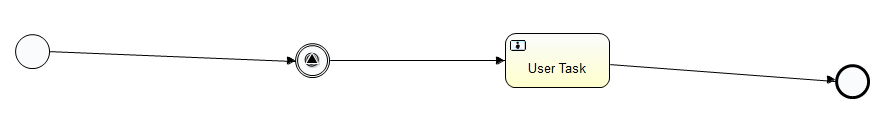
流程引擎测试

# 流程信号事件测试

信号事件分为两类

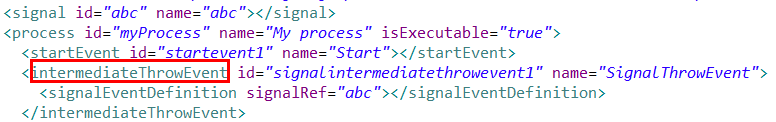
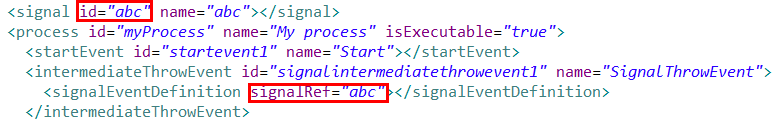
## 抛出事件



在BPMN2的标准中，这个为信号节点抛出事件，为实心倒三角型。这个节点会关联到一个信号定义。



XML如下：



id:abc就是必须定义的。

intermediateThrowEvent ：表示这是一个抛出信号事件。

在我的这个流程图的例子中表示：

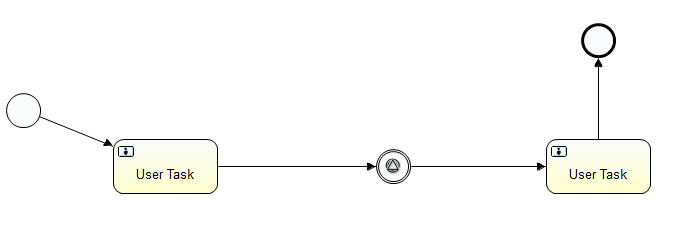
当流程启动时，流程引擎会去查找订阅了这个信号灯的事件，然后一个个让他们往下执行。在流程图中就是会触发捕获事件，驱动捕获流程往下执行。

启动流程时，不会在这个节点上停留，会触发捕获事件后，往下执行。上图会停留在“用户任务1”上。

注意：

抛出信号事件为全局事件，它会执行所有订阅了该信号的捕获事件，驱动流程往下执行。

## 捕获事件



在BPMN2的标准中，这个节点节点抛出事件,为空心倒三角型。



这个流程执行的顺序是：

流程执行完成用户任务1后，就会停留在捕获事件节点上，等待有相同的信号抛出。当接收到抛出信号后，流程会继续往下走，产生出用户任务2.

当然流程引擎在事件节点停留后。用户也可以调用API驱动流程往下执行。不一定得通过抛出信号进行触发。

API如下：

RuntimeService.signalEventReceived(String signalName);  
RuntimeService.signalEventReceived(String signalName, String executionId);

第一个API:

输入参数：

signalName：信号灯名称，在本例中为abc。

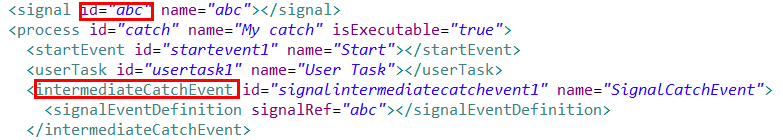
第二个API:

输入参数：

signalName:信号灯名称，本例中为abc。

executionId:流程运行实例ID。

流程定义XML如下：



信号：abc

intermediateCatchEvent:捕获事件。

查询订阅了某信号事件的运行实例：

List<Execution> executions = runtimeService.createExecutionQuery()

.signalEventSubscriptionName("abc")

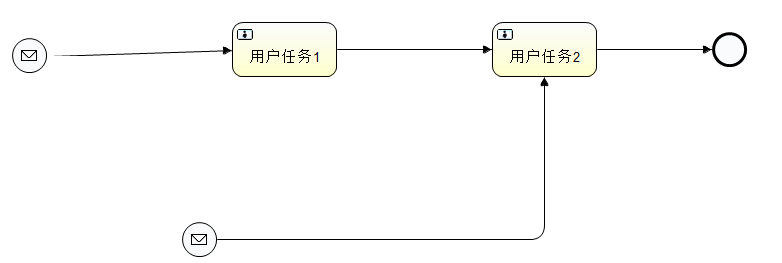
.list();

# 使用消息启动流程

在流程引擎中支持使用消息启动流程。这种可以实现如下需求：

举个例子：

流程描述如下：



A需要先完成一步审批到审批节点2。

B 直接就到审批节点2.

流程定义的XML如下：



这里定义了两个消息，在流程发起节点中引用了这两个消息。

启动流程方式：

ProcessInstance p= runtimeService.startProcessInstanceByMessage("message2");

ProcessInstance p= runtimeService.startProcessInstanceByMessage("mymessage");

实现逻辑如下：

发布流程的时候消息定义信息时，流程引擎会在表ACT\_RU\_EVENT\_SUBSCR中记录如下记录.



在发起流程时，程序首先根据消息定位流程定义ID，再根据消息节点ID定位到发起节点对象，然根据发起节点对象启动流程。

## 注意事项

1.消息唯一性。

消息在系统中不能重复，如果重复那么发布将会出错。抛出错误如下：



2.消息开始事件只支持顶级流程。消息开始事件不支持内嵌子流程。如果流程定义有多个消息开始事件，runtimeService.startProcessInstanceByMessage(...) 会选择对应的开始事件。如果流程定义有多个消息开始事件和一个空开始事件。 runtimeService.startProcessInstanceByKey(...)和 runtimeService.startProcessInstanceById(...)会使用空开始事件启动流程实例。

3.如果流程定义有多个消息开始事件，而且没有空开始事件，runtimeService.startProcessInstanceByKey(...)和 runtimeService.startProcessInstanceById(...)会抛出异常。**这个测试过可以启动流程。**

4.如果流程定义只有一个消息开始事件，

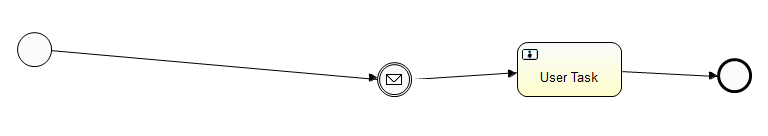
runtimeService.startProcessInstanceByKey(...)和 runtimeService.startProcessInstanceById(...)会使用这个消息开始事件启动流程实例。

5.如果流程被调用环节（callActivity）启动，消息开始事件只支持如下情况：

在消息开始事件以外，还有一个单独的空开始事件

流程只有一个消息开始事件，没有空开始事件。

# 使用消息捕获事件



流程定义xml如下：



在intermediateCatchEvent:中引用了一个消息定义。

这个流程执行的过程如下：

1.启动流程。

2.流程停留在捕获事件代码节点。

3.需要使用下面的api驱动节点往下执行。

void messageEventReceived(String messageName, String executionId);

参数：

messageName:消息名称

executionId:流程运行ID。

## 根据消息查询流程定义

ProcessDefinition processDefinition = repositoryService.createProcessDefinitionQuery()

.messageEventSubscriptionName("mymessage")

.singleResult();

输入参数：消息名称。

一个消息对应唯一一条流程。

一条流程都可以对应多个消息。

## 根据消息名查询运行实例对象

List<Execution> execution = runtimeService.createExecutionQuery()

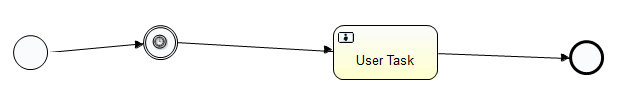
.messageEventSubscriptionName("paymentMessage")

.list();

输入参数：消息名称

可以查到订阅该消息的运行实例数据。

# 定时器

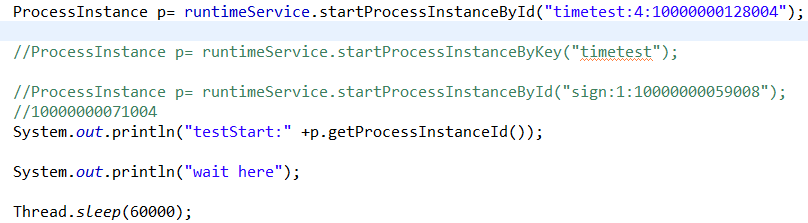


定时器设置：



这里定时10秒，意思是流程启动10秒中，自动产生用户任务。

不过在测试的过程中。



这个时候发现流程实例已经产生，但是60秒过后，这个任务并没有自动产生出来，在停止测试后这个后数据库产生。



不过在实际的环境中，这个应该会直接消失。我测试了另外一个用例去查询实例，时间为60秒，这个任务会自动产生出来。

注意需要定时器工作需要将工作引擎定时器打开，具体配置如下：

<property name="jobExecutorActivate" value="true" />

## 定时器时间配置

定时器的时间可以配置3种形式：

### 指定未来的一个日期

使用 [ISO 8601](http://en.wikipedia.org/wiki/ISO_8601" \l "Dates" \t "_top) 格式指定一个确定的时间，触发事件的时间。

<timerEventDefinition>

<timeDate>2011-03-11T12:13:14</timeDate>

</timerEventDefinition>

### 指定等待时长

指定定时器之前要等待多长时间， *timeDuration*可以设置为*timerEventDefinition*的子元素。 使用[ISO 8601](http://en.wikipedia.org/wiki/ISO_8601" \l "Durations" \t "_top)规定的格式 （由BPMN 2.0规定）。示例（等待10天）。

<timerEventDefinition>

<timeDuration>P10D</timeDuration>

</timerEventDefinition>

这个时间为等待10天。

P表示周期，这个字符放在时间字符串开始。

Y表示年份。

M表示月份的值。

W表示周数。

D表示天数。

T表示时间的开始。

H 表示小时数。

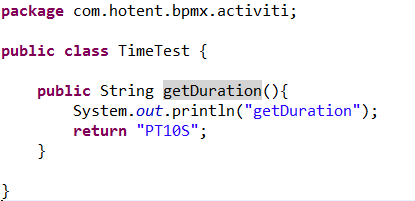
M 表示分钟数。

S表示秒数。

P1Y2M4DT2H3M10S表示一年，两月，四天，两小时，三分钟，十秒后。

这个时间可以使用表达式。比如：

我们定义如下代码：



在配置spring中如下配置：

<bean id="timeTest" class="com.hotent.bpmx.activiti.TimeTest"></bean>

流程定义如下：

<timerEventDefinition>

<timeDuration>${timeTest.getDuration()}</timeDuration>

</timerEventDefinition>

### 循环执行

指定重复执行的间隔， 可以用来定期启动流程实例，或为超时时间发送多个提醒。 timeCycle元素可以使用两种格式。第一种是 ISO 8601 标准的格式。示例（重复3次，每次间隔10小时）：

<timerEventDefinition>

<timeCycle>R3/PT10H</timeCycle>

</timerEventDefinition>

另外，你可以使用cron表达式指定timeCycle，下面的例子是从整点开始，每5分钟执行一次：

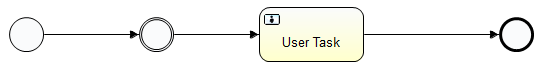
0 0/5 \* \* \* ?

# 中间触发空事件

触发事件在流程的执行过程中都不会在它上面停止。

流程触发空事件表示一个触发事件，但是没有定义任何类型的事件。也就是没有如时间事件，信号事件，消息事件等定义。

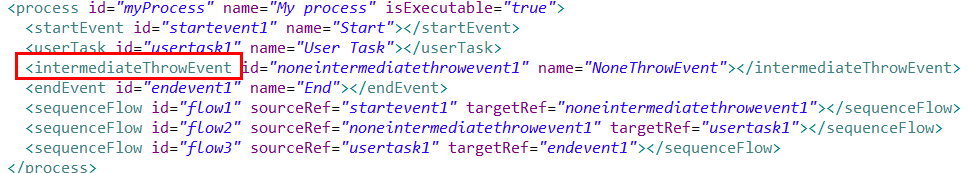
这个空事件的作用是：标识流程执行到了这里，我们可以对这个节点定义监听事件，这样就可以监控一些KPI。



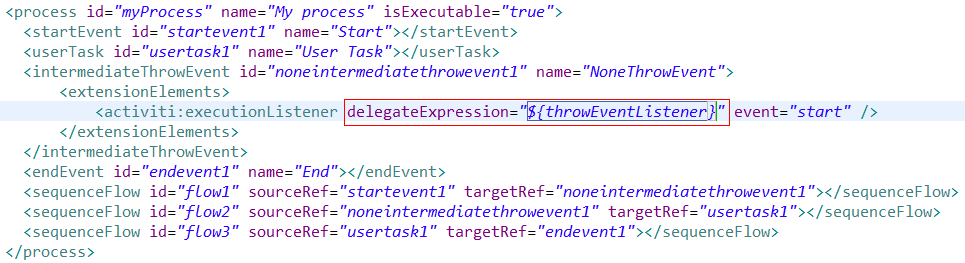
在流程图中，空的触发事件在流程图中用表示。



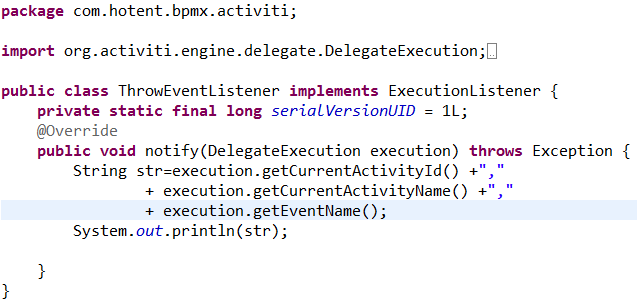
流程事件代码如下：



定义监听器如下：



监听器代码：

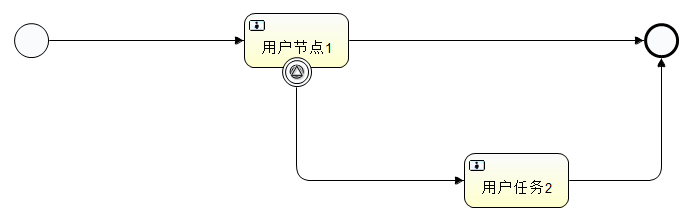


我们在这个监听器做相关的处理。

spring配置如下：



# 边界信号事件

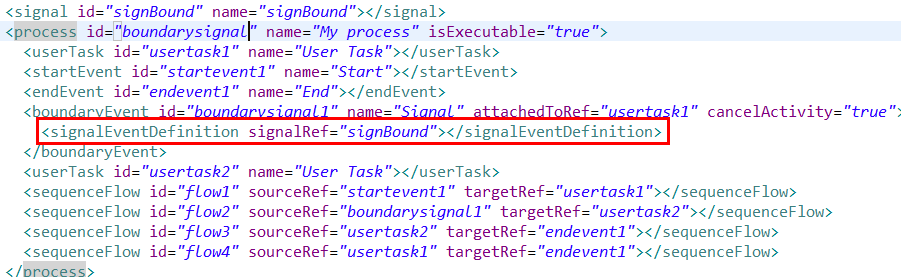


1.启动流程后。

2.可以完成用户任务1，流程结束。

3.也可以完成信号事件，产生用户任务2.

流程定义如下：



测试代码如下：

1.启动流程

ProcessInstance p= runtimeService.startProcessInstanceById("signBoundProcess:1:10000000149004");

2.当任务1产生时，同时会在事件订阅表中增加记录。

3.可以完成任务1，驱动流程执行。

3.可以调用API触发捕获事件往下执行。

驱动所有订阅了signBound信号的事件往下执行。

runtimeService.signalEventReceived("signBound");

驱动指定实例的代码往下执行。

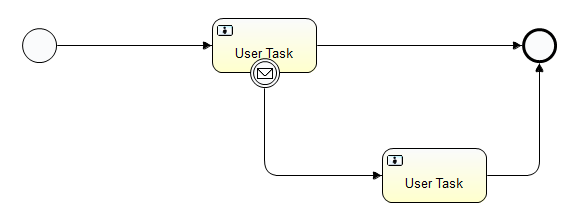
runtimeService.signalEventReceived("signBound",executionId);

# 消息边界事件

在启动流程时。

1.会产生用户任务1。

2.会产生消息捕获事件。



ACT\_RU\_EXECUTION



1.完成用户任务1，流程结束。

2.执行边界事件。

runtimeService.messageEventReceived("msgBound", "10000000153003");

这个时候会完成“用户任务1”，另外产生“用户任务2”。

# 流程引擎支持挂起

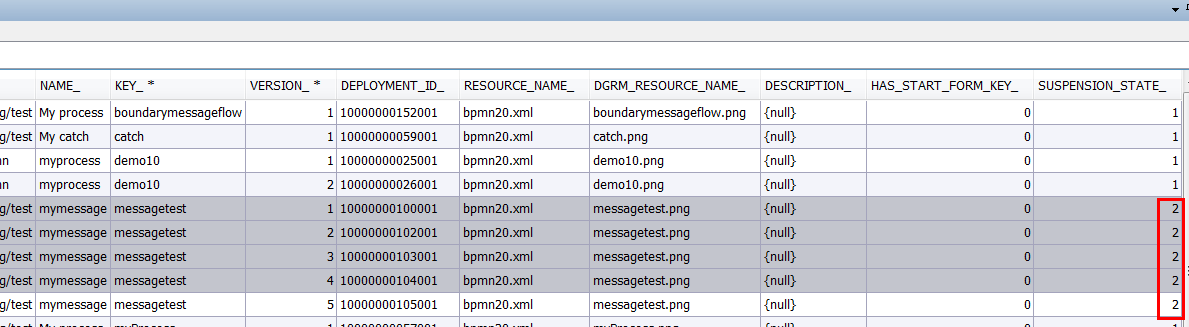
## 流程定义挂起

//挂起

repositoryService.suspendProcessDefinitionByKey("messagetest");

挂起后流程不能再启动。

挂起后，流程挂起状态为2。



也可以根据流程定义ID挂起流程。

repositoryService.suspendProcessDefinitionById("流程定义ID");

//激活流程

repositoryService.activateProcessDefinitionByKey("messagetest");

流程挂起后，流程实例还是可以继续执行。

## 挂起流程流程实例

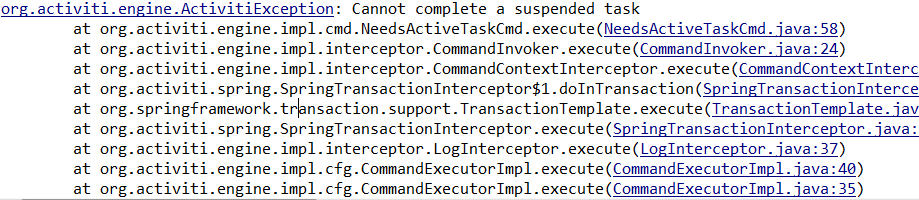
repositoryService.suspendProcessDefinitionByKey("messagetest",true,null);

挂起某个流程，true为挂起流程实例。

当挂起流程实例时：

任务不能执行。

测试抛出如下错误：



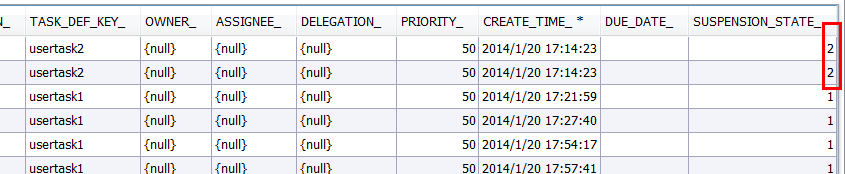
也可以根据日期进行挂起实例。

repositoryService.suspendProcessDefinitionByKey("messagetest",true,"未来的一个日期");

这个需要启动定时任务，定时器会根据日期，将流程或流程实例进行挂起。

挂起流程实例后数据表变化如下：

ACT\_RU\_TASK



ACT\_RU\_EXECUTION

