

# 尚马教育 JAVA 高级课程

## Activiti workflow引擎

文档编号：C15

创建日期：2021-02-26

最后修改日期：2021-02-26

版本号：V1.0

电子版文件名：尚马教育-第三阶段-15.activiti workflow引擎.docx

文档修改记录：

更新日期	更新作者	更新说明	版本号
2021-02-26	冯勇涛		V1.0

## 目录

尚马教育 JAVA 高级课程 .....	1
Activiti 工作流引擎 .....	1
1. Activiti 介绍 .....	3
1.1. 认识 activiti .....	3
1.2. 认识 BPMN .....	3
2. 环境安装 .....	3
2.1. 数据库环境 .....	4
2.2. 流程设计之 Eclipse 插件 .....	7
2.3. 代码运行环境 .....	11
3. IdentityService 服务 .....	13
4. 流程设计 .....	15
5. RepositoryService 服务 .....	17
6. 创建请假申请单业务表与实体类 .....	19
7. RuntimeService 服务 .....	20
8. TaskService 服务 .....	21
8.1. 查询待办任务 .....	21
8.2. 多人任务签收 .....	22
8.3. 签收后审批 .....	22
9. HistoryService 流程跟踪 .....	23
9.1. 申请人查询历史任务 .....	24
9.2. 审批人查询已办任务 .....	25

## 1. Activiti 介绍

### 1.1. 认识 activiti

Activiti 基于 Apache 许可的开源 BPM（业务流程管理）平台，创始人 Tom Baeyens 是 JBoss jBPM 的项目架构师，因此 activiti 很多思想与 jbpm 流程管理框架类似，activiti 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。

在企业日常运营中，企业业务流程审批是非常常见的，比如：财务报销流程，贴发票到报销单，提交给组长，组长审批后提交主管，主管审批后提交财务部，财务部主管审批后，向申请人银行卡打钱。这样一个财务报销流程就执行完成。

### 1.2. 认识 BPMN

Bpmn 是业务流程建模与标注 (Business Process Model and Notation, BPMN) 的一种标记语言，描述流程的基本符号，包括这些图元如何组合成一个业务流程图 (Business Process Diagram)，activiti 的 eclipse 插件即是一套 bpmn 流程定义的实现插件。

## 2. 环境安装

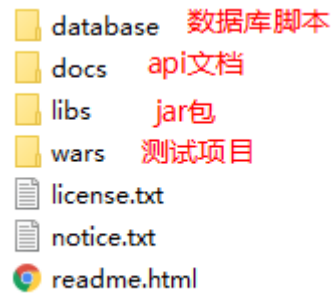
Ssm 环境下，Activiti 目前主流应用版本 5.22。

版本 7 主要针对 springboot 提供支持。

下载地址：




<https://github.com/Activiti/Activiti/releases/download/activiti-5.22.0/activiti-5.22.0.zip>

解压压缩包后，如下图：




























## 2.1. 数据库环境

找到 database 文件夹下的 created 脚本，执行 mysql 数据库的三个脚本文件。

 activiti.mysql.create.engine.sql	201
 activiti.mysql.create.history.sql	201
 activiti.mysql.create.identity.sql	201

创建成功后，数据库有 25 张表：

 act_ru_variable	InnoDB	16 KB	0
 act_ru_task	InnoDB	16 KB	0
 act_ru_job	InnoDB	16 KB	0
 act_ru_identitylink	InnoDB	16 KB	0
 act_ru_execution	InnoDB	16 KB	0
 act_ru_event_subscr	InnoDB	16 KB	0
 act_re_procdef	InnoDB	16 KB	0
 act_re_model	InnoDB	16 KB	0
 act_re_deployment	InnoDB	16 KB	0
 act_procdef_info	InnoDB	16 KB	0
 act_id_user	InnoDB	16 KB	4
 act_id_membership	InnoDB	16 KB	4
 act_id_info	InnoDB	16 KB	0
 act_id_group	InnoDB	16 KB	3
 act_hi_varinst	InnoDB	16 KB	16
 act_hi_taskinst	InnoDB	16 KB	10
 act_hi_procin	InnoDB	16 KB	4
 act_hi_identitylink	InnoDB	16 KB	14
 act_hi_detail	InnoDB	16 KB	0
 act_hi_comment	InnoDB	16 KB	0
 act_hi_attachment	InnoDB	16 KB	0
 act_hi_actinst	InnoDB	16 KB	22
 act_ge_property	InnoDB	16 KB	3
 act_ge_bytearray	InnoDB	2576 KB	2
 act_evt_log	1 InnoDB	16 KB	0

数据库表介绍：

表名默认以“ACT\_”开头,并且表名的第二部分用两个字母表明表的用例,而这个用例也基本上跟 Service API 匹配。

ACT\_GE\_\*: “GE”代表“General”（通用），用在各种情况下；

ACT\_HI\_\*: “HI”代表“History”（历史），这些表中保存的都是历史数据，比如执行过的流程实例、变量、任务，等等。Activiti 默认提供了 4 种历史级别：

ACT\_ID\_\*: “ID”代表“Identity”（身份），这些表中保存的都是身份信息，如用户和组以及两者之间的关系。如果 Activiti 被集成在某一系统当中的话，这些表可以不用，可以直接使用现有系统中的用户或组信息；

ACT\_RE\_\*: “RE”代表“Repository”（仓库），这些表中保存一些‘静态’信息，如流程定义和流程资源（如图片、规则等）；

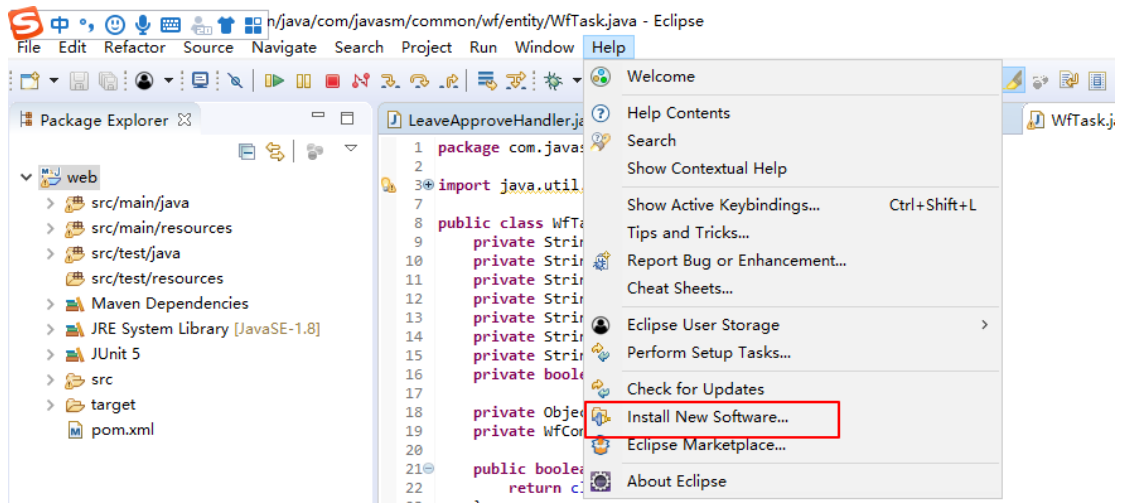
ACT\_RU\_\*: “RU”代表“Runtime”（运行时），这些表中保存一些流程实例、用户任务、变量等的运行时数据。Activiti 只保存流程实例在执行过程中的运行时数据，并且当流程结束后会立即移除这些数据，这是为了保证运行时表尽量的小并运行的足够快；

表分类	表名称	表含义
一般数据	act_evt_log	事件处理日志表
	act_ge_bytearray	通用的流程定义和流程资源
	act_ge_property	系统相关属性
流程历史记录	act_hi_actinst	历史的流程实例
	act_hi_attachment	历史的流程附件
	act_hi_comment	历史的说明性信息
	act_hi_detail	历史的流程运行中的细节信息
	act_hi_identitylink	历史的流程运行过程中用户关系
	act_hi_procinst	历史的流程实例

表分类	表名称	表含义
	act_hi_taskinst	历史的任务实例
	act_hi_varinst	历史的流程运行中的变量信息
用户用户组表	act_id_group	身份信息-组信息
	act_id_info	身份信息-组信息
	act_id_membership	身份信息-用户和组关系的中间表
	act_id_user	身份信息-用户信息
	act_procdef_info	死信任务
流程定义表	act_re_deployment	部署单元信息
	act_re_model	模型信息
	act_re_procdef	已部署的流程定义
运行实例表	act_ru_deadletter_job	执行失败任务表
	act_ru_event_subscr	运行时事件
	act_ru_execution	运行时流程执行实例
	act_ru_identitylink	运行时用户关系信息
	act_ru_job	运行时作业
	act_ru_suspended_job	运行时暂停任务
	act_ru_task	运行时任务
	act_ru_timer_job	运行时定时任务
	act_ru_variable	运行时变量表

## 2.2. 流程设计之 Eclipse 插件

第一步：点击 eclipse 上方工具栏的 Help，选择 Install New Software。

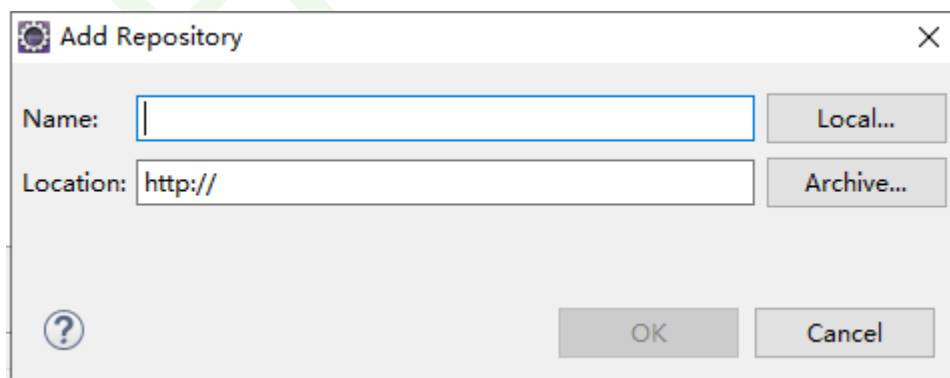


第二步：在弹出窗口下点击 add 按钮，填写插件名称与安装地址。

### ● 在线安装：

Name: Activiti BPMN 2.0 designer

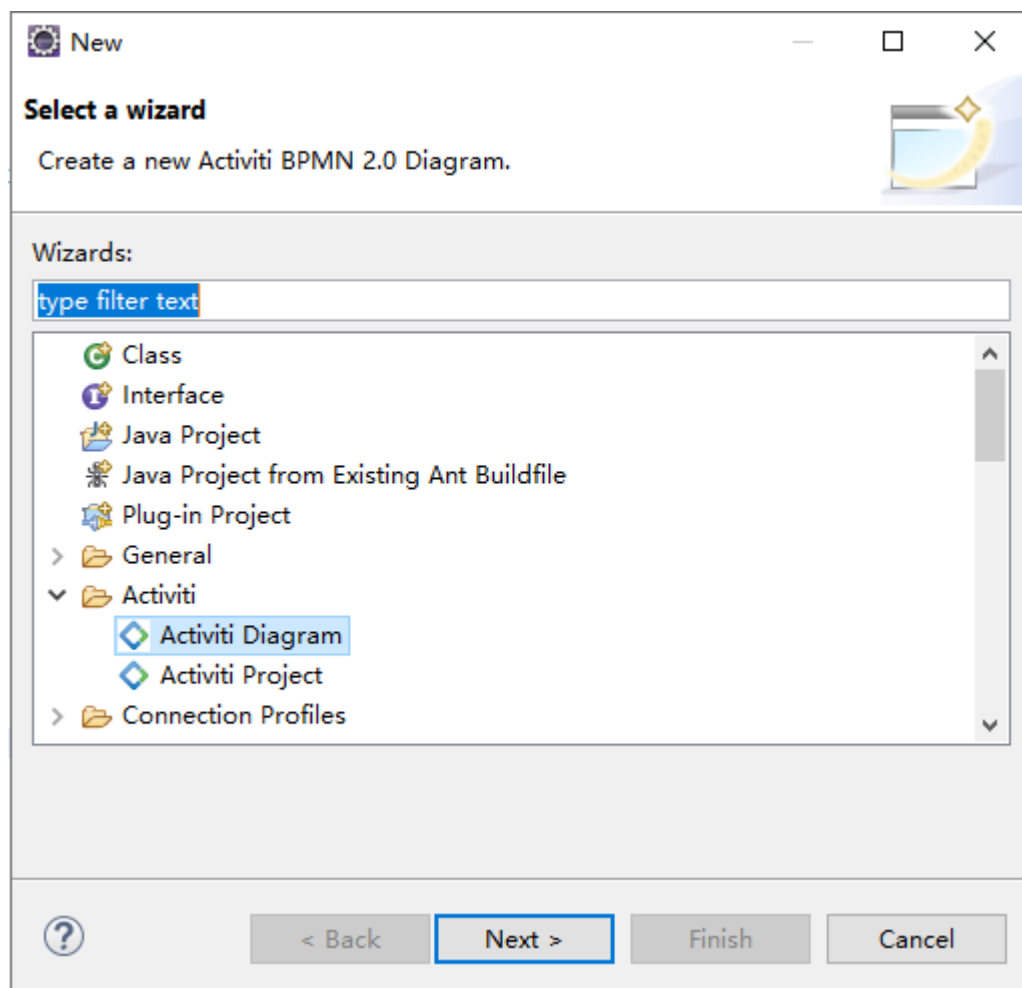
Location: <http://activiti.org/designer/update/>



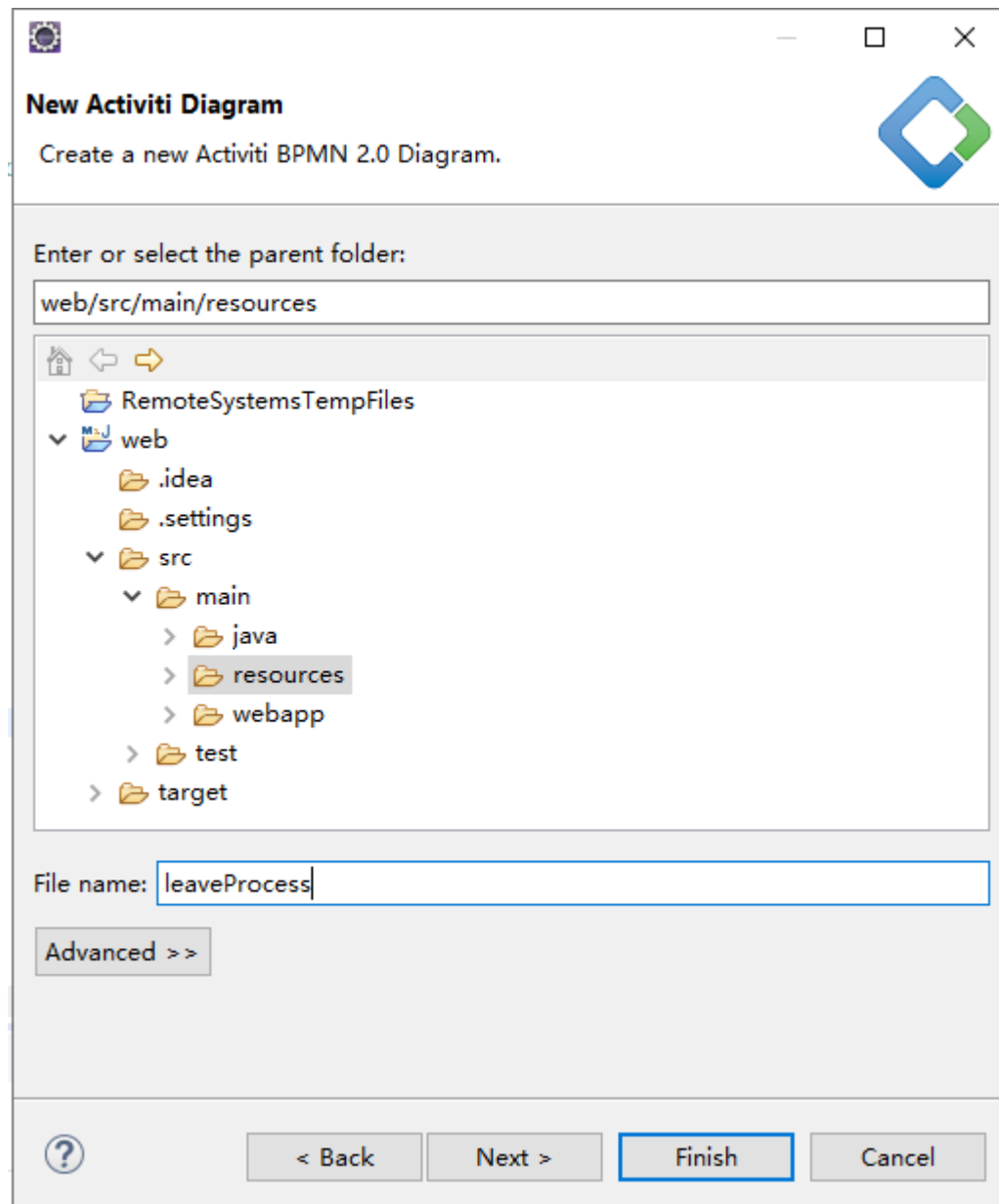
### ● 离线安装

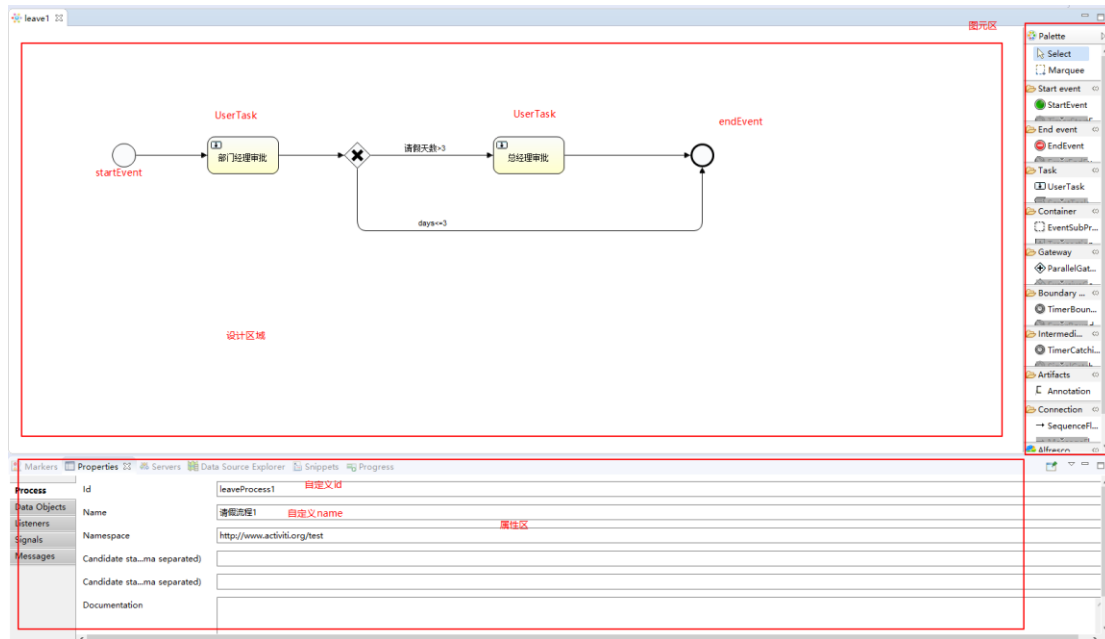
在添加窗口点击 Local 按钮，选择本地磁盘下的插件文件夹。课件：15.activiti\designer 插件。

插件安装成功后，在 new 创建下，选择如下：





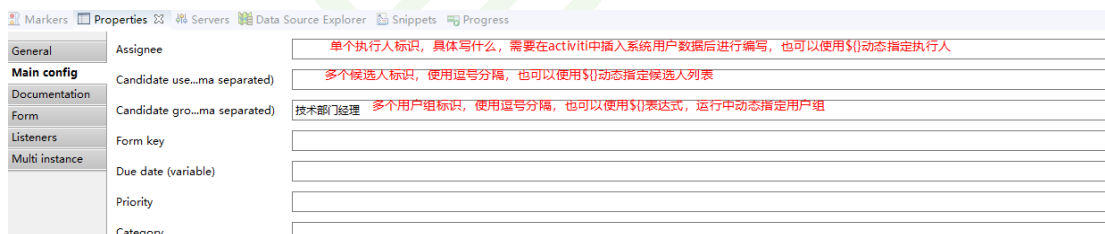




注意点 1: id 与 name 必须自定义为流程 id 与流程名, 不能重复。

注意点 2: 一个流程中, 有一个 StartEvent, 可以有多个 endEvent;

注意点 3: UserTask 中的 Assignee, Candidate users, Candidate groups 三者中选择一个指定任务参与人。

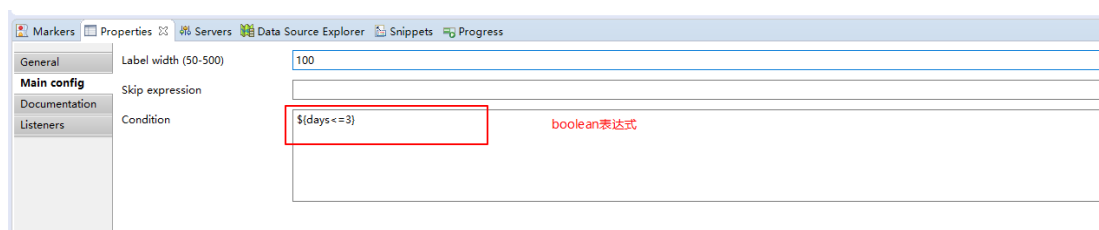


The screenshot shows the Properties panel for a UserTask. The "Main config" tab is selected, and the following properties are visible:

Property	Value
Assignee	单个执行人标识, 具体写什么, 需要在activiti中插入系统用户数据后进行编写, 也可以使用\${}动态指定执行人
Candidate use...ma separated)	多个候选人标识, 使用逗号分隔, 也可以使用\${}动态指定候选人列表
Candidate gro...ma separated)	技术部门经理 多个用户组标识, 使用逗号分隔, 也可以使用\${}表达式, 运行中动态指定用户组
Form key	
Due date (variable)	
Priority	
Category	

注意点 4: 当流程产生重要分支, 需要加 Gateway 网关进行分支, 多选一分支, 使用 ExclusiveGateway 排它网关; 并行分支, 选用 ParallelGateway 并行网关。

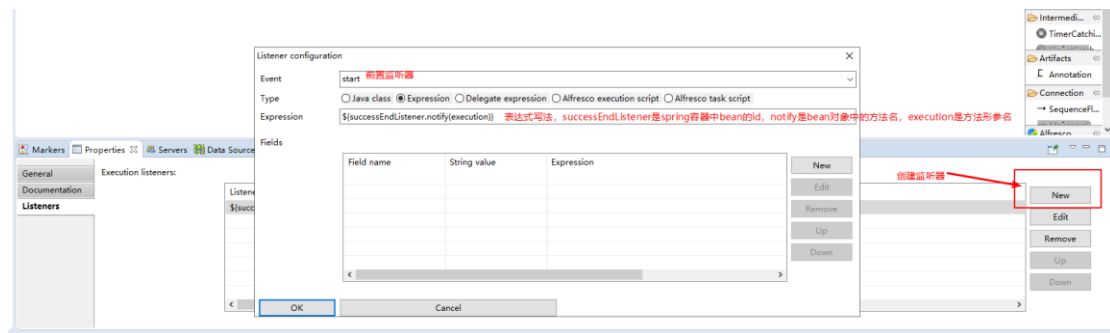
注意点 5: 分支后的 connection 连接线, 需要指定分支条件。



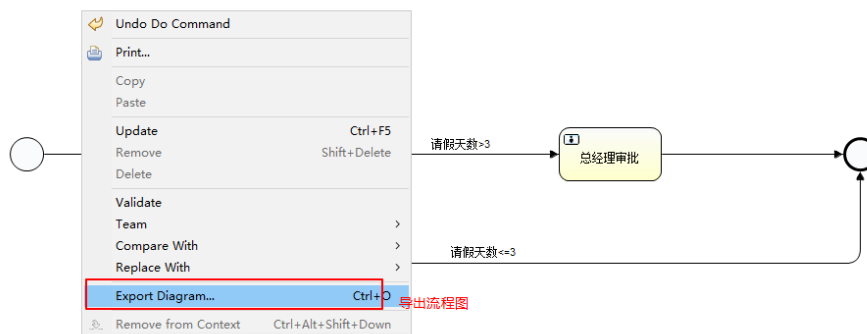
The screenshot shows the Properties panel for a connection. The "Main config" tab is selected, and the following properties are visible:

Property	Value
Label width (50-500)	100
Skip expression	
Condition	\${days <= 3} boolean表达式

注意点 6: UserTask 或 endEvent, startEvent 可以指定 listener 监听器, 当执行到当前节点时, 执行监听器代码, 比如流程审批通过发邮件或短信通知申请人。以 endEvent 为例:



注意点 7: 流程设计完成后, 导入图片。



## 2.3. 代码运行环境

在 ssm 项目环境下安装 activiti 依赖包。

### 2.3.1. 添加 maven 依赖

```
<dependency>
  <groupId>org.activiti</groupId>
  <artifactId>activiti-engine</artifactId>
  <version>5.22.0</version>
</dependency>
<dependency>
```

```
<groupId>org.activiti</groupId>

<artifactId>activiti-spring</artifactId>

<version>5.22.0</version>

</dependency>
```

### 2.3.2. 创建 activiti 的配置文件

在 src/main/resources 下创建 activiti.xml，spring 风格。

```
<!--ProcessEngineConfiguration-->
<bean id="processEngineConfiguration"
class="org.activiti.spring.SpringProcessEngineConfiguration">
    <property name="dataSource" ref="dataSource" />
    <property name="transactionManager" ref="transactionManager"
/>

    <!--在创建 ProcessEngine 对象的时候是否创建更新数据库表-->
    <property name="databaseSchemaUpdate" value="false" />
    <property name="jobExecutorActivate" value="false" />
</bean>

<!--ProcessEngine 流程引擎核心对象-->
<bean id="processEngine"
class="org.activiti.spring.ProcessEngineFactoryBean">
    <property name="processEngineConfiguration"
ref="processEngineConfiguration" />
</bean>

<!--用来操作用户与角色信息-->
<bean id="identityService" factory-bean="processEngine" factory-
method="getIdentityService"></bean>

<!--用来进行流程部署-->
<bean id="repositoryService" factory-bean="processEngine"
```

```
factory-method="getRepositoryService" />

<!--用来启动流程-->

<bean id="runtimeService" factory-bean="processEngine" factory-
method="getRuntimeService" />

<!--用来在流程启动后，执行任务签收，完成任务-->

<bean id="taskService" factory-bean="processEngine" factory-
method="getTaskService" />

<!--用来对于已结束的历史流程与历史任务查询-->

<bean id="historyService" factory-bean="processEngine" factory-
method="getHistoryService" />
```

注意点：切记把 activiti.xml 的加载一定要配置。

### 3. IdentityService 服务

该服务用来管理引擎内依赖的用户与用户组数据。

代码如下：

```
/**
 * 用户与用户组数据不能删除，不能修改 id
 */
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:spring.xml")
public class TestIdentityService {

    @Resource
    private IdentityService ds;

    //添加引擎依赖用户，在系统管理下用户管理功能处加入如下代码，维护两套用户
    数据。

    @Test
```

```

public void test1_addUser() {
    //用户唯一标识（登录账号）

    User user1 = ds.newUser("jpc");
    User user2 = ds.newUser("zbw");
    User user3 = ds.newUser("lhd");
    User user4 = ds.newUser("wts");

    ds.saveUser(user1);
    ds.saveUser(user2);
    ds.saveUser(user3);
    ds.saveUser(user4);
}

//添加引擎依赖用户组，在角色管理模块处加入如下代码，维护两套用户组数据
@Test
public void test1_addGroup() {
    Group g1 = ds.newGroup("普通员工");
    Group g2 = ds.newGroup("技术部门经理");
    Group g3 = ds.newGroup("总经理");
    ds.saveGroup(g1);
    ds.saveGroup(g2);
    ds.saveGroup(g3);
}

//为用户分配用户组
@Test
public void test1_memberShip() {
    ds.createMembership("jpc", "普通员工");
    ds.createMembership("zbw", "技术部门经理");
    ds.createMembership("lhd", "技术部门经理");
}

```

```
ds.createMembership("wts", "总经理");

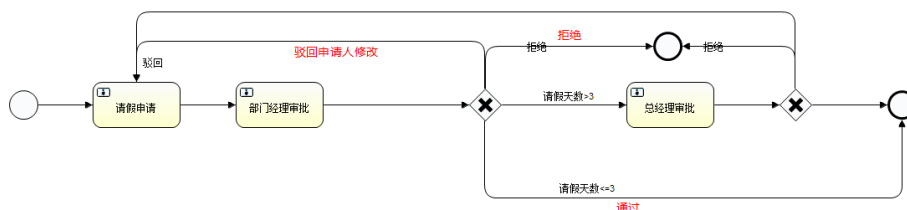
}

}
```

注意点：在流程设计时，为 UserTask 指定的执行人候选人，用户组数据都是这里维护的数据标识。

## 4. 流程设计

假设审批人有三种审批结果：通过，驳回，拒绝。



注意点 1：UserTask 一定要指定执行人或候选人或候选用户组。

注意点 2：这里为两个 endEvent 分别指定了 Listener，成功结束的如下：

拒绝的如下：

Listener configuration

Event

start

Type

☐ Java class
☒ Expression
☐ Delegate expression
☐ Alfresco execution script
☐ Alfresco task script

Expression

Fields

Field name	String value	Expression

New

Edit

Remove

Up

Down

OK

Cancel

监听器代码如下：

```

@Service("successEndListener")
public class SuccessEndListener {

    @Resource

    private ILeaveService ls;

    public void notify(DelegateExecution execution){

        String processInstanceId =
execution.getProcessInstanceId();

        System.out.println("审核通过结束: "+processInstanceId);

        //把请假申请单的状态修改为审核通过

        ls.updateStatusByProcessId(processInstanceId, "审批通过");

    }

}

```

```

@Service("errorEndListener")
public class ErrorEndListener {

    @Resource

    private ILeaveService ls;

    public void notify(DelegateExecution execution){

```



```
String processInstanceId =
execution.getProcessInstanceId();

System.out.println("审核不通过结束: "+processInstanceId);

//把请假申请单的状态修改为审核未通过

ls.updateStatusByProcessId(processInstanceId, "审批未通过");

}

}
```

## 5. RepositoryService 服务

该服务用来管理流程部署，把设计的流程定义文件部署到数据库。

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:spring.xml")
public class TestRepositoryService {

    @Resource

    private RepositoryService rs;

    //部署流程

    @Test
    public void deploy() {
        DeploymentBuilder deployment = rs.createDeployment();
        deployment.addClasspathResource("workflow/leave2.bpmn");
        deployment.addClasspathResource("workflow/leave2.png");
        deployment.name("请假流程 2 部署");
        Deployment deploy = deployment.deploy();//向数据库中添加流程
        定义文件

        System.out.println(deploy.getId()); //2501
    }
}
```

```
}

//查询流程部署记录

@Test

public void queryDeploy(){

    DeploymentQuery query = rs.createDeploymentQuery();

    List<Deployment> list = query.list();

}

//删除流程部署

@Test

public void del(){

    rs.deleteDeployment("30001");

}

}
```

## 6. 创建请假申请单业务表与实体类

wf\_leave @aa (local) - 表 \*

文件 编辑 窗口 帮助

新建 保存 另存为 添加栏位 插入栏位 删除栏位 主键 上移 下移

栏位 索引 外键 触发器 选项 注释 SQL 预览

名	类型	长度	小数点	允许空值 (	
lid	int	11	0	<input type="checkbox"/>	1
luser	varchar	255	0	<input checked="" type="checkbox"/>	
ldays	int	255	0	<input checked="" type="checkbox"/>	
lbegin	datetime	255	0	<input checked="" type="checkbox"/>	
lreason	varchar	255	0	<input checked="" type="checkbox"/>	
processid	varchar	255	0	<input checked="" type="checkbox"/>	
lstatus	varchar			<input checked="" type="checkbox"/>	

默认: 草稿

注释: 申请单状态

字符集:

排序规则:

键长度:

☐ 二进制

栏位数: 7

对应实体类

```
public class WfLeave {
    private Integer lid;
    private String luser;
    private Integer ldays;
    private String lreason;
    private String lbegin;
    private String processid; //流程实例id
    private String lstatus; //审批状态,默认草稿状态
}
```

对应的 service 层与 dao 层基本 curd 代码，省略。

## 7. RuntimeService 服务

启动流程

```
/**
 * 提交流程审批
 * @param leave, 请假申请单对象, 该对象数据要完整, 有 id, days, luser 信息
 */
@Override
public void startProcess(WfLeave leave) {
    //1. 把请假申请单保存到请假申请表, 默认草稿状态
    this.addLeaveApply(leave);

    //2. 启动流程, "leaveProcess2"是流程设计时, 指定的 id, days 与
    loginUser 是流程设计时需要的流程变量

    Map<String, Object> vali = new HashMap<>();
    vali.put("days", leave.getLdays());
    vali.put("loginUser", leave.getLuser());

    ProcessInstance procInst =
rs.startProcessInstanceByKey("leaveProcess2", vali); //流程实例对象
    //新启动的流程实例 id
    String procId = procInst.getId();
    //把 ProcessId 更新到业务表中的 processid 外键
    this.updateProcessId2Leave(procId, leave.getLid() + "");
    //把请假申请节点走完

    List<Task> todoTask = this.getTodoTask(leave.getLuser());
    String taskid = todoTask.get(0).getId();

    this.completeTask(taskid, null);
}
```

```

@Override

public void updateProcessId2Leave(String processId, String
leaveId) {

    WfLeave l = new WfLeave();

    l.setLid(Integer.parseInt(leaveId));

    l.setProcessid(processId);

    l.setLstatus("审批中");

    lm.updateByPrimaryKeySelective(l);

}

@Override

public void updateStatusByProcessId(String processInstanceId,
String result) {

    lm.updateStatusByProcessId(processInstanceId, result);

}

```

## 8. TaskService 服务

### 8.1. 查询待办任务

```

@Override

public List<Task> getTodoTask(String loginUserId) {

    TaskQuery taskQuery = ts.createTaskQuery();

    taskQuery.taskCandidateOrAssigned(loginUserId);

    //如果不指定查询流程定义的 key。则查询所有业务的待办任务,适合于用在
    工作台欢迎页面

    taskQuery.processDefinitionKey("leaveProcess2");//这里表示
    查询请假审批的待办任务

    //      long count = taskQuery.count();//查询总记录数
    //      taskQuery.listPage(1,3);//分页查询

```

```

        List<Task> list = taskQuery.list();

        return list;
    }

```

## 8.2. 多人任务签收

```

@Override
public void doClaim(String loginUser, String taskid) {
    ts.claim(taskid,loginUser);
}

```

## 8.3. 签收后审批

```

/**
 *
 * @param taskid, 审批任务 id
 * @param result: 审批结果 (通过, 拒绝, 驳回)
 * @param comment: 审批意见
 */
@Transactional
@Override
public void doApprove(String taskid, String result, String
comment) {

    TaskQuery taskQuery = ts.createTaskQuery();

    taskQuery.taskId(taskid);

    Task task = taskQuery.singleResult();

    String processInstanceId = task.getProcessInstanceId();

    ts.addComment(taskid,processInstanceId,result,comment);
}

```

```

        this.completeTask(taskid, result);
    }

    @Override
    public void completeTask(String taskid, String result) {
        if(result==null){
            ts.complete(taskid);
            return;
        }

        Map<String, Object> vali = new HashMap<>();

        //1: 通过, 2: 不通过, 3: 驳回
        vali.put("approveResult", result);

        ts.complete(taskid, vali);
    }

```

## 9. HistoryService 流程跟踪

创建实体

```

public class HistoryTaskAndComment {
    private String taskid; //任务 id
    private String processId; //流程实例 id
    private String taskName; //任务名
    private String createTime; //任务开始时间
    private String endTime; //结束时间
    private String result; //审批结果
    private String comment; //审批意见
    private String assignee; //任务执行人
}

```

}

## 9.1. 申请人查询历史任务

```

/**
 * 作为申请人，查看申请单下的已执行节点，以及任务审批意见
 * @param processId
 * @return
 */
@Override
public List<HistoryTaskAndComment>
getHistoryTaskByProcessId(String processId) {
    List<HistoryTaskAndComment> result = new ArrayList<>();
    //得到执行流程实例下的所有审批意见
    List<Comment> comments =
ts.getProcessInstanceComments(processId);
    //查询历史任务
    HistoricTaskInstanceQuery htq =
hs.createHistoricTaskInstanceQuery(); //历史任务查询
    htq.processInstanceId(processId);
    htq.orderByTaskCreateTime();
    htq.asc();

    //把历史任务和审批意见组合
    List<HistoricTaskInstance> list = htq.list();
    for (HistoricTaskInstance t : list) {
        HistoryTaskAndComment tc = new
HistoryTaskAndComment(t.getId(), t.getProcessInstanceId(), t.getNa
me(), DateUtils.formatYMDHmS(t.getStartTime()), DateUtils.formatYM
DHmS(t.getEndTime()), t.getAssignee());
    }
}

```



```

        Comment c = this.getCommentsByTaskId(comments, t.getId());

        if (c != null)
        {
            tc.setResult(c.getType());

            tc.setComment(c.getFullMessage());

        }

        result.add(tc);

    }

    return result;
}

private Comment getCommentsByTaskId(List<Comment>
comments, String taskId) {

    for (Comment comment : comments) {

        if (taskId.equals(comment.getTaskId())) {

            return comment;

        }

    }

    return null;

}

```

## 9.2. 审批人查询已办任务

```

/**
 * 某节点审批人登录系统，查询自身已办的任务
 * @param loginUserId
 */
@Override
public void getHistoryTaskByAssignee(String loginUserId) {

```

```
HistoricTaskInstanceQuery htq =  
hs.createHistoricTaskInstanceQuery();  
  
htq.taskAssignee(loginuserId);  
  
htq.processDefinitionKey(key2);  
  
List<HistoricTaskInstance> list = htq.list();  
  
for (HistoricTaskInstance t : list) {  
    System.out.println(t.getId()+"-"  
"+t.getProcessInstanceId()+"-"+t.getAssignee()+"-"  
"+t.getEndTime()+"-"+t.getName());  
}  
}
```