作者：千锋-索尔

版本：QF1.0

一、工作流概述

1.工作流介绍

工作流（Workflow），指“业务过程的部分或整体在计算机应用环境下的自动化”。是对工作流程及其各操作步骤之间业务规则的抽象、概括描述。在计算机中，工作流属于计算机支持的协同工作（CSCW）的一部分。后者是普遍地研究一个群体如何在计算机的帮助下实现协同工作的。

工作流主要解决的主要问题是：为了实现某个业务目标，利用计算机在多个参与者之间按某种预定规则自动传递文档、信息或者任务。

工作流概念起源于生产组织和办公自动化领域，是针对日常工作中具有固定程序活动而提出的一个概念，目的是通过将工作分解成定义良好的任务或角色，按照一定的规则和过程来执行这些任务并对其进行监控，达到提高工作效率、更好的控制过程、增强对客户的服务、有效管理业务流程等目的。

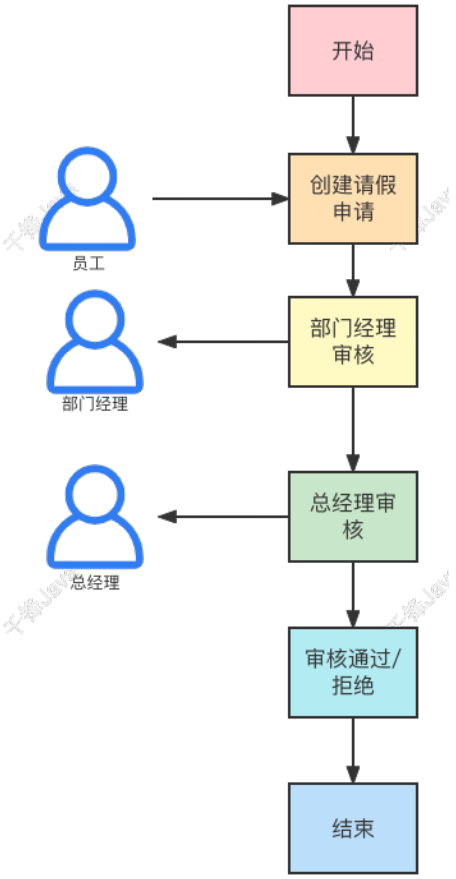
2.工作流应用场景

在众多工作流应用场景中，较为典型的应用场景有以下几个：

- 行政管理中的各类申请：请假申请、加班申请、出差申请等
- 客户服务类的各种处理：退货处理、换货处理等
- 人事管理类的各种场景：员工升职流程、岗位调配流程等

3.如何实现

为了实现工作流的流程控制，通常是采用状态字段的方式来跟踪流程的变化情况。通过状态变化来控制不同的业务行为。比如以请假申请为例，工作流程分成了多个状态，不同状态对应不同的业务行为。



通过状态的变化，把请假申请投递给不同的人进行相应的业务处理。这种方式的好处是非常自然和简单的。但是这种方式存在一些问题，流程中的某一个人只能看到当前流程的某一部分，不能看到完整的流程，需要借助需求文档等其他工具。另一个问题，当流程发生变更时，此时需要整个流程发生变化。

二、Activiti概述

1.Activiti介绍

Activiti项目是一项新的基于Apache许可的开源BPM平台，从基础开始构建，旨在提供支持新的BPMN 2.0标准，包括支持对象管理组（OMG），面对新技术的机遇，诸如互操作性和云架构，提供技术实现。

创始人Tom Baeyens是JBoss jBPM的项目架构师，以及另一位架构师Joram Barrez，一起加入到创建Alfresco这项首次实现Apache开源许可的BPMN 2.0引擎开发中来。

Activiti是一个独立运作和经营的开源项目品牌，并将独立于Alfresco开源ECM系统运行。 Activiti将是一种轻量级，可嵌入的BPM引擎，而且还设计适用于可扩展的云架构。 Activiti将提供宽松的Apache许可2.0，以便这个项目可以广泛被使用，同时促进Activiti BPM引擎和BPMN 2.0的匹配，该项目现正由OMG通过标准审定。 加入Alfresco Activiti项目的是VMware的SpringSource分支，Alfresco的计划把该项目提交给Apache基础架构，希望吸引更多方面的BPM专家和促进BPM的创新。

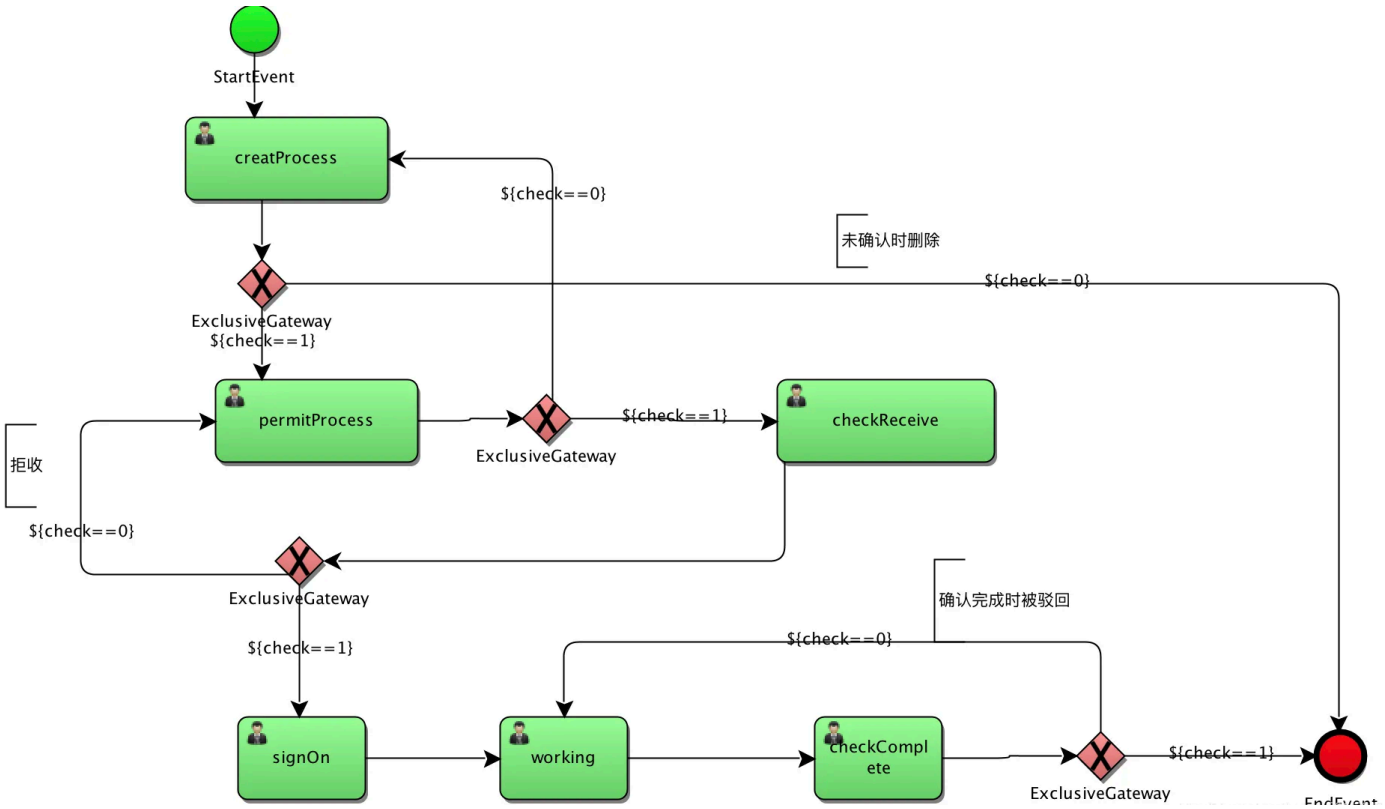
2.BPM

BPM，即Business Process Management，业务流程管理，是一种以规范化的构造端到端的卓越业务流程为中心，以持续的提高组织业务绩效为目的的系统化方法，常见商业管理教育如EMBA、MBA等均将BPM包含在内。

通常，BPM也指针对流程管理的信息化系统，其特点是注重流程驱动为核心，实现端到端全流程信息化管理。

3.BPMN

BPMN，即Business Process Modeling Notation，业务流程建模符号。BPMN定义了一个业务流程图（Business Process Diagram），该业务流程图基于一个流程图（flowcharting），该流程图被设计用于创建业务流程操作的图形化模型。而一个业务流程模型（Business Process Model），指一个由图形对象（graphical objects）组成的网状图，图形对象包括活动（activities)和用于定义这些活动执行顺序的流程控制器（flow controls）。



三、Activiti快速开始

1.创建Maven工程

引入依赖

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.qf</groupId>
    <artifactId>activiti-demo</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>8</maven.compiler.source>
        <maven.compiler.target>8</maven.compiler.target>
```

```
14          <slf4j.version>1.6.6</slf4j.version>
15          <log4j.version>1.2.12</log4j.version>
16      </properties>
17
18      <dependencies>
19          <dependency>
20              <groupId>org.activiti</groupId>
21              <artifactId>activiti-engine</artifactId>
22              <version>7.0.0.Beta1</version>
23          </dependency>
24
25          <dependency>
26              <groupId>org.activiti</groupId>
27              <artifactId>activiti-spring</artifactId>
28              <version>7.0.0.Beta1</version>
29          </dependency>
30
31          <dependency>
32              <groupId>org.activiti</groupId>
33              <artifactId>activiti-bpmn-model</artifactId>
34              <version>7.0.0.Beta1</version>
35          </dependency>
36
37          <dependency>
38              <groupId>org.activiti</groupId>
39              <artifactId>activiti-bpmn-converter</artifactId>
40              <version>7.0.0.Beta1</version>
41          </dependency>
42
43          <dependency>
44              <groupId>org.activiti</groupId>
45              <artifactId>activiti-json-converter</artifactId>
46              <version>7.0.0.Beta1</version>
47          </dependency>
48
49          <!--  <dependency>
50              <groupId>org.activiti</groupId>
```

```
51              <artifactId>activiti-bpmn-layout</artifactId>
52              <version>7.0.0.Beta1</version>
53            </dependency>-->
54
55          <dependency>
56              <groupId>org.activiti.cloud</groupId>
57              <artifactId>activiti-cloud-services-api</artifactId>
58              <version>7.0.0.Beta1</version>
59          </dependency>
60
61          <dependency>
62              <groupId>mysql</groupId>
63              <artifactId>mysql-connector-java</artifactId>
64              <version>8.0.23</version>
65          </dependency>
66
67          <dependency>
68              <groupId>junit</groupId>
69              <artifactId>junit</artifactId>
70              <version>4.12</version>
71          </dependency>
72
73          <!-- log start -->
74          <dependency>
75              <groupId>log4j</groupId>
76              <artifactId>log4j</artifactId>
77              <version>${log4j.version}</version>
78          </dependency>
79          <dependency>
80              <groupId>org.slf4j</groupId>
81              <artifactId>slf4j-api</artifactId>
82              <version>${slf4j.version}</version>
83          </dependency>
84          <dependency>
85              <groupId>org.slf4j</groupId>
86              <artifactId>slf4j-log4j12</artifactId>
87              <version>${slf4j.version}</version>
```

```
        </dependency>
        <!-- log end -->

        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
            <version>3.4.5</version>
        </dependency>

        <dependency>
            <groupId>commons-dbcp</groupId>
            <artifactId>commons-dbcp</artifactId>
            <version>1.4</version>
        </dependency>
    </dependencies>

</project>
```

2.创建日志的配置文件

```
# Set root category priority to INFO and its only appender to CONSOLE.
#log4j.rootCategory=INFO, CONSOLE            debug   info   warn error fatal
log4j.rootCategory=debug, CONSOLE, LOGFILE

# Set the enterprise logger category to FATAL and its only appender to CONSOLE.
log4j.logger.org.apache.axis.enterprise=FATAL, CONSOLE

# CONSOLE is set to be a ConsoleAppender using a PatternLayout.
log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender
log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout
log4j.appender.CONSOLE.layout.ConversionPattern=%d{ISO8601} %-6r [%15.15t] %-5p %30.30c %x - %m\n
```

```
12
13  # LOGFILE is set to be a File appender using a PatternLayout.
14  log4j.appender.LOGFILE=org.apache.log4j.FileAppender
15  log4j.appender.LOGFILE.File=/Users/zeleishi/Documents/code/log/log
    .log
16  log4j.appender.LOGFILE.Append=true
17  log4j.appender.LOGFILE.layout=org.apache.log4j.PatternLayout
18  log4j.appender.LOGFILE.layout.ConversionPattern=%d{ISO8601} %-6r
    [%15.15t] %-5p %30.30c %x - %m\n
```

3.创建activiti配置文件

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <beans xmlns="http://www.springframework.org/schema/beans"
3          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd">
5
6       <!--数据源-->
7       <bean id="dataSource"
    class="org.apache.commons.dbcp.BasicDataSource">
8           <property name="driverClassName"
    value="com.mysql.cj.jdbc.Driver"/>
9           <property name="url"
    value="jdbc:mysql://localhost:3306/activiti?serverTimezone=UTC"/>
10          <property name="username" value="root"/>
11          <property name="password" value="qf123456"/>
12      </bean>
13
14      <!--配置Activiti使用的processEngine对象 默认命名为
    processEngineConfiguration-->
15      <bean id="processEngineConfiguration"
    class="org.activiti.engine.impl.cfg.StandaloneProcessEngineConfigu
    ration">
```

```
        <!--注入数据源-->
        <property name="dataSource" ref="dataSource"/>
        <!--指定数据库生成策略,自动生成Activiti的表-->
        <property name="databaseSchemaUpdate" value="true"/>
    </bean>
</beans>
```

4.启动Activiti

```
package com.qf.activiti.demo;

import org.activiti.engine.ProcessEngine;
import org.activiti.engine.ProcessEngineConfiguration;
import org.activiti.engine.ProcessEngines;

/**
 * @author Thor
 * @公众号 Java架构栈
 */
public class TestActiviti {


    public static void main(String[] args) {
        //加载配置
        ProcessEngineConfiguration configuration =
ProcessEngineConfiguration.createProcessEngineConfigurationFromRes
ource("activiti.cfg.xml");
        //获取ProcessEngine对象
        ProcessEngine processEngine =
configuration.buildProcessEngine();
    }
}

```

5.Activiti表结构说明

1）GE通用数据

这里的数据可以用在各种情况下

表名	解释	详细
ACT_GE_BYTEARRAY	通用的流程定义和流程资源	保存流程定义图片和xml、Serializable(序列化)的变量,即保存所有二进制数据。
ACT_GE_PROPERTY	系统相关属性	存储整个流程引擎级别的数据

2）HI历史数据

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。
Activit默认提供了4种历史级别：

- none：不保存任何历史信息，可以提高系统性能；
- activity：保存所有的流程实例、任务、活动信息；
- audit：默认级别，保存所有的流程实例、任务、活动、表单属性；
- full：最完整的历史记录，除了audit级别的信息外还可以保存详细，例如流程变量。

表名	解释	详细
ACT_HI_ACTINST	历史节点表	记录所有节点
ACT_HI_ATTACHMENT	历史流程附件	
ACT_HI_COMMENT	历史说明性信息	
ACT_HI_DETAIL	历史流程中的细节信息	流程中产生的变量详细，包括控制流程流转的变量，业务表单中填写的流程需要用到的变量等。
ACT_HI_IDENTITYLINK	历史流程运行中的用户关系	
ACT_HI_PROCINST	历史流程实例	
ACT_HI_TASKINST	历史任务实例	
ACT_HI_VARINST	历史流程运行中的变量信息	

3）ID身份信息

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。

表名	解释	详细
ACT_ID_GROUP	用户组信息	
ACT_ID_INFO	用户组信息拓展表	
ACT_ID_MEMBERSHIP	用户与分组对应信息	用来保存用户的分组信息
ACT_ID_USER	用户信息	

4）RE静态数据

代表仓库，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）。

表名	解释	详细
ACT_RE_DEPLOYMENT	部署信息表	部署流程定义时需要被持久化保存下来的信息
ACT_RE_MODEL	模型部署表	流程设计器设计流程后，保存数据到该表
ACT_RE_PROCDEF	流程定义数据表	业务流程定义数据表

5）RU运行实例

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。

表名	解释	详细
ACT_RU_EVENT_SUBSCR	运行时事件表	
ACT_RU_EXECUTION	运行时执行实例表	
ACT_RU_IDENTITYLINK	运行时流程人员表	任务参与者数据表。主要存储当前节点参与者的信息。
ACT_RU_JOB	运行时定时任务数据表	
ACT_RU_TASK	运行时任务节点表	
ACT_RU_VARIABLE	运行时流程变量数据表	

6.Activiti的六大Service服务

Activiti工作流框架除了提供数据库的25张表外，还提供了Service服务层供我们使用。在项目中直接使用Service服务层接口，就可以实现工作流相关的业务功能。

1） RepositoryService仓储服务

```
1    //仓储服务
2    @Autowired
3    private RepositoryService repositoryService;
```

仓储服务可以用来部署我们的流程图，还可以创建我们的流程部署查询对象，用于查询刚刚部署的流程列表，便于我们管理流程，方法如下。

```
1
2     //部署流程的方法，流程图以inputStream流的形式传入
3     DeploymentBuilder builder =
    repositoryService.createDeployment();
4     builder.name(process.getName());
5     builder.addInputStream(fileName, inputStream);
6     Deployment deployment = builder.deploy();
7     //流程部署列表查询的方法
8     DeploymentQuery deploymentQuery =
    repositoryService.createDeploymentQuery();
9     //可以根据很多条件查询，我这是根据部署名称模糊查询
10    List<Deployment> list =
    deploymentQuery.deploymentNameLike("%"+name+"%")
```

2）RuntimeService运行时服务

```
1     //运行时服务
2     @Autowired
3     private RuntimeService runtimeService;
```

运行时服务主要用来开启流程实例，一个流程实例对应多个任务，也就是多个流程节点，比如请假审批是一个流程实例，部门主管，部门经理，总经理都是节点，我们开启服务是通过流程定义key或者流程定义id来开启的，方法如下：

```
1     //根据部署id创建流程定义
2     ProcessDefinition def =
    repositoryService.createProcessDefinitionQuery().
3         deploymentId(form.getDeployId()).singleResult();
4     //根据流程定义id或者key开启流程实例
5     ProcessInstance proInst =
    runtimeService.startProcessInstanceById(def.getId());
```

当我们用仓储服务部署了流程图之后，就会产生一个流程部署id，一个流程部署id对应一个流程定义，一个流程定义对应多个流程实例，流程定义和流程实例之间的关系就好比是类和对象的关系。一个流程实例对应多个任务节点。

3）TaskService任务服务

```
1    //任务服务
2    @Autowired
3    private TaskService taskService;
```

任务服务是用来可以用来领取，完成，查询任务列表功能的，使用方法分别如下：

```
1  //根据任务id和用户领取任务
2  taskService.claim(String taskId, String userId)
3  //根据任务id完成自己节点的任务
4  taskService.complete(String taskId)
5  //创建任务查询对象之后根据候选人也就是任务处理人查询自己的任务列表
6  taskService.createTaskQuery().taskAssignee(String assignee)
```

4）HistoryService历史服务

```
1    //历史服务
2    @Autowired
3    private HistoryService historyService;
```

历史服务可以查看审批人曾经审批完成了哪些项目，审批项目总共花了多少时间，以及在哪个环节比较耗费时间等等，便于审批人查看历史信息，方法如下。

```
1    //根据审批人查看该审批人审批了哪些项目
2    List<HistoricTaskInstance> =
   historyService.createHistoricTaskInstanceQuery().
3      taskAssignee(String assignee).finished().list();
```

历史任务对象HistoricTaskInstance，它里面封装了任务开始时间，结束时间，该节点花费的时间等等信息。

5）FormService表单服务

```
1    //表单服务
2    @Autowired
3    private FormService formService;
```

实现任务表单管理的，可选服务。

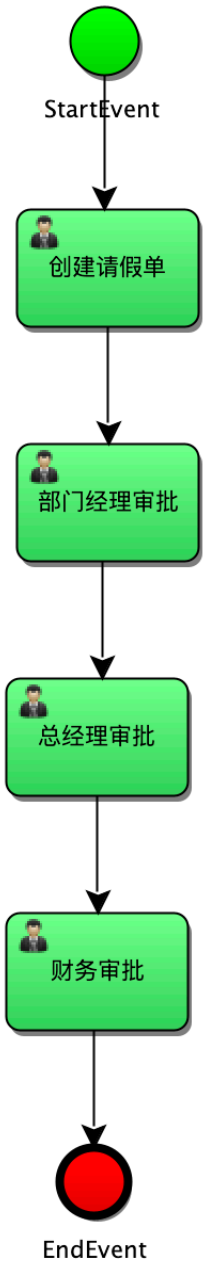
6）IdentityService实体服务

```
1    //实体服务
2    @Autowired
3    private IdentityService identityService;
```

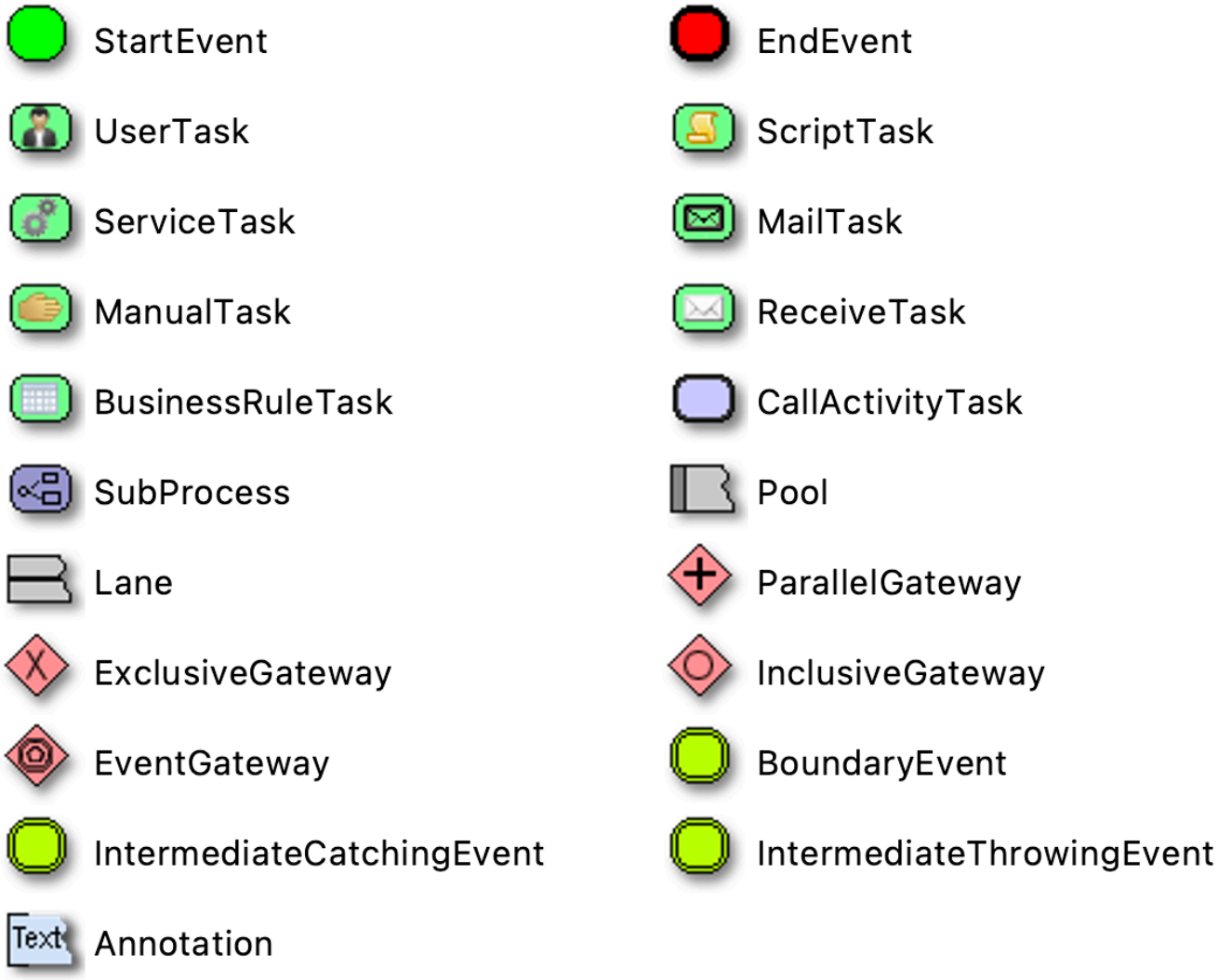
操作用户信息，用户分组信息等，组信息包括如部门表和职位表。

7.绘制流程图

使用Idea的bpmn组件绘制流程图，绘制内容如下：



Bpmn建模符号如下:



对应图标的内容解释如下：

名称	描述
UserTask	用户任务，即用户操作的任务
ScriptTask	脚本任务，脚本任务是一个自动化活动。当一个流程执行到达脚本任务时，执行相应的脚本。Activiti脚本任务比较少用，脚本任务通常是用在当前的监听器或者监听服务类都不能知足的情形下面，或者说后期系统维护，忽然在不想改动系统的状况下须要对流程作一些适当的改变。仅仅是几个变量或者仅仅是一个计算公式等等。这个时候可使用脚本任务。
ServiceTask	服务任务不同于用户任务（需人工处理），服务任务一般是一段可自动执行的任务而无需人工干预。

MailTask	允许通过自动邮件服务任务来增强业务流程，这些任务将电子邮件发送给一个或多个收件人，多个人以逗号隔开，包括抄送，密件抄送，支持发送HTML内容等。请注意，邮件任务不是 BPMN 2.0 的正式任务规格（因此没有专用图标）。因此，在Activiti中，邮件任务也是serviceTask，只是使用activiti:type="mail"属性来表示此服务任务为邮件任务。
ManualTask	手工任务就是一个自动执行的过程。手动任务几乎不在程序中做什么事情，只是在流程的历史中留下一点痕迹，表明流程是走过某些节点的。而且这个任务是无法用taskservice查询到的。手工任务就是一个直接通过的任务。可以使用它来自动执行一些可以直接通过的任务。
ReceiveTask	接受任务其实和Activiti的手动任务是差不多的，不过手动任务是直接通过，而接受任务则在任务启动到该节点的时候停下来等待信号。当任务接收到信号的时候，该流程就会继续往下执行。
BusinessRuleTask	规则任务可以通过制定一系列的规则来实现流程自动化
CallActivityTask	唤醒（调用）子流程，主流程走到某个节点后唤醒其它子流程，当子流程走完后，主流程继续走。其中子流程可以作为一个独立的流程启动互不影响
SubProcess	内嵌子流程，子流程元素内嵌在主流程元素之内，只能在该流程中使用该子流程，外部是无法访问到的。这种子流程一般作为局部通用逻辑处理，或者因为特定业务需要，使得比较复杂的单个主流程设计清晰直观
ParallelGateWay	它可以将分支（fork，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。）为多个路径，也可以合并（join，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）多个入口路径。并行网关的前提条件：基于出口顺序流和入口顺序流。

ExclusiveGateway	排他网关经常使用流程变量决定流程下一步要选择的路径，为流程中的决策建模。当执行到达这个网关时，所有出口顺序流会按照它们定义的顺序进行计算。条件计算为true的顺序流（当没有设置条件时，认为顺序流定义为true）会被选择用于继续流程。用排他网关时，只会选择一条顺序流。当多条顺序 流的条件都计算为true时，其中在XML中定义的第一条（也只有这条）会被选择，用于继续流程。如果没有可选的顺序流，会抛出异常。
InclusiveGateway	包容网关就是并行网关（Parallel Gateway）和排他网关（Exclusive Gateway）的组合。可以在出口顺序流上定义条件，包容网关会计算它们。然而主要的区别是，包容网关与并行网关一样，可以选择多于一条（出口）顺序流
EventGateway	事件网关，允许基于事件做选择。网关的每一条出口顺序流，都需要连接至一个捕获中间事件。当流程执行到达基于事件的网关时，网关类似等待状态地动作：执行被暂停。并且，为每一条出口顺序流，创建一个事件订阅。流程的走向完全是由于中间事件的选择。而由哪一个事件来决定流程的走向则是由最先触发的事件来决定的。

8.部署流程图

将流程图部署到Activiti框架中，生成具体的流程信息数据。

```
1       /*
2       部署流程图
3        */
4       @Test
5       public void test1(){
6           //1.获取ProcessEngine对象
7           ProcessEngine engine =
    ProcessEngines.getDefaultProcessEngine();
8           //2.获取用于部署的RepositoryService
9           RepositoryService repositoryService =
    engine.getRepositoryService();
```

```
        //3.执行部署
        Deployment deploy = repositoryService.createDeployment()
                //添加资源
                .addClasspathResource("bpmn/leave.bpmn")
                .addClasspathResource("bpmn/leave.png")
                .name("请假申请流程")
                .deploy();
        System.out.println("流程部署的id:"+deploy.getId());
        System.out.println("流程name:"+deploy.getName());
    }
```

9.创建流程实例

根据流程定义来创建流程实例。

```
     /*
    创建流程实例
     */
    @Test
    public void test2(){
        ProcessEngine engine =
ProcessEngines.getDefaultProcessEngine();
        RepositoryService repositoryService =
engine.getRepositoryService();
        //根据部署id创建流程定义
        RuntimeService runtimeService =
engine.getRuntimeService();
        ProcessDefinition def =
repositoryService.createProcessDefinitionQuery().
                deploymentId("2501").singleResult();
        ProcessInstance processInstance =
runtimeService.startProcessInstanceById(def.getId());
        System.out.println("流程定义的
id:"+processInstance.getProcessDefinitionId());
        System.out.println("实例id: "+processInstance.getId());
```

```
15      }
```

10.任务查询

流程实例启动后，不同的用户可以查询到他在这个流程中的任务，前提是流程已经到达该用户。

```
1       /*
2       任务查询
3        */
4       @Test
5       public void test3(){
6           ProcessEngine engine =
    ProcessEngines.getDefaultProcessEngine();
7           TaskService taskService = engine.getTaskService();
8           List<Task> tasks = taskService.createTaskQuery()
9                   .processDefinitionId("myProcess_1:1:2504")
10                  .taskAssignee("王工")
11                  .list();
12          for (Task task : tasks) {
13              System.out.println("流程定义
    Id: "+task.getProcessDefinitionId());
14              System.out.println("流程实例
    Id:"+task.getProcessInstanceId());
15              System.out.println("任务Id:"+task.getId());
16              System.out.println("任务名称: "+task.getName());
17          }
18      }
```

11.处理任务

指定的处理人可以处理任务，让整个流程进行下去，去往下一个任务点。

```
    /*
      任务的处理
       */
      @Test
      public void test4(){
          //1.获得engine
          ProcessEngine engine =
    ProcessEngines.getDefaultProcessEngine();
          //2.获得TaskService
          TaskService taskService = engine.getTaskService();
          //3.查询出王工的任务
          Task task = taskService.createTaskQuery()
                  .processDefinitionId("myProcess_1:1:2504")
                  .taskAssignee("王工")
                  .singleResult();
          //完成任务
          taskService.complete(task.getId());
      }
```

四、流程的基本操作

1.流程资源下载

通过RepositoryService下载流程图和流程文件。

```
/*
      下载流程图图片和bpmn文件
       */
      @Test
      public void test5() throws IOException {
```

```
6           //1.获得engine对象
7           ProcessEngine engine =
    ProcessEngines.getDefaultProcessEngine();
8           //2.获得RepositoryService对象
9           RepositoryService repositoryService =
    engine.getRepositoryService();
10          //3.获得流程定义对象
11          ProcessDefinition processDefinition =
    repositoryService.createProcessDefinitionQuery()
12                  .processDefinitionId("myProcess_1:1:2504")
13                  .singleResult();
14          //4.获得png图片的输入流
15          InputStream pngInputStream =
    repositoryService.getResourceAsStream(processDefinition.getDeploym
    entId(),
16                  processDefinition.getDiagramResourceName());
17          //5.获得bpmn文件的输入流
18          InputStream bpmnInputStream =
    repositoryService.getResourceAsStream(processDefinition.getDeploym
    entId(),
19                  processDefinition.getResourceName());
20          FileOutputStream pngFos = new
    FileOutputStream("leave1.png");
21          FileOutputStream bpmnFos = new
    FileOutputStream("leave1.bpmn");
22          //6.读写数据
23          int b = 0;
24          while((b = pngInputStream.read())!=-1){
25              pngFos.write(b);
26          }
27
28          int b1 = 0;
29          while((b1 = bpmnInputStream.read())!=-1){
30              bpmnFos.write(b);
31          }
32          System.out.println("下载完成了");
33
```

```
34      }
```

2.流程历史信息查询

在数据库的`ACT_HI_ACTINST`中存放所有节点的历史信息，Activiti的HistoryService提供了查询历史信息的API。

```
1       /*
2       查询流程历史信息
3        */
4       @Test
5       public void test6(){
6           //1.获得engine对象
7           ProcessEngine engine =
    ProcessEngines.getDefaultProcessEngine();
8           //2.获得HistoryService对象
9           HistoryService historyService =
    engine.getHistoryService();
10          //3.查询历史信息
11          List<HistoricActivityInstance> results =
    historyService.createHistoricActivityInstanceQuery()
12                  //传入流程定义id
13                  .processDefinitionId("myProcess_1:1:2504")
14                  //根据节点的开始时间逆序的排序
15                  .orderByHistoricActivityInstanceStartTime().desc()
16                  //返回结果列表
17                  .list();
18          for (HistoricActivityInstance result : results) {
19              System.out.println(result.getActivityId());
20              System.out.println(result.getActivityName());
21              System.out.println(result.getActivityType());
22              System.out.println(result.getAssignee());
23              System.out.println("------------");
24          }
25      }
```

3.流程实例绑定业务id

Activiti提供的数据是针对于该框架所需要的流程控制维护的数据，也就是数据库25张表存放的数据，但是在业务系统中，业务数据如何与Activiti框架进行关联？通过流程实例数据的BusinessKey字段来实现关联。

在创建流程实例时，指明BusinessKey即可。

```
    /*
    创建流程实例，关联业务id
     */
    @Test
    public void test1(){
        //1.获得engine对象
        ProcessEngine engine =
ProcessEngines.getDefaultProcessEngine();
        //2.获得RuntimeService对象
        RuntimeService runtimeService =
engine.getRuntimeService();
        //3.创建流程实例，并指明业务id
        ProcessInstance processInstance =
runtimeService.startProcessInstanceById("myProcess_1:1:2504",
"1001");
        System.out.println("流程实例已经创建");
        System.out.println("业务
ID: "+processInstance.getBusinessKey());
    }
```

4.删除流程定义

删除流程定义，可以删除有未结束的流程实例的流程定义或者已结束的流程定义。

```
    /*
    删除流程定义
     */
    @Test
    public void test2(){
        //1.获得engine
        ProcessEngine engine =
ProcessEngines.getDefaultProcessEngine();
        //2.获得RepositoryService
        RepositoryService repositoryService =
engine.getRepositoryService();
        //3.删除部署的流程,cascade:true，删除正在进行的流程实例，否则无法
删除
        repositoryService.deleteDeployment("2501",true);
    }
```

5.流程的挂起和激活

当因为业务需要，将执行的流程挂起，或者将已被挂起的流程激活，可以通过Activiti框架，对流程进行挂起和激活。

```
    /*
    流程挂起
     */
    @Test
    public void test3(){
        //1.获得engine对象
        ProcessEngine engine =
ProcessEngines.getDefaultProcessEngine();
        //2.获得RuntimeService
        RuntimeService runtimeService =
engine.getRuntimeService();
        //3.获得流程实例对象
        ProcessInstance instance =
runtimeService.createProcessInstanceQuery()
                .processInstanceId("22501").singleResult();
        //4.先判断该流程是否属于正常状态（非挂起状态）
```

```
14          boolean suspended = instance.isSuspended();
15          if(!suspended){
16              //执行流程实例的挂起操作
17
 runtimeService.suspendProcessInstanceById(instance.getId());
18              System.out.println(instance.getId()+"流程被挂起");
19          }
20
21      }
22
23      /*
24      激活一个被挂起的流程
25       */
26      @Test
27      public void test4(){
28          ProcessEngine engine =
ProcessEngines.getDefaultProcessEngine();
29          RuntimeService runtimeService =
engine.getRuntimeService();
30          ProcessInstance processInstance =
runtimeService.createProcessInstanceQuery().processInstanceId("225
01").singleResult();
31          boolean suspended = processInstance.isSuspended();
32          if(suspended){
33              //当前流程实例已被挂起，激活的操作
34
 runtimeService.activateProcessInstanceById(processInstance.getId(
));
35              System.out.println(processInstance.getId()+"流程被激
活");
36          }
37
38      }
```

6.使用变量设置Assignee

之前设置指派人时，是在画图时直接写明，不能通过程序更改。因此可以通过变量设值的方式进行。

使用UEL表达式在创建bpmn的任务节点时指明Assignee的值为变量`${assignee0}`,再在程序中通过代码传入。

```
1       /*
2       创建流程实例，为Assignee变量设置参数
3        */
4       @Test
5       public void test6(){
6           ProcessEngine engine =
    ProcessEngines.getDefaultProcessEngine();
7           RuntimeService runtimeService =
    engine.getRuntimeService();
8           //创建存放变量名-变量值的map
9           Map<String,Object> map = new HashMap<>();
10          map.put("assignee0","张三");
11          map.put("assignee1","李四");
12          map.put("assignee2","王五");
13          map.put("assignee3","赵六");
14          //创建流程实例
15          ProcessInstance instance =
    runtimeService.startProcessInstanceById("myProcess_1:2:32504",
    map);
16      }
17
```

7.使用变量驱动流程走向

通过设置流程分支的变量条件，依据流程变量来控制流程走向。

- 设置流程分支的变量条件：

```
    <sequenceFlow id="_10" sourceRef="_4" targetRef="_7">
      <conditionExpression xsi:type="tFormalExpression">
        <![CDATA[${leave.days<3}]]>
      </conditionExpression>
    </sequenceFlow>
    <sequenceFlow id="_11" sourceRef="_7" targetRef="_6"/>
    <sequenceFlow id="_12" sourceRef="_4" targetRef="_5">
      <conditionExpression xsi:type="tFormalExpression">
        <![CDATA[${leave.days>2}]]>
      </conditionExpression>
    </sequenceFlow>
```

- 设置流程变量，创建流程实例

```
  /*
     根据变量控制流程图的走向
      */
     @Test
     public void test7(){
         ProcessEngine engine =
ProcessEngines.getDefaultProcessEngine();
         RuntimeService runtimeService =
engine.getRuntimeService();
         Map<String, Object> map = new HashMap<>();
         map.put("assignee0","zhangsan");
         map.put("assignee1","lisi");
         map.put("assignee2","wangwu");
         map.put("assignee3","zhaoliu");
         //leave.days
         Leave leave = new Leave();
         leave.setDays(3);
         map.put("leave",leave);
         ProcessInstance processInstance =
runtimeService.startProcessInstanceByKey("leave", map);
     }
```

- 执行流程，检验分支效果

```
 @Test
    public void test8(){
        ProcessEngine processEngine =
ProcessEngines.getDefaultProcessEngine();
        TaskService taskService = processEngine.getTaskService();
        Task task =
taskService.createTaskQuery().processDefinitionKey("leave")
                .taskAssignee("zhaoliu").singleResult();
        if(task!=null){
            taskService.complete(task.getId());
            System.out.println("任务执行完毕");
        }
    }
```

8.使用Local局部变量

在流程实例运行过程中，局部变量只针对当前任务生效，当进入到下一个任务时，该局部变量失效。

```
    /*
    设置局部变量
     */
    @Test
    public void test9(){
        ProcessEngine processEngine =
ProcessEngines.getDefaultProcessEngine();
        TaskService taskService = processEngine.getTaskService();
        Task task =
taskService.createTaskQuery().processDefinitionKey("leave")
                .taskAssignee("zhaoliu").singleResult();
        if(task!=null){
            //为任务设置局部变量
            taskService.setVariableLocal(task.getId(),"days","3");
            taskService.complete(task.getId());
            System.out.println("任务执行完毕");
        }
    }
```

9.组任务

如果任务负责人是单一存在的，且任务负责人因为默写原因没有办法完成任务，那么流程就没办法执行下去了。

此时可以为任务设置候选人，通过候选人拾取、执行、归还等操作，完成组任务相关的操作。

1）任务候选人

创建任务候选人，允许一个任务可以被多个负责人领取。通过流程图或者程序来指明任务的候选人。

查询候选人查询任务的代码如下：

```
/*
    根据候选人来查询组任务
     */
    @Test
    public void test3(){
        String user = "李工";
        ProcessEngine engine =
ProcessEngines.getDefaultProcessEngine();
        TaskService taskService = engine.getTaskService();
        List<Task> tasks = taskService.createTaskQuery()
                .processDefinitionKey("leave3")
                .taskCandidateUser(user)
                .list();
        for (Task task : tasks) {
            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）候选人拾取任务

任务候选人先拾取任务，才能执行任务。

```
    /*
      拾取任务
       */
      @Test
      public void test4(){
          ProcessEngine engine =
ProcessEngines.getDefaultProcessEngine();
          TaskService taskService = engine.getTaskService();
          String taskId = "67505";
          String user = "李工";
          //拾取任务
          Task task = taskService.createTaskQuery().taskId(taskId)
                  .taskCandidateUser(user)
                  .singleResult();
          if(task!=null){
              taskService.claim(taskId,user);
              System.out.println("完成拾取");
          }
      }
```

3）候选人归还任务

候选人拾取任务后，在没执行任务之前，可以归还任务。

```
     /*
      归还任务
       */
      @Test
      public void test5(){
          ProcessEngine engine =
ProcessEngines.getDefaultProcessEngine();
          TaskService taskService = engine.getTaskService();
```

```
        String taskId = "67505";
        String user = "李工";
        Task task = taskService.createTaskQuery()
                .taskId(taskId)
                .taskAssignee(user)
                .singleResult();
        if(task!=null){
            //归还任务
            taskService.setAssignee(taskId,null);
        }

    }

```

4）任务转交

任务候选人拾取完任务后，可以选择归还任务，也可以选择把任务转交给其他候选人。

```
    /*
     转交任务给其他候选人
      */
     @Test
     public void test6(){
         ProcessEngine engine =
ProcessEngines.getDefaultProcessEngine();
         TaskService taskService = engine.getTaskService();
         String taskId = "67505";
         String user = "李工";
         String targetUser = "王工";
         Task task = taskService.createTaskQuery().taskId(taskId)
                 .taskAssignee(user)
                 .singleResult();
         if(task!=null){
             //转交任务
             taskService.setAssignee(taskId,targetUser);
         }
```

```
18          }
```

五、网关

网关(Gateway)用于控制流程走向（在BPMN2.0规范中称为“执行令牌”）。根据功能不同可以划分为以下4种网关：

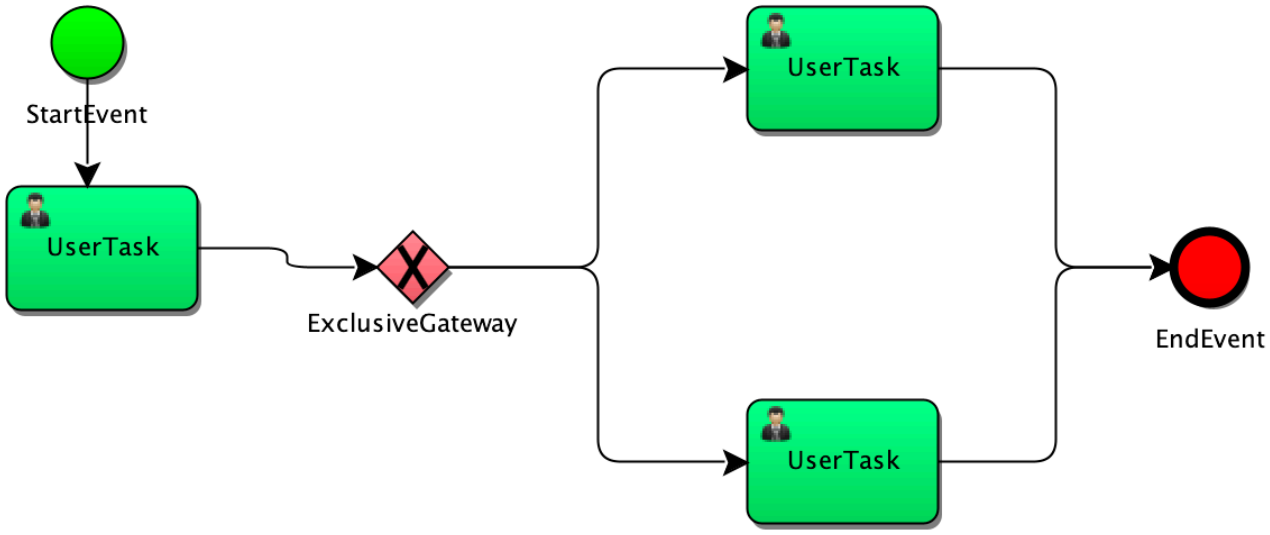
- 排他网关
- 并行网关
- 包容网关
- 事件网关

1.排他网关

排他网关（Exclusive Gateway）用来对流程中的决定进行建模。流程执行到该网关时，按照输出流的顺序逐个计算，当条件计算结果为true时。继续执行当前网关的输出流。

在排他网关中，如果多个线路的计算结果都为true，那么只会执行第一个值为true的网关，忽略其他表达式的值为true的网关。如果多个网关计算结果没有为true的值，则引擎会抛出异常。

排他网关需要和条件顺序流配合使用，一个排他网关可以连接多个条件顺序流，每个条件顺序流设置一个条件在运行时由引擎计算并根据结果是否为true决定执行与否。

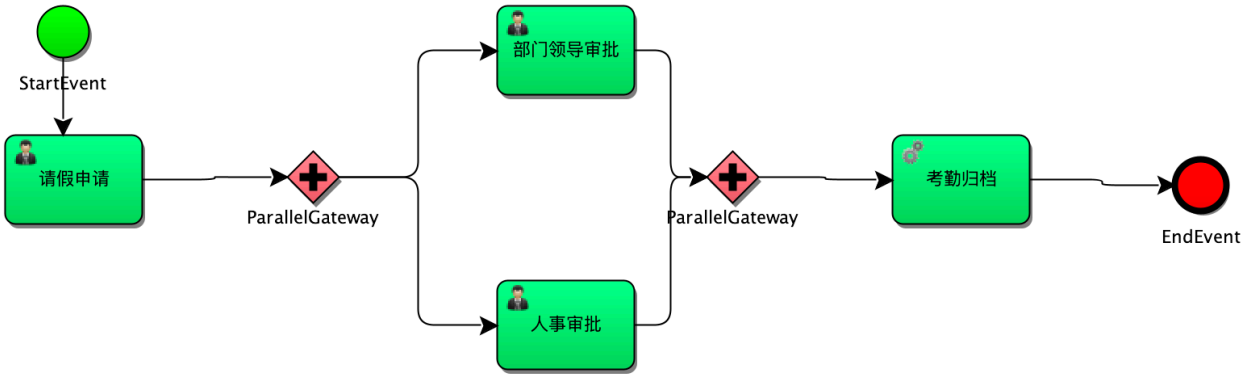


2.并行网关

并行网关用来对并发的任务进行流程建模，它能把单条线路任务拆分成多个路径并行执行获将多条线路合并。

并行网关的图形化表示是在菱形中嵌入一个加号。并行网关的功能取决于输入、输出顺序流。

- 拆分：并行执行所有的输出顺序流，并且为每一条顺序流创建一个并行执行线路。
- 合并：所有从并行网关拆分兵执行完成的线路均在此等待，直到所有的线路都执行完成才继续向下执行。

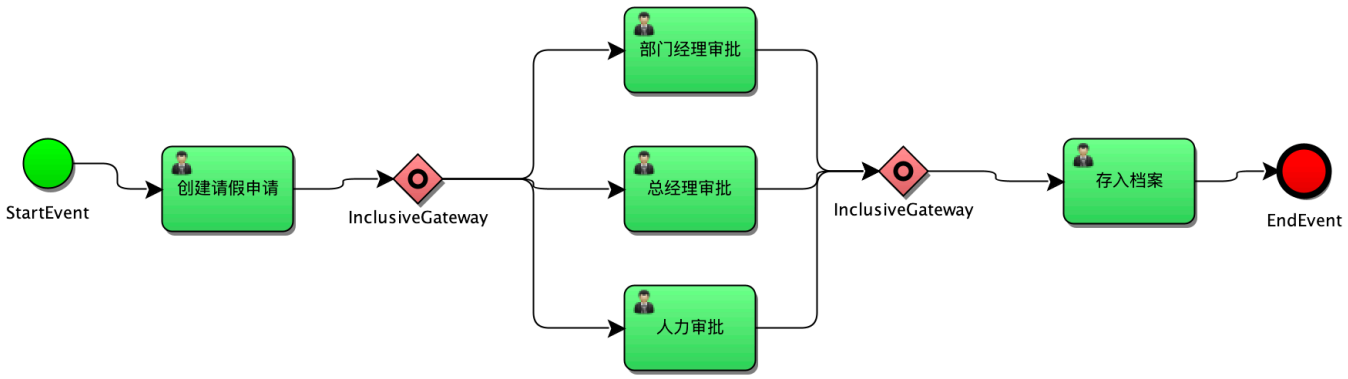


3.包容网关

包容网关融合了排他网关和并行网关的特性，排他网关允许在每条线路上设置条件，并行网关可以同时执行多条线路，包容网关既可以同时执行多条线路，又允许在网关上设置条件。

和并行网关一样，包容网关的功能也取决于输入、输出顺序流。

- 拆分：计算每条线路上的表达式，当表达式计算结果为true时，创建一个并行线路并继续执行。
- 合并：所有从并行网关拆分并执行完成的线路均在此等待，直到所有的线路都执行完才继续向下执行。



4.事件网关

事件网关是专门为中间捕获事件设置的，它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，因为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。

六、Activiti在项目中使用

1.Spring中使用Activiti

- 创建maven工程
- 引入依赖

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.qf</groupId>
    <artifactId>spring-activiti-demo</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>8</maven.compiler.source>
        <maven.compiler.target>8</maven.compiler.target>
        <slf4j.version>1.6.6</slf4j.version>
        <log4j.version>1.2.12</log4j.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.activiti</groupId>
            <artifactId>activiti-engine</artifactId>
            <version>7.0.0.Beta1</version>
        </dependency>
        <dependency>
            <groupId>org.activiti</groupId>
            <artifactId>activiti-spring</artifactId>
            <version>7.0.0.Beta1</version>
        </dependency>
        <dependency>
            <groupId>org.activiti</groupId>
            <artifactId>activiti-bpmn-model</artifactId>
            <version>7.0.0.Beta1</version>
```

```
        </dependency>
        <dependency>
            <groupId>org.activiti</groupId>
            <artifactId>activiti-bpmn-converter</artifactId>
            <version>7.0.0.Beta1</version>
        </dependency>
        <dependency>
            <groupId>org.activiti</groupId>
            <artifactId>activiti-json-converter</artifactId>
            <version>7.0.0.Beta1</version>
        </dependency>
        <dependency>
            <groupId>org.activiti</groupId>
            <artifactId>activiti-bpmn-layout</artifactId>
            <version>7.0.0.Beta1</version>
        </dependency>
        <dependency>
            <groupId>org.activiti.cloud</groupId>
            <artifactId>activiti-cloud-services-api</artifactId>
            <version>7.0.0.Beta1</version>
        </dependency>
        <dependency>
            <groupId>org.aspectj</groupId>
            <artifactId>aspectjweaver</artifactId>
            <version>1.6.11</version>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.23</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
        </dependency>
        <dependency>
```

```
69              <groupId>log4j</groupId>
70              <artifactId>log4j</artifactId>
71              <version>${log4j.version}</version>
72          </dependency>
73          <dependency>
74              <groupId>org.slf4j</groupId>
75              <artifactId>slf4j-api</artifactId>
76              <version>${slf4j.version}</version>
77          </dependency>
78          <dependency>
79              <groupId>org.slf4j</groupId>
80              <artifactId>slf4j-log4j12</artifactId>
81              <version>${slf4j.version}</version>
82          </dependency>
83          <dependency>
84              <groupId>org.springframework</groupId>
85              <artifactId>spring-test</artifactId>
86              <version>5.0.8.RELEASE</version>
87          </dependency>
88
89          <dependency>
90              <groupId>org.mybatis</groupId>
91              <artifactId>mybatis</artifactId>
92              <version>3.4.5</version>
93          </dependency>
94
95          <dependency>
96              <groupId>commons-dbcp</groupId>
97              <artifactId>commons-dbcp</artifactId>
98              <version>1.4</version>
99          </dependency>
100
101
102      </dependencies>
103
104  </project>
```

- 编写spring配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

    <!--数据源-->
    <bean id="dataSource"
class="org.apache.commons.dbcp.BasicDataSource">
        <property name="driverClassName"
value="com.mysql.cj.jdbc.Driver"/>
        <property name="url"
value="jdbc:mysql://localhost:3306/activiti?serverTimezone=UTC"/>
        <property name="username" value="root"/>
        <property name="password" value="qf123456" />
    </bean>

    <!--配置Activiti使用的processEngine对象-->
    <bean id="processEngineConfiguration"
class="org.activiti.engine.impl.cfg.StandaloneProcessEngineConfigu
ration">
        <!--注入数据源-->
        <property name="dataSource" ref="dataSource"/>
        <!--自动的生成Activiti数据库中的表-->
        <property name="databaseSchemaUpdate" value="true"/>
    </bean>

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

```

```
27      <!--RepositoryService-->
28      <bean id="repositoryService" factory-bean="processEngine"
    factory-method="getRepositoryService" />
29
30      <!--RuntimeService-->
31      <bean id="runtimeService" factory-bean="processEngine"
    factory-method="getRuntimeService" />
32
33      <!--TaskService-->
34      <bean id="taskService" factory-bean="processEngine" factory-
    method="getTaskService" />
35
36      <!--HistoryService-->
37      <bean id="historyService" factory-bean="processEngine"
    factory-method="getHistoryService" />
38
39
40  </beans>
```

- 启动Spring工程并验证

2.Springboot中使用Activiti

- 引入依赖

```
1           <!--activiti驱动-->
2           <dependency>
3               <groupId>org.activiti</groupId>
4               <artifactId>activiti-spring-boot-starter</artifactId>
5               <version>7.0.0.Beta1</version>
6           </dependency>
7
8           <!--mysql驱动-->
9           <dependency>
10              <groupId>mysql</groupId>
11              <artifactId>mysql-connector-java</artifactId>
```

```
        </dependency>

        <!--druid连接-->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid-spring-boot-starter</artifactId>
            <version>1.1.10</version>
        </dependency>
```

- 编写配置文件

```
# 应用名称
spring.application.name=spring-boot-activiti-demo
# 应用服务 WEB 访问端口
server.port=8080
# 配置数据源
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/activiti?serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=qf123456

# 配置activiti
# true：activiti会对数据库中所有表进行更新操作。如果表不存在，则自动创建。
# flase： 默认值，activiti在启动时，会对比数据库表中保存的版本，如果没有表或者版本不匹配，将抛出异常。
spring.activiti.database-schema-update=true

# activiti7需要手动开启历史记录
spring.activiti.db-history-used=true

# 配置历史记录级别
# none：不保存任何的历史数据，因此，在流程执行过程中，这是最高效的。
# activity：级别高于none，保存流程实例与流程行为，其他数据不保存。
# audit：除activity级别会保存的数据外，还会保存全部的流程任务及其属性。audit为history的默认值。
```

```
# full：保存历史数据的最高级别，除了会保存audit级别的数据外，还会保存其他全部流程相关的细节数据，包括一些流程参数等。
spring.activiti.history-level=full

```

- 复制流程文件

将流程图文件bpmn和png复制到`resources/processes`文件夹内，Activiti7框架将自动进行部署。

3.整合Spring Security

- 创建Spring Security工具类

```
package com.qf.my.spring.boot.activiti.demo;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.GrantedAuthority;
import
org.springframework.security.core.context.SecurityContextHolder;
import
org.springframework.security.core.context.SecurityContextImpl;
import org.springframework.security.core.userdetails.UserDetails;
import
org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.stereotype.Component;

import java.util.Collection;

/**
 * @author Thor
 * @公众号 Java架构栈
 */
@Component
public class SecurityUtil {

```

```
22
23      @Autowired
24      private UserDetailsService userDetailsService;
25
26      public void logInAs(String username) {
27
28          UserDetails user =
    userDetailsService.loadUserByUsername(username);
29          if (user == null) {
30              throw new IllegalStateException("User " + username + "
    doesn't exist, please provide a valid user");
31          }
32          SecurityContextHolder.setContext(new
    SecurityContextImpl(new Authentication() {
33              @Override
34              public Collection<? extends GrantedAuthority>
    getAuthorities() {
35                  return user.getAuthorities();
36              }
37
38              @Override
39              public Object getCredentials() {
40                  return user.getPassword();
41              }
42
43              @Override
44              public Object getDetails() {
45                  return user;
46              }
47
48              @Override
49              public Object getPrincipal() {
50                  return user;
51              }
52
53              @Override
54              public boolean isAuthenticated() {
```

```
55                  return true;
56              }
57
58              @Override
59              public void setAuthenticated(boolean isAuthenticated)
    throws IllegalArgumentException {
60
61              }
62
63              @Override
64              public String getName() {
65                  return user.getUsername();
66              }
67          }));
68
     org.activiti.engine.impl.identity.Authentication.setAuthenticated
    UserId(username);
69      }
70
71  }
72
```

- 创建Security配置类

```
1  package com.qf.my.spring.boot.activiti.demo;
2
3  import org.slf4j.LoggerFactory;
4  import org.springframework.context.annotation.Bean;
5  import org.springframework.context.annotation.Configuration;
6  import
   org.springframework.security.core.authority.SimpleGrantedAuthority
   ;
7  import org.springframework.security.core.userdetails.User;
8  import
   org.springframework.security.core.userdetails.UserDetailsService;
9  import
   org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
```

```
10  import
    org.springframework.security.crypto.password.PasswordEncoder;
11  import
    org.springframework.security.provisioning.InMemoryUserDetailsManag
    er;
12
13  import java.util.Arrays;
14  import java.util.List;
15  import java.util.logging.Logger;
16  import java.util.stream.Collectors;
17
18  /**
19   * @author Thor
20   * @公众号 Java架构栈
21   */
22  @Configuration
23  public class SecurityConfig {
24
25      @Bean
26      public UserDetailsService myUserDetailsService() {
27
28          InMemoryUserDetailsManager inMemoryUserDetailsManager =
    new InMemoryUserDetailsManager();
29
30          //用户
31          String[][] usersGroupsAndRoles = {
32                  {"王工", "password", "ROLE_ACTIVITI_USER"},
33                  {"李部门经理", "password", "ROLE_ACTIVITI_USER"},
34                  {"张总经理", "password", "ROLE_ACTIVITI_USER"},
35                  {"孙人力总监", "password", "ROLE_ACTIVITI_USER"}
36          };
37
38          for (String[] user : usersGroupsAndRoles) {
39              List<String> authoritiesStrings =
    Arrays.asList(Arrays.copyOfRange(user, 2, user.length));
40              inMemoryUserDetailsManager.createUser(new
    User(user[0], passwordEncoder().encode(user[1]),
```

```
                    authoritiesStrings.stream().map(s -> new
 SimpleGrantedAuthority(s)).collect(Collectors.toList())));
        }


        return inMemoryUserDetailsManager;

    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}

```

4.创建流程实例

```
    //启动流程实例
    @Test
    public void test2(){
        securityUtil.logInAs("王工");
        ProcessInstance processInstance =
processRuntime.start(ProcessPayloadBuilder.start()
                .withProcessDefinitionKey("myProcess_1")
                .build());
        System.out.println("流程实例的id:"+processInstance.getId());
    }
```

5.执行任务

```
    //完成任务
    @Test
    public void test3(){
        securityUtil.logInAs("王工");
        //获得当前登陆用户的待处理的前10个任务
        Page<Task> tasks = taskRuntime.tasks(Pageable.of(0,10));
```

```
7           if(tasks!=null && tasks.getTotalItems()>0){
8               for (Task task : tasks.getContent()) {
9                   //执行任务
10                  taskRuntime.complete(TaskPayloadBuilder.complete()
11                          .withTaskId(task.getId()).build());
12                  System.out.println("任务执行完成");
13              }
14          }
15
16      }
```

千锋教育Java教研院 关注公众号【Java架构栈】下载所有课程代码课件及工具 让技术回归本该有的纯静!