id/流程实例id来查询。

|  |
| --- |
| @Test  **public** **void** queryHiProcessTask(){  String instanceId = "1301";  HistoryService service = engine.getHistoryService();  List<HistoricTaskInstance> list = service.createHistoricTaskInstanceQuery()  .processInstanceId(instanceId)  .list();    **if** (list != **null** && list.size() != 0) {  **for** (HistoricTaskInstance historicTaskInstance : list) {  System.***out***.print("历史流程实例任务id：" + historicTaskInstance.getId() + " ");  System.***out***.print("历史流程定义id：" + historicTaskInstance.getProcessDefinitionId() + " ");  System.***out***.print("历史流程实例任务名称：" + historicTaskInstance.getName() + " ");  System.***out***.println("历史流程任务办理人姓名：" + historicTaskInstance.getAssignee() + " ");  }  }  } |



# 流程变量

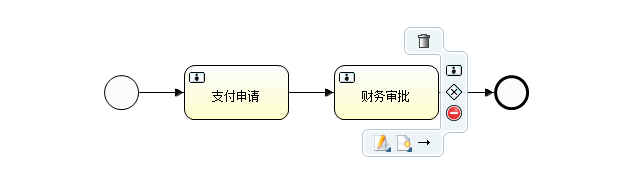
**流程变量的作用：**

1. 传递业务参数
2. 动态指定代理人（请假人是动态的，对应的处理人也是动态的）
3. 指定连接，完成任务（同意或拒绝）

**涉及到的表：**

1. Act\_ru\_variable：正在执行的流程变量表。
2. Act\_hi\_varinst：流程变量历史表。

**做一个简单的流程BPMN：**



不用去设置办理人，我们用流程变量去动态的实现。

**部署流程：**

|  |
| --- |
| ProcessEngine engine = ProcessEngines.*getDefaultProcessEngine*();    @Test  **public** **void** deploy(){  RepositoryService repositoryService = engine.getRepositoryService();  Deployment deploy = repositoryService.createDeployment()  .addClasspathResource("diagrams/payBill.bpmn")  .addClasspathResource("diagrams/payBill.png")  .name("支付流程")  .category("办公类别")  .deploy();  System.***out***.println(deploy.getId());  } |

**开始流程：**

|  |
| --- |
| @Test  **public** **void** startProcess(){  RuntimeService service = engine.getRuntimeService();  ProcessInstance instance = service.startProcessInstanceByKey("payBill");  System.***out***.println(instance.getId());  System.***out***.println(instance.getProcessInstanceId());  System.***out***.println(instance.getProcessDefinitionId());  } |

**设置流程变量：**

通过taskservice和runtimeservice来设置。

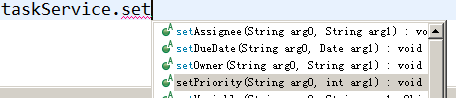
SetValiable和setLocalValiable的区别？

作用域不同。

**方法汇总：**

|  |
| --- |
| @Test  **public** **void** getAndSetProcessVariable(){  // 两种服务设置流程变量  TaskService taskService = engine.getTaskService();  RuntimeService runtimeService = engine.getRuntimeService();  /\*  \* 设置值：  \* 1.通过runtimeService来设置流程变量  \* runtimeService.setVariable(执行对象id, 流程变量名, 变量值（object）);作用域是整个流程实例。  \* runtimeService.setVariableLocal(执行对象id, 流程变量名, 变量值（object）); 设置执行对象的变量，该变量的作用域只作用在当前的执行对象（分支）中。  \* runtimeService.setVariables(执行对象id, map<String, object>); 设置多个变量。  \* runtimeService.setVariablesLocal(执行对象id, map<String, object>);同第二种方式。  \*  \* 2.通过taskService来设置流程变量  \* taskService.setVariable(任务id, 流程变量名, 变量值（object）);  \* taskService.setVariableLocal(任务id, 流程变量名, 变量值（object）);  \* taskService.setVariables(任务id, map<String, object>); 设置多个变量。  \* taskService.setVariablesLocal(任务id, map<String, object>);  \*  \* 3.当流程开始执行的时候，可以设置流程变量  \* engine.getRuntimeService().startProcessInstanceByKey(流程定义key, map<String, object>)  \*  \* 4.当执行任务时，可以设置流程变量  \* engine.getTaskService().complete(任务id, map<String, object>);  \* \*\*/    /\*  \* 取值：  \* 1.通过runtimeService来取变量值  \* runtimeService.getVariable(执行对象id, 流程变量名);  \* runtimeService.getVariableLocal(执行对象id, 流程变量名);  \* runtimeService.getVariables(执行对象id); 返回map  \* runtimeService.getVariablesLocal(执行对象id);  \*  \* 2.通过taskService来设置流程变量  \* taskService.getVariable(任务id, 流程变量名);  \* taskService.getVariableLocal(任务id, 流程变量名);  \* taskService.getVariables(任务id);返回map  \* taskService.getVariablesLocal(任务id);  \* \*\*/    } |

其他方法：



**代码实例：**



注意：

1. taskservice.setvariable形式设置的变量，不能使用taskservice.getvariableLocal获取到，因为作用域不同，没有对应的taskid。
2. 不同的任务中设置各自的local变量，互不影响。
3. 当整个流程结束时，ru表中的变量将不存在，但是历史中会有所有的变量。

**支持的数据类型：**

1. 简单的数据类型：String、boolean、Integer、double、date。
2. 自定义的对象类型（必须实现序列化：实现Serializable）。数据库表中存储的type是serializable

Service：

|  |
| --- |
| TaskService taskService = engine.getTaskService();  RuntimeService runtimeService = engine.getRuntimeService();  TaskInfo taskInfo = **new** TaskInfo();  taskInfo.setName("王五");  taskInfo.setCost(1999);  taskService.setVariable("2004", "info", taskInfo); |

Pojo：

|  |
| --- |
| **public** **class** TaskInfo **implements** Serializable{  /\*\* serialVersionUID\*/  **private** **static** **final** **long** ***serialVersionUID*** = 1L;  **private** String name;  **private** **int** cost;  **public** **int** getCost() {  **return** cost;  }  **public** **void** setCost(**int** cost) {  **this**.cost = cost;  }  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }    } |

测试：

|  |
| --- |
| TaskService taskService = engine.getTaskService();  RuntimeService runtimeService = engine.getRuntimeService();  TaskInfo object = (TaskInfo) taskService.getVariable("2004", "info");  System.***out***.println(object.getName()); |

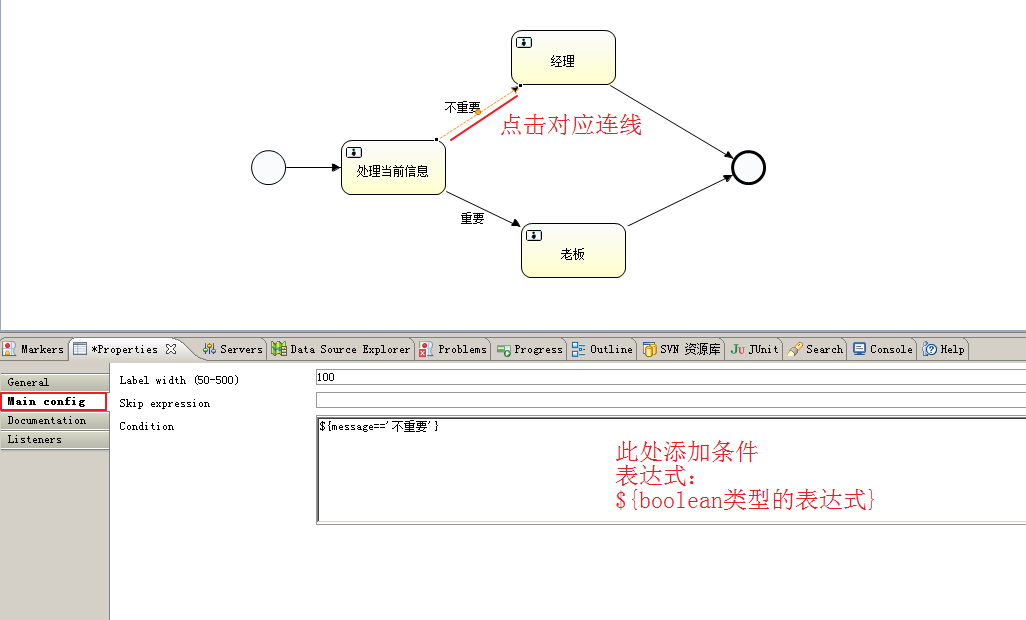
序列化之后的对象会被存到act\_ge\_bytearray表中。因为序列化是流的形式存储，所以存储的形式就是字节数组或者字节。在表中会出现两条记录。Hist.开头的是历史流程变量中的数据，另外一个是运行时流程变量的值。

业务管理链：

1. 开始业务流程时获取到流程实例id
2. 根据流程实例id查询任务id
3. 再根据任务id去设置属性，或者获取属性

# 连线

制作bpmn流程图：



设置变量可以使用#或者$，$会自动为值加上引号。

部署、开始、查询、完成任务等流程同样需要实现：

|  |
| --- |
| **public** **class** Activiti5 {  ProcessEngine engine = ProcessEngines.*getDefaultProcessEngine*();  @Test  **public** **void** deploy() {  RepositoryService repositoryService = engine.getRepositoryService();  Deployment deploy = repositoryService.createDeployment().addClasspathResource("diagrams/sequenceFlow.bpmn")  .addClasspathResource("diagrams/sequenceFlow.png").name("连线流程").category("办公类别").deploy();  System.***out***.println(deploy.getId());  }  @Test  **public** **void** startProcess() {  RuntimeService service = engine.getRuntimeService();  ProcessInstance instance = service.startProcessInstanceByKey("sequenceBill");//bpmn的id  System.***out***.println(instance.getId());  System.***out***.println(instance.getProcessInstanceId());  System.***out***.println(instance.getProcessDefinitionId());  }  @Test  **public** **void** Query() {  TaskService service = engine.getTaskService();  TaskQuery query = service.createTaskQuery();  query.processInstanceId("2301");  List<Task> list = query.list();  **for** (Task task : list) {  System.***out***.println(task.getId());  }  }  @Test  **public** **void** compileProcess() {  TaskService service = engine.getTaskService();  service.complete("2304");  }  } |

如果我们要完成这个任务，那我们必须传入一个参数message？也就是在完成任务时设置一个流程变量。

**代码改动如下：**

|  |
| --- |
| @Test  **public** **void** compileProcess() {  TaskService service = engine.getTaskService();  Map<String, Object> params = **new** HashMap<>();  params.put("message", "重要");  service.complete("2304", params);  } |

根据message值的不同，就会执行对应的任务。

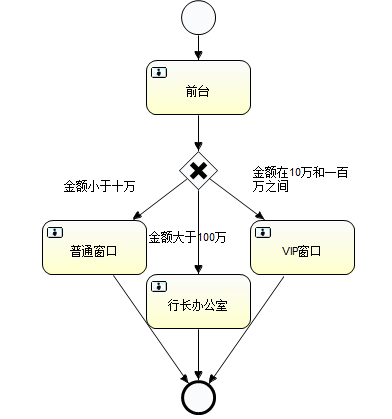
如果变量的赋值不正确，则流程就无法继续，所以在设计流程时，需要我们把条件写严谨，愿望就是要考虑else的情况。

注意：

1. 一个流程中可以有多个连线，但是开始只有一个，结束中没有。
2. 如果只有一个不用去写条件（condition）。

# 排他网关

利用排他网关做一个银行的业务流程：

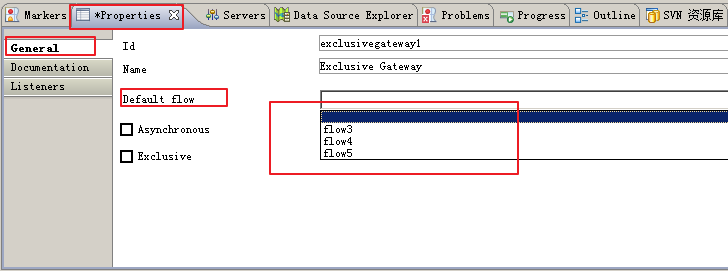


代码实现：

|  |
| --- |
| @Test  **public** **void** compileProcess() {  TaskService service = engine.getTaskService();  Map<String, Object> params = **new** HashMap<>();  params.put("cost", 300000);  service.complete("2604", params);  } |

排他网关与连线相比的优点：

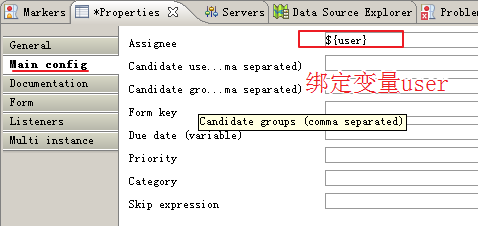
1. 可以设置一个缺省的条件，来当作else处理。
2. 可以在网关properties中设置一个默认的连线，缺省连线会自动被设置为默认连线。



不满足条件的情况，会走默认的连线。

# 指定任务处理者

绑定变量：



在完成前一个任务时设置变量值：

|  |
| --- |
| @Test  **public** **void** compileProcess() {  TaskService service = engine.getTaskService();  Map<String, Object> params = **new** HashMap<>();  params.put("user", "老罗");  service.complete("3204", params);  } |

在开始流程时设置变量值：

|  |
| --- |
| @Test  **public** **void** startProcess() {  RuntimeService service = engine.getRuntimeService();  Map<String, Object> params = **new** HashMap<>();  params.put("user", "老罗");  ProcessInstance instance = service.startProcessInstanceByKey("payBill", params);//bpmn的id  System.***out***.println(instance.getId());  System.***out***.println(instance.getProcessInstanceId());  System.***out***.println(instance.getProcessDefinitionId());  } |