# SaaS-IHRM 项目-Activiti7

# 工作流引擎

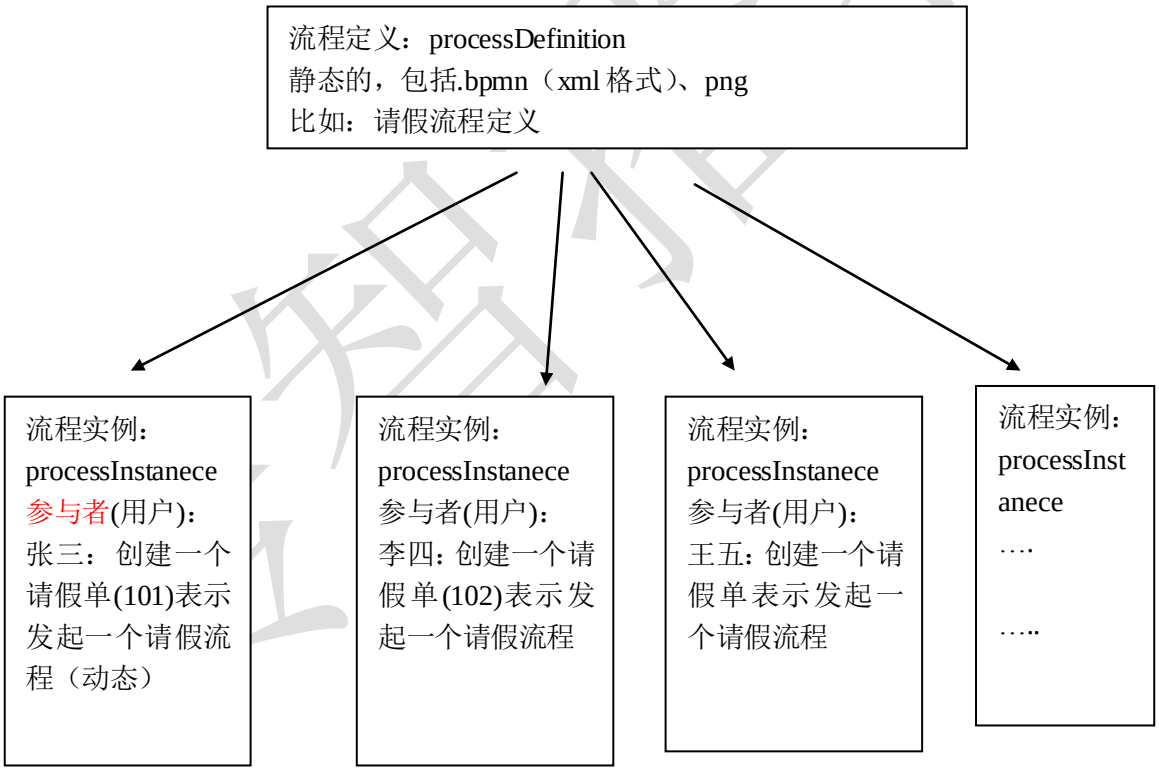
传智播客-研究院

# 第1章 流程实例

## 1.1 什么是流程实例

参与者（可以是用户也可以是程序）按照流程定义内容发起一个流程，这就是一个流程实例。是动态的。

流程定义和流程实例的图解：




## 1.2 启动流程实例

流程定义部署在 activiti 后，就可以在系统中通过 activiti 去管理该流程的执行，执行流程表示流程的一次执行。比如部署系统请假流程后，如果某用户要申请请假这时就需要执行这个流程，如果另外一个用户也要申请请假则也需要执行该流程，每个执行互不影响，每个执行是单独的流程实例。

执行流程首先需要启动流程实例。

```
@Test
    public void startProcessInstance() {

        // 流程定义key
        String processDefinitionKey = "";
        // 获取RunTimeService
        RuntimeService runtimeService = processEngine.getRuntimeService();
        // 根据流程定义key启动流程
        ProcessInstance processInstance = runtimeService
                .startProcessInstanceByKey(processDefinitionKey);

        System.out
                .println("    流    程    定    义    id    :    "    +
processInstance.getProcessDefinitionId());
        System.out.println("流程实例id: " + processInstance.getId());
        System.out.println("当前活动Id: " + processInstance.getActivityId());
        }
```

## 1.3 Businesskey(业务标识)

启动流程实例时，指定的businesskey，就会在act_ru_execution #流程实例的执行表中存储businesskey。

Businesskey：业务标识，通常为业务表的主键，业务标识和流程实例一一对应。业务标识来源于业务系统。存储业务标识就是根据业务标识来关联查询业务系统的数据。
比如：请假流程启动一个流程实例，就可以将请假单的 id 作为业务标识存储到 activiti 中，将来查询 activiti 的流程实例信息就可以获取请假单的 id 从而关联查询业务系统数据库得到请假单信息。

代码：

```
// 根据流程定义的key启动一个流程实例
        ProcessInstance processInstance = runtimeService
                .startProcessInstanceByKey(processDefinitionKey,
businessKey);
```

Activiti 中存储业务标识：

| ID_ | REV_ | PROC_INST_ID_ | BUSINESS_KEY_ | PARENT_ID_ | PROC_DEF_ID_ |
|---|---|---|---|---|---|
| 1401 | 1 | 1401 | (NULL) | (NULL) | purchasingflow:1:13 |
| 1501 | 1 | 1501 | 10101 | (NULL) | purchasingflow:1:13 |

## 1.4 操作数据库表

启动流程实例，操作如下数据库表：

SELECT * FROM act_ru_execution #流程实例执行表，记录当前流程实例的执行情况

| ID | REV_ | PROC_INST_ID_ | BUSINESS_KEY_ | PARENT_ID_ | PROC_DEF_ID_ | SUPER_EXEC_ | ACT_ID_ |
|---|---|---|---|---|---|---|---|
| 101 | 1 | 101 | (NULL) | (NULL) | purchasingflow01:1:4 | (NULL) | createOrder |

说明：
流程实例执行，如果当前只有一个分支时，一个流程实例只有一条记录且执行表的主键 id 和流程实例 id 相同，如果当前有多个分支正在运行则该执行表中有多条记录，存在执行表的主键和流程实例 id 不相同的记录。**不论当前有几个分支总会有一条记录的执行表的主键和流程实例 id 相同**
一个流程实例运行完成，此表中与流程实例相关的记录删除。

SELECT * FROM act_ru_task #任务执行表，记录当前执行的任务

| ID_ | REV_ | EXECUTION_ID_ | PROC_INST_ID_ | PROC_DEF_ID_ | NAME_ | PARENT_TASK_ID_ | DESCRIPTION_ | TASK_DEF_KEY_ |
|---|---|---|---|---|---|---|---|---|
| 105 | 1 | 101 | 101 | purchasingflow01:1:4 | 创建采购单 | (NULL) | (NULL) | createOrder |
| * | (NULL) | (NULL) | (NULL) | (NULL) | (NULL) | (NULL) | (NULL) | (NULL) |

说明：启动流程实例，流程当前执行到第一个任务结点，此表会插入一条记录表示当前任务的执行情况，如果任务完成则记录删除。

SELECT * FROM act_ru_identitylink #任务参与者，记录当前参与任务的用户或组

| ID_ | REV_ | GROUP_ID_ | TYPE_ | USER_ID_ | TASK_ID_ | PROC_INST_ID_ |
|---|---|---|---|---|---|---|
| 106 | 1 | (NULL) | participant | zhangsan | (NULL) | 101 |
| | (NULL) | (NULL) | (NULL) | (NULL) | (NULL) | (NULL) |

SELECT * FROM act_hi_procinst #流程实例历史表

| ID_ | PROC_INST_ID_ | BUSINESS_KEY_ | PROC_DEF_ID | START_TIME_ | END_TIME_ | DURATION_ | START_USER_ID_ | START_ACT_ID_ |
|---|---|---|---|---|---|---|---|---|
| 101 | 101 | (NULL) | purchasingflow01:1:4 | 2014-11-03 14:11:26 | (NULL) | (NULL) | (NULL) | startevent1 |
| (NULL) | (NULL) | (NULL) | (NULL) | (NULL) | (NULL) | (NULL) | (NULL) | (NULL) |

流程实例启动，会在此表插入一条记录，流程实例运行完成记录也不会删除。

SELECT * FROM act_hi_taskinst #任务历史表，记录所有任务

| ID_ | PROC_DEF_ID_ | TASK_DEF_KEY_ | PROC_INST_ID_ | EXECUTION_ID_ | NAME_ | P.. | D.. | O.. | ASSIGNEE_ | START_TIME_ | C.. | END_TIME_ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 105 | purchasingflow01:1:4 | createOrder | 101 | 101 | 创建采购单 | (NU | (NU | (NU | zhangsan | 2014-11-03 14:11:26 | (NU | (NULL) |
| (NULL) | (NULL) | (NULL) | (NULL) | (NULL) | (NULL) | (NU | (NU | (NU | (NULL) | (NULL) | (NU | (NULL) |

开始一个任务，不仅在 act_ru_task 表插入记录，也会在历史任务表插入一条记录，任务历史表的主键就是任务 id，任务完成此表记录不删除。

SELECT * FROM act_hi_actinst #活动历史表，记录所有活动

| ID_ | PROC_DEF_ID_ | PROC_INST_ID | EXECUTION_ID_ | ACT_ID_ | TASK_ID_ | CALL_PROC_INST_ID_ | ACT_NAME_ | ACT_TYPE_ | ASSIGNEE_ |
|---|---|---|---|---|---|---|---|---|---|
| 103 | purchasingflow01:1:4 | 101 | 101 | startevent1 | (NULL) | (NULL) | Start | startEvent | (NULL) |
| 104 | purchasingflow01:1:4 | 101 | 101 | createOrder | 105 | (NULL) | 创建采购单 | userTask | zhangsan |

活动包括任务，所以此表中不仅记录了任务，还记录了流程执行过程的其它活动，比如开始事件、结束事件。

## 1.5 查询流程实例

流程在运行过程中可以查询流程实例的状态，当前运行结点等信息。

```
@Test
    public void queryProcessInstance() {
        // 流程定义key
        String processDefinitionKey = "holiday";
        // 获取RunTimeService
        RuntimeService runtimeService = processEngine.getRuntimeService();
        List<ProcessInstance> list = runtimeService
                .createProcessInstanceQuery()
                .processDefinitionKey(processDefinitionKey)//
                .list();

        for (ProcessInstance processInstance : list) {
            System.out.println("----------------------------");
            System.out.println("流程实例id：" 
                    + processInstance.getProcessInstanceId());
            System.out.println("所属流程定义id：" 
                    + processInstance.getProcessDefinitionId());
            System.out.println("是否执行完成：" + processInstance.isEnded());
            System.out.println("是否暂停：" + processInstance.isSuspended());
            System.out.println("当前活动标识：" +
processInstance.getActivityId());
        }
    }
```

### 1.5.1 关联 businessKey

需求：
在 activiti 实际应用时，查询流程实例列表时可能要显示出业务系统的一些相关信息，比如：查询当前运行的请假流程列表需要将请假单名称、请假天数等信息显示出来，请假天数等信息在业务系统中存在，而并没有在 activiti 数据库中存在，所以是无法通过 activiti 的 api 查询到请假天数等信息。

实现：

在查询流程实例时，通过 businessKey（业务标识 ）关联查询业务系统的请假单表，查询出请假天数等信息。

通过下面的代码就可以获取 activiti 中所对应实例保存的业务 Key。而这个业务 Key 一般都会保存相关联的业务操作表的主键，再通过主键 ID 去查询业务信息，比如通过请假单的 ID，去查询更多的请假信息（请假人，请假时间，请假天数，请假事由等）

String businessKey = processInstance.getBusinessKey();

在 activiti 的 act_ru_execution 表，字段 BUSINESS_KEY 就是存放业务 KEY 的。

| | ID_ | REV_ | PROC_INST_ID_ | BUSINESS_KEY_ | PARENT_ID_ | PROC_DEF_ID_ | SUPER_EXEC_ | ROOT_PROC_INST_ID_ | ACT_ID_ |
|---|---|---|---|---|---|---|---|---|---|
| * | | (NULL) | (NULL) | (NULL) | (NULL) | (NULL) | (NULL) | (NULL) | (NULL) |

## 1.6 挂起、激活流程实例

某些情况可能由于流程变更需要将当前运行的流程暂停而不是直接删除，流程暂停后将不会继续执行。

### 1.6.1 全部流程实例挂起

操作流程定义为挂起状态，该流程定义下边所有的流程实例全部暂停：

流程定义为挂起状态该流程定义将不允许启动新的流程实例，同时该流程定义下所有的流程实例将全部挂起暂停执行。

```
// 挂起激活流程定义
    @Test
    public void suspendOrActivateProcessDefinition() {
        // 流程定义id
        String processDefinitionId = "";

        RepositoryService repositoryService = processEngine
                .getRepositoryService();
        // 获得流程定义
        ProcessDefinition processDefinition = repositoryService
                .createProcessDefinitionQuery()
                .processDefinitionId(processDefinitionId).singleResult();
        //是否暂停
        boolean suspend = processDefinition.isSuspended();
        if(suspend){
            //如果暂停则激活，这里将流程定义下的所有流程实例全部激活
           repositoryService.activateProcessDefinitionById(processDefinitionId,
true, null);
```

```
            System.out.println("流程定义：" +processDefinitionId+"激活");
        }else{
            //如果激活则挂起，这里将流程定义下的所有流程实例全部挂起
            repositoryService.suspendProcessDefinitionById(processDefinitionId,
true, null);
            System.out.println("流程定义：" +processDefinitionId+"挂起");
        }
    }
```

### 1.6.2 单个流程实例挂起

操作流程实例对象，针对单个流程执行挂起操作，某个流程实例挂起则此流程不再继续执行，完成该流程实例的当前任务将报异常。

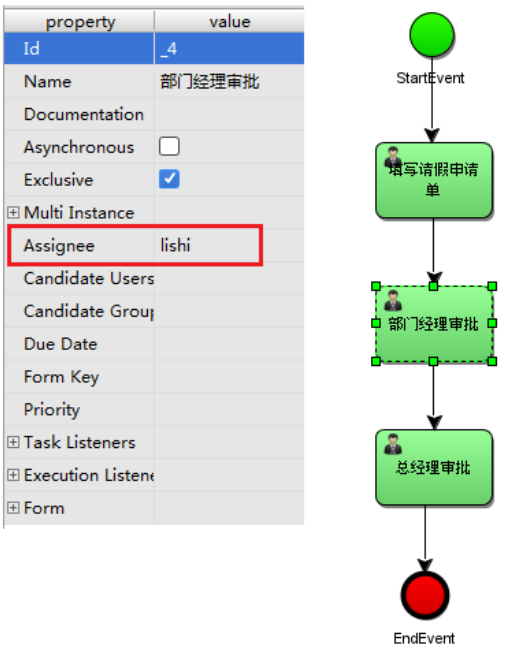
```
@Test
    public void suspendOrActiveProcessInstance() {
        // 流程实例id
        String processInstanceId = "";
        // 获取RunTimeService
        RuntimeService runtimeService = processEngine.getRuntimeService();
        //根据流程实例id查询流程实例
        ProcessInstance                    processInstance                    =
runtimeService.createProcessInstanceQuery()
                .processInstanceId(processInstanceId).singleResult();
        boolean suspend = processInstance.isSuspended();
        if(suspend){
            //如果暂停则激活
            runtimeService.activateProcessInstanceById(processInstanceId);
            System.out.println("流程实例：" +processInstanceId+"激活");
        }else{
            //如果激活则挂起
            runtimeService.suspendProcessInstanceById(processInstanceId);
            System.out.println("流程实例：" +processInstanceId+"挂起");
        }
    }
```

# 第2章 个人任务

## 2.1 分配任务负责人

### 2.1.1 固定分配

在进行业务流程建模时指定固定的任务负责人，




在 properties 视图中，填写 Assignee 项为任务负责人。

#### 2.1.1.1 注意事项

由于固定分配方式，任务只管一步一步执行任务，执行到每一个任务将按照 bpmn 的配置去分配任务负责人。

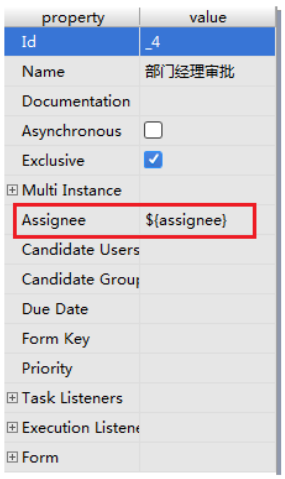
### 2.1.2 表达式分配

#### 2.1.2.1 UEL 表达式

Activiti 使用 UEL 表达式，UEL 是 java EE6 规范的一部分，UEL（**Unified Expression Language**）即

统一表达式语言，activiti 支持两个 UEL 表达式：UEL-value 和 UEL-method。

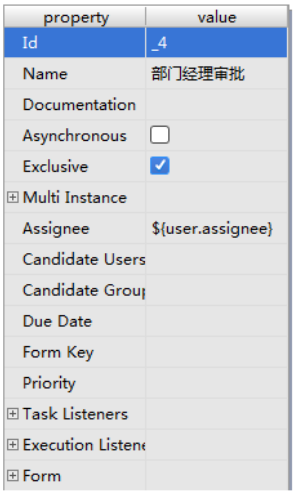
- UEL-value 定义如下：





assignee 这个变量是 activiti 的一个流程变量。

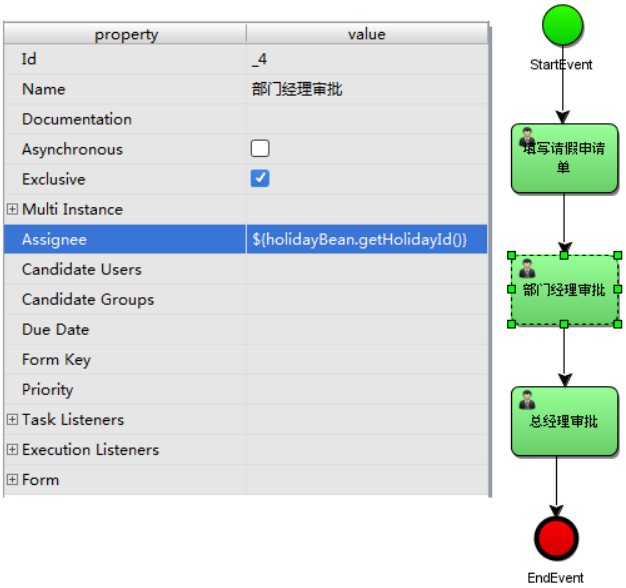
或：





user 也是 activiti 的一个流程变量，user.assignee 表示通过调用 user 的 getter 方法获取值。

- UEL-method 方式如下：




userBean 是 spring 容器中的一个 bean，表示调用该 bean 的 getUserId()方法。

- UEL-method 与 UEL-value 结合

再比如：

${ldapService.findManagerForEmployee(emp)}

ldapService 是 spring 容器的一个 bean，findManagerForEmployee 是该 bean 的一个方法，emp 是 activiti 流程变量，emp 作为参数传到 ldapService.findManagerForEmployee 方法中。

- 其它

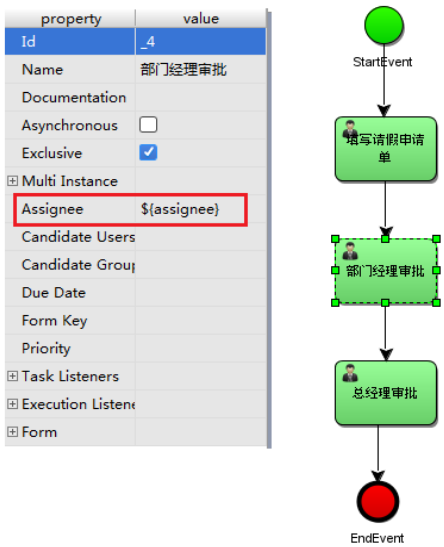
表达式支持解析基础类型、bean、list、array 和 map，也可作为条件判断。

如下：

${order.price > 100 && order.price < 250}

### 2.1.2.2 使用流程变量分配任务

- 定义任务分配流程变量



- 设置流程变量

在启动流程实例时设置流程变量，如下：

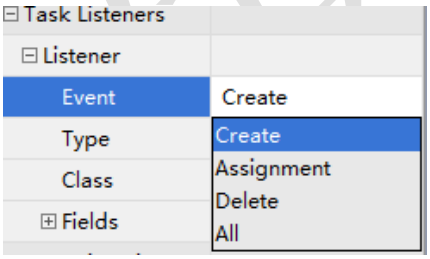
```
//启动流程实例时设计流程变量
        //定义流程变量
        Map<String, Object> variables = new HashMap<String, Object>();
        //设置流程变量assignee
        variables.put("assignee", "张三");
        ProcessInstance processInstance = runtimeService
                .startProcessInstanceByKey(processDefinitionKey, variables);
```

#### 2.1.2.3 注意事项

由于使用了表达式分配，必须保证在任务执行过程表达式执行成功，比如：
某个任务使用了表达式${order.price > 100 && order.price < 250}，当执行该任务时必须保证 order 在流程变量中存在，否则 activiti 异常。

### 2.1.3 监听器分配

任务监听器是发生对应的任务相关事件时执行自定义 java 逻辑 或表达式。
任务相当事件包括：




Create：任务创建后触发
Assignment：任务分配后触发
Delete：任务完成后触发
All：所有事件发生都触发

java 逻辑 或表达式：
表达式参考上边的介绍的 UEL 表达式，这里主要介绍监听类使用。

定义任务监听类，且类必须实现 org.activiti.engine.delegate.TaskListener 接口

```
public class MyTaskListener implements TaskListener {
    @Override
    public void notify(DelegateTask delegateTask) {
        //这里指定任务负责人
        delegateTask.setAssignee("张三");
    }
}
```

#### 2.1.3.1 注意事项

使用监听器分配方式，按照监听事件去执行监听类的 notify 方法，方法如果不能正常执行也会影响任务的执行。

## 2.2 查询任务

查询任务负责人的待办任务：

```
// 查询当前个人待执行的任务
    @Test
    public void findPersonalTaskList() {
        // 流程定义key
        String processDefinitionKey = "holiday";
        // 任务负责人
        String assignee = "张三丰";
        // 创建TaskService
        TaskService taskService = processEngine.getTaskService();
        List<Task> list = taskService.createTaskQuery()//
                .processDefinitionKey(processDefinitionKey)//
                .includeProcessVariables().taskAssignee(assignee).list();

        for (Task task : list) {
            System.out.println("----------------------------");
            System.out.println("流程实例id: " + task.getProcessInstanceId());
            System.out.println("任务id: " + task.getId());
            System.out.println("任务负责人: " + task.getAssignee());
            System.out.println("任务名称: " + task.getName());

        }
    }
```

### 2.2.1 关联 businessKey

需求：
在 activiti 实际应用时，查询待办任务可能要显示出业务系统的一些相关信息，比如：查询待审批请假单任务列表需要将请假单的日期、请假天数等信息显示出来，请假天数等信息在业务系统中存在，而并没有在 activiti 数据库中存在，所以是无法通过 activiti 的 api 查询到请假天数等信息。

实现：
在查询待办任务时，通过 businessKey（业务标识 ）关联查询业务系统的请假单表，查询出请假天数等信息。

```
    //2.创建RuntimeService对象
    RuntimeService runtimeService = processEngine.getRuntimeService();
    TaskService taskService = processEngine.getTaskService();

    //3.通过TaskService查询到个人任务
    Task task = taskService.createTaskQuery().processDefinitionKey("holiday")
            .taskAssignee("zhangsan").singleResult();

    //4.通过task对象，得到任务id
    String processInstanceId = task.getProcessInstanceId();

    //5.通过流程实例id,得到流程实例对象
    ProcessInstance processInstance = runtimeService.createProcessInstanceQuery()
            .processInstanceId(processInstanceId)
            .singleResult();

    //6.对象processInstance对象，得到businessKey
    String businessKey = processInstance.getBusinessKey();

    //7.根据businessKey就可以得到请假单的信息
    System.out.println("businessKey:"+businessKey);
}
```

### 2.3 处理任务

指定任务 id，调用 TaskService 完成任务：

```
// 完成任务
    @Test
    public void completTask() {
        //任务id
```

```
        String taskId = "10305";
        // 创建TaskService
        TaskService taskService = processEngine.getTaskService();
        taskService.complete(taskId);
        System.out.println("完成任务");
    }
```

注意：在实际应用中，完成任务前需要校验任务的负责人是否具有该任务的办理权限。

```
// 指定要办理任务id
String taskId = "2802";

// 任务负责人
String assignee = "zhaoliu";

TaskService takService = processEngine.getTaskService();

// 这里只指定任务id并没有指定任务负责人，只要调用complete，该任务就完成
// 所以说必须完成任务前校验
// 校验方法：
// 根据任务id和任务负责人assignee查询当前任务，如果查到说该用户有完成该任务权限，否则 没有权限

Task task = takService.createTaskQuery().taskId(taskId)
        .taskAssignee(assignee).singleResult();

if (task != null) {
    takService.complete(taskId);

    System.out.println("完成任务！！！");
}
```

# 第3章 流程变量

## 3.1 什么是流程变量

流程变量在 activiti 中是一个非常重要的角色，流程运转有时需要靠流程变量，业务系统和 activiti 结合时少不了流程变量，流程变量就是 activiti 在管理工作流时根据管理需要而设置的变量。

比如在请假流程流转时如果请假天数大于 3 天则由总经理审核，否则由人事直接审核，请假天数就可以设置为流程变量，在流程流转时使用。

**注意：虽然流程变量中可以存储业务数据可以通过 activiti 的 api 查询流程变量从而实现 查询业务数据，但是不建议这样使用，因为业务数据查询由业务系统负责，activiti 设置流程变量是为了流程**

执行需要而创建。

## 3.2 流程变量类型

Table 15.9. Variable Types

| Type name | Description |
|---|---|
| string | Value is threaded as a java.lang.String. Raw JSON-t |
| integer | Value is threaded as a java.lang.Integer. When writir text. |
| short | Value is threaded as a java.lang.Short. When writing, |
| long | Value is threaded as a java.lang.Long. When writing, |
| double | Value is threaded as a java.lang.Double. When writing text. |
| boolean | Value is threaded as a java.lang.Boolean. When writir |
| date | Value is treated as a java.util.Date. When writing, the |
| binary | Binary variable, threated as an array of bytes. The valu stream. |
| serializable | Serialized representation of a Serializable Java-object. URL pointing to the raw binary stream. All serializable variable of this type. |

注意：如果将 pojo 存储到流程变量中，必须实现序列化接口 serializable，为了防止由于新增字段无法反序列化，需要生成 serialVersionUID。

## 3.3 流程变量作用域

流程变量的作用域默认是一个流程实例(processInstance)，也可以是一个任务(task)或一个执行实例(execution)，这三个作用域流程实例的范围最大，可以**称为 global 变量**，任务和执行实例仅仅是针对一个任务和一个执行实例范围，范围没有流程实例大，**称为 local 变量**。

如下图：

一个流程实例

Global 变量：userId（变量名）：zhangsan（变量值）

变量名不允许重复

global 变量中变量名不允许重复，设置相同名称的变量，后设置的值会覆盖前设置的变量值。
Local 变量由于在不同的任务或不同的执行实例中，作用域互不影响，变量名可以相同没有影响。
Local 变量名也可以和 global 变量名相同，没有影响。

## 3.4 流程变量的使用方法

第一步：设置流程变量
第二步：通过 UEL 表达式使用流程变量

1> 可以在 assignee 处设置 UEL 表达式，表达式的值为任务的负责人
比如：**${assignee}**，assignee 就是一个流程变量名称

Activiti 获取 UEL 表达式的值 ，即流程变量 assignee 的值 ，将 assignee 的值作为任务的负责人进行任务分配

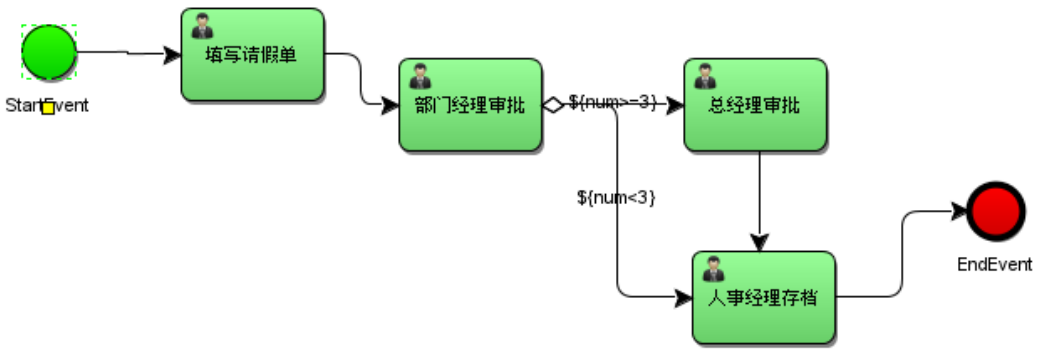
2> 可以在连线上设置 UEL 表达式，决定流程走向
比如：${price>=10000}和${price<10000}：price 就是一个流程变量名称，uel 表达式结果类型为布尔类型

如果 UEL 表达式是 true，要决定 流程执行走向。

## 3.5 使用 Global 变量控制流程

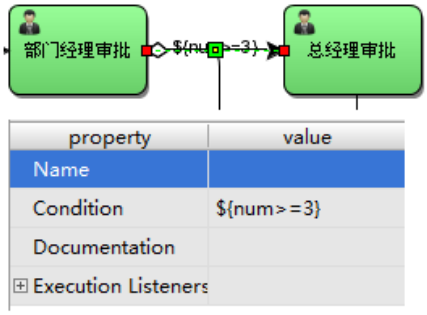
### 3.5.1 需求：

员工创建请假申请单，由部门经理审核，部门经理审核通过后请假 3 天及以下由人事经理直接审核，3 天以上先由总经理审核，总经理审核通过再由人事经理存档。

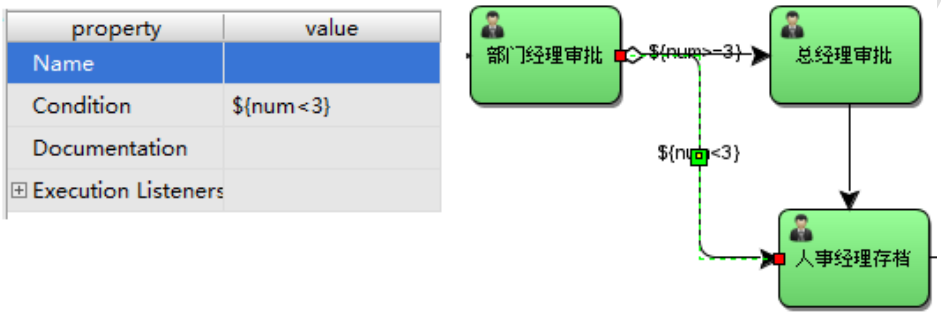


## 3.5.2 流程程定义

请假天数大于等于 3 连线条件：




请假天数小于 3 连线条件：




## 3.5.3 设置 global 流程变量

在部门经理审核前设置流程变量，变量值为请假单信息（包括请假天数），部门经理审核后可以根据流程变量的值决定流程走向。

### 3.5.3.1 启动流程时设置

在启动流程时设置流程变量，变量的作用域是整个流程实例。

通过 map<key,value>设置流程变量，map 中可以设置多个变量，这个 key 就是流程变量的名字。

```
// 启动流程时设置流程变量
    @Test
    public void startProcessInstance() {
        // 流程定义key
        String processDefinitionKey = "";
```

```
        Holiday holiday = new Holiday();
        holiday.setNum(3);
        // 定义流程变量
        Map<String, Object> variables = new HashMap<String, Object>();
        //变量名是num，变量值是holiday.getNum(),变量名也可以是一个对象
variables.put("num", holiday.getNum());

        RuntimeService runtimeService = processEngine.getRuntimeService();
        ProcessInstance processInstance = runtimeService
                .startProcessInstanceByKey(processDefinitionKey, variables);

        System.out.println("流程实例id:" +
processInstance.getProcessInstanceId());

    }
```

说明：
startProcessInstanceByKey(processDefinitionKey, variables)流程变量作用域是一个流程实例，流程变量使用 Map 存储，同一个流程实例设置变量 map 中 key 相同，后者覆盖前者。

### 3.5.3.2 任务办理时设置

在完成任务时设置流程变量，该流程变量只有在该任务完成后其它结点才可使用该变量，它的作用域是整个流程实例，如果设置的流程变量的 key 在流程实例中已存在相同的名字则后设置的变量替换前边设置的变量。

这里需要在创建请假单任务完成时设置流程变量

```
// 办理任务时设置流程变量
    @Test
    public void completTask() {

        //任务id
        String taskId = "";

        TaskService taskService = processEngine.getTaskService();
        Holiday holiday = new Holiday();
        holiday.setNum(4);
        // 定义流程变量
        Map<String, Object> variables = new HashMap<String, Object>();
        //变量名是holiday，变量值是holiday对象
        variables.put("holiday", holiday);
        taskService.complete(taskId, variables);
```

```
    }
```

说明：

通过当前任务设置流程变量，需要指定当前任务 id，如果当前执行的任务 id 不存在则抛出异常。

任务办理时也是通过 map<key,value>设置流程变量，一次可以设置多个变量。

#### 3.5.3.3 通过当前流程实例设置

通过流程实例 id 设置全局变量，该流程实例必须未执行完成。

```
public void setGlobalVariableByExecutionId(){

        //当前流程实例执行 id，通常设置为当前执行的流程实例
        String executionId="2601";

        RuntimeService runtimeService = processEngine.getRuntimeService();
        Holiday holiday = new Holiday();
        holiday.setNum(3);

        //通过流程实例 id设置流程变量
        runtimeService.setVariable(executionId, "holiday", holiday);

        //一次设置多个值
        //runtimeService.setVariables(executionId, variables)

    }
```

注意：

executionId 必须当前未结束 流程实例的执行 id，通常此 id 设置流程实例 的 id。

也可以通过 runtimeService.getVariable()获取流程变量

#### 3.5.3.4 通过当前任务设置

```
@Test
    public void setGlobalVariableByTaskId(){

        //当前待办任务id
        String taskId="1404";

        TaskService taskService = processEngine.getTaskService();
        Holiday holiday = new Holiday();
```

```
        holiday.setNum(3);
        //通过任务设置流程变量
        taskService.setVariable(taskId, "holiday", holiday);
        //一次设置多个值
        //taskService.setVariables(taskId, variables)
    }
```

注意：
任务id必须是当前待办任务id，act_ru_task中存在。如果该任务已结束，报错：

```
ROR [main] - Error while closing command context
g.activiti.engine.ActivitiObjectNotFoundException: Cannot find task with id 3803
        at org.activiti.engine.impl.cmd.NeedsActiveTaskCmd.execute(NeedsActiveTaskCmd.java:54)
        at org.activiti.engine.impl.interceptor.CommandInvoker.execute(CommandInvoker.java:24)
        at org.activiti.engine.impl.interceptor.CommandContextInterceptor.execute(CommandContextInterc
```

也可以通过 taskService.getVariable()获取流程变量。

## 3.5.4 测试

正常测试：
设置流程变量的值大于等于 3 天
设计流程变量的值小于 3 天
异常测试：
流程变量不存在
流程变量的值为空 NULL，price 属性为空
UEL 表达式都不符合条件
不设置连线的条件

## 3.5.5 注意事项

1、 如果 UEL 表达式中流程变量名不存在则报错。
2、 如果 UEL 表达式中流程变量值为空 NULL，流程不按 UEL 表达式去执行，而流程结束 。
3、 如果 UEL 表达式都不符合条件，流程结束
4、 如果连线不设置条件，会走 flow 序号小的那条线

操作数据库表

设置流程变量会在当前执行流程变量表插入记录，同时也会在历史流程变量表也插入记录。

SELECT * FROM act_ru_variable #当前流程变量表
记录当前运行流程实例可使用的流程变量，包括 global 和 local 变量
Id_：主键
Type_：变量类型
Name_：变量名称
Execution_id_：所属流程实例执行 id，global 和 local 变量都存储
Proc_inst_id_：所属流程实例 id，global 和 local 变量都存储
Task_id_：所属任务 id，local 变量存储
Bytearray_：serializable 类型变量存储对应 act_ge_bytearray 表的 id
Double_：double 类型变量值
Long_：long 类型变量值
Text_：text 类型变量值

SELECT * FROM act_hi_varinst #历史流程变量表
记录所有已创建的流程变量，包括 global 和 local 变量
字段意义参考当前流程变量表。

## 3.6 设置 local 流程变量

### 3.6.1 任务办理时设置

任务办理时设置 local 流程变量，当前运行的流程实例只能在该任务结束前使用，任务结束该变量无法在当前流程实例使用，可以通过查询历史任务查询。

```
// 办理任务时设置local流程变量
    @Test
    public void completTask() {
        //任务id
        String taskId = "";

        TaskService taskService = processEngine.getTaskService();
        // 定义流程变量
        Map<String, Object> variables = new HashMap<String, Object>();
        Holiday holiday = new Holiday ();
        holiday.setNum(3);
        // 定义流程变量
```

```
        Map<String, Object> variables = new HashMap<String, Object>();
        //变量名是holiday，变量值是holiday对象
        variables.put("holiday", holiday);
        // 设置local变量，作用域为该任务
        taskService.setVariablesLocal(tasked, variables);
        taskService.complete(taskId);

    }
```

说明：
设置作用域为任务的 local 变量，每个任务可以设置同名的变量，互不影响。

## 3.6.2 通过当前任务设置

```
@Test
    public void setLocalVariableByTaskId(){

        //当前待办任务id
        String taskId="1404";

        TaskService taskService = processEngine.getTaskService();
        Holiday holiday = new Holiday ();
        holiday.setNum(3);
        //通过任务设置流程变量
        taskService.setVariableLocal(taskId, "holiday", holiday);
        //一次设置多个值
        //taskService.setVariablesLocal(taskId, variables)
    }
```

注意：
任务 id 必须是当前待办任务 id，act_ru_task 中存在。

## 3.6.3 Local 变量测试 1

如果上边例子中设置 global 变量改为设置 local 变量是否可行？为什么？

Local 变量在任务结束后无法在当前流程实例执行中使用，如果后续的流程执行需要用到此变量则会报错。

### 3.6.4 Local 变量测试 2

在部门经理审核、总经理审核、人事经理审核时设置 local 变量，可通过 historyService 查询每个历史任务时将流程变量的值也查询出来。

代码如下：

```
// 创建历史任务查询对象
      HistoricTaskInstanceQuery historicTaskInstanceQuery = historyService
            .createHistoricTaskInstanceQuery();
      // 查询结果包括 local变量
      historicTaskInstanceQuery.includeTaskLocalVariables();

for (HistoricTaskInstance historicTaskInstance : list) {

         System.out.println("================================");
         System.out.println("  任  务  id  :  " + historicTaskInstance.getId());
         System.out.println("  任  务  名  称  :  " + historicTaskInstance.getName());
         System.out.println("  任  务  负  责  人  :  " + historicTaskInstance.getAssignee());
         System.out.println("  任  务  local  变  量  :  "+ historicTaskInstance.getTaskLocalVariables());
}
```

注意：查询历史流程变量，特别是查询 pojo 变量需要经过反序列化，不推荐使用。

# 第4章 组任务

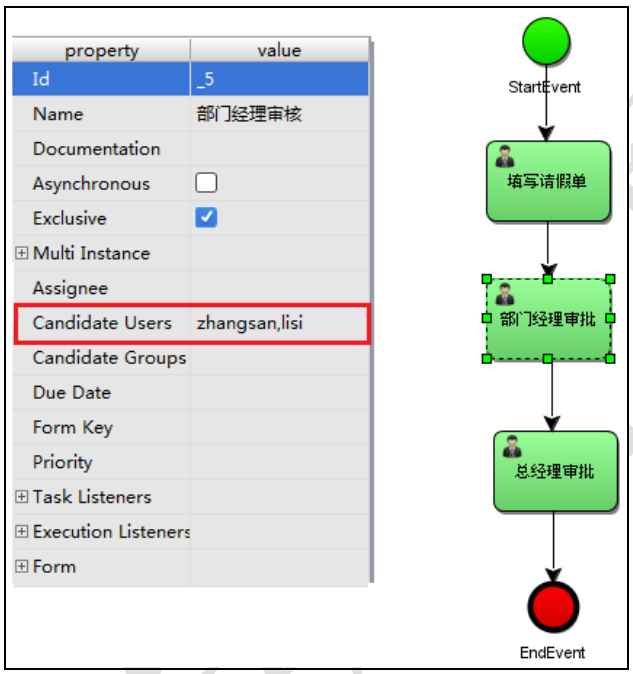
# 4.1 Candidate-users 候选人

## 4.1.1 需求

在流程定义中在任务结点的 assignee 固定设置任务负责人，在流程定义时将参与者固定设置在.bpmn 文件中，如果临时任务负责人变更则需要修改流程定义，系统可扩展性差。
针对这种情况可以给任务设置多个候选人，可以从候选人中选择参与者来完成任务。

### 设置任务候选人

在流程图中任务节点的配置中设置 candidate-users(候选人)，多个候选人之间用逗号分开。




查看 bpmn 文件：

```
<userTask activiti:assignee="xiaozhang" activiti:exclusive="true" id="_4" name="填写请假申请单"/>
<userTask activiti:candidateUsers="zhangsan,lishi" activiti:exclusive="true" id="_5" name="部门经理审核"/>
<userTask activiti:assignee="xiaowang" activiti:exclusive="true" id="_6" name="总经理审核"/>
```

我们可以看到部门经理的审核人已经设置为 zhangsan,lishi 这样的一组候选人，可以使用 activiti:candiateUsers=”用户 1,用户 2,用户 3”的这种方式来实现设置一组候选人。

# 办理组任务

## 4.1.1.1 组任务办理流程

第一步：查询组任务
指定候选人，查询该候选人当前的待办任务。
候选人不能办理任务。

第二步：拾取(claim)任务
该组任务的所有候选人都能拾取。
将候选人的组任务，变成个人任务。原来候选人就变成了该任务的负责人。
***如果拾取后不想办理该任务？
　需要将已经拾取的个人任务归还到组里边，将个人任务变成了组任务。

第三步：查询个人任务
查询方式同个人任务部分，根据 assignee 查询用户负责的个人任务。

第四步：办理个人任务

## 4.1.1.2 用户查询组任务

根据候选人查询组任务

```
@Test
    public void findGroupTaskList() {
        // 流程定义key
        String processDefinitionKey = "holiday4";
        // 任务候选人
        String candidateUser = "zhangsan";
        // 创建TaskService
        TaskService taskService = processEngine.getTaskService();
        //查询组任务
        List<Task> list = taskService.createTaskQuery()//
                .processDefinitionKey(processDefinitionKey)//
                .taskCandidateUser(candidateUser)//根据候选人查询
                .list();
```

```
        for (Task task : list) {
            System.out.println("----------------------------");
            System.out.println("流程实例id: " + task.getProcessInstanceId());
            System.out.println("任务id: " + task.getId());
            System.out.println("任务负责人: " + task.getAssignee());
            System.out.println("任务名称: " + task.getName());

        }
    }
```

#### 4.1.1.3 用户拾取组任务

候选人员拾取组任务后该任务变为自己的个人任务。

```
@Test
    public void claimTask(){
        TaskService taskService = processEngine.getTaskService();
        //要拾取的任务id
        String taskId = "6302";
        //任务候选人id
        String userId = "lisi";
        //拾取任务
        //即使该用户不是候选人也能拾取(建议拾取时校验是否有资格)

        //校验该用户有没有拾取任务的资格
        Task task = taskService.createTaskQuery()//
                .taskId(taskId)
                .taskCandidateUser(userId)//根据候选人查询
                .singleResult();
        if(task!=null){
            taskService.claim(taskId, userId);
            System.out.println("任务拾取成功");
        }


    }
```

说明：即使该用户不是候选人也能拾取，建议拾取时校验是否有资格
组任务拾取后，该任务已有负责人，通过候选人将查询不到该任务

#### 4.1.1.4 用户查询个人待办任务

查询方式同个人任务查询

```
@Test
    public void findPersonalTaskList() {
        // 流程定义key
        String processDefinitionKey = "holiday4";
        // 任务负责人
        String assignee = "zhangsan";
        // 创建TaskService
        TaskService taskService = processEngine.getTaskService();
        List<Task> list = taskService.createTaskQuery()//
                .processDefinitionKey(processDefinitionKey)//
                .taskAssignee(assignee).list();

        for (Task task : list) {
            System.out.println("----------------------------");
            System.out.println("流程实例id：" +
task.getProcessInstanceId());
            System.out.println("任务id：" + task.getId());
            System.out.println("任务负责人：" + task.getAssignee());
            System.out.println("任务名称：" + task.getName());
        }
    }
```

#### 4.1.1.5 用户办理个人任务

同个人任务办理

```
/**完成任务*/
    @Test
    public void completeTask(){
        //任务ID
        String taskId = "12304";
        processEngine.getTaskService()//
                    .complete(taskId);
        System.out.println("完成任务：" +taskId);
    }
```

说明：建议完成任务前校验该用户是否是该任务的负责人。

#### 4.1.1.6 归还组任务

如果个人不想办理该组任务，可以归还组任务，归还后该用户不再是该任务的负责人

```
// 归还组任务，由个人任务变为组任务，还可以进行任务交接
    @Test
    public void setAssigneeToGroupTask() {
        // 查询任务使用TaskService
        TaskService taskService = processEngine.getTaskService();
        // 当前待办任务
        String taskId = "6004";
        // 任务负责人
        String userId = "zhangsan2";

        // 校验userId是否是taskId的负责人，如果是负责人才可以归还组任务
        Task task = taskService.createTaskQuery().taskId(taskId)
                .taskAssignee(userId).singleResult();

        if (task != null) {
            // 如果设置为null，归还组任务,该 任务没有负责人
            taskService.setAssignee(taskId, null);

        }

    }
```

说明：建议归还任务前校验该用户是否是该任务的负责人
也可以通过 setAssignee 方法将任务委托给其它用户负责，注意被委托的用户可以不是候选人（建议不要这样使用）

#### 4.1.1.7 任务交接

任务交接,任务负责人将任务交给其它候选人办理该任务

```
@Test
    public void setAssigneeToCandidateUser() {
        // 查询任务使用TaskService
```

```
        TaskService taskService = processEngine.getTaskService();
        // 当前待办任务
        String taskId = "6004";
        // 任务负责人
        String userId = "zhangsan2";
// 将此任务交给其它候选人办理该 任务
            String candidateuser = "zhangsan";

        // 校验userId是否是taskId的负责人，如果是负责人才可以归还组任务
        Task task = taskService.createTaskQuery().taskId(taskId)
                .taskAssignee(userId).singleResult();

        if (task != null) {
            taskService.setAssignee(taskId, candidateuser);
        }
    }
```

#### 4.1.1.8 数据库表操作

SELECT * FROM act_ru_task #任务执行表，记录当前执行的任务，由于该任务当前是组任务，所有 assignee 为空，当拾取任务后该字段就是拾取用户的 id

| ID_ | REV_ | EXECUTION_ID_ | PROC_INST_ID_ | PROC_DEF_ID_ | NAME_ | PARENT_TASK_ID_ | DESCRIPTION_ | TASK_DEF_KEY_ | OWNER_ | ASSIGNEE_ |
|---|---|---|---|---|---|---|---|---|---|---|
| 704 | 3 | 701 | 701 | purchasingflow02:1:325 | 填写请假单 | (NULL) | (NULL) | createOrder | (NULL) | (NULL) |

SELECT * FROM act_ru_identitylink #任务参与者，记录当前参考任务用户或组，当前任务如果设置了候选人，会向该表插入候选人记录，有几个候选就插入几个

| ID_ | REV_ | GROUP_ID_ | TYPE_ | USER_ID_ | TASK_ID_ | PROC_INST_ID_ | PROC_DEF_ID_ |
|---|---|---|---|---|---|---|---|
| 705 | 1 | (NULL) | candidate | zhangsan | 704 | (NULL) | (NULL) |
| 707 | 1 | (NULL) | candidate | zhangsan2 | 704 | (NULL) | (NULL) |

于 act_ru_identitylink 对应的还有一张历史表 act_hi_identitylink，向 act_ru_identitylink 插入记录的同时也会向历史表插入记录。任务完成

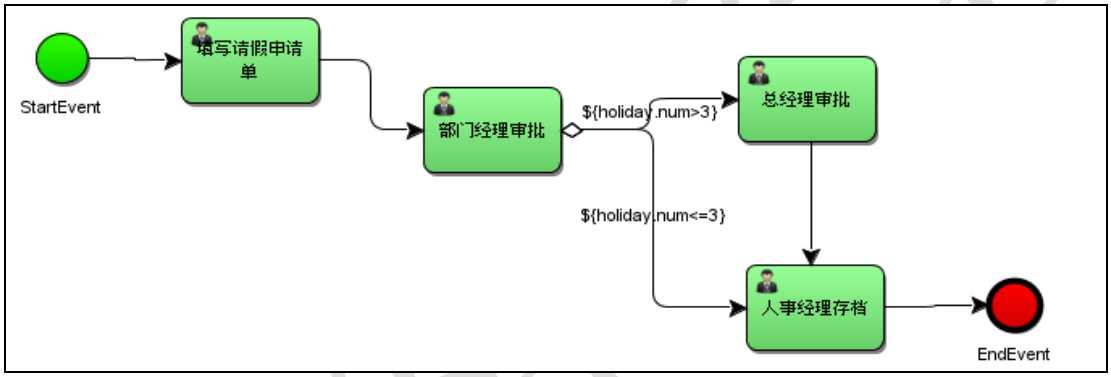
# 第5章 网关

## 5.1 排他网关

### 5.1.1 什么是排他网关：

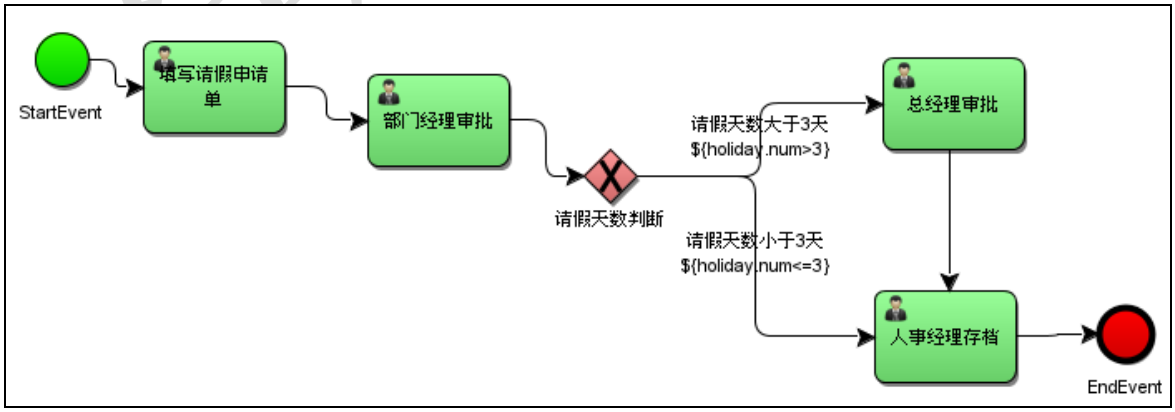
排他网关（也叫异或（XOR）网关，或叫基于数据的排他网关），用来在流程中实现决策。 当流程执行到这个网关，所有分支都会判断条件是否为 true，如果为 true 则执行该分支，
**注意，排他网关只会选择一个为 true 的分支执行**。(即使有两个分支条件都为 true，排他网关也会只选择一条分支去执行)

为什么要用排他网关？
不用排他网关也可以实现分支，如下图：




上图中，在连线的 condition 条件上设置分支条件。
缺点：
如果条件都不满足，不使用排他网关，流程就结束了(是异常结束)。

如果 使用排他网关决定分支的走向，如下：



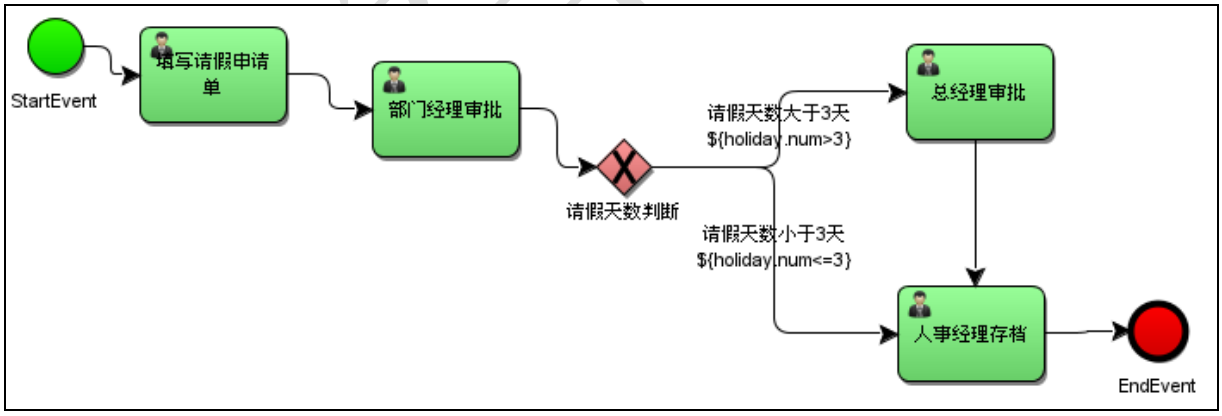
如果从网关出去的线所有条件都不满足则系统抛出异常。

```
org.activiti.engine.ActivitiException: No outgoing sequence flow of the exclusive gateway 'exclusivegateway1' could be selected for continuing the process
	at org.activiti.engine.impl.bpmn.behavior.ExclusiveGatewayActivityBehavior.leave(ExclusiveGatewayActivityBehavior.java:85)
```

说明 ：经过排他网关必须要有一条且只有一条分支走。

## 5.1.2 流程定义

图标：


## 5.1.3 测试

在部门经理审核后，走排他网关，从排他网关出来的分支有两条，一条是判断请假天数是否大于 3 天，另一条是判断请假天数是否小于等于 3 天。

设置分支条件时，如果所有分支条件都不是 true，报错：

org.activiti.engine.ActivitiException: No outgoing sequence flow of the exclusive gateway 'exclusivegateway1' could be selected for continuing the process

at org.activiti.engine.impl.bpmn.behavior.ExclusiveGatewayActivityBehavior.leave(ExclusiveGatewayActivityBehavior.java:85)

# 5.2 并行网关

## 5.2.1 什么是并行网关

并行网关允许将流程分成多条分支，也可以把多条分支汇聚到一起，并行网关的功能是基于进入和外出顺序流的：

- fork 分支：

并行后的所有外出顺序流，为每个顺序流都创建一个并发分支。

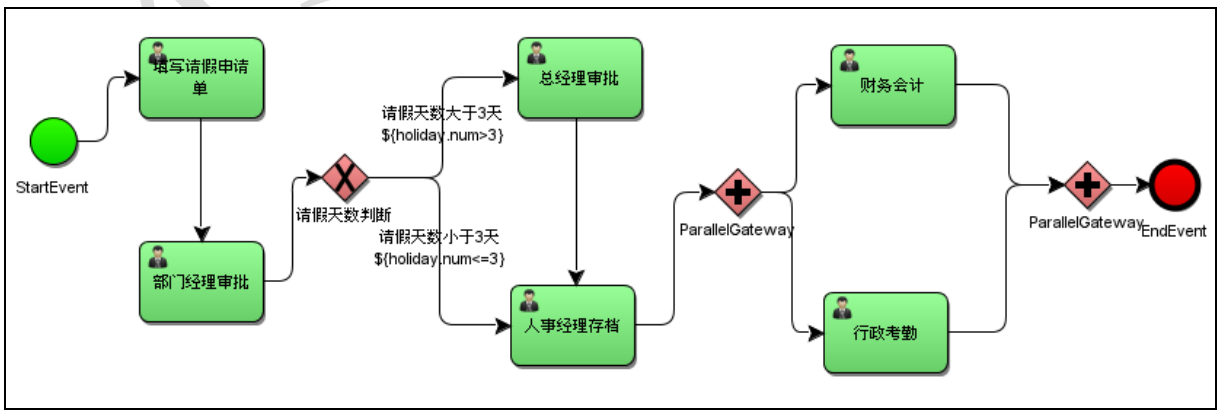
- join 汇聚：

所有到达并行网关，在此等待的进入分支， 直到所有进入顺序流的分支都到达以后， 流程就会通过汇聚网关。

注意，如果同一个并行网关有多个进入和多个外出顺序流， 它就同时具有分支和汇聚功能。 这时，网关会先汇聚所有进入的顺序流，然后再切分成多个并行分支。

**与其他网关的主要区别是，并行网关不会解析条件。 即使顺序流中定义了条件，也会被忽略。**

例子：

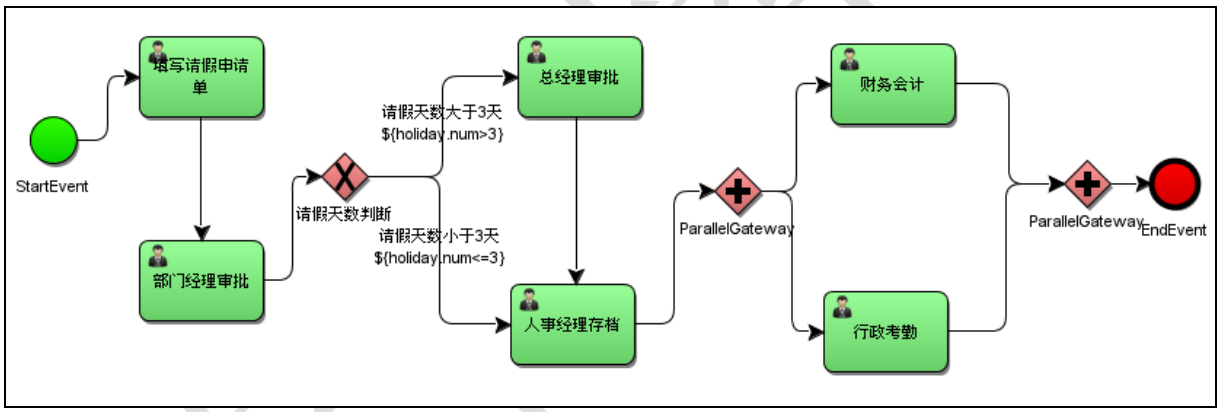



说明：

财务结算和入库是两个 execution 分支，在 act_ru_execution 表有两条记录分别是财务会计和行政考勤，act_ru_execution 还有一条记录表示该流程实例。
待财务会计和行政考勤任务全部完成，在汇聚点汇聚，通过 parallelGateway 并行网关。
并行网关在业务应用中常用于会签任务，会签任务即多个参与者共同办理的任务。

## 5.2.2 流程定义

图标：


## 5.2.3 测试

当执行到并行网关数据库跟踪如下：

当前任务表：SELECT * FROM act_ru_task #当前任务表

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 7305 | 1 | 7302 | 6501 | purchasingflow:12:6404 | 财务会计 | (NULL) | (NULL) | settlement |
| 7308 | 1 | 7303 | 6501 | purchasingflow:12:6404 | 行政考勤 | (NULL) | (NULL) | kaoqing |

上图中：有两个(多个)任务当前执行。

通过流程实例执行表：SELECT * FROM act_ru_execution #流程实例的执行表

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 6501 | 5 | 6501 | (NULL) | (NULL) | purchasingflow:12:6404 | (NULL) | parallelgateway1 |
| 7302 | 1 | 6501 | (NULL) | 6501 | purchasingflow:12:6404 | (NULL) | settlement |
| 7303 | 1 | 6501 | (NULL) | 6501 | purchasingflow:12:6404 | (NULL) | storage |

上图中，说明当前流程实例有多个分支(两个)在运行。

对并行任务的执行：
并行任务执行不分前后，由任务的负责人去执行即可。

当完成并任务中一个任务后：
已完成的任务在当前任务表 act_ru_task_已被删除。
在流程实例执行表：SELECT * FROM act_ru_execution 有中多个分支存在且有并行网关的汇聚结点。

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 6501 | 6 | 6501 | (NULL) | (NULL) | purchasingflow:12:6404 | (NULL) | parallelgateway1 |
| 7302 | 1 | 6501 | (NULL) | 6501 | purchasingflow:12:6404 | (NULL) | settlement |
| 7303 | 2 | 6501 | (NULL) | 6501 | purchasingflow:12:6404 | (NULL) | parallelgateway2 |


有并行网关的汇聚结点：说明有一个分支已经到汇聚，等待其它的分支到达。

当所有分支任务都完成，都到达汇聚结点后：
流程实例执行表：SELECT * FROM act_ru_execution，执行流程实例不存在，说明流程执行结束。

总结：所有分支到达汇聚结点，并行网关执行完成。

# 5.3 包含网关

## 5.3.1 什么是包含网关

包含网关可以看做是排他网关和并行网关的结合体。 和排他网关一样，你可以在外出顺序流上定义条件，包含网关会解析它们。 但是主要的区别是包含网关可以选择多于一条顺序流，这和并行网关一样。
包含网关的功能是基于进入和外出顺序流的：

- 分支：

所有外出顺序流的条件都会被解析，结果为 true 的顺序流会以并行方式继续执行， 会为每个顺序流

创建一个分支。

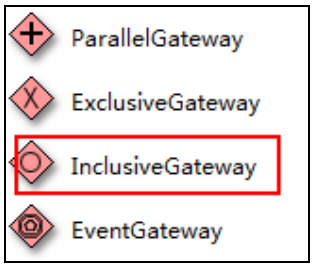
- 汇聚：

所有并行分支到达包含网关，会进入等待状态， 直到每个包含流程 token 的进入顺序流的分支都到达。 这是与并行网关的最大不同。换句话说，包含网关只会等待被选中执行了的进入顺序流。 在汇聚之后，流程会穿过包含网关继续执行。

## 5.3.2 流程定义：

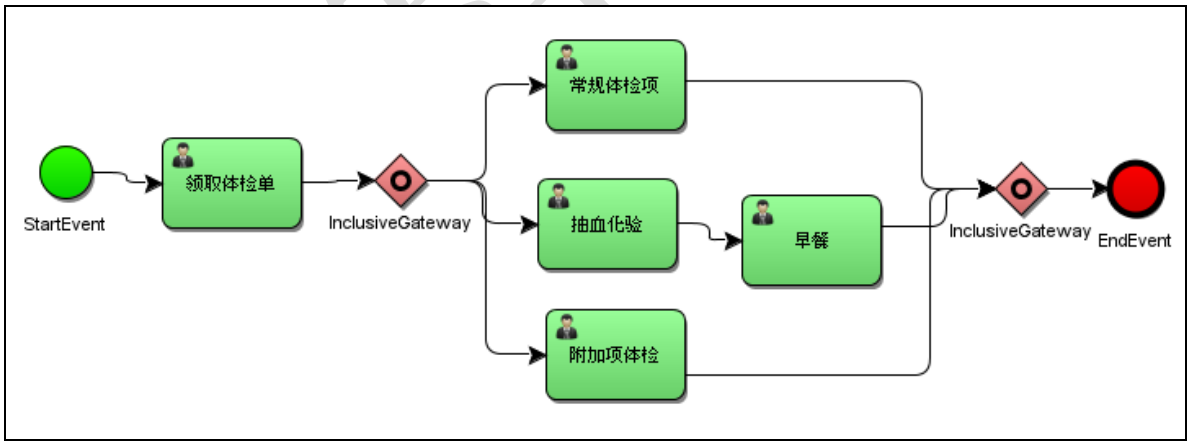
企业体检流程，公司全体员工进行常规项检查、抽血化验，公司管理层除常规检查和抽血化验还要进行增加项检查。

图标：




员工类型：

通过流程变量 userType 来表示，如果等于 1 表示普通员工，如果等于 2 表示领导




注意：通过包含网关的每个分支的连线上设置 condition 条件。

## 5.3.3 测试

如果包含网关设置的条件中，流程变量不存在，报错；
org.activiti.engine.ActivitiException: Unknown property used in expression: ${userType=='1' || userType=='2'}
需要在流程启动时设置流程变量 userType

当执行到包含网关：
流程实例执行表：SELECT * FROM act_ru_execution

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 7901 | 2 | 7901 | (NULL) | (NULL) | inclusiveGateway:1:7604 | (NULL) | inclusivegateway1 |
| 8002 | 1 | 7901 | (NULL) | 7901 | inclusiveGateway:1:7604 | (NULL) | usertask2 |
| 8003 | 1 | 7901 | (NULL) | 7901 | inclusiveGateway:1:7604 | (NULL) | usertask4 |

第一条记录：包含网关分支。
后两条记录：两个分支：常规项体检，抽血化验

当前任务表：ACT_RU_TASK_

| | | | | | |
|---|---|---|---|---|---|
| 8005 | 1 | 8002 | 7901 | inclusiveGateway:1:7604 | 常规项体检 |
| 8007 | 1 | 8003 | 7901 | inclusiveGateway:1:7604 | 抽血化验 |

上图中，常规项体检，抽血化验都是当前的任务，在并行执行。

如果有一个分支执行到汇聚：

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 901 | 3 | 7901 | (NULL) | (NULL) | inclusiveGateway:1:7604 | (NULL) | inclusivegateway1 |
| 002 | 1 | 7901 | (NULL) | 7901 | inclusiveGateway:1:7604 | (NULL) | usertask2 |
| 003 | 3 | 7901 | (NULL) | 7901 | inclusiveGateway:1:7604 | (NULL) | inclusivegateway2 |

包含网关汇聚结点

先走到汇聚结点的分支，要等待其它分支走到汇聚。
等所有分支走到汇聚，包含网关就执行完成。

包含网关执行完成，分支和汇聚就从 act_ru_execution 删除。

小结：在分支时，需要判断条件，**符合条件的分支，将会执行**，符合条件的分支最终才进行汇聚。

# 第6章 课程总结

什么是工作流？
就是通过计算机对业务流程进行自动化管理，实现多个参与者按照预定义的流程去自动执行业务流程。

什么 activiti？
Activiti 是一个工作流的引擎，开源的架构，基本 bpmn2.0 标准进行流程定义，它的是前身是 jbpm。
Activiti 通过是要嵌入到业务系统开发使用。

如何使用 activiti 开发？

- 第一步：部署 activiti 的环境。

环境包括：jar 包和数据库（25 张表）
业务系统通过 spring 和 activiti 整合进行开发。

- 第二步：使用 activiti 提供流程设计器（和 idea 或 eclipse 集成的 designer）工具进行流程定义

流程定义生成两个文件：.bpmn 和.png（不是必须的）。

- 第三步：将流程定义文件部署到 activiti 的数据库

SELECT * FROM act_re_deployment #流程定义部署表
一次部署插入一条记录，记录流程定义的部署信息

SELECT * FROM act_re_procdef #流程定义表
一次部署流程定义信息，如果一次部署两个流程定义，插入两条记录
建议：一次部署只部署一个流程定义，这样 act_re_deployment 和 act_re_procdef 一对一关系

常用两个方法：单个文件部署和 zip 文件部署。
建议单个文件部署。

- 第四步： 启动一个流程实例

业务系统就可以按照流程定义去执行业务流程，执行前需要启动一个流程实例
根据流程定义来启动一个流程实例。
指定一个流程定义的 key 启动一个流程实例，activiti 根据 key 找最新版本的流程定义。
指定一个流程定义的 id 启动一个流程实例。

启动一个实例需要指定 businesskey（业务标识），businessKey 是 activiti 和业务系统整合时桥梁。
比如：请假流程，businessKey 就是请假单 id。
启动一个实例还可以指定流程变量，流程变量是全局变量（生命期是整个流程实例，流程实例结束，变量就消失）

- 第五步：查询待办任务

查询个人任务：使用 taskService，根据 assignee 查询该用户当前的待办任务。
查询组任务：使用 taskService，根据 candidateuser 查询候选用户当前的待办组任务。

- 第六步：办理任务

办理个人任务：调用 taskService 的 complete 方法完成任务。
如果是组任务，需要先拾取任务，调用 taskService 的 claim 方法拾取任务，拾取任务之后组任务就变成了个人任务（该任务就有负责人）。

网关：
排他网关：任务执行之后的分支，经过排他网关分支只有一条有效。
并行网关：任务执行后，可以多条分支，多条分支总会汇聚，汇聚完成，并行网关结束。
包含网关：是排他网关和并行网关结合体。