Activity

# 概览

## 结构图



## Service

|  |  |
| --- | --- |
| **RepositoryService** | **管理流程定义** |
| **RuntimeService** | **执行管理，包括启动、推进、删除流程实例等操作** |
| **TaskService** | **任务管理** |
| **HistoryService** | **历史管理(执行完的数据的管理)** |
| **IdentityService** | **组织机构管理** |
| FormService | 一个可选服务，任务表单管理 |
| ManagerService |  |

## 数据库表格

|  |  |  |
| --- | --- | --- |
| ACT\_RE\_\* | repository | 包含了流程定义和流程静态资源 （图片，规则，等等） |
| ACT\_RU\_\* | runtime | 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,只在流程实例执行过程中保存数据，在流程结束时删除记录, 这样运行时表可以一直很小速度很快。 |
| ACT\_ID\_\* | identity | 包含身份信息，比如用户，组等 |
| ACT\_HI\_\* | history | 包含历史数据，比如历史流程实例， 变量，任务等 |
| ACT\_GE\_\* | general | 用于不同场景下，如存放资源文件 |

## Activiti数据表清单

|  |  |  |
| --- | --- | --- |
| 表分类 | 表名 | 解释 |
| 一般数据 | [ACT\_GE\_BYTEARRAY](file:///C:\\Users\\Administrator\\Desktop\\Activiti%E5%B7%A5%E4%BD%9C%E6%B5%81%E6%95%B0%E6%8D%AE%E5%BA%93%E8%A1%A8%E7%BB%93%E6%9E%84.docx" \l "LOCAL_1-ACT_GE_BYTEARRAY" \t "_blank) | 通用的流程定义和流程资源 |
|  | [ACT\_GE\_PROPERTY](file:///C:\\Users\\Administrator\\Desktop\\Activiti%E5%B7%A5%E4%BD%9C%E6%B5%81%E6%95%B0%E6%8D%AE%E5%BA%93%E8%A1%A8%E7%BB%93%E6%9E%84.docx" \l "LOCAL_1-ACT_GE_PROPERTY" \t "_blank) | 系统相关属性 |
| 流程历史记录 | [ACT\_HI\_ACTINST](file:///C:\\Users\\Administrator\\Desktop\\Activiti%E5%B7%A5%E4%BD%9C%E6%B5%81%E6%95%B0%E6%8D%AE%E5%BA%93%E8%A1%A8%E7%BB%93%E6%9E%84.docx" \l "LOCAL_1-ACT_HI_ACTINST" \t "_blank) | 历史的流程实例 |
|  | [ACT\_HI\_ATTACHMENT](file:///C:\\Users\\Administrator\\Desktop\\Activiti%E5%B7%A5%E4%BD%9C%E6%B5%81%E6%95%B0%E6%8D%AE%E5%BA%93%E8%A1%A8%E7%BB%93%E6%9E%84.docx" \l "LOCAL_1-ACT_HI_ATTACHMENT" \t "_blank) | 历史的流程附件 |
|  | [ACT\_HI\_COMMENT](file:///C:\\Users\\Administrator\\Desktop\\Activiti%E5%B7%A5%E4%BD%9C%E6%B5%81%E6%95%B0%E6%8D%AE%E5%BA%93%E8%A1%A8%E7%BB%93%E6%9E%84.docx" \l "LOCAL_1-ACT_HI_COMMENT" \t "_blank) | 历史的说明性信息 |
|  | [ACT\_HI\_DETAIL](file:///C:\\Users\\Administrator\\Desktop\\Activiti%E5%B7%A5%E4%BD%9C%E6%B5%81%E6%95%B0%E6%8D%AE%E5%BA%93%E8%A1%A8%E7%BB%93%E6%9E%84.docx" \l "LOCAL_1-ACT_HI_DETAIL" \t "_blank) | 历史的流程运行中的细节信息 |
|  | [ACT\_HI\_IDENTITYLINK](file:///C:\\Users\\Administrator\\Desktop\\Activiti%E5%B7%A5%E4%BD%9C%E6%B5%81%E6%95%B0%E6%8D%AE%E5%BA%93%E8%A1%A8%E7%BB%93%E6%9E%84.docx" \l "LOCAL_1-ACT_HI_IDENTITYLINK" \t "_blank) | 历史的流程运行过程中用户关系 |
|  | [ACT\_HI\_PROCINST](file:///C:\\Users\\Administrator\\Desktop\\Activiti%E5%B7%A5%E4%BD%9C%E6%B5%81%E6%95%B0%E6%8D%AE%E5%BA%93%E8%A1%A8%E7%BB%93%E6%9E%84.docx" \l "LOCAL_1-ACT_HI_PROCINST" \t "_blank) | 历史的流程实例 |
|  | [ACT\_HI\_TASKINST](file:///C:\\Users\\Administrator\\Desktop\\Activiti%E5%B7%A5%E4%BD%9C%E6%B5%81%E6%95%B0%E6%8D%AE%E5%BA%93%E8%A1%A8%E7%BB%93%E6%9E%84.docx" \l "LOCAL_1-ACT_HI_TASKINST" \t "_blank) | 历史的任务实例 |
|  | [ACT\_HI\_VARINST](file:///C:\\Users\\Administrator\\Desktop\\Activiti%E5%B7%A5%E4%BD%9C%E6%B5%81%E6%95%B0%E6%8D%AE%E5%BA%93%E8%A1%A8%E7%BB%93%E6%9E%84.docx" \l "LOCAL_1-ACT_HI_VARINST" \t "_blank) | 历史的流程运行中的变量信息 |
| 用户用户组表 | [ACT\_ID\_GROUP](file:///C:\\Users\\Administrator\\Desktop\\Activiti%E5%B7%A5%E4%BD%9C%E6%B5%81%E6%95%B0%E6%8D%AE%E5%BA%93%E8%A1%A8%E7%BB%93%E6%9E%84.docx" \l "LOCAL_1-ACT_ID_GROUP" \t "_blank) | 身份信息-组信息 |
|  | [ACT\_ID\_INFO](file:///C:\\Users\\Administrator\\Desktop\\Activiti%E5%B7%A5%E4%BD%9C%E6%B5%81%E6%95%B0%E6%8D%AE%E5%BA%93%E8%A1%A8%E7%BB%93%E6%9E%84.docx" \l "LOCAL_1-ACT_ID_INFO" \t "_blank) | 身份信息-组信息 |
|  | [ACT\_ID\_MEMBERSHIP](file:///C:\\Users\\Administrator\\Desktop\\Activiti%E5%B7%A5%E4%BD%9C%E6%B5%81%E6%95%B0%E6%8D%AE%E5%BA%93%E8%A1%A8%E7%BB%93%E6%9E%84.docx" \l "LOCAL_1-ACT_ID_MEMBERSHIP" \t "_blank) | 身份信息-用户和组关系的中间表 |
|  | [ACT\_ID\_USER](file:///C:\\Users\\Administrator\\Desktop\\Activiti%E5%B7%A5%E4%BD%9C%E6%B5%81%E6%95%B0%E6%8D%AE%E5%BA%93%E8%A1%A8%E7%BB%93%E6%9E%84.docx" \l "LOCAL_1-ACT_ID_USER" \t "_blank) | 身份信息-用户信息 |
| 流程定义表 | [ACT\_RE\_DEPLOYMENT](file:///C:\\Users\\Administrator\\Desktop\\Activiti%E5%B7%A5%E4%BD%9C%E6%B5%81%E6%95%B0%E6%8D%AE%E5%BA%93%E8%A1%A8%E7%BB%93%E6%9E%84.docx" \l "LOCAL_1-ACT_RE_DEPLOYMENT" \t "_blank) | 部署单元信息 |
|  | [ACT\_RE\_MODEL](file:///C:\\Users\\Administrator\\Desktop\\Activiti%E5%B7%A5%E4%BD%9C%E6%B5%81%E6%95%B0%E6%8D%AE%E5%BA%93%E8%A1%A8%E7%BB%93%E6%9E%84.docx" \l "LOCAL_1-ACT_RE_MODEL" \t "_blank) | 模型信息 |
|  | [ACT\_RE\_PROCDEF](file:///C:\\Users\\Administrator\\Desktop\\Activiti%E5%B7%A5%E4%BD%9C%E6%B5%81%E6%95%B0%E6%8D%AE%E5%BA%93%E8%A1%A8%E7%BB%93%E6%9E%84.docx" \l "LOCAL_1-ACT_RE_PROCDEF" \t "_blank) | 已部署的流程定义 |
| 运行实例表 | [ACT\_RU\_EVENT\_SUBSCR](file:///C:\\Users\\Administrator\\Desktop\\Activiti%E5%B7%A5%E4%BD%9C%E6%B5%81%E6%95%B0%E6%8D%AE%E5%BA%93%E8%A1%A8%E7%BB%93%E6%9E%84.docx" \l "LOCAL_1-ACT_RU_EVENT_SUBSCR" \t "_blank) | 运行时事件 |
|  | [ACT\_RU\_EXECUTION](file:///C:\\Users\\Administrator\\Desktop\\Activiti%E5%B7%A5%E4%BD%9C%E6%B5%81%E6%95%B0%E6%8D%AE%E5%BA%93%E8%A1%A8%E7%BB%93%E6%9E%84.docx" \l "LOCAL_1-ACT_RU_EXECUTION" \t "_blank) | 运行时流程执行实例 |
|  | [ACT\_RU\_IDENTITYLINK](file:///C:\\Users\\Administrator\\Desktop\\Activiti%E5%B7%A5%E4%BD%9C%E6%B5%81%E6%95%B0%E6%8D%AE%E5%BA%93%E8%A1%A8%E7%BB%93%E6%9E%84.docx" \l "LOCAL_1-ACT_RU_IDENTITYLINK" \t "_blank) | 运行时用户关系信息 |
|  | [ACT\_RU\_JOB](file:///C:\\Users\\Administrator\\Desktop\\Activiti%E5%B7%A5%E4%BD%9C%E6%B5%81%E6%95%B0%E6%8D%AE%E5%BA%93%E8%A1%A8%E7%BB%93%E6%9E%84.docx" \l "LOCAL_1-ACT_RU_JOB" \t "_blank) | 运行时作业 |
|  | [ACT\_RU\_TASK](file:///C:\\Users\\Administrator\\Desktop\\Activiti%E5%B7%A5%E4%BD%9C%E6%B5%81%E6%95%B0%E6%8D%AE%E5%BA%93%E8%A1%A8%E7%BB%93%E6%9E%84.docx" \l "LOCAL_1-ACT_RU_TASK" \t "_blank) | 运行时任务 |
|  | [ACT\_RU\_VARIABLE](file:///C:\\Users\\Administrator\\Desktop\\Activiti%E5%B7%A5%E4%BD%9C%E6%B5%81%E6%95%B0%E6%8D%AE%E5%BA%93%E8%A1%A8%E7%BB%93%E6%9E%84.docx" \l "LOCAL_1-ACT_RU_VARIABLE" \t "_blank) | 运行时变量表 |

## 核心API

### ProcessInstance

代表流程定义的执行实例。流程实例就表示一个流程从开始到结束的最大的流程分支，即一个流程中流程实例只有一个。

### Execution

执行对象Execution,流程按照流程定义的规则执行一次的过程;

* 并发执行的流程, 总线路代表ProcessInstance，而分线路中每个活动代表Execution
* 一个流程中，执行对象并发执行的流程, 总线路代表ProcessInstance，而分线路中每个活动代表Execution
* 可以存在多个，但是流程实例ProcessInstance只能有一个

# Method & table

## DeploymentBuilder.deploy()

Deployment deployment = processEngine.getRepositoryService()//

.createDeployment()//

.addClasspathResource("diagrams/LeaveBill.bpmn")//

.addClasspathResource("diagrams/LeaveBill.png")//

.deploy();

这一步在数据库中将操作三张表：

* 1. act\_re\_deployment（部署对象表）

存放流程定义的显示名和部署时间，每部署一次增加一条记录

* 1. act\_re\_procdef（流程定义表）

存放流程定义的属性信息，部署每个新的流程定义都会在这张表中增加一条记录。

注意：当流程定义的key相同的情况下，使用的是版本升级

* 1. act\_ge\_bytearray（资源文件表）

存储流程定义相关的部署信息。即流程定义文档的存放地。每部署一次就会增加两条记录，一条是关于bpmn规则文件的，一条是图片的（如果部署时只指定了bpmn一个文件，activiti会在部署时解析bpmn文件内容自动生成流程图）。两个文件不是很大，都是以二进制形式存储在数据库中。

* ACT\_RE\_PROCDEF

流程解析表，解析成功了，在该表保存一条记录。业务流程定义数据表

|  |  |  |
| --- | --- | --- |
| ACT\_RE\_PROCDEF(act\_re\_procdef) | | |
| NAME | **字段名** | **字段描述** |
| ID\_ | 流程ID | {processDefinitionKey}:  {processDefinitionVersion}:  {generated-id} |
| REV\_ | 版本号 |  |
| CATEGORY\_ | 流程命名空间 | 流程定义的Namespace就是类别 |
| NAME\_ | 流程名称 | 该编号就是流程文件process元素的name属性值 |
| KEY\_ | 流程定义ID | 该编号就是流程文件process元素的id属性值 |
| VERSION\_ | 流程版本号 | 由程序控制，新增即为1，修改后依次加1来完成的 |
| DEPLOYMENT\_ID\_ | 部署编号 | 部署表ID |
| RESOURCE\_NAME\_ | 流程bpmn文件名称 |  |
| DGRM\_RESOURCE\_NAME\_ | png流程图片名称 |  |
| DESCRIPTION\_ | 描述信息 | 描述 |
| HAS\_START\_FORM\_KEY\_ | start节点是否存在formKey | 0否  1是 |
|
| SUSPENSION\_STATE\_ | 是否挂起 | 1激活 2挂起 |

此表和ACT\_RE\_DEPLOYMENT是多对一的关系，即一个部署的bar包里可能包含多个流程定义文件，每个流程定义文件都会有一条记录在ACT\_RE\_PROCDEF表内，每个流程定义的数据，都会对于ACT\_GE\_BYTEARRAY表内的一个资源文件和PNG图片文件。和ACT\_GE\_BYTEARRAY的关联是通过程序用ACT\_GE\_BYTEARRAY.NAME与ACT\_RE\_PROCDEF.NAME\_完成的, 在数据库表结构中没有体现

* ACT\_RE\_DEPLOYMENT

用来存储部署时需要持久化保存下来的信息

|  |  |  |
| --- | --- | --- |
| 是否主键 | 字段名 | 字段描述 |
| ID\_ |  | 部署编号，自增长 |
| NAME\_ | 部署包的名称 |  |
| CATEGORY\_ | 类型 | VARCHAR(255) |
| DEPLOY\_TIME\_ | 部署时间 | TIMESTAMP |

* ACT\_GE\_BYTEARRAY

保存所有二进制数据

LeaveBill.bpmn和LeaveBill.png都是ResourceEntity,被添加入表中

|  |  |  |
| --- | --- | --- |
| ACT\_GE\_BYTEARRAY(act\_ge\_bytearray) | | |
| Name | **字段名** | **字段描述** |
| ID\_ | ID\_ | 主键ID，资源文件编号，自增长 |
| REV\_ | 版本号 | INT(11) |
| NAME\_ | 部署的文件名称， |  |
| DEPLOYMENT\_ID\_ | 来自于父表ACT\_RE\_DEPLOYMENT的主键 |  |
| BYTES\_ | 大文本类型，存储文本字节流 | LONGBLOB |
| GENERATED\_ | 是否是引擎生成 | 0为用户生成,1为Activiti生成 |

## RuntimeService().startProcessInstanceByKey()

当流程到达一个节点时,会在ACT\_RU\_EXECUTION表中产生1条数据,如果当前节点是用户任务节点,这是会在ACT\_RU\_TASK表中添加一条记录

* ACT\_RU\_TASK(运行时任务数据表)

（执行中实时任务）代办任务查询表

|  |  |  |
| --- | --- | --- |
| ACT\_RU\_TASK(act\_ru\_task) | | |
| 是否主键 | **字段名** | **字段描述** |
| ID\_ |  |  |
| REV\_ | 版本号 | INT(11) |
| EXECUTION\_ID\_ | 实例id（外键EXECUTION\_ID\_） | 执行实例ID |
| PROC\_INST\_ID\_ | 流程实例ID（外键PROC\_INST\_ID\_） | VARCHAR(64) |
| PROC\_DEF\_ID\_ | 流程定义ID | VARCHAR(64) |
| NAME\_ | 任务名称 | 节点定义名称 |
| PARENT\_TASK\_ID\_ | 父节任务ID | 父节点实例ID |
| DESCRIPTION\_ | 任务描述 | 节点定义描述 |
| TASK\_DEF\_KEY\_ | 任务定义key | 任务定义的ID |
| OWNER\_ | 所属人(老版) | 拥有者（一般情况下为空，只有在委托时才有值） |
| ASSIGNEE\_ | 代理人员(受让人) | 签收人或委托人 |
| DELEGATION\_ | 代理团 | 委托类型，DelegationState分为两种：PENDING，RESOLVED。如无委托则为空VARCHAR(64) |
| PRIORITY\_ | 优先权 | INT(11) |
| CREATE\_TIME\_ | 创建时间 | TIMESTAMP |
| DUE\_DATE\_ | 执行时间 | 耗时 |
| SUSPENSION\_STATE\_ | 暂停状态 | 1代表激活 2代表挂起 |

* ACT\_RU\_EXECUTION（运行时流程执行实例）

核心，我的待办任务查询表

|  |  |  |
| --- | --- | --- |
| ACT\_RU\_EXECUTION(act\_ru\_execution) | | |
| 是否主键 | **字段名** | **字段描述** |
| ID\_ | ID\_ |  |
| REV\_ | 版本号 | INT(11) |
| PROC\_INST\_ID\_ | 流程实例编号 | VARCHAR(64) |
| BUSINESS\_KEY\_ | 业务主键ID | VARCHAR(255) |
| PARENT\_ID\_ | 父执行流程 | 父节点实例ID |
| PROC\_DEF\_ID\_ | 流程定义Id | VARCHAR(64) |
| SUPER\_EXEC\_ | VARCHAR(64) | 64 |
| ACT\_ID\_ | 实例id | 节点实例ID即ACT\_HI\_ACTINST中ID |
| IS\_ACTIVE\_ | 激活状态 | 是否存活 |
| IS\_CONCURRENT\_ | 并发状态 | 是否为并行(true/false） |
| IS\_SCOPE\_ |  | TINYINT(4) |
| IS\_EVENT\_SCOPE\_ |  | TINYINT(4) |
| SUSPENSION\_STATE\_ | 暂停状态\_ | 挂起状态   1激活 2挂起 |
| CACHED\_ENT\_STATE\_ | 缓存结束状态\_ | INT(11) |

* ACT\_RU\_IDENTITYLINK（身份联系）

主要存储当前节点参与者的信息,任务参与者数据表。

|  |  |  |
| --- | --- | --- |
| ACT\_RU\_IDENTITYLINK(act\_ru\_identitylink) | | |
| 是否主键 | **字段名** | **字段描述** |
| ID\_ | ID\_ |  |
| REV\_ | 版本号 | INT(11) |
| GROUP\_ID\_ | 用户组ID | VARCHAR(255) |
| TYPE\_ | 用户组类型 | 主要分为以下几种：assignee、candidate、owner、starter、participant。即：受让人,候选人,所有者、起动器、参与者 |
| USER\_ID\_ | 用户ID | VARCHAR(255) |
| TASK\_ID\_ | 任务Id | VARCHAR(64) |
| PROC\_INST\_ID\_ | 流程实例ID | VARCHAR(64) |
| PROC\_DEF\_ID\_ | 流程定义Id | VARCHAR(64) |

* ACT\_HI\_ACTINST（历史节点表）

历史活动信息。这里记录流程流转过的所有节点，与HI\_TASKINST不同的是，taskinst只记录usertask内容

|  |  |  |
| --- | --- | --- |
| ACT\_HI\_ACTINST(act\_hi\_actinst) | | |
| 是否主键 | **字段名** | **字段描述** |
| ID\_ | ID\_ |  |
| PROC\_DEF\_ID\_ | 流程定义ID | VARCHAR(64) |
| PROC\_INST\_ID\_ | 流程实例ID | VARCHAR(64) |
| EXECUTION\_ID\_ | 流程执行ID | VARCHAR(64) |
| ACT\_ID\_ | 活动ID | 节点定义ID |
| TASK\_ID\_ | 任务ID | 任务实例ID 其他节点类型实例ID在这里为空 |
| CALL\_PROC\_INST\_ID\_ | 请求流程实例ID | 调用外部流程的流程实例ID' |
| ACT\_NAME\_ | 活动名称 | 节点定义名称 |
| ACT\_TYPE\_ | 活动类型 | 如startEvent、userTask |
| ASSIGNEE\_ | 代理人员 | 节点签收人 |
| START\_TIME\_ | 开始时间 | DATETIME |
| END\_TIME\_ | 结束时间 | DATETIME |
| DURATION\_ | 时长，耗时 | 毫秒值 |

# 教程

## setVariable和setVariableLocal的区别

* setVariable

设置流程变量的时候，流程变量名称相同的时候，后一次的值替换前一次的值，且TASK\_ID的字段不会存放任务ID的值

* setVariableLocal

设置流程变量的时候，针对当前活动的节点设置流程变量，如果一个流程中存在2个活动节点(分支)，对每个活动节点都设置流程变量，即使流程变量的名称相同，后一次的版本的值也不会替换前一次版本的值，它会使用不同的任务ID作为标识，存放2个流程变量值，而且可以看到TASK\_ID的字段会存放任务ID的值

## 排他网关

只会返回一条结果。当流程执行到排他网关时，流程引擎会自动检索网关出口，从上到下检索如果发现第一条决策结果为true或者没有设置条件的(默认为成立)，则流出。

使用流程变量，设置连线的条件，并按照连线的条件执行工作流，如果没有条件符合的条件，则以默认的连线离开.需要设置流程变量,不设置时抛异常,不会走,默认网关

## 并行网关

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。

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

## 接收活动receiveTask

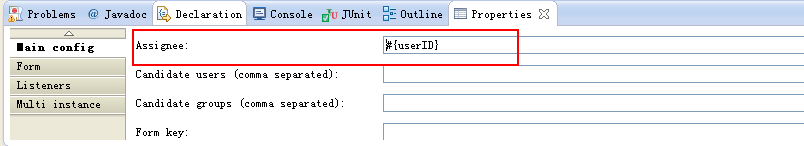
在任务创建后，意味着流程会进入等待状态， 直到引擎接收了一个特定的消息， 这会触发流程穿过接收任务继续执行.

ReceiceTask任务，机器自动完成的任务,只会在act\_ru\_execution表中产生一条数据

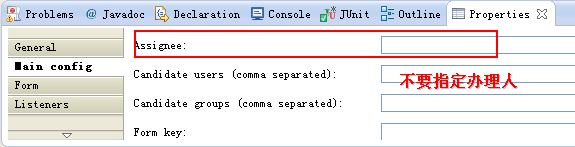
## 个人任务

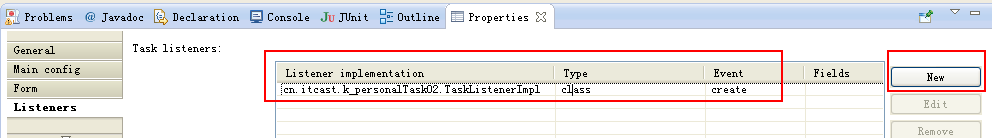
设置任务办理人

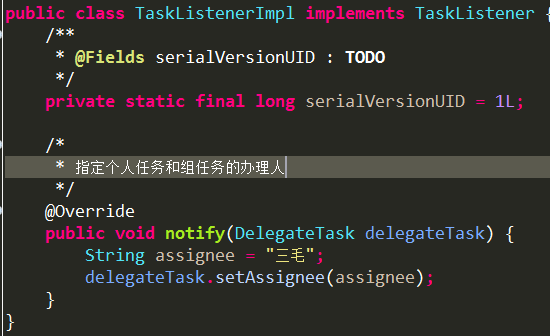
### 使用流程变量



### 使用类



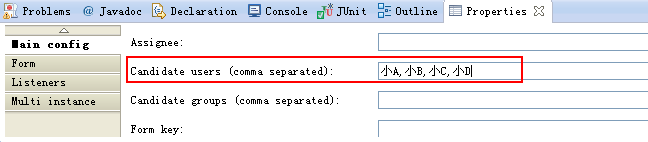




通过processEngine.getTaskService().setAssignee(taskId, userId);将个人任务从一个人分配给另一个人,会覆盖监听器设置的代办人

## 组任务

* 直接指定办理人



查询组任务成员列表

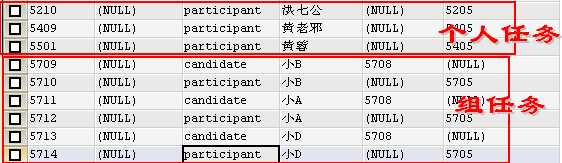
List<IdentityLink> list = processEngine.getTaskService().getIdentityLinksForTask(taskId)

act\_ru\_identitylink表存放任务的办理人，包括个人任务和组任务，表示正在执行的任务

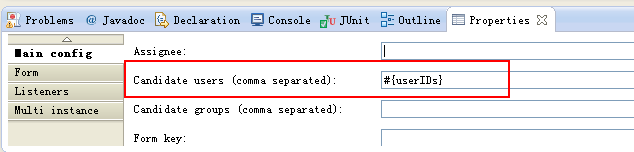
act\_hi\_identitylink表存放任务的办理人，包括个人任务和组任务，表示历史任务

区别在于：如果是个人任务TYPE的类型表示participant（参与者）

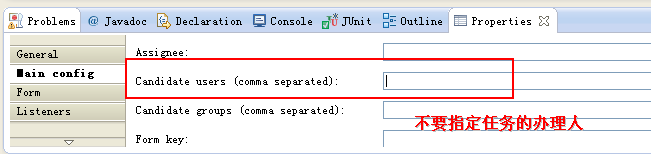
如果是组任务TYPE的类型表示candidate（候选者）和participant（参与者）

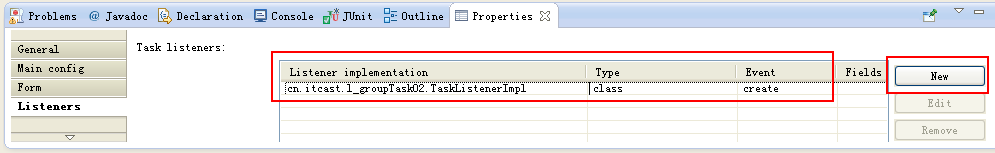


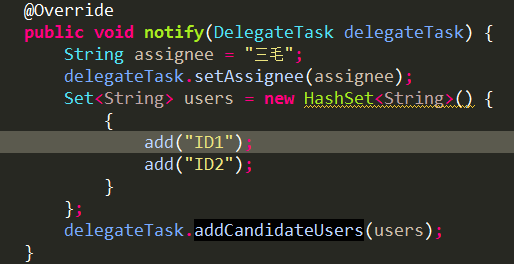
* 使用流程变量



* 类







通过processEngine.getTaskService().claim (taskId, userId);将组任务分配给个人任务，也叫认领任务，即指定某个人去办理这个任务.此时不会校验userId是否在候选人中,如果不在,则将userId加入到候选人列表,对应IdentityLink表格

领任务的时候，可以是组任务成员中的人，也可以不是组任务成员的人，此时通过Type的类型为participant来指定任务的办理人

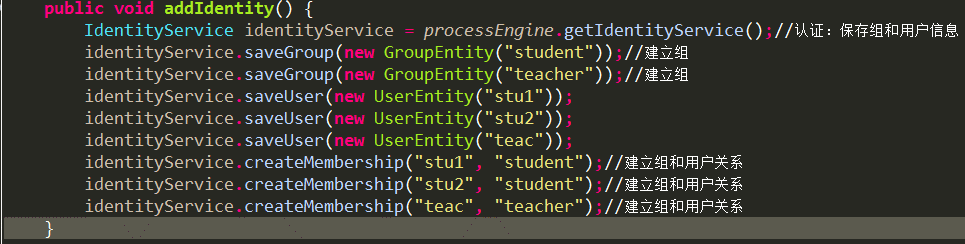
## 角色组

act\_id\_group：角色组表

act\_id\_user：用户表：

act\_id\_membership：用户角色表

在部署流程定义和启动流程实例的中间，设置组任务的办理人，向Activiti表中存放组和用户的信息



# Activity实战

## 任务

### 用户任务

* 任务监听

Create 创建任务,assignment分配任务,complete完成任务

Expression,delegateExpression,Alfresco script

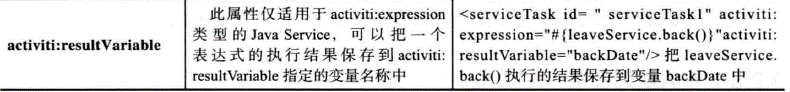
activity:initiator=”xxx”:可以把启动流程实例的操作人以变量名称”xxx”保存到数据库中,需要配合identityService.setAuthenticatedUserId(authenticatedUserId)使用

### 脚本任务

可以运行脚本语言

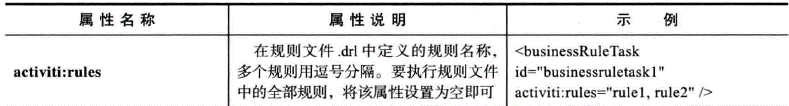
### Service 任务

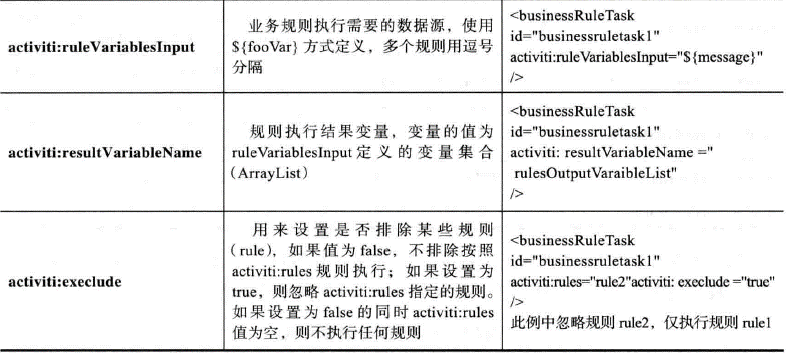




### 业务规则任务

可以和drools整合





### 邮件任务

在serviceTask的基础上由activiti扩展而来,可以发邮件

### Camel

在serviceTask的基础上由activiti扩展而来,是用来解决消息路由的框架

### Manual task

不做任何处理,流程引擎无需关系如何处理它,activity把手动任务当做一个空任务来处理,当到达此任务时由引擎自动完成并转向下一个任务

### Receive task

在任务创建后开始等待消息的到来,直到被触发才会完成任务,可以通过RuntimeService接口的signal()方法发送信号触发接收任务

### 多实例

允许一个任务甚至子流程可以重复执行多次如一个申请由多人审批,可以选择顺序执行和并行执行

可以设置重复执行,设置完成条件

## 网关gateway

### 排他网关

如果多个线路的计算结果为true,则会执行第一个值为true的网关,若线路都没有true,则抛出异常

### 并行网关

并行网关汇集处,只有当所有进入的流程都进来了,才能往下走流程

### 包容网关inclusivegateway

融合了排他网关和并行网关的特性,既可以同时执行多条线路,又允许在网关上设置条件

### 事件网关?

事件网关是专门为中间捕获事件设置,允许设置多个输出流指向多个不同的中间捕获事件(最少两个).在流程执行到事件网关后,流程处于”等待”状态,中间捕获事件需要依赖中间抛出事件触发才能更改等待状态为活动状态

## 子流程与调用活动

### 子流程

子流程可以包含流程规范的大部分模型

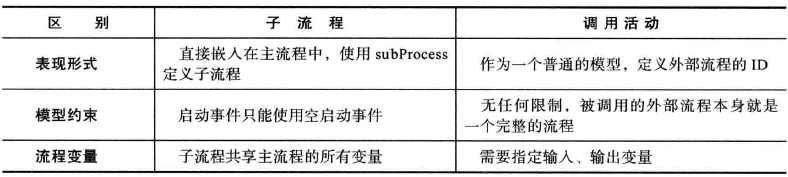
最常用的是把最通用的流程独立成子流程,嵌入到各个流程当中

* 约束

1. 只能且仅能包含一个空启动事件
2. 至少要有一个结束事件
3. 在子流程中顺序流不能直接设置输出流到子流程之外的活动上,可以通过边界事件代替

### 调用活动 call activity

和子流程作用一致,但表现方式不同,可以使用一个调用活动取代嵌入子流程的活动,通过创建一个调用活动模型并制定外部流程的id方式作为主流程的一个子活动



### 事件子流程

和子流程类似,不同的是事件子流程不能直接启动,而是被动地由其他的事件触发启动

可以由异常事件,信号事件,消息事件,定时器事件,补偿事件等触发

### 事务子流程

该子流程拥有事务性,ACID,回滚

## 边界与中间事件

边界事件是绑定在活动上的”捕获型”事件,一旦触发边界事件,当前活动会被中断然后按照边界事件之后的输出流执行

一个活动只能绑定一个边界事件

部分边界事件中可以设置cancelActivity属性值,以控制是否取消执行输出流指定的活动

### 定时器边界事件

定时启动事件:用于在指定的时间启动一个新的流程

定时器边界事件:附属在一个非自动任务,调用活动,子流程上,在上游任务执行完成之后开始倒计时准备触发事件

### 异常边界事件

用来捕获嵌入子流程或调用活动抛出的异常

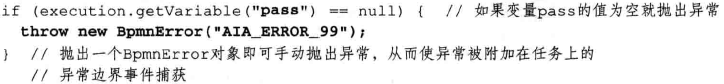
异常抛出之后被主流程的异常边界事件捕获,同时嵌入子流程或调用活动中的活动也被中断执行

异常边界事件可以直接在流程定义图

<sequenceFlow id="flow-treasurerAudit" name="财务不同意" sourceRef="exclusivegateway-treasurerAudit" targetRef="errorendevent2">

也可以直接抛出异常





### 信号边界事件

信号边界事件可以捕获流程执行中抛出的信号,可以附加在各种活动和子流程上

信号抛出事件是全局的:信号边界事件不仅可以捕获本流程的信号,还可以捕获到其他流程的信号事件,如果定义了多个信号边界事件并监听同一个信号,则会同时触发多个边界事件

### 取消边界事件

专门针对事务子流程所设计的,用来捕获子流程中抛出的取消事件,不能附加到其他活动上

1. 一个事务主流程只允许附加一个取消边界事件
2. 如果事务子流程中嵌套了子流程,仅仅触发已经完成了的子流程补偿事件?
3. 对于多实例的事务子流程,若其中一个实例触发取消事件,则全部实例都会被触发取消边界事件

### 补偿边界事件

用于事务子流程中针对事务失败后的业务逻辑进行补偿

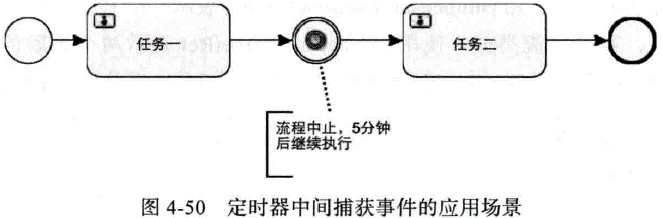
若补偿边界事件附加的活动室多实例的,当抛出补偿事件时,每一个实例都会触发补偿边界事件

### 中间捕获事件

根据事件不同的类型需要使用不同的方式才能继续执行后续的输出流的活动

中间捕获事件必须连接一个输入流和输出流

* 定时器中间捕获事件



* 信号中间捕获事件

用来捕获抛出的信号事件,信号id一致即可捕获

广播式传播

* 消息中间捕获事件

定向一对一传播,一次只能把一个消息发给一个指定的流程实例

### 中间抛出事件

中间抛出事件需要有对应的捕获事件接收才有意义

中间抛出事件一般用在一个任务完成后需要发送通知或执行其他系统任务的场景,工作流引擎会对抛出的事件进行传播

* 空中间抛出事件

不执行流程功能,可以借助activiti对大多数活动/事件添加的扩展功能使其更有意义,可以作为中间状态/结果的处理器

* 信号中间抛出事件

可以同步也可以异步

同步时信号发出后等待信号被捕获处理完成后才能继续往下执行流程,中间出错则回滚

异步时发出信号后立即执行下一流程, 不会回滚

## 监听器

### 执行监听器

允许在执行流程中执行java代码

监听类型:start,end,take

可以捕获的事件:

1. 流程实例启动,结束
2. 输出流捕获
3. 活动启动,结束
4. 路由开始,结束
5. 中间事件开始,结束
6. 触发开始事件,触发结束事件

3种监听器执行类型

1. Class:实现接口
2. Expression:表达式,类似EL
3. DelegateExpression:可以以参数形式设置实现的的接口类

### 任务监听器

只能应用于用户任务

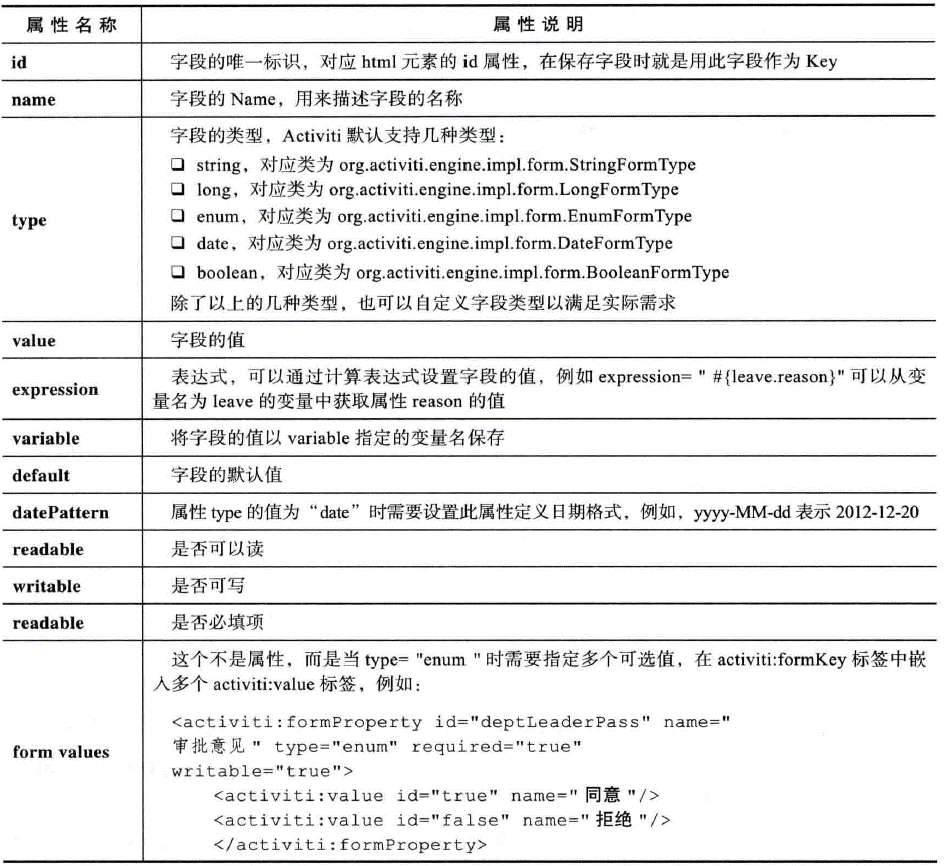
监听类型

1. Assignment:在任务被分配给某个办理人之后触发,在create之前
2. Create:需要逐一处理任务的办理人,候选人,候选组属性
3. complete:任务完成,运行时运行数据被删除之前

顺序: assignment-> create->complete

## 任务表单

表单属性

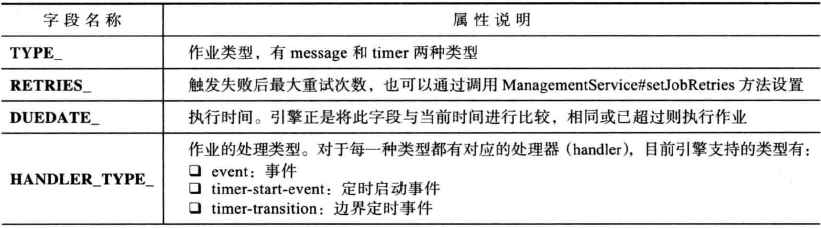


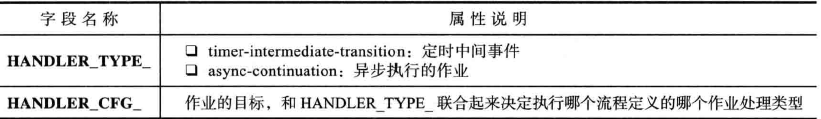
流程变量是整个流程实例公用的,即使是表单上要求必填的属性,在流程中已有值时就可以不填

# 定时作业

引擎在部署流程之后,引擎对部署的流程定义做一些初始化的工作,其中就包含对定时作业的注册和对消息事件的注册

ACT\_RU\_JOB表的主要字段说明



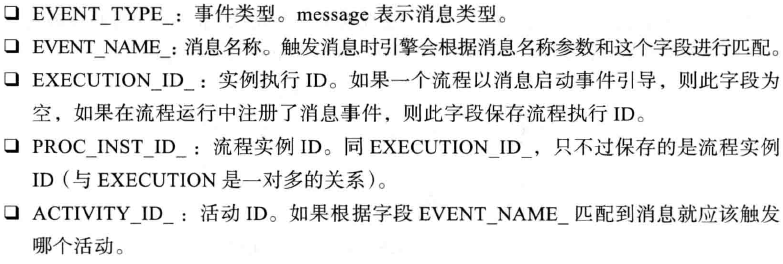


定时启动事件可以按照预设时间启动是因为引擎不断刷新数据库表ACT\_RU\_JOB的记录,根据时间匹配作业,命中之后就执行作业.

# 消息启动事件

在部署流程之后引擎会在初始化中处理消息事件,把消息的类型注册到数据act\_ru\_event\_subscr中

在流程执行过程中遇到了消息类型事件或通过API触发消息事件触发该表读取数据,并且根据消息的属性调用消息处理器





在调用代码

processInstance = runtimeService.startProcessInstanceByMessage("启动XXX流程");

流程会找到act\_ru\_event\_subscr表中的EVENT\_NAME匹配消息名称,找到就启动

# 结束事件

终止结束事件:终止整个流程

空结束事件:只结束一条输出流的执行

在历史任务实例中HistoricActivityInstance,有个HistoricActivityInstance.getDeleteReason(),对应act\_hi\_taskinst表的DELETE\_REASON\_字段

如果一个任务正常完成(通过complete方法),则DELETE\_REASON\_值为completed,如果任务是被删除的,则删除原因为deleted

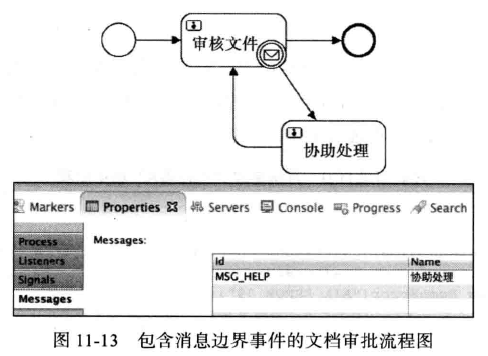
比如在空结束事件没有complete,而主流程就completed了,则子流程的记录就为deleted

# 消息边界事件

消息边界事件,信号边界事件与异常边界事件会多出一个cancelActivity属性,可取值为:

True:在边界事件触发后取消已注册的消息事件

False:在边界事件触发后仍然保留已注册的事件,可以再次触发





runtimeService.messageEventReceived(messageName, executionId); //触发消息边界事件

同理,触发信号事件runtimeService.signalEventReceived(signalName, executionId);

如果一个任务有附加的消息边界事件,当任务执行完成且边界事件没有被触发时,那么已经注册的事件将在任务完成时被删除

# 任务

## 任务委托人与任务办理人

委托人:owner\_

办理人:assignee\_

参与人: taskService.addUserIdentityLink(taskId, userId, identityLinkType);

taskService.getIdentityLinksForTask(taskId);

签收: processEngine.getTaskService().claim(task.getId(), " user "); // 数据库字段owner\_为user

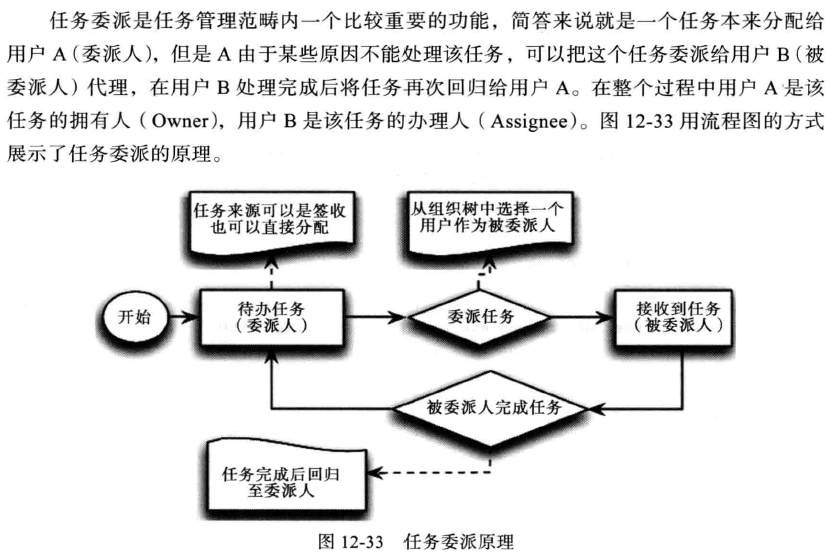
委派: processEngine.getTaskService().delegateTask(task.getId(), "henryyan"); // 数据库字段assignee \_为henryyan

被委派人完成任务: taskService.resolveTask(task.getId());

最终Assignee:user1,owen:user1

owner是任务所有者只能是一个，assignee是任务确定的接收者，也只能是一个，candidate是可领取任务的人，那么就会有很多，所以candidate单独放在identitylink表中关联task，而前两个就直接在task表中标明即可

当调用taskService.getIdentityLinksForTask("task102");时,这三种情况(两个表)都要统计进去。



## 签收与反签收

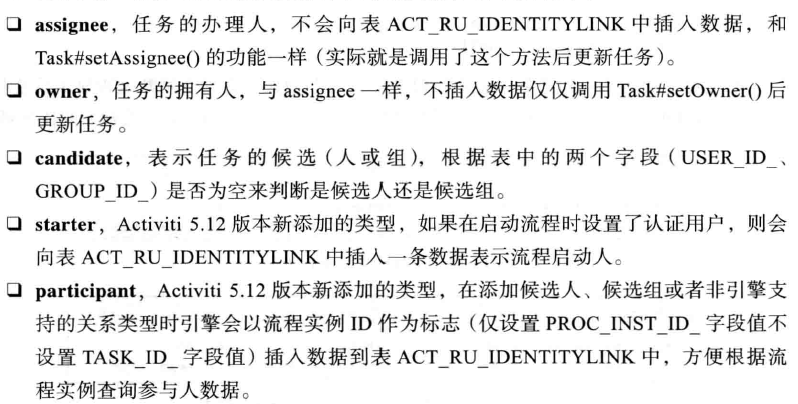
签收: taskService.claim(taskId, userId);

反签收: taskService.claim(taskId, null); // 把已签收的任务办理人置空

## 候选人与候选人组

taskService.addUserIdentityLink(taskId, userId, identityLinkType)

identityLinkType有分成几类人



设置认证用户有什么用?

## 子任务

一个任务可以拆分出多个任务交给不同的人处理

当主任务complete时,子任务也会默认完成,从ru表删除进入hi表

Task task2 = taskService.newTask();

task2.setParentTaskId(task.getId());

task2.setAssignee("sub task");

taskService.saveTask(task2); //记录到数据库中

# Question

1. 通过父流程查看子流程

List<Execution> processInstances = processEngine.getRuntimeService().createExecutionQuery().processInstanceId("13201").list();