

Activiti7

一、工作流介绍

1.1 概念

工作流(Workflow)，就是通过计算机对业务流程自动化执行管理。它主要解决的是“使在多个参与者之间按照某种预定义的规则自动进行传递文档、信息或任务的过程，从而实现某个预期的业务目标，或者促使此目标的实现”。

1.2 工作流系统

一个软件系统中具有工作流的功能，我们把它称为工作流系统，一个系统中工作流的功能是什么？就是对系统的业务流程进行自动化管理，所以工作流是建立在业务流程的基础上，所以一个软件的系统核心根本上还是系统的业务流程，工作流只是协助进行业务流程管理。即使没有工作流业务系统也可以开发运行，只不过有了工作流可以更好的管理业务流程，提高系统的可扩展性。

1.3 适用行业

消费品行业，制造业，电信服务业，银证险等金融服务业，物流服务业，物业服务业，物业管理，大中型进出口贸易公司，政府事业机构，研究院所及教育服务业等，特别是大的跨国企业和集团公司。

1.4 具体应用

- 1、关键业务流程：订单、报价处理、合同审核、客户电话处理、供应链管理等
- 2、行政管理类：出差申请、加班申请、请假申请、用车申请、各种办公用品申请、购买申请、日报周报等凡是原来手工流转处理的行政表单。
- 3、人事管理类：员工培训安排、绩效考评、职位变动处理、员工档案信息管理等。
- 4、财务相关类：付款请求、应收款处理、日常报销处理、出差报销、预算和计划申请等。
- 5、客户服务类：客户信息管理、客户投诉、请求处理、售后服务管理等。
- 6、特殊服务类：ISO系列对应流程、质量管理对应流程、产品数据信息管理、贸易公司报关处理、物流公司货物跟踪处理等各种通过表单逐步手工流转完成的任务均可应用工作流软件自动规范地实施。

1.5 实现方式

在没有专门的工作流引擎之前，我们之前为了实现流程控制，通常的做法就是采用状态字段的值来跟踪流程的变化情况。这样不用角色的用户，通过状态字段的取值来决定记录是否显示。

针对有权限可以查看的记录，当前用户根据自己的角色来决定审批是否合格的操作。如果合格将状态字段设置一个值，来代表合格；当然如果不合格也需要设置一个值来代表不合格的情况。

这是一种最为原始的方式。通过状态字段虽然做到了流程控制，但是当我们的流程发生变更的时候，这种方式所编写的代码也要进行调整。

那么有没有专业的方式来实现工作流的管理呢？并且可以做到业务流程变化之后，我们的程序可以不用改变，如果可以实现这样的效果，那么我们的业务系统的适应能力就得到了极大提升。

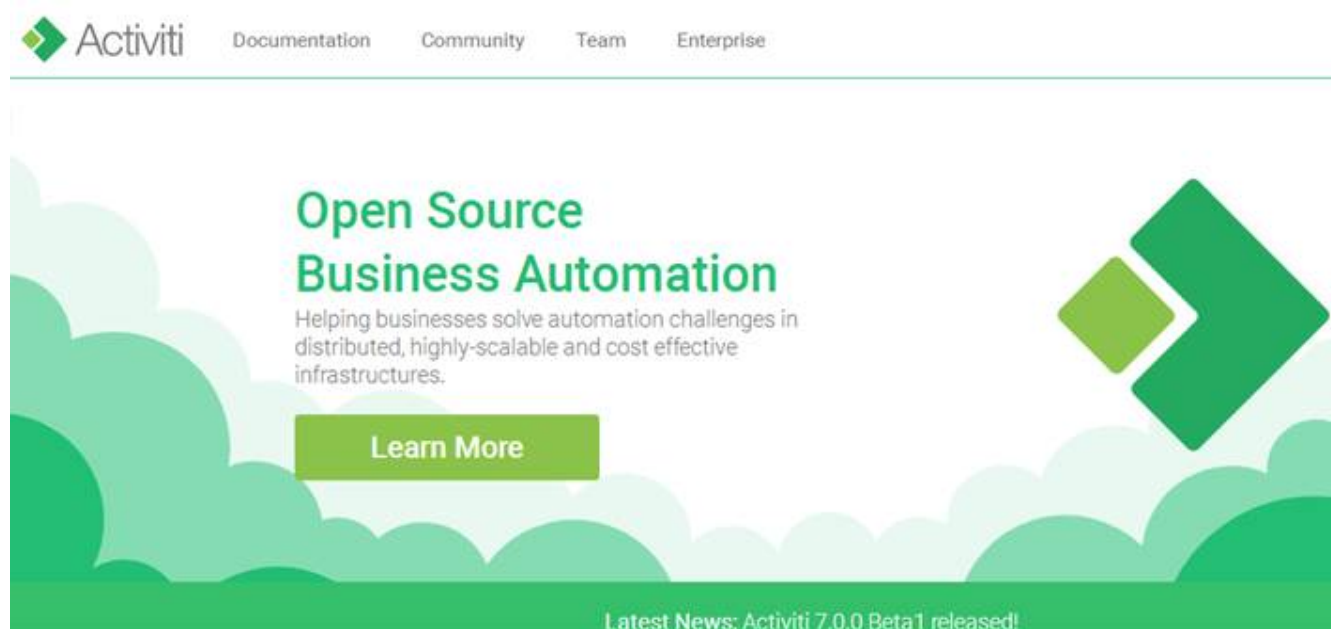
二、Activiti7概述

2.1 介绍

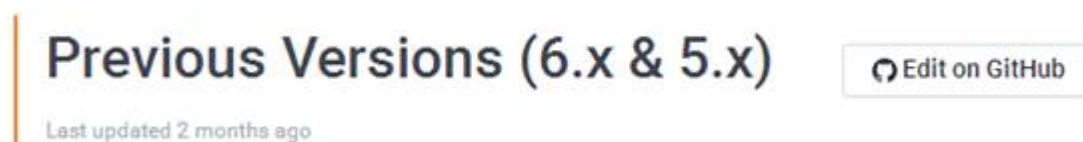
Alfresco软件在2010年5月17日宣布Activiti业务流程管理（BPM）开源项目的正式启动，其首席架构师由业务流程管理BPM的专家 Tom Baeyens担任，Tom Baeyens就是原来jbpm的架构师，而jbpm是一个非常有名的工作流引擎，当然activiti也是一个工作流引擎。

Activiti是一个工作流引擎，activiti可以将业务系统中复杂的业务流程抽取出来，使用专门的建模语言BPMN2.0进行定义，业务流程按照预先定义的流程进行执行，实现了系统的流程由activiti进行管理，减少业务系统由于流程变更进行系统升级改造的工作量，从而提高系统的健壮性，同时也减少了系统开发维护成本。

官方网站：<https://www.activiti.org/>



经历版本:



目前最新版本：Activiti7.0.0.Beta

2.1.1 BPM

BPM (Business Process Management) , 即业务流程管理, 是一种规范化的构造端到端的业务流程, 以持续的提高组织业务效率。常见商业管理教育如EMBA、MBA等都将BPM包含在内。

2.1.2 BPM软件

BPM软件就是根据企业中业务环境的变化, 推进人与人之间、人与系统之间以及系统与系统之间的整合及调整的经营方法与解决方案的IT工具。

通过BPM软件对企业内部及外部的业务流程的整个生命周期进行建模、自动化、管理监控和优化, 使企业成本降低, 利润得以大幅提升。

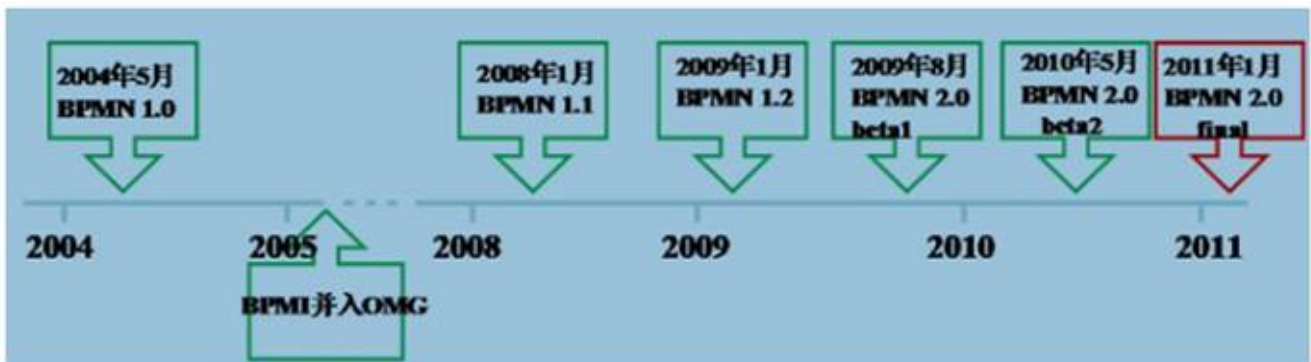
BPM软件在企业中应用领域广泛, 凡是有业务流程的地方都可以BPM软件进行管理, 比如企业人事办公管理、采购流程管理、公文审批流程管理、财务管理等。

2.1.3 BPMN

BPMN (Business Process Model And Notation) - 业务流程模型和符号 是由BPMI (Business Process Management Initiative) 开发的一套标准的业务流程建模符号, 使用BPMN提供的符号可以创建业务流程。

2004年5月发布了BPMN1.0规范.BPMI于2005年9月并入OMG (The Object Management Group对象管理组织)组织。OMG于2011年1月发布BPMN2.0的最终版本。

具体发展历史如下:



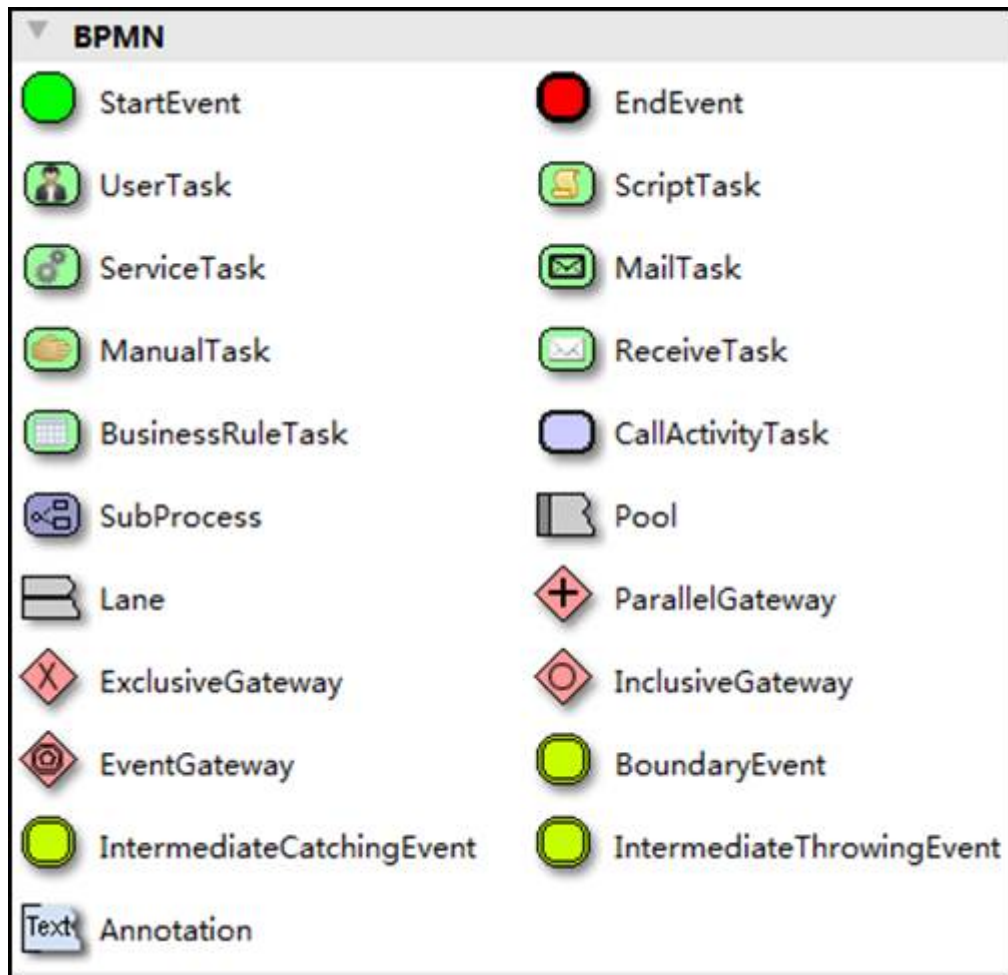
BPMN 是目前被各 BPM 厂商广泛接受的 BPM 标准。Activiti 就是使用 BPMN 2.0 进行流程建模、流程执行管理, 它包括很多的建模符号, 比如:

Event

用一个圆圈表示, 它是流程中运行过程中发生的事情。



活动用圆角矩形表示，一个流程由一个活动或多个活动组成



Bpmn图形其实是通过xml表示业务流程，上边的.bpmn文件使用文本编辑器打开：

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:activiti="http://activiti.org/bpmn"
    xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
    xmlns:omgdc="http://www.omg.org/spec/DD/20100524/DC"
    xmlns:omgdi="http://www.omg.org/spec/DD/20100524/DI"
    typeLanguage="http://www.w3.org/2001/XMLSchema"
    expressionLanguage="http://www.w3.org/1999/XPath"
    targetNamespace="http://www.activiti.org/test">
3    <process id="myProcess" name="My process" isExecutable="true">

```

```

4      <startEvent id="startevent1" name="Start"></startEvent>
5      <userTask id="usertask1" name="创建请假单"></userTask>
6      <sequenceFlow id="flow1" sourceRef="startevent1" targetRef="usertask1"></sequenceFlow>
7      <userTask id="usertask2" name="部门经理审核"></userTask>
8      <sequenceFlow id="flow2" sourceRef="usertask1" targetRef="usertask2"></sequenceFlow>
9      <userTask id="usertask3" name="人事复核"></userTask>
10     <sequenceFlow id="flow3" sourceRef="usertask2" targetRef="usertask3"></sequenceFlow>
11     <endEvent id="endevent1" name="End"></endEvent>
12     <sequenceFlow id="flow4" sourceRef="usertask3" targetRef="endevent1"></sequenceFlow>
13 </process>
14 <bpmndi:BPMNDiagram id="BPMNDiagram_myProcess">
15     <bpmndi:BPMNPlane bpmnElement="myProcess" id="BPMNPlane_myProcess">
16         <bpmndi:BPMNShape bpmnElement="startevent1" id="BPMNShape_startevent1">
17             <omgdc:Bounds height="35.0" width="35.0" x="130.0" y="160.0"></omgdc:Bounds>
18         </bpmndi:BPMNShape>
19         <bpmndi:BPMNShape bpmnElement="usertask1" id="BPMNShape_usertask1">
20             <omgdc:Bounds height="55.0" width="105.0" x="210.0" y="150.0"></omgdc:Bounds>
21         </bpmndi:BPMNShape>
22         <bpmndi:BPMNShape bpmnElement="usertask2" id="BPMNShape_usertask2">
23             <omgdc:Bounds height="55.0" width="105.0" x="360.0" y="150.0"></omgdc:Bounds>
24         </bpmndi:BPMNShape>
25         <bpmndi:BPMNShape bpmnElement="usertask3" id="BPMNShape_usertask3">
26             <omgdc:Bounds height="55.0" width="105.0" x="510.0" y="150.0"></omgdc:Bounds>
27         </bpmndi:BPMNShape>
28         <bpmndi:BPMNShape bpmnElement="endevent1" id="BPMNShape_endevent1">
29             <omgdc:Bounds height="35.0" width="35.0" x="660.0" y="160.0"></omgdc:Bounds>
30         </bpmndi:BPMNShape>
31         <bpmndi:BPMNEdge bpmnElement="flow1" id="BPMNEdge_flow1">
32             <omgdi:waypoint x="165.0" y="177.0"></omgdi:waypoint>
33             <omgdi:waypoint x="210.0" y="177.0"></omgdi:waypoint>
34         </bpmndi:BPMNEdge>
35         <bpmndi:BPMNEdge bpmnElement="flow2" id="BPMNEdge_flow2">
36             <omgdi:waypoint x="315.0" y="177.0"></omgdi:waypoint>
37             <omgdi:waypoint x="360.0" y="177.0"></omgdi:waypoint>
38         </bpmndi:BPMNEdge>
39         <bpmndi:BPMNEdge bpmnElement="flow3" id="BPMNEdge_flow3">
40             <omgdi:waypoint x="465.0" y="177.0"></omgdi:waypoint>
41             <omgdi:waypoint x="510.0" y="177.0"></omgdi:waypoint>
42         </bpmndi:BPMNEdge>
43         <bpmndi:BPMNEdge bpmnElement="flow4" id="BPMNEdge_flow4">
44             <omgdi:waypoint x="615.0" y="177.0"></omgdi:waypoint>
45             <omgdi:waypoint x="660.0" y="177.0"></omgdi:waypoint>
46         </bpmndi:BPMNEdge>
47     </bpmndi:BPMNPlane>
48 </bpmndi:BPMNDiagram>
49 </definitions>
50

```

2.2 使用步骤

部署activiti

Activiti是一个工作流引擎（其实就是一堆jar包API），业务系统访问(操作)activiti的接口，就可以方便的操作流程相关数据，这样就可以把工作流环境与业务系统的环境集成在一起。

流程定义

使用activiti流程建模工具(activity-designer)定义业务流程(.bpmn文件)。

.bpmn文件就是业务流程定义文件，通过xml定义业务流程。

流程定义部署

activiti部署业务流程定义（.bpmn文件）。

使用activiti提供的api把流程定义内容存储起来，在Activiti执行过程中可以查询定义的内容

Activiti执行把流程定义内容存储在数据库中

启动一个流程实例

流程实例也叫：ProcessInstance

启动一个流程实例表示开始一次业务流程的运行。

在员工请假流程定义部署完成后，如果张三要请假就可以启动一个流程实例，如果李四要请假也启动一个流程实例，两个流程的执行互相不影响。

用户查询待办任务(Task)

因为现在系统的业务流程已经交给activiti管理，通过activiti就可以查询当前流程执行到哪了，当前用户需要办理什么任务了，这些activiti帮我们管理了，而不需要开发人员自己编写在sql语句查询。

用户办理任务

用户查询待办任务后，就可以办理某个任务，如果这个任务办理完成还需要其它用户办理，比如采购单创建后由部门经理审核，这个过程也是由activiti帮我们完成了。

流程结束

当任务办理完成没有下一个任务结点了，这个流程实例就完成了。

三、Activiti环境

3.1 开发环境

Jdk1.8或以上版本

Mysql 5及以上的版本

Tomcat8.5

IDEA

注意：activiti的流程定义工具插件可以安装在IDEA下，也可以安装在Eclipse工具下

3.2 Activiti环境

我们使用：Activiti7.0.0.Beta1 默认支持spring5

3.2.1 下载activiti7

Activiti下载地址：<http://activiti.org/download.html>，Maven的依赖如下：

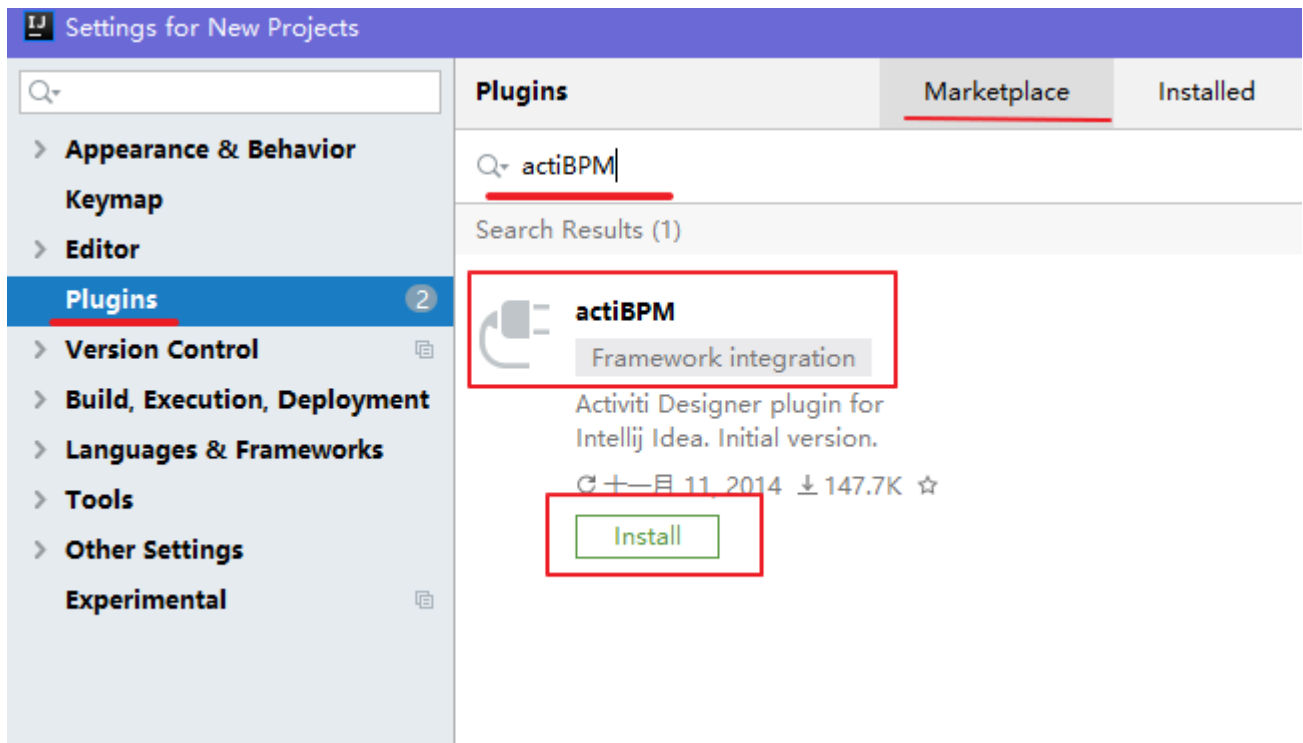
```
1  <dependencyManagement>
2      <dependencies>
3          <dependency>
4              <groupId>org.activiti</groupId>
5              <artifactId>activiti-dependencies</artifactId>
6              <version>7.0.0.Beta1</version>
7              <scope>import</scope>
8              <type>pom</type>
9          </dependency>
10     </dependencies>
11 </dependencyManagement>
```

1) Database：

activiti运行需要有数据库的支持，支持的数据库有：h2, mysql, oracle, postgres, mssql, db2。

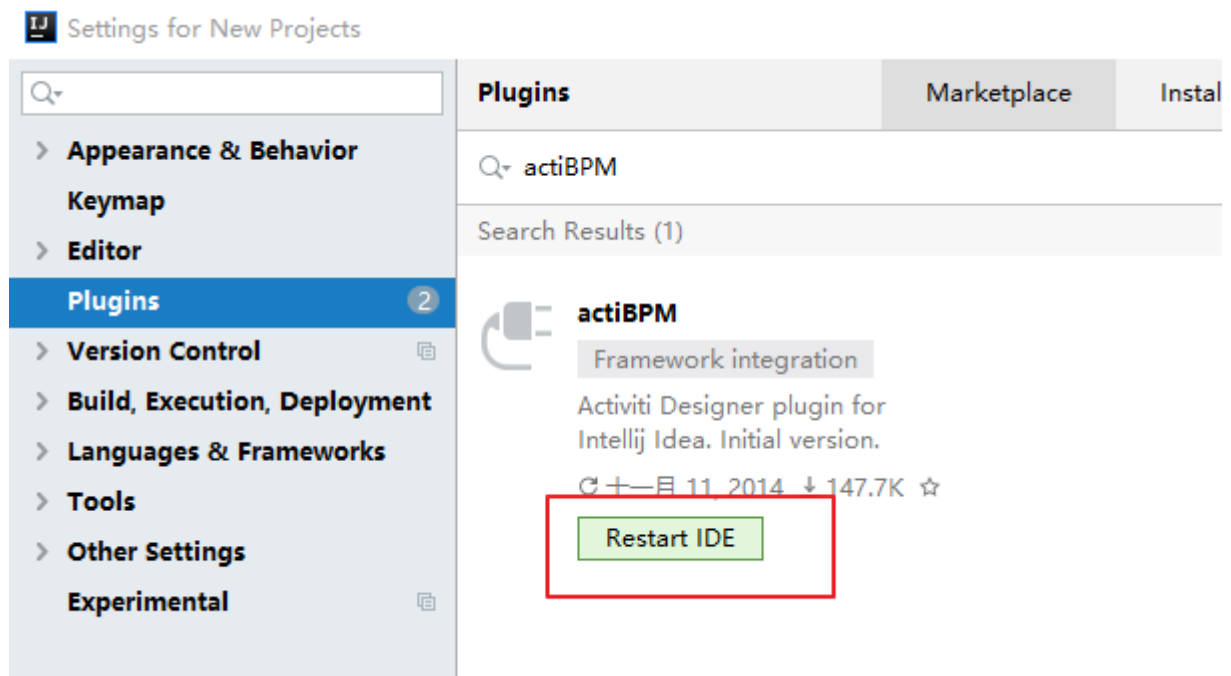
3.2.2 流程设计器IDEA下安装

在IDEA的File菜单中找到子菜单”Settings”,后面我们再选择左侧的“plugins”菜单，如下图所示：



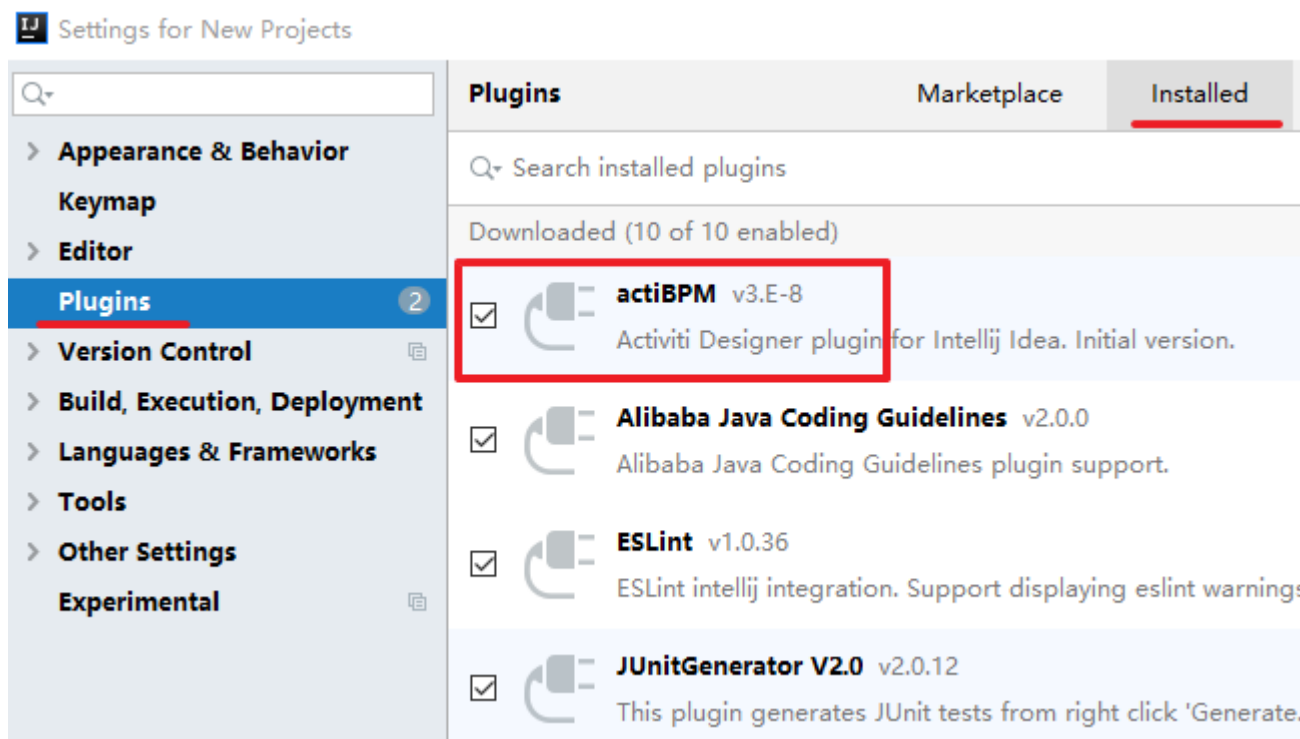
此时我们就可以搜索到actiBPM插件，它就是Activiti Designer的IDEA版本，我们点击Install安装。

安装好后，页面如下：



提示需要重启idea，点击重启。

重启完成后，再次打开Settings 下的 Plugins（插件列表），点击右侧的Installed（已安装的插件），在列表中看到actiBPM，就说明已经安装成功了，如下图所示：



后面的课程里，我们会使用这个流程设计器进行Activiti的流程设计。

3.3 Activiti的数据库支持

Activiti 在运行时需要数据库的支持，使用25张表，把流程定义节点内容读取到数据库表中，以供后续使用。

3.3.1 Activiti 支持的数据库

activiti 支持的数据库和版本如下：

数据库类型	版本	JDBC连接示例	说明
h2	1.3.168	jdbc:h2:tcp://localhost/activiti	默认配置的数据库
mysql	5.1.21	jdbc:mysql://localhost:3306/activiti? autoReconnect=true	使用 mysql-connector-java 驱动测试
oracle	11.2.0.1.0	jdbc:oracle:thin:@localhost:1521:xe	
postgres	8.1	jdbc:postgresql://localhost:5432/activiti	
db2	DB2 10.1 using db2jcc4	jdbc:db2://localhost:50000/activiti	
mssql	2008 using sqljdbc4	jdbc:sqlserver://localhost:1433/activiti	

3.3.2 在MySQL生成表

3.3.2.1 创建数据库

创建 mysql 数据库 activiti （名字任意）：

```
CREATE DATABASE activiti DEFAULT CHARACTER SET utf8;
```

3.3.2.2 使用java代码生成表

1) 创建 java 工程

使用idea 创建 java 的maven工程，取名：activiti01。

2) 加入 maven 依赖的坐标（jar 包）

首先需要在 java 工程中加入 ProcessEngine 所需要的 jar 包，包括：

1) activiti-engine-7.0.0.beta1.jar 2) activiti 依赖的 jar 包： mybatis、 alf4j、 log4j 等

3) activiti 依赖的 spring 包

4) mysql数据库驱动

5) 第三方数据连接池 dbcp 6) 单元测试 Junit-4.12.jar

我们使用 maven 来实现项目的构建，所以应当导入这些 jar 所对应的坐标到 pom.xml 文件中。

完整的依赖内容如下：

```
1  <properties>
2      <slf4j.version>1.6.6</slf4j.version>
3      <log4j.version>1.2.12</log4j.version>
4      <activiti.version>7.0.0.Beta1</activiti.version>
5  </properties>
6  <dependencies>
7      <dependency>
8          <groupId>org.activiti</groupId>
9          <artifactId>activiti-engine</artifactId>
10         <version>${activiti.version}</version>
11     </dependency>
12     <dependency>
13         <groupId>org.activiti</groupId>
14         <artifactId>activiti-spring</artifactId>
15         <version>${activiti.version}</version>
16     </dependency>
17     <!-- bpmn 模型处理 -->
18     <dependency>
19         <groupId>org.activiti</groupId>
20         <artifactId>activiti-bpmn-model</artifactId>
21         <version>${activiti.version}</version>
22     </dependency>
23     <!-- bpmn 转换 -->
24     <dependency>
25         <groupId>org.activiti</groupId>
26         <artifactId>activiti-bpmn-converter</artifactId>
27         <version>${activiti.version}</version>
28     </dependency>
29     <!-- bpmn json数据转换 -->
```

```
30     <dependency>
31         <groupId>org.activiti</groupId>
32         <artifactId>activiti-json-converter</artifactId>
33         <version>${activiti.version}</version>
34     </dependency>
35     <!-- bpmn 布局 -->
36     <dependency>
37         <groupId>org.activiti</groupId>
38         <artifactId>activiti-bpmn-layout</artifactId>
39         <version>${activiti.version}</version>
40     </dependency>
41     <!-- activiti 云支持 -->
42     <dependency>
43         <groupId>org.activiti.cloud</groupId>
44         <artifactId>activiti-cloud-services-api</artifactId>
45         <version>${activiti.version}</version>
46     </dependency>
47     <!-- mysql驱动 -->
48     <dependency>
49         <groupId>mysql</groupId>
50         <artifactId>mysql-connector-java</artifactId>
51         <version>5.1.40</version>
52     </dependency>
53     <!-- mybatis -->
54     <dependency>
55         <groupId>org.mybatis</groupId>
56         <artifactId>mybatis</artifactId>
57         <version>3.4.5</version>
58     </dependency>
59     <!-- 链接池 -->
60     <dependency>
61         <groupId>commons-dbcp</groupId>
62         <artifactId>commons-dbcp</artifactId>
63         <version>1.4</version>
64     </dependency>
65     <dependency>
66         <groupId>junit</groupId>
67         <artifactId>junit</artifactId>
68         <version>4.12</version>
69     </dependency>
70     <!-- log start -->
71     <dependency>
72         <groupId>log4j</groupId>
73         <artifactId>log4j</artifactId>
74         <version>${log4j.version}</version>
75     </dependency>
76     <dependency>
77         <groupId>org.slf4j</groupId>
78         <artifactId>slf4j-api</artifactId>
79         <version>${slf4j.version}</version>
80     </dependency>
81     <dependency>
82         <groupId>org.slf4j</groupId>
```

```

83         <artifactId>slf4j-log4j12</artifactId>
84         <version>${slf4j.version}</version>
85     </dependency>
86 </dependencies>

```

3) 添加log4j日志配置

我们使用log4j日志包，可以对日志进行配置

在resources 下创建log4j.properties

```

1  # Set root category priority to INFO and its only appender to CONSOLE.
2  #log4j.rootCategory=INFO, CONSOLE debug info warn error fatal
3  log4j.rootCategory=debug, CONSOLE, LOGFILE
4  # Set the enterprise logger category to FATAL and its only appender to CONSOLE.
5  log4j.logger.org.apache.axis.enterprise=FATAL, CONSOLE
6  # CONSOLE is set to be a ConsoleAppender using a PatternLayout.
7  log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender
8  log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout
9  log4j.appender.CONSOLE.layout.ConversionPattern=%d{ISO8601} %-6r[%15.15t] %-5p %30.30c %x -
    %m\n
10 # LOGFILE is set to be a File appender using a PatternLayout.
11 log4j.appender.LOGFILE=org.apache.log4j.FileAppender
12 log4j.appender.LOGFILE.File=f:\act\activiti.log
13 log4j.appender.LOGFILE.Append=true
14 log4j.appender.LOGFILE.layout=org.apache.log4j.PatternLayout
15 log4j.appender.LOGFILE.layout.ConversionPattern=%d{ISO8601} %-6r[%15.15t] %-5p %30.30c %x -
    %m\n

```

4) 添加activiti配置文件

我们使用activiti提供的默认方式来创建mysql的表。

默认方式的要求是在 resources 下创建 activiti.cfg.xml 文件，注意：默认方式目录和文件名不能修改，因为activiti的源码中已经设置，到固定的目录读取固定文件名的文件。

```

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  xmlns:context="http://www.springframework.org/schema/context"
5  xmlns:tx="http://www.springframework.org/schema/tx"
6  xsi:schemaLocation="http://www.springframework.org/schema/beans
7  http://www.springframework.org/schema/beans/spring-beans.xsd
8  http://www.springframework.org/schema/context
9  http://www.springframework.org/schema/context/spring-context.xsd
10 http://www.springframework.org/schema/tx
11 http://www.springframework.org/schema/tx/spring-tx.xsd">
12 </beans>

```

5) 在 activiti.cfg.xml 中进行配置

默认方式要在在activiti.cfg.xml中bean的名字叫processEngineConfiguration，名字不可修改

在这里有2中配置方式：一种是单独配置数据源，一种是不单独配置数据源

1、直接配置processEngineConfiguration

processEngineConfiguration 用来创建 ProcessEngine，在创建 ProcessEngine 时会执行数据库的操作。

```
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        xmlns:context="http://www.springframework.org/schema/context"
5        xmlns:tx="http://www.springframework.org/schema/tx"
6        xsi:schemaLocation="http://www.springframework.org/schema/beans
7                             http://www.springframework.org/schema/beans/spring-beans.xsd
8                             http://www.springframework.org/schema/context
9                             http://www.springframework.org/schema/context/spring-context.xsd
10                            http://www.springframework.org/schema/tx
11                            http://www.springframework.org/schema/tx/spring-tx.xsd">
12    <!-- 默认id对应的值 为processEngineConfiguration -->
13    <!-- processEngine Activiti的流程引擎 -->
14    <bean id="processEngineConfiguration"
15          class="org.activiti.engine.impl.cfg.StandaloneProcessEngineConfiguration">
16        <property name="jdbcDriver" value="com.mysql.jdbc.Driver"/>
17        <property name="jdbcUrl" value="jdbc:mysql:///activiti"/>
18        <property name="jdbcUsername" value="root"/>
19        <property name="jdbcPassword" value="123456"/>
20        <!-- activiti数据库表处理策略 -->
21        <property name="databaseSchemaUpdate" value="true"/>
22    </bean>
23 </beans>
```

2、配置数据源后，在processEngineConfiguration 引用

首先配置数据源

```
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        xmlns:context="http://www.springframework.org/schema/context"
5        xmlns:tx="http://www.springframework.org/schema/tx"
6        xsi:schemaLocation="http://www.springframework.org/schema/beans
7                             http://www.springframework.org/schema/beans/spring-beans.xsd
8                             http://www.springframework.org/schema/context
9                             http://www.springframework.org/schema/context/spring-context.xsd
10                            http://www.springframework.org/schema/tx
11                            http://www.springframework.org/schema/tx/spring-tx.xsd">
12
13    <!-- 这里可以使用 链接池 dbcp -->
14    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
15        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
16        <property name="url" value="jdbc:mysql:///activiti" />
17        <property name="username" value="root" />
18        <property name="password" value="123456" />
19        <property name="maxActive" value="3" />
20        <property name="maxIdle" value="1" />
21    </bean>
22
```

```

23     <bean id="processEngineConfiguration"
24           class="org.activiti.engine.impl.cfg.StandaloneProcessEngineConfiguration">
25       <!-- 引用数据源 上面已经设置好了-->
26       <property name="dataSource" ref="dataSource" />
27       <!-- activiti数据库表处理策略 -->
28       <property name="databaseSchemaUpdate" value="true" />
29     </bean>
30 </beans>

```

6) java类编写程序生成表

创建一个测试类，调用activiti的工具类，生成activiti需要的数据库表。

直接使用activiti提供的工具类ProcessEngines，会默认读取classpath下的activiti.cfg.xml文件，读取其中的数据库配置，创建 ProcessEngine，在创建ProcessEngine 时会自动创建表。

代码如下：

```

1  package com.itheima.activiti01.test;
2
3  import org.activiti.engine.ProcessEngine;
4  import org.activiti.engine.ProcessEngineConfiguration;
5  import org.junit.Test;
6
7  public class TestDemo {
8      /**
9       * 生成 activiti的数据库表
10     */
11     @Test
12     public void testCreateDbTable() {
13         //使用classpath下的activiti.cfg.xml中的配置创建processEngine
14         ProcessEngine processEngine = ProcessEngines.getDefaultProcessEngine();
15         System.out.println(processEngine);
16     }
17 }
18

```

说明： 1、运行以上程序段即可完成 activiti 表创建，通过改变 activiti.cfg.xml 中databaseSchemaUpdate 参数的值执行不同的数据表处理策略。 2、上边的方法 getDefaultProcessEngine方法在执行时，从activiti.cfg.xml 中找固定的名称 processEngineConfiguration 。

在测试程序执行过程中，idea的控制台会输出日志，说明程序正在创建数据表，类似如下,注意红线内容：

```

2019-11-04 15:14:01,631 4798 [      main] DEBUG mpl.interceptor.LogInterceptor - --- starting SchemaOperationsProcessEngineBuild -----
2019-11-04 15:14:01,636 4803 [      main] DEBUG da.DefaultActivitiEngineAgenda - Operation class org.activiti.engine.impl.interceptor.(
2019-11-04 15:14:01,647 4814 [      main] DEBUG ti.engine.impl.db.DbSqlSession - Executing performSchemaOperationsProcessEngineBuild w:
2019-11-04 15:14:01,647 4814 [      main] DEBUG ansaction.jdbc.JdbcTransaction - Opening JDBC Connection
2019-11-04 15:14:01,649 4816 [      main] DEBUG ansaction.jdbc.JdbcTransaction - Setting autocommit to false on JDBC Connection [jdbc:
2019-11-04 15:14:01,659 4826 [      main] INFO  ti.engine.impl.db.DbSqlSession - performing create on engine with resource org/activit:
2019-11-04 15:14:01,659 4826 [      main] INFO  ti.engine.impl.db.DbSqlSession - Found MySQL: majorVersion=5 minorVersion=7
2019-11-04 15:14:01,661 4828 [      main] DEBUG ti.engine.impl.db.DbSqlSession - SQL: create table ACT_GE_PROPERTY (
NAME_ varchar(64),
VALUE_ varchar(300),
REV_ integer,
primary key (NAME_)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE utf8_bin
2019-11-04 15:14:01,720 4887 [      main] DEBUG ti.engine.impl.db.DbSqlSession - SQL: insert into ACT_GE_PROPERTY
values ('schema.version', '7.0.0.0', 1)
2019-11-04 15:14:01,724 4891 [      main] DEBUG ti.engine.impl.db.DbSqlSession - SQL: insert into ACT_GE_PROPERTY
values ('schema.history', 'create(7.0.0.0)', 1)
2019-11-04 15:14:01,725 4892 [      main] DEBUG ti.engine.impl.db.DbSqlSession - SQL: insert into ACT_GE_PROPERTY

```

执行完成后我们查看数据库， 创建了 25 张表， 结果如下：



到这，我们就完成activiti运行需要的数据库和表的创建。

3.4 表结构介绍

3.4.1 表的命名规则和作用

看到刚才创建的表，我们发现Activiti 的表都以 ACT_ 开头。

第二部分是表示表的用途的两个字母标识。用途也和服务的 API 对应。ACT_RE：'RE'表示 repository。这个前缀的表包含了流程定义和流程静态资源（图片，规则，等等）。ACT_RU：'RU'表示 runtime。这些运行时的表，包含流程实例，任务，变量，异步任务，等运行中的数据。Activiti 只在流程实例执行过程中保存这些数据，在流程结束时就会删除这些记录。这样运行时表可以一直很小速度很快。ACT_HI：'HI'表示 history。这些表包含历史数据，比如

历史流程实例， 变量， 任务等等。 ACT_GE : GE 表示 general。 通用数据， 用于不同场景下

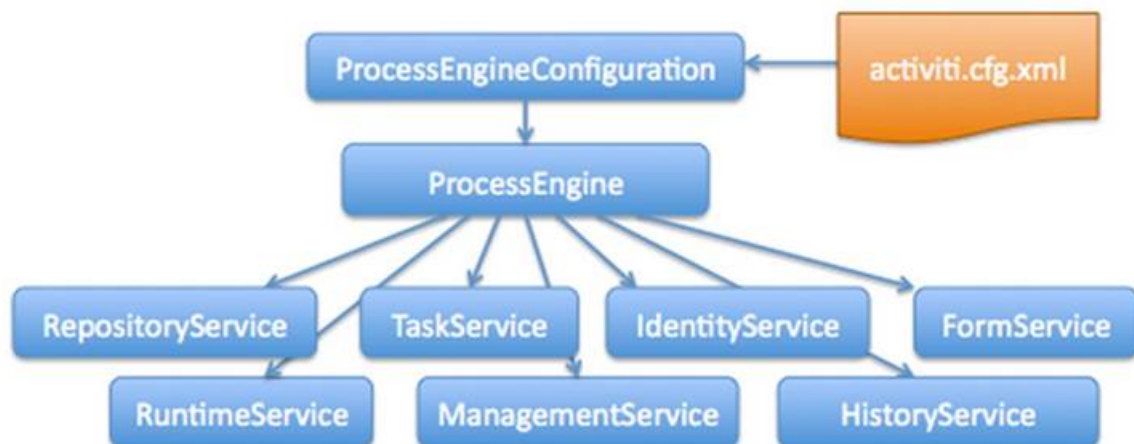
3.4.2 Activiti数据表介绍

表分类	表名	解释
一般数据		
	[ACT_GE_BYTEARRAY]	通用的流程定义和流程资源
	[ACT_GE_PROPERTY]	系统相关属性
流程历史记录		
	[ACT_HI_ACTINST]	历史的流程实例
	[ACT_HI_ATTACHMENT]	历史的流程附件
	[ACT_HI_COMMENT]	历史的说明性信息
	[ACT_HI_DETAIL]	历史的流程运行中的细节信息
	[ACT_HI_IDENTITYLINK]	历史的流程运行过程中用户关系
	[ACT_HI_PROCINST]	历史的流程实例
	[ACT_HI_TASKINST]	历史的任务实例
	[ACT_HI_VARINST]	历史的流程运行中的变量信息
流程定义表		
	[ACT_RE_DEPLOYMENT]	部署单元信息
	[ACT_RE_MODEL]	模型信息
	[ACT_RE_PROCDEF]	已部署的流程定义
运行实例表		
	[ACT_RU_EVENT_SUBSCR]	运行时事件
	[ACT_RU_EXECUTION]	运行时流程执行实例
	[ACT_RU_IDENTITYLINK]	运行时用户关系信息， 存储任务节点与参与者的相关信息
	[ACT_RU_JOB]	运行时作业
	[ACT_RU_TASK]	运行时任务
	[ACT_RU_VARIABLE]	运行时变量表

四、Activiti类关系图

上面我们完成了Activiti数据库表的生成， java代码中我们调用Activiti的工具类， 下面来了解Activiti的类关系

4.1 类关系图



在新版本中，我们通过实验可以发现IdentityService，FormService两个Service都已经删除了。

所以后面我们对于这两个Service也不讲解了，但老版本中还是有这两个Service，同学们需要了解一下

4.2 activiti.cfg.xml

activiti的引擎配置文件，包括：ProcessEngineConfiguration的定义、数据源定义、事务管理等，此文件其实就是一个spring配置文件。

4.3 流程引擎配置类

流程引擎的配置类（ProcessEngineConfiguration），通过ProcessEngineConfiguration可以创建工作流引擎ProceccEngine，常用的两种方法如下：

4.3.1 StandaloneProcessEngineConfiguration

使用StandaloneProcessEngineConfigurationActiviti可以单独运行，来创建ProcessEngine，Activiti会自己处理事务。

配置文件方式：

通常在activiti.cfg.xml配置文件中定义一个id为 processEngineConfiguration 的bean.

方法如下：

```
1 <bean id="processEngineConfiguration"
2     class="org.activiti.engine.impl.cfg.StandaloneProcessEngineConfiguration">
3     <!--配置数据库相关的信息-->
4     <!--数据库驱动-->
5     <property name="jdbcDriver" value="com.mysql.jdbc.Driver"/>
6     <!--数据库链接-->
7     <property name="jdbcUrl" value="jdbc:mysql:///activiti"/>
```

```

8      <!--数据库用户名-->
9      <property name="jdbcUsername" value="root" />
10     <!--数据库密码-->
11     <property name="jdbcPassword" value="123456" />
12     <!--activiti数据库表在生成时的策略 true - 如果数据库中已经存在相应的表，那么直接使用，如果不存
在，那么会创建-->
13     <property name="databaseSchemaUpdate" value="true" />
14 </bean>

```

还可以加入连接池:

```

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        xmlns:context="http://www.springframework.org/schema/context"
5        xmlns:tx="http://www.springframework.org/schema/tx"
6        xsi:schemaLocation="http://www.springframework.org/schema/beans
7                             http://www.springframework.org/schema/beans/spring-beans.xsd
8                             http://www.springframework.org/schema/context
9                             http://www.springframework.org/schema/context/spring-context.xsd
10                            http://www.springframework.org/schema/tx
11                            http://www.springframework.org/schema/tx/spring-tx.xsd">
12    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
13        <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
14        <property name="url" value="jdbc:mysql:///activiti"/>
15        <property name="username" value="root"/>
16        <property name="password" value="123456"/>
17        <property name="maxActive" value="3"/>
18        <property name="maxIdle" value="1"/>
19    </bean>
20    <!--在默认方式下 bean的id 固定为 processEngineConfiguration-->
21    <bean id="processEngineConfiguration"
22          class="org.activiti.engine.impl.cfg.StandaloneProcessEngineConfiguration">
23        <!--引入上面配置好的 链接池-->
24        <property name="dataSource" ref="dataSource"/>
25        <!--activiti数据库表在生成时的策略 true - 如果数据库中已经存在相应的表，那么直接使用，如果不存
在，那么会创建-->
26        <property name="databaseSchemaUpdate" value="true"/>
27    </bean>
28 </beans>

```

4.3.2 SpringProcessEngineConfiguration

通过org.activiti.spring.SpringProcessEngineConfiguration 与Spring整合。

创建spring与activiti的整合配置文件:

activity-spring.cfg.xml (名称可修改)

```

1  <beans xmlns="http://www.springframework.org/schema/beans"
2        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3        xmlns:mvc="http://www.springframework.org/schema/mvc"

```

```

3   xmlns:context="http://www.springframework.org/schema/context"
4   xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:tx="http://www.springframework.org/schema/tx"
5   xsi:schemaLocation="http://www.springframework.org/schema/beans
6       http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
7       http://www.springframework.org/schema/mvc
8       http://www.springframework.org/schema/mvc/spring-mvc-3.1.xsd
9       http://www.springframework.org/schema/context
10      http://www.springframework.org/schema/context/spring-context-3.1.xsd
11      http://www.springframework.org/schema/aop
12      http://www.springframework.org/schema/aop/spring-aop-3.1.xsd
13      http://www.springframework.org/schema/tx
14      http://www.springframework.org/schema/tx/spring-tx-3.1.xsd ">
15   <!--  workflow引擎配置bean -->
16   <bean id="processEngineConfiguration"
class="org.activiti.spring.SpringProcessEngineConfiguration">
17       <!-- 数据源 -->
18       <property name="dataSource" ref="dataSource" />
19       <!-- 使用spring事务管理器 -->
20       <property name="transactionManager" ref="transactionManager" />
21       <!-- 数据库策略 -->
22       <property name="databaseSchemaUpdate" value="drop-create" />
23       <!-- activiti的定时任务关闭 -->
24       <property name="jobExecutorActivate" value="false" />
25   </bean>
26   <!-- 流程引擎 -->
27   <bean id="processEngine" class="org.activiti.spring.ProcessEngineFactoryBean">
28       <property name="processEngineConfiguration" ref="processEngineConfiguration" />
29   </bean>
30   <!-- 资源服务service -->
31   <bean id="repositoryService" factory-bean="processEngine"
32       factory-method="getRepositoryService" />
33   <!-- 流程运行service -->
34   <bean id="runtimeService" factory-bean="processEngine"
35       factory-method="getRuntimeService" />
36   <!-- 任务管理服务 -->
37   <bean id="taskService" factory-bean="processEngine"
38       factory-method="getTaskService" />
39   <!-- 历史管理服务 -->
40   <bean id="historyService" factory-bean="processEngine" factory-
method="getHistoryService" />
41   <!-- 用户管理服务 -->
42   <bean id="identityService" factory-bean="processEngine" factory-
method="getIdentityService" />
43   <!-- 引擎管理服务 -->
44   <bean id="managementService" factory-bean="processEngine" factory-
method="getManagementService" />
45   <!-- 数据源 -->
46   <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
47       <property name="driverClassName" value="com.mysql.jdbc.Driver" />
48       <property name="url" value="jdbc:mysql://localhost:3306/activiti" />
49       <property name="username" value="root" />
50       <property name="password" value="mysql" />

```

```

51     <property name="maxActive" value="3" />
52     <property name="maxIdle" value="1" />
53 </bean>
54 <!-- 事务管理器 -->
55 <bean id="transactionManager"
56     class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
57     <property name="dataSource" ref="dataSource" />
58 </bean>
59 <!-- 通知 -->
60 <tx:advice id="txAdvice" transaction-manager="transactionManager">
61     <tx:attributes></tx:attributes>
62     <!-- 传播行为 -->
63     <tx:method name="save*" propagation="REQUIRED" />
64     <tx:method name="insert*" propagation="REQUIRED" />
65     <tx:method name="delete*" propagation="REQUIRED" />
66     <tx:method name="update*" propagation="REQUIRED" />
67     <tx:method name="find*" propagation="SUPPORTS" read-only="true" />
68     <tx:method name="get*" propagation="SUPPORTS" read-only="true" />
69 </tx:attributes>
70 </tx:advice>
71 <!-- 切面，根据具体项目修改切点配置 -->
72 <aop:config proxy-target-class="true">
73     <aop:advisor advice-ref="txAdvice" pointcut="execution(*
74 com.itheima.ihrm.service.impl.*(..))*" />
75 </aop:config>
76 </beans>

```

创建processEngineConfiguration

```

1 ProcessEngineConfiguration configuration =
    ProcessEngineConfiguration.createProcessEngineConfigurationFromResource("activiti.cfg.xml")

```

上边的代码要求activiti.cfg.xml中必须有一个processEngineConfiguration的bean

也可以使用下边的方法，更改bean 的名字：

```

1 ProcessEngineConfiguration.createProcessEngineConfigurationFromResource(String resource,
    String beanName);

```

4.4 workflow引擎创建

workflow引擎（ProcessEngine），相当于一个门面接口，通过ProcessEngineConfiguration创建processEngine，通过ProcessEngine创建各个service接口。

4.4.1 默认创建方式

将activiti.cfg.xml文件名及路径固定，且activiti.cfg.xml文件中有 processEngineConfiguration的配置，可以使用如下代码创建processEngine：

```
1 //直接使用工具类 ProcessEngines, 使用classpath下的activiti.cfg.xml中的配置创建processEngine
2 ProcessEngine processEngine = ProcessEngines.getDefaultProcessEngine();
3 System.out.println(processEngine);
```

4.4.2 一般创建方式

```
1 //先构建ProcessEngineConfiguration
2 ProcessEngineConfiguration configuration =
  ProcessEngineConfiguration.createProcessEngineConfigurationFromResource("activiti.cfg.xml");
3 //通过ProcessEngineConfiguration创建ProcessEngine, 此时会创建数据库
4 ProcessEngine processEngine = configuration.buildProcessEngine();
```

4.5 Service服务接口

Service是 workflow 引擎提供用于进行 workflow 部署、执行、管理的服务接口，我们使用这些接口可以就是操作服务对应的数据表

4.5.1 Service创建方式

通过ProcessEngine创建Service

方式如下：

```
1 RuntimeService runtimeService = processEngine.getRuntimeService();
2 RepositoryService repositoryService = processEngine.getRepositoryService();
3 TaskService taskService = processEngine.getTaskService();
```

4.5.2 Service总览

service名称	service作用
RepositoryService	activiti的资源管理类
RuntimeService	activiti的流程运行管理类
TaskService	activiti的任务管理类
HistoryService	activiti的历史管理类
ManagerService	activiti的引擎管理类

简单介绍：

RepositoryService

是activiti的资源管理类，提供了管理和控制流程发布包和流程定义的操作。使用 workflow 建模工具设计的业务流程图需要使用此service将流程定义文件的内容部署到计算机。

除了部署流程定义以外还可以：查询引擎中的发布包和流程定义。

暂停或激活发布包，对应全部和特定流程定义。暂停意味着它们不能再执行任何操作了，激活是对应的反向操作。获得多种资源，像是包含在发布包里的文件，或引擎自动生成的流程图。

获得流程定义的pojo版本，可以用来通过java解析流程，而不必通过xml。

RuntimeService

Activiti的流程运行管理类。可以从这个服务类中获取很多关于流程执行相关的信息

TaskService

Activiti的任务管理类。可以从这个类中获取任务的信息。

HistoryService

Activiti的历史管理类，可以查询历史信息，执行流程时，引擎会保存很多数据（根据配置），比如流程实例启动时间，任务的参与者，完成任务的时间，每个流程实例的执行路径，等等。这个服务主要通过查询功能来获得这些数据。

ManagementService

Activiti的引擎管理类，提供了对 Activiti 流程引擎的管理和维护功能，这些功能不在工作流驱动的应用程序中使用，主要用于 Activiti 系统的日常维护。

五、Activiti入门

在本章内容中，我们来创建一个Activiti工作流，并启动这个流程。

创建Activiti工作流主要包含以下几步：

- 1、定义流程，按照BPMN的规范，使用流程定义工具，用**流程符号**把整个流程描述出来
- 2、部署流程，把画好的流程定义文件，加载到数据库中，生成表的数据
- 3、启动流程，使用java代码来操作数据库表中的内容

5.1 流程符号

BPMN 2.0是业务流程建模符号2.0的缩写。

它由Business Process Management Initiative这个非营利协会创建并不断发展。作为一种标识，BPMN 2.0是使用一些**符号**来明确业务流程设计流程图的一整套符号规范，它能增进业务建模时的沟通效率。

目前BPMN2.0是最新的版本，它用于在BPM上下文中进行布局和可视化的沟通。

接下来我们先来了解在流程设计中常见的 符号。

BPMN2.0的**基本符合**主要包含：

事件 Event



[Start Event](#)

开始事件



[Intermediate Event](#)

中间事件

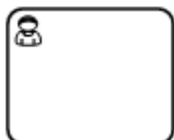


[End Event](#)

结束事件

活动 Activity

活动是工作或任务的一个通用术语。一个活动可以是一个任务，还可以是一个当前流程的子处理流程；其次，你还可以为活动指定不同的类型。常见活动如下：



User Task

用户任务



Service Task

服务任务



[Sub Process](#)

子流程

网关 GateWay

网关用来处理决策，有几种常用网关需要了解：



排他网关



并行网关



包容网关



综合网关



事件网关

排他网关 (x)

——只有一条路径会被选择。流程执行到该网关时，按照输出流的顺序逐个计算，当条件的计算结果为true时，继续执行当前网关的输出流；

如果多条线路计算结果都是 true，则会执行第一个值为 true 的线路。如果所有网关计算结果没有true，则引擎会抛出异常。

排他网关需要和条件顺序流结合使用，default 属性指定默认顺序流，当所有的条件不满足时会执行默认顺序流。

并行网关 (+)

——所有路径会被同时选择

拆分 —— 并行执行所有输出顺序流，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并执行完成的线路均在此等候，直到所有的线路都执行完成才继续向下执行。

包容网关 (+)

——可以同时执行多条线路，也可以在网关上设置条件

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候，直到所有的线路都执行完成才继续向下执行。

事件网关 (+)

—— 专门为中间捕获事件设置的，允许设置多个输出流指向多个不同的中间捕获事件。当流程执行到事件网关后，流程处于等待状态，需要等待抛出事件才能将等待状态转换为活动状态。

流向 Flow

流是连接两个流程节点的连线。常见的流向包含以下几种：



5.2 流程设计器使用

Activiti-Designer使用

Palette（画板）

在idea中安装插件即可使用，画板中包括以下结点：

Connection—连接

Event---事件

Task---任务

Gateway---网关

Container—容器

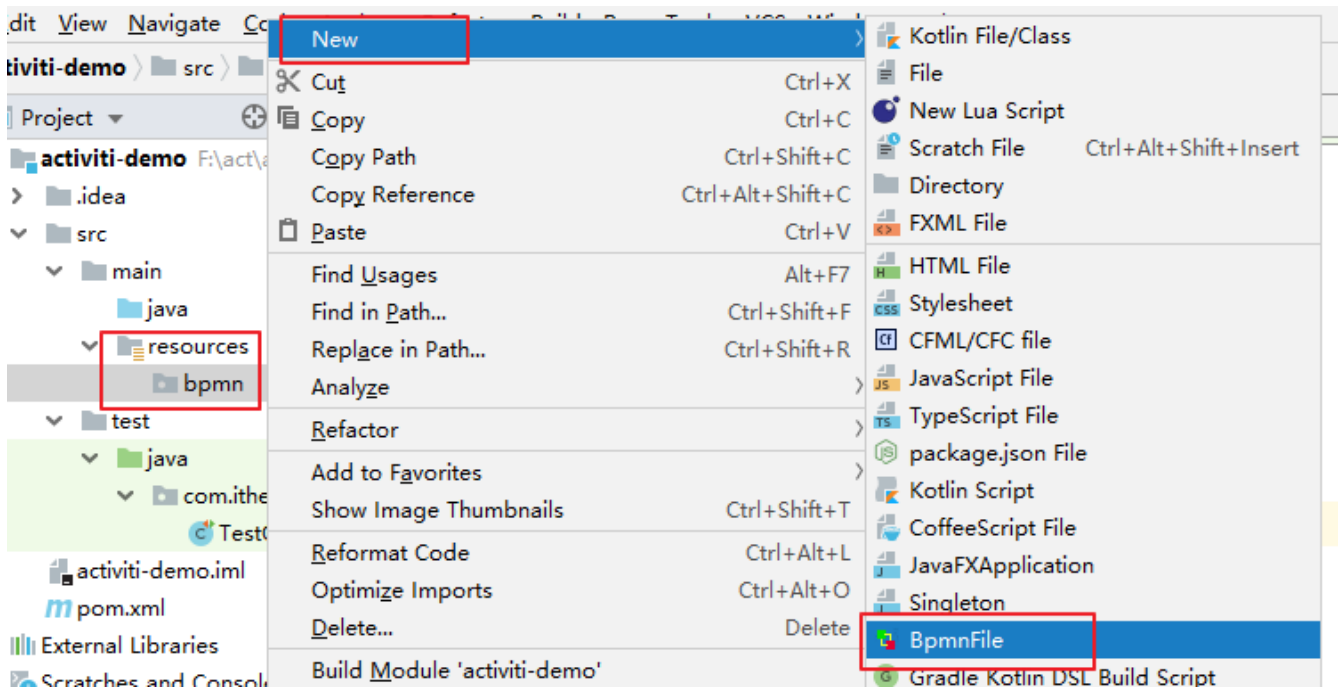
Boundary event—边界事件

Intermediate event- -中间事件

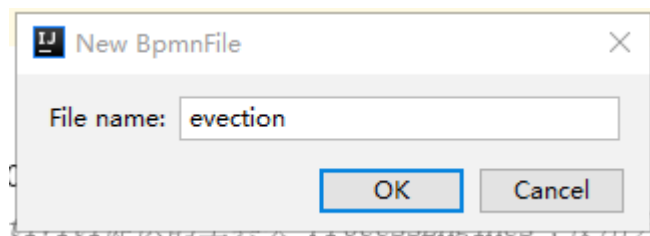
流程图设计完毕保存生成.bpmn文件

新建流程(IDEA工具)

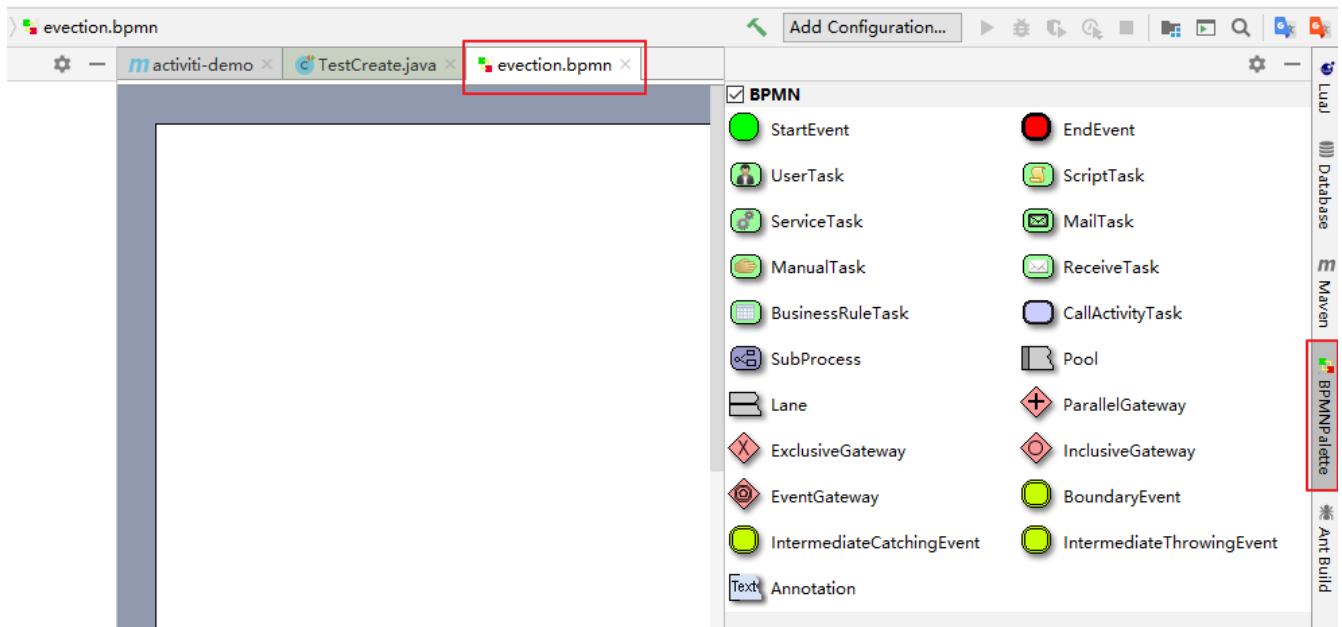
首先选中存放图形的目录(选择resources下的bpmn目录)，点击菜单：New -> BpmnFile，如图：



弹出如下图所示框，输入evection 表示 出差审批流程：



起完名字evection后（默认扩展名为bpmn），就可以看到流程设计页面，如图所示：

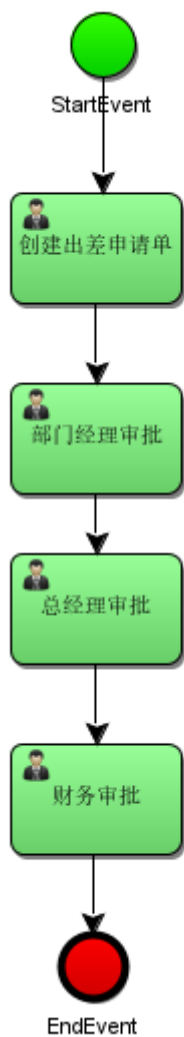


左侧区域是绘图区，右侧区域是palette画板区域

鼠标先点击画板的元素即可在左侧绘图

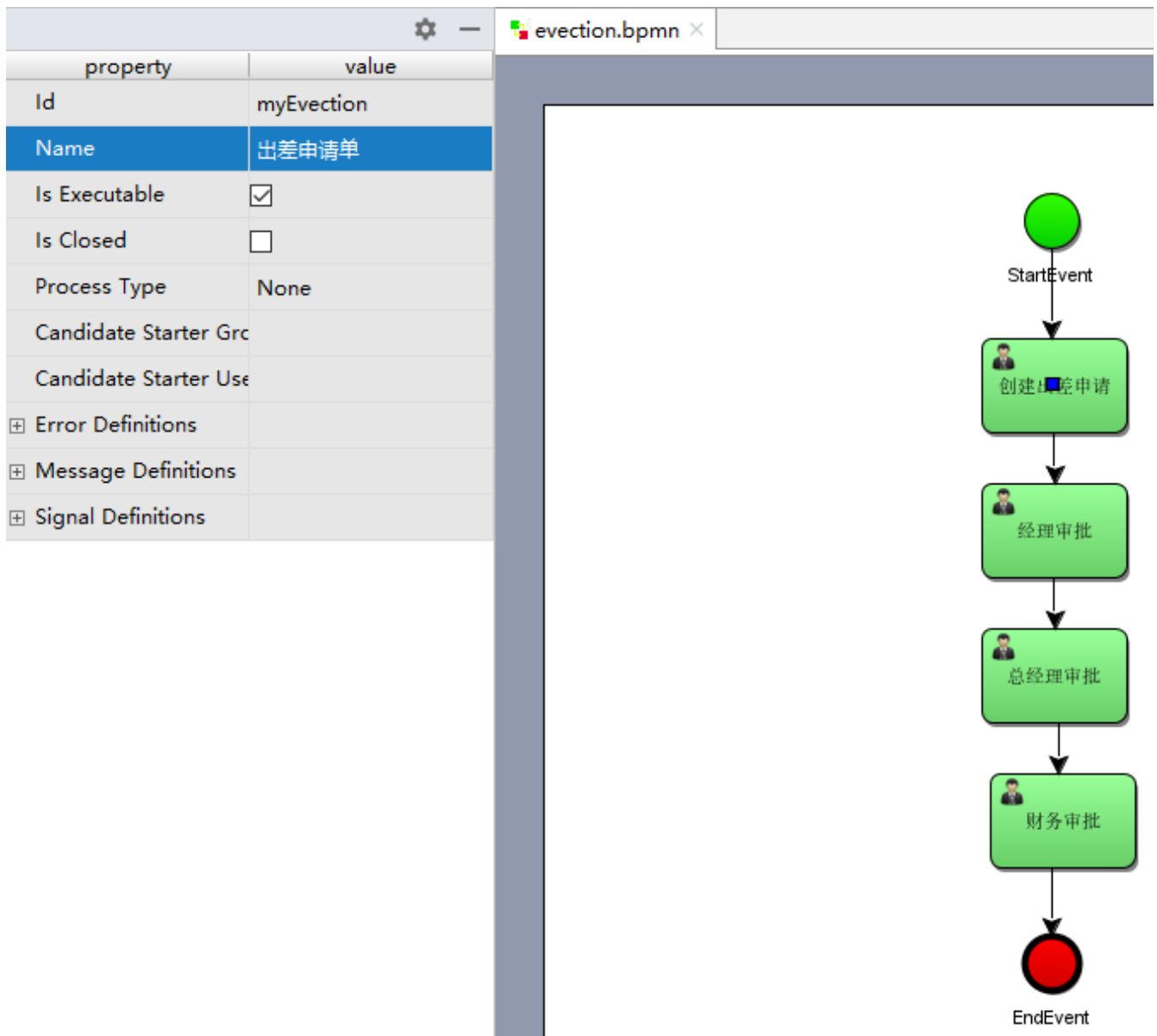
绘制流程

使用滑板来绘制流程，通过从右侧把图标拖拽到左侧的画板，最终效果如下：



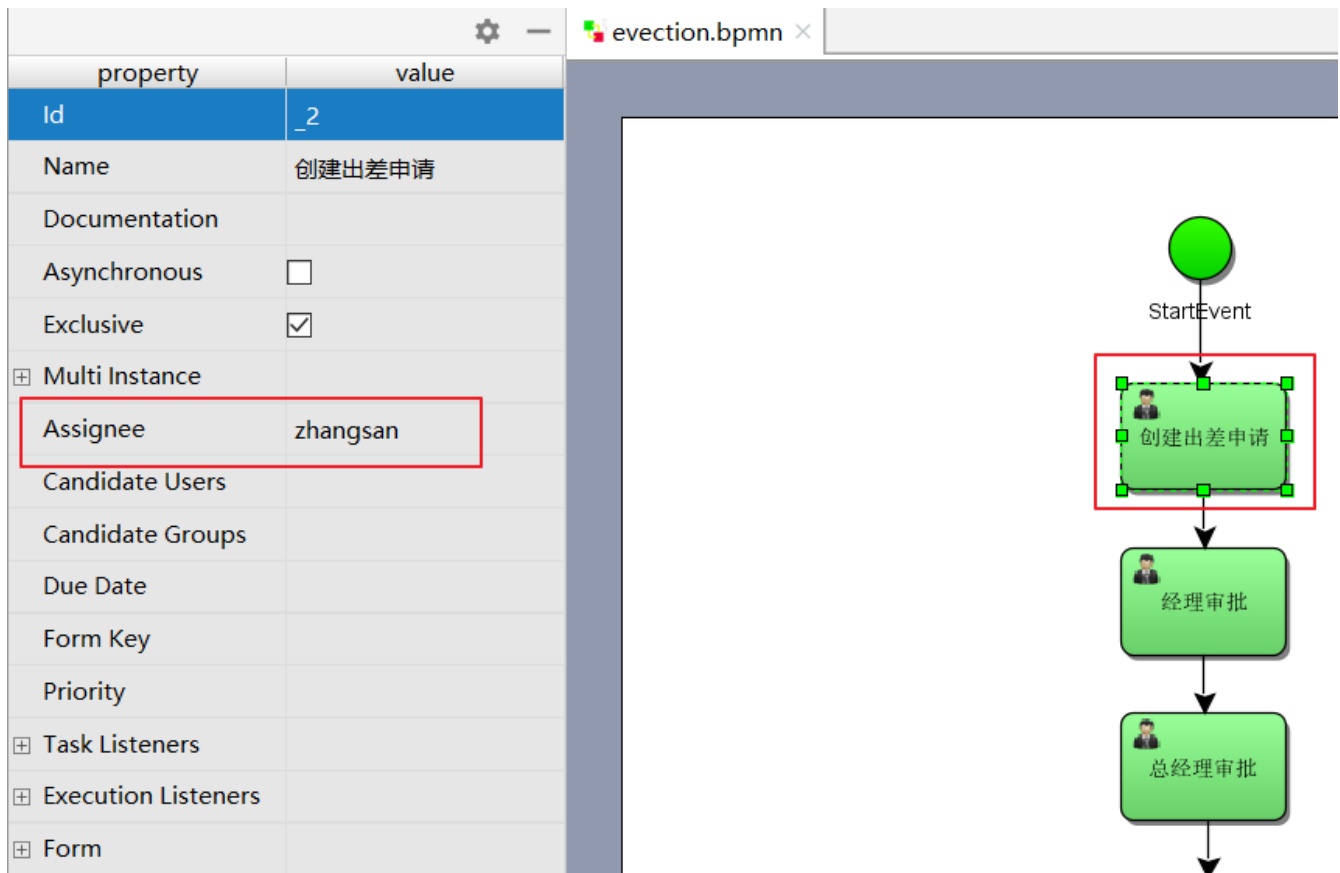
指定流程定义Key

流程定义key即流程定义的标识，通过properties视图查看流程的key



指定任务负责人

在properties视图指定每个任务结点的负责人，如：填写出差申请的负责人为 zhangsan



经理审批负责人为 jerry

总经理审批负责人为 jack

财务审批负责人为 rose

六、流程操作

6.1 流程定义

概述

流程定义是线下按照bpmn2.0标准去描述 业务流程，通常使用idea中的插件对业务流程进行建模。

使用idea下的designer设计器绘制流程，并会生成两个文件：.bpmn和.png

.bpmn文件

使用activiti-designer设计业务流程，会生成.bpmn文件，上面我们已经创建好了bpmn文件

BPMN 2.0根节点是definitions节点。这个元素中，可以定义多个流程定义（不过我们建议每个文件只包含一个流程定义，可以简化开发过程中的维护难度）。注意，definitions元素 最少也要包含xmlns 和 targetNamespace的声明。targetNamespace可以是任意值，它用来对流程实例进行分类。

流程定义部分：定义了流程每个结点的描述及结点之间的流程流转。

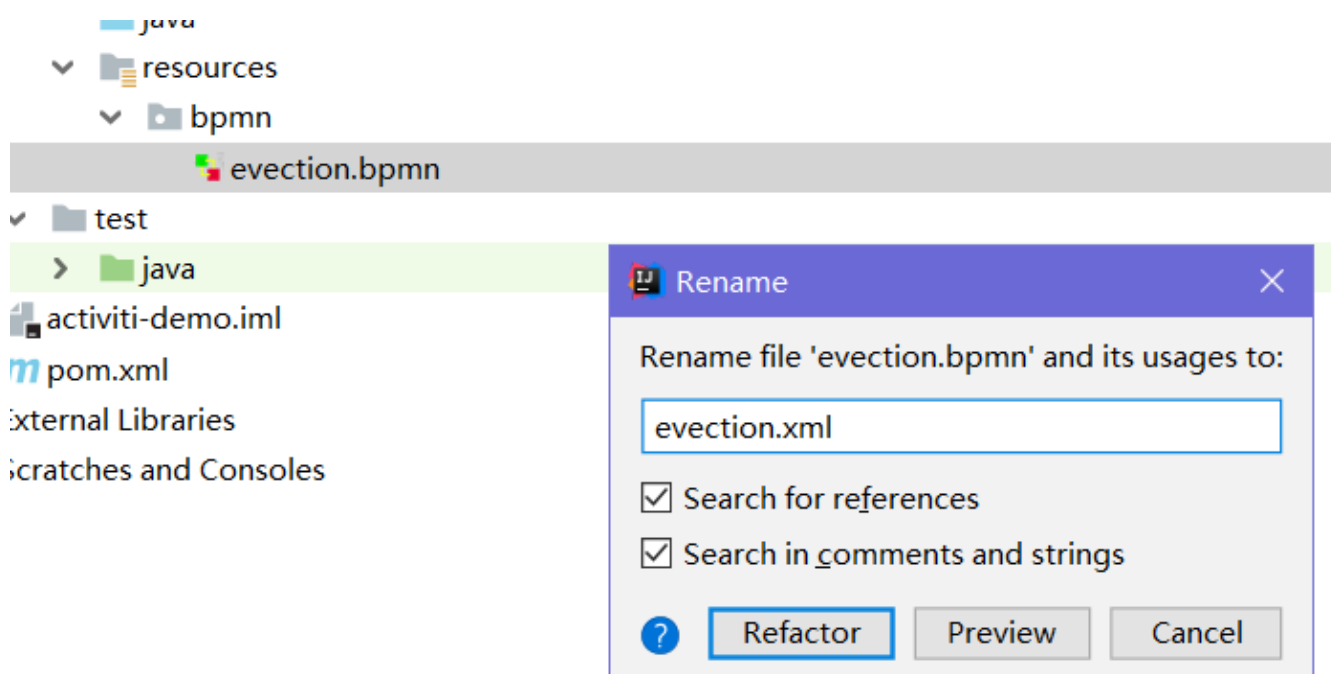
流程布局定义：定义流程每个结点在流程图上的位置坐标等信息。

生成.png图片文件

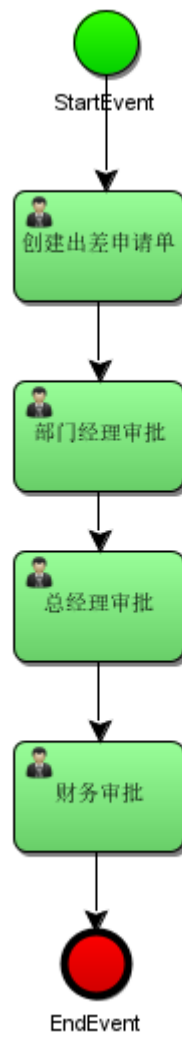
IDEA工具中的操作方式

1、修改文件后缀为xml

首先将evection.bpmn文件改名为evection.xml，如下图：

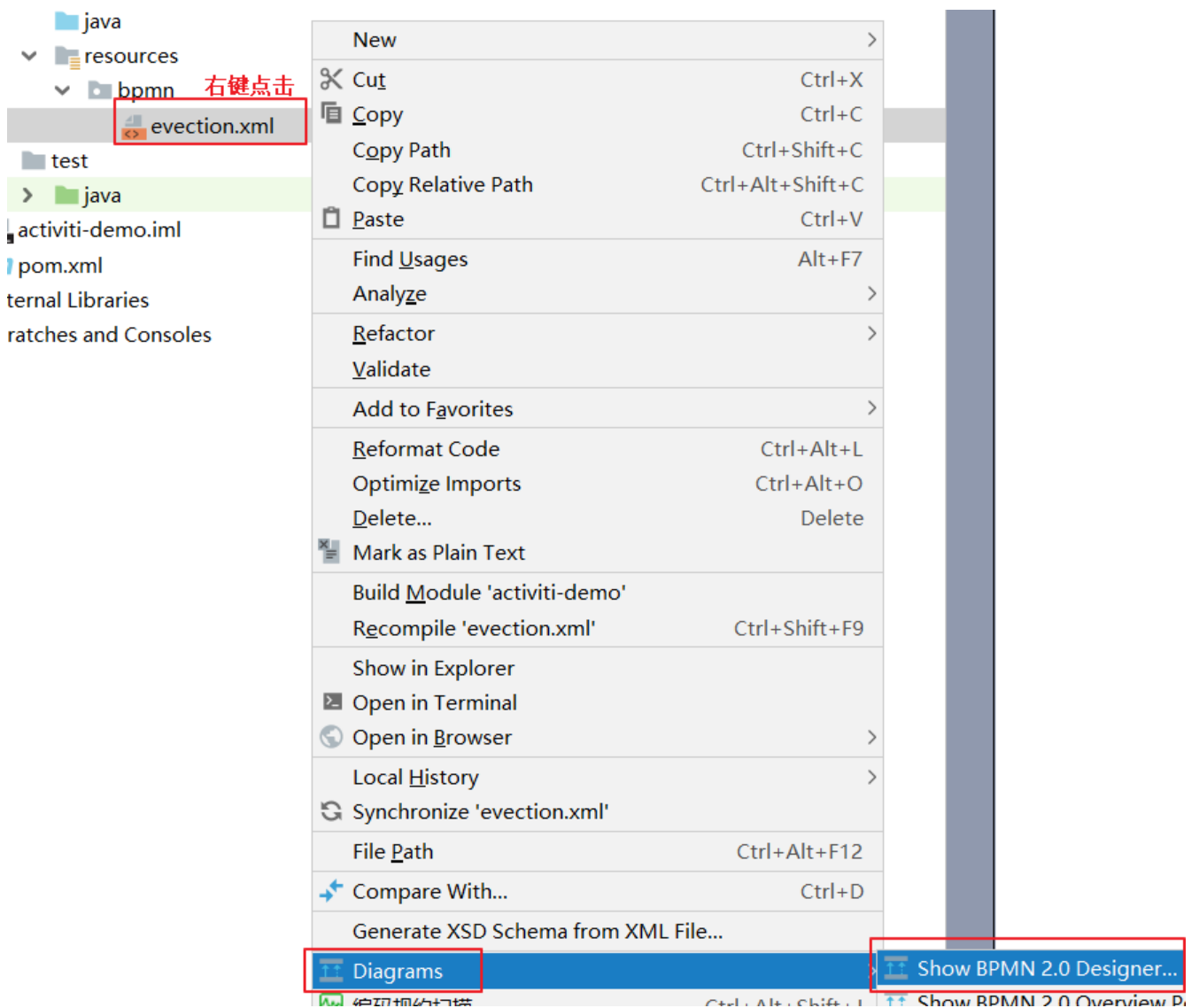


evection.xml修改前的bpmn文件，效果如下：



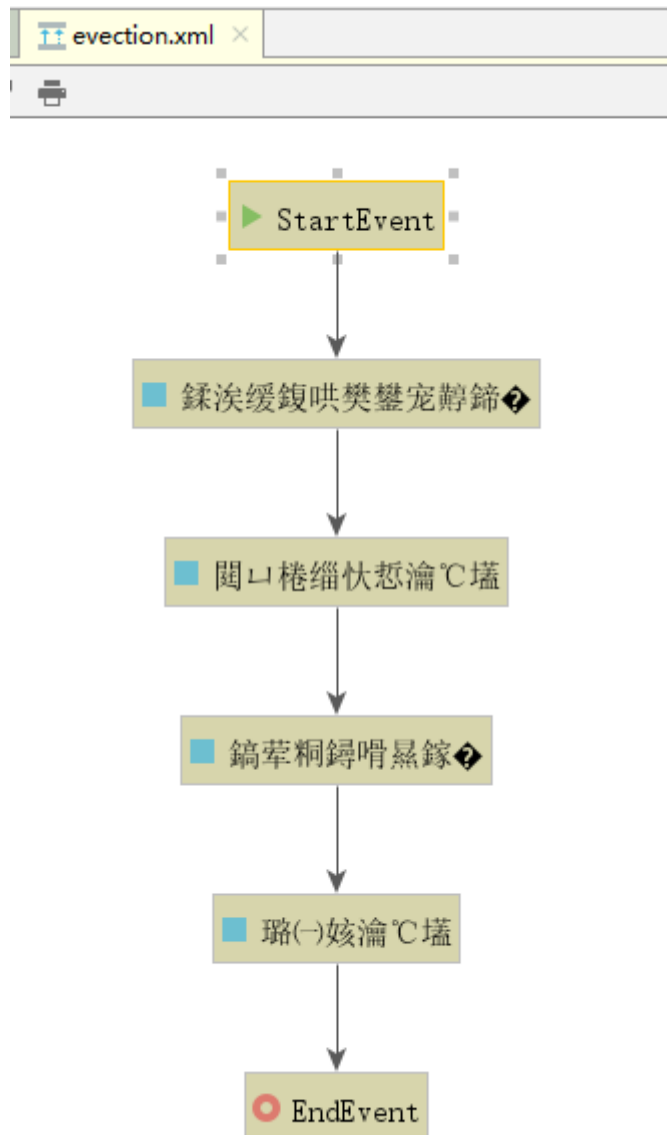
2、使用designer设计器打开.xml文件

在evection.xml文件上面，点右键并选择Diagrams菜单，再选择Show BPMN2.0 Designer...



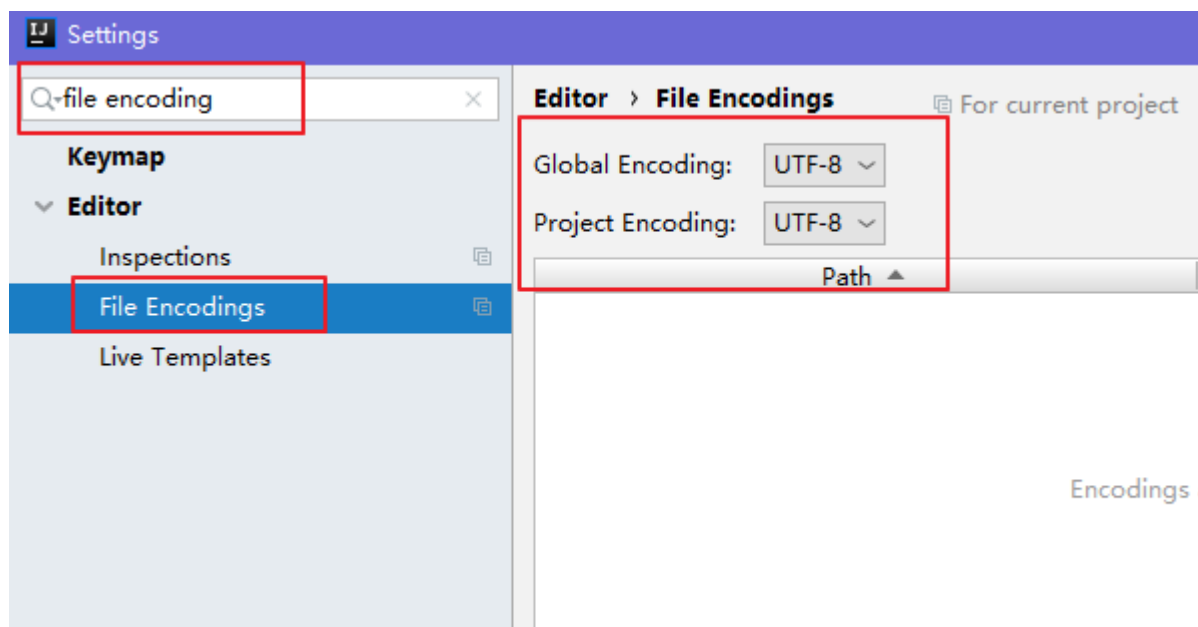
3、查看打开的文件

打开后，却出现乱码，如图：

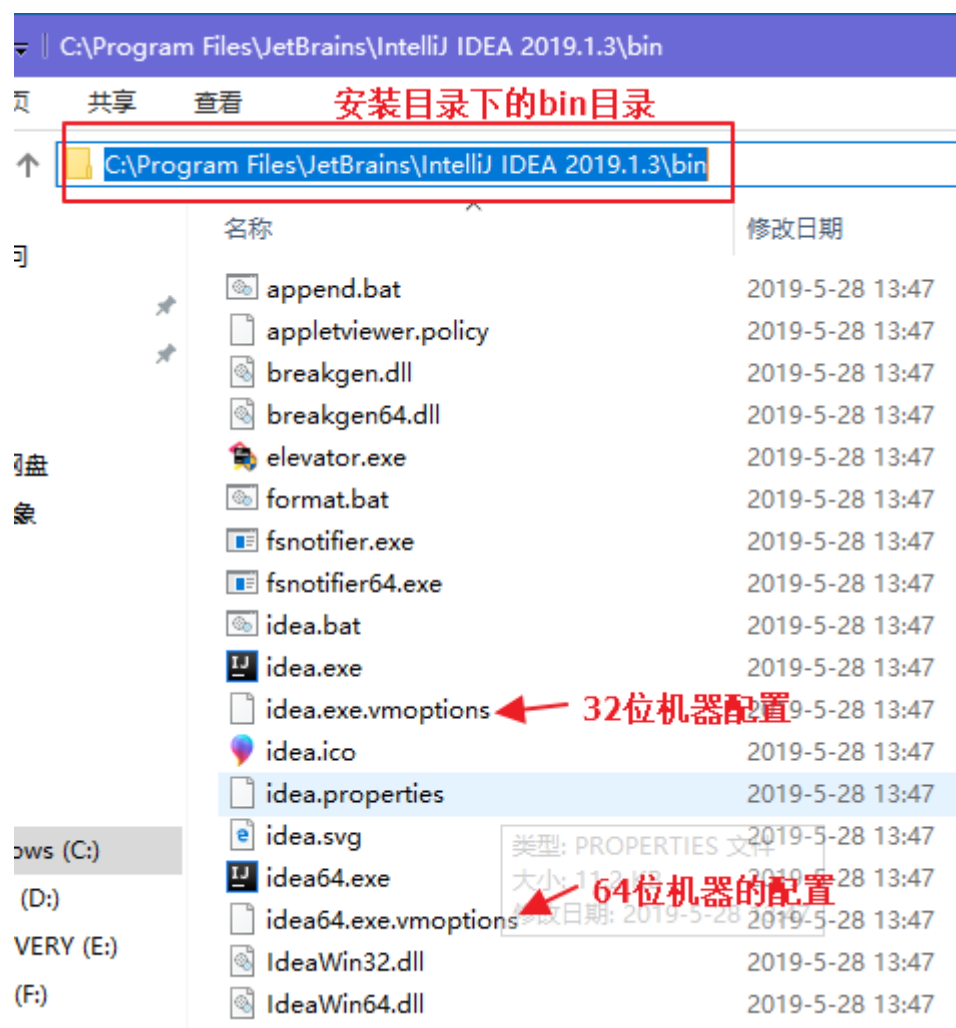


4、解决中文乱码

- 1、打开Settings，找到File Encodings，把encoding的选项都选择UTF-8




2、打开IDEA安装路径，找到如下的安装目录



根据自己所安装的版本来决定，我使用的是64位的idea，所以在idea64.exe.vmoptions文件的最后一行追加一条命令：
-Dfile.encoding=UTF-8

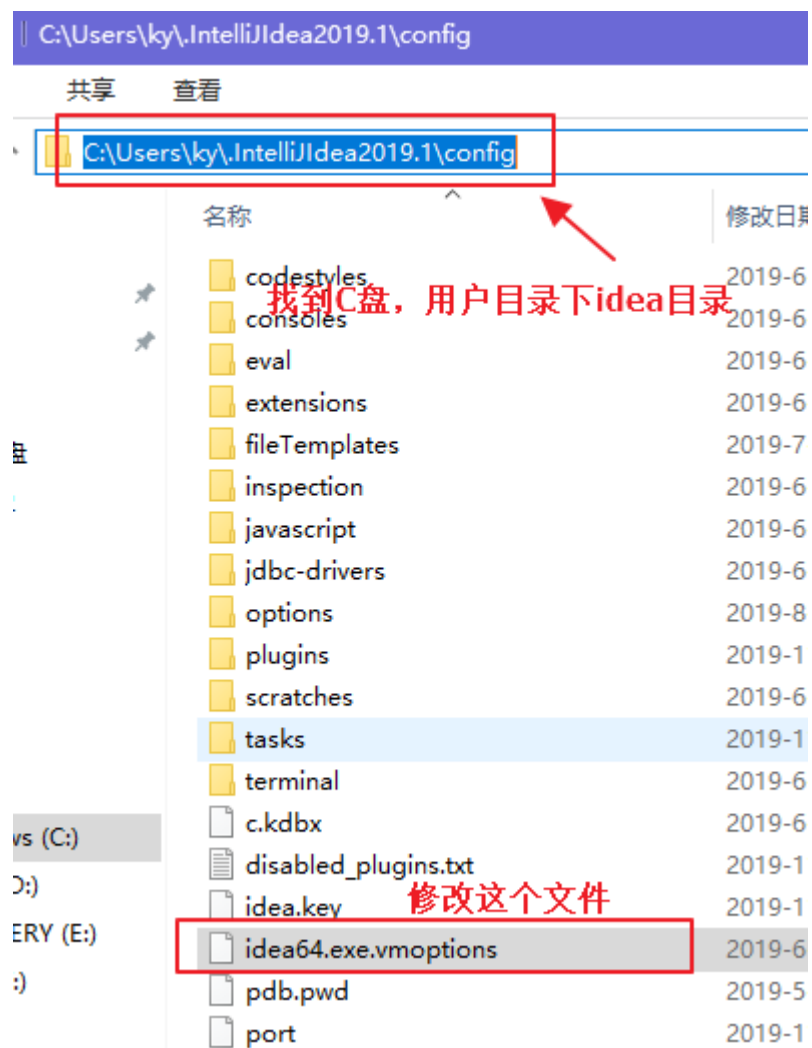
如下所示：

```
1 -Xms128m
2 -Xmx750m
3 -XX:ReservedCodeCacheSize=240m
4 -XX:+UseConcMarkSweepGC
5 -XX:SoftRefLRUPolicyMSPerMB=50
6 -ea
7 -Dsun.io.useCanonCaches=false
8 -Djava.net.preferIPv4Stack=true
9 -XX:+HeapDumpOnOutOfMemoryError
10 -XX:-OmitStackTraceInFastThrow
11 |-Dfile.encoding=UTF-8
```

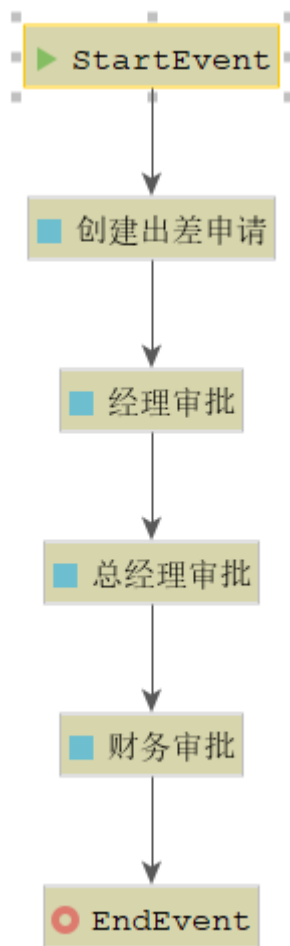


一定注意，不要有空格，否则重启IDEA时会打不开，然后 重启IDEA。

如果以上方法已经做完，还出现乱码，就再修改一个文件，并在文件的末尾添加：-Dfile.encoding=UTF-8，然后重启idea，如图：



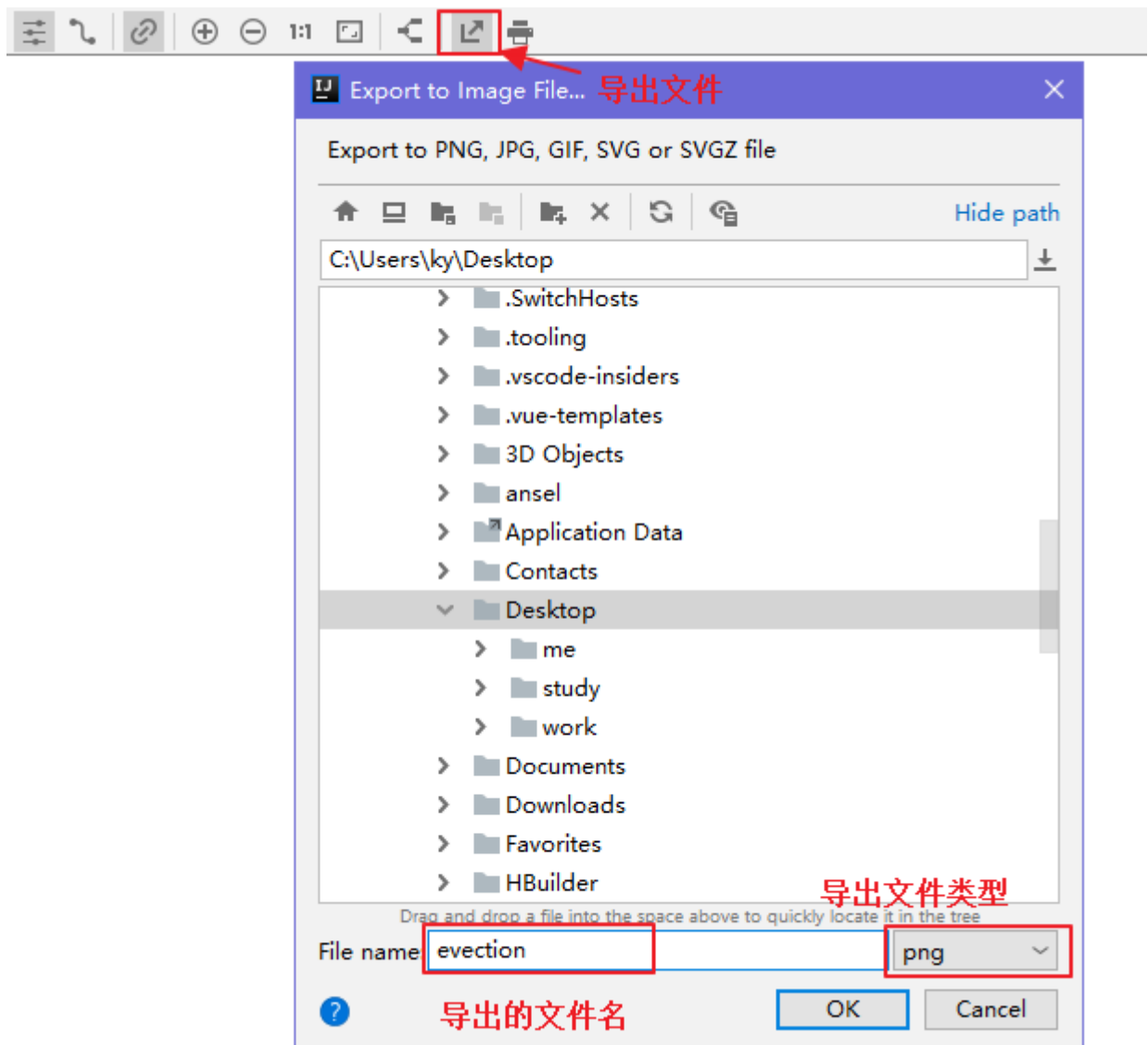
最后重新在evection.xml文件上面，点右键并选择Diagrams菜单，再选择Show BPMN2.0 Designer...，看到生成图片，如图：



到此，解决乱码问题

5、导出为图片文件

点击Export To File的小图标，打开如下窗口，注意填写文件名及扩展名，选择好保存图片的位置：



然后，我们把png文件拷贝到resources下的bpmn目录，并且把evection.xml改名为evection.bpmn。

6.2 流程定义部署

概述

将上面在设计器中定义的流程部署到activiti数据库中，就是流程定义部署。

通过调用activiti的api将流程定义的bpmn和png两个文件一个一个添加部署到activiti中，也可以将两个文件打成zip包进行部署。

单个文件部署方式

分别将bpmn文件和png图片文件部署。

```
1 package com.itheima.test;
2
3 import org.activiti.engine.ProcessEngine;
4 import org.activiti.engine.ProcessEngines;
```

```

5  import org.activiti.engine.RepositoryService;
6  import org.activiti.engine.repository.Deployment;
7  import org.junit.Test;
8
9  public class ActivitiDemo {
10     /**
11      * 部署流程定义
12      */
13     @Test
14     public void testDeployment(){
15         //      1、创建ProcessEngine
16         ProcessEngine processEngine = ProcessEngines.getDefaultProcessEngine();
17         //      2、得到RepositoryService实例
18         RepositoryService repositoryService = processEngine.getRepositoryService();
19         //      3、使用RepositoryService进行部署
20         Deployment deployment = repositoryService.createDeployment()
21             .addClasspathResource("bpmn/evaction.bpmn") // 添加bpmn资源
22             .addClasspathResource("bpmn/evaction.png") // 添加png资源
23             .name("出差申请流程")
24             .deploy();
25         //      4、输出部署信息
26         System.out.println("流程部署id: " + deployment.getId());
27         System.out.println("流程部署名称: " + deployment.getName());
28     }
29 }
30

```

执行此操作后activiti会将上边代码中指定的bpm文件和图片文件保存在activiti数据库。

压缩包部署方式

将evaction.bpmn和evaction.png压缩成zip包。

```

1  @Test
2  public void deployProcessByZip() {
3      // 定义zip输入流
4      InputStream inputStream = this
5          .getClass()
6          .getClassLoader()
7          .getResourceAsStream(
8              "bpmn/evaction.zip");
9      ZipInputStream zipInputStream = new ZipInputStream(inputStream);
10     // 获取repositoryService
11     RepositoryService repositoryService = processEngine
12         .getRepositoryService();
13     // 流程部署
14     Deployment deployment = repositoryService.createDeployment()
15         .addZipInputStream(zipInputStream)
16         .deploy();
17     System.out.println("流程部署id: " + deployment.getId());
18     System.out.println("流程部署名称: " + deployment.getName());
19 }
20

```

执行此操作后activiti会将上边代码中指定的bpm文件和图片文件保存在activiti数据库。

操作数据表

流程定义部署后操作activiti的3张表如下：

act_re_deployment 流程定义部署表，每部署一次增加一条记录

act_re_procdef 流程定义表，部署每个新的流程定义都会在这张表中增加一条记录

act_ge_bytearray 流程资源表

接下来我们来看看，写入了什么数据：

```
1 SELECT * FROM act_re_deployment #流程定义部署表，记录流程部署信息
```

结果：

ID_	NAME_	CATEGORY_	KEY_	TENANT_ID_
1	出差申请流程	(NULL)	(NULL)	
*	(NULL)	(NULL)	(NULL)	

```
1 SELECT * FROM act_re_procdef #流程定义表，记录流程定义信息
```

结果：

注意，KEY 这个字段是用来唯一识别不同流程的关键字

ID_	REV_	CATEGORY_	NAME_	KEY_	VERSION_	DEPLOYMENT_ID_	RESOURCE_NAME_	DGRM_RESOURCE_NAME_
myEvection:1:4	1	http://www.activiti.org/test	出差申请单	myEvection	1	1	bpmn/evection.bpmn	bpmn/evection.png
*	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

```
1 SELECT * FROM act_ge_bytearray #资源表
```

结果：

ID_	REV_	NAME_	DEPLOYMENT_ID_	BYTES_	GENERATED_
2	1	bpmn/evection.png	1	(Binary/Image)	8K
3	1	bpmn/evection.bpmn	1	<?xml version="1.0" encoding="UTF-8" standalone="yes"?><d...	5K
*	(NULL)	(NULL)	(NULL)	(NULL)	0K

注意：

act_re_deployment和act_re_procdef一对多关系，一次部署在流程部署表生成一条记录，但一次部署可以部署多个流程定义，每个流程定义在流程定义表生成一条记录。每一个流程定义在act_ge_bytearray会存在两个资源记录，bpmn和png。

建议：一次部署一个流程，这样部署表和流程定义表是一对一有关系，方便读取流程部署及流程定义信息。

6.3 启动流程实例

流程定义部署在activiti后就可以通过工作流管理业务流程了，也就是说上边部署的出差申请流程可以使用了。

针对该流程，启动一个流程表示发起一个新的出差申请单，这就相当于java类与java对象的关系，类定义好后需要new创建一个对象使用，当然可以new多个对象。对于请出差申请流程，张三发起一个出差申请单需要启动一个流程实例，出差申请单发起一个出差单也需要启动一个流程实例。

代码如下：

```
1      /**
2      * 启动流程实例
3      */
4      @Test
5      public void testStartProcess(){
6          //      1、创建ProcessEngine
7          ProcessEngine processEngine = ProcessEngines.getDefaultProcessEngine();
8          //      2、获取RunTimeService
9          RuntimeService runtimeService = processEngine.getRuntimeService();
10         //      3、根据流程定义Id启动流程
11         ProcessInstance processInstance = runtimeService
12             .startProcessInstanceByKey("myEvection");
13         //      输出内容
14         System.out.println("流程定义id: " + processInstance.getProcessDefinitionId());
15         System.out.println("流程实例id: " + processInstance.getId());
16         System.out.println("当前活动Id: " + processInstance.getActivityId());
17     }
18
```

输出内容如下：

```
流程定义id: myEvection:1:4
流程实例id: 2501
当前活动Id: null
```

操作数据表

act_hi_actinst 流程实例执行历史

act_hi_identitylink 流程的参与用户历史信息

act_hi_procinst 流程实例历史信息

act_hi_taskinst 流程任务历史信息

act_ru_execution 流程执行信息

act_ru_identitylink 流程的参与用户信息

act_ru_task 任务信息

6.4 任务查询

流程启动后，任务的负责人就可以查询自己当前需要处理的任务，查询出来的任务都是该用户的待办任务。

```
1      /**
```

```

2      * 查询当前个人待执行的任务
3      */
4      @Test
5      public void testFindPersonalTaskList() {
6          //      任务负责人
7          String assignee = "zhangsan";
8          ProcessEngine processEngine = ProcessEngines.getDefaultProcessEngine();
9          //      创建TaskService
10         TaskService taskService = processEngine.getTaskService();
11         //      根据流程key 和 任务负责人 查询任务
12         List<Task> list = taskService.createTaskQuery()
13             .processDefinitionKey("myEvection") //流程Key
14             .taskAssignee(assignee) //只查询该任务负责人的任务
15             .list();
16
17         for (Task task : list) {
18
19             System.out.println("流程实例id: " + task.getProcessInstanceId());
20             System.out.println("任务id: " + task.getId());
21             System.out.println("任务负责人: " + task.getAssignee());
22             System.out.println("任务名称: " + task.getName());
23
24         }
25     }

```

输出结果如下：

```

1  流程实例id: 2501
2  任务id: 2505
3  任务负责人: zhangsan
4  任务名称: 创建出差申请

```

6.5 流程任务处理

任务负责人查询待办任务，选择任务进行处理，完成任务。

```

1  // 完成任务
2  @Test
3  public void completTask(){
4      //      获取引擎
5      ProcessEngine processEngine = ProcessEngines.getDefaultProcessEngine();
6      //      获取taskService
7      TaskService taskService = processEngine.getTaskService();
8
9      //      根据流程key 和 任务的负责人 查询任务
10     //      返回一个任务对象
11     Task task = taskService.createTaskQuery()
12         .processDefinitionKey("myEvection") //流程Key
13         .taskAssignee("zhangsan") //要查询的负责人
14         .singleResult();
15

```

```
16 //      完成任务,参数: 任务id
17      taskService.complete(task.getId());
18  }
19
```

6.6 流程定义信息查询

查询流程相关信息，包含流程定义，流程部署，流程定义版本

```
1      /**
2      * 查询流程定义
3      */
4      @Test
5      public void queryProcessDefinition(){
6          //      获取引擎
7          ProcessEngine processEngine = ProcessEngines.getDefaultProcessEngine();
8          //      repositoryService
9          RepositoryService repositoryService = processEngine.getRepositoryService();
10         //      得到ProcessDefinitionQuery 对象
11         ProcessDefinitionQuery processDefinitionQuery =
            repositoryService.createProcessDefinitionQuery();
12         //      查询出当前所有的流程定义
13         //      条件: processDefinitionKey =evection
14         //      orderByProcessDefinitionVersion 按照版本排序
15         //      desc倒叙
16         //      list 返回集合
17         List<ProcessDefinition> definitionList =
            processDefinitionQuery.processDefinitionKey("myEvection")
18             .orderByProcessDefinitionVersion()
19             .desc()
20             .list();
21         //      输出流程定义信息
22         for (ProcessDefinition processDefinition : definitionList) {
23             System.out.println("流程定义 id="+processDefinition.getId());
24             System.out.println("流程定义 name="+processDefinition.getName());
25             System.out.println("流程定义 key="+processDefinition.getKey());
26             System.out.println("流程定义 Version="+processDefinition.getVersion());
27             System.out.println("流程部署ID =" +processDefinition.getDeploymentId());
28         }
29     }
30 }
```

输出结果：

```
1  流程定义id: myEvection:1:4
2  流程定义名称: 出差申请单
3  流程定义key: myEvection
4  流程定义版本: 1
```

6.7 流程删除

```
1 public void deleteDeployment() {
2     // 流程部署id
3     String deploymentId = "1";
4
5     ProcessEngine processEngine = ProcessEngines.getDefaultProcessEngine();
6     // 通过流程引擎获取repositoryService
7     RepositoryService repositoryService = processEngine
8         .getRepositoryService();
9     //删除流程定义，如果该流程定义已有流程实例启动则删除时出错
10    repositoryService.deleteDeployment(deploymentId);
11    //设置true 级联删除流程定义，即使该流程有流程实例启动也可以删除，设置为false非级联删除方式，如果流
程
12    //repositoryService.deleteDeployment(deploymentId, true);
13 }
14
```

说明：

- 1) 使用repositoryService删除流程定义，历史表信息不会被删除
- 2) 如果该流程定义下没有正在运行的流程，则可以用普通删除。

如果该流程定义下存在已经运行的流程，使用普通删除报错，可用级联删除方法将流程及相关记录全部删除。

先删除没有完成流程节点，最后就可以完全删除流程定义信息

项目开发中级联删除操作一般只开放给超级管理员使用。

6.8 流程资源下载

现在我们的流程资源文件已经上传到数据库了，如果其他用户想要查看这些资源文件，可以从数据库中把资源文件下载到本地。

解决方案有：

- 1、jdbc对blob类型，clob类型数据读取出来，保存到文件目录
- 2、使用activiti的api来实现

使用commons-io.jar 解决IO的操作

引入commons-io依赖包

```
1 <dependency>
2     <groupId>commons-io</groupId>
3     <artifactId>commons-io</artifactId>
4     <version>2.6</version>
5 </dependency>
```

通过流程定义对象获取流程定义资源，获取bpmn和png


```

1  import org.apache.commons.io.IOUtils;
2
3  @Test
4      public void deleteDeployment(){
5      //      获取引擎
6          ProcessEngine processEngine = ProcessEngines.getDefaultProcessEngine();
7      //      获取repositoryService
8          RepositoryService repositoryService = processEngine.getRepositoryService();
9      //      根据部署id 删除部署信息,如果想要级联删除,可以添加第二个参数, true
10         repositoryService.deleteDeployment("1");
11     }
12
13     public void queryBpmnFile() throws IOException {
14     //      1、得到引擎
15         ProcessEngine processEngine = ProcessEngines.getDefaultProcessEngine();
16     //      2、获取repositoryService
17         RepositoryService repositoryService = processEngine.getRepositoryService();
18     //      3、得到查询器: ProcessDefinitionQuery, 设置查询条件,得到想要的流程定义
19         ProcessDefinition processDefinition =
20 repositoryService.createProcessDefinitionQuery()
21             .processDefinitionKey("myEvection")
22             .singleResult();
23     //      4、通过流程定义信息, 得到部署ID
24         String deploymentId = processDefinition.getDeploymentId();
25     //      5、通过repositoryService的方法, 实现读取图片信息和bpmn信息
26     //      png图片的流
27         InputStream pngInput = repositoryService.getResourceAsStream(deploymentId,
28 processDefinition.getDiagramResourceName());
29     //      bpmn文件的流
30         InputStream bpmnInput = repositoryService.getResourceAsStream(deploymentId,
31 processDefinition.getResourceName());
32     //      6、构造OutputStream流
33         File file_png = new File("d:/evectionflow01.png");
34         File file_bpmn = new File("d:/evectionflow01.bpmn");
35         FileOutputStream bpmnOut = new FileOutputStream(file_bpmn);
36         FileOutputStream pngOut = new FileOutputStream(file_png);
37     //      7、输入流, 输出流的转换
38         IOUtils.copy(pngInput, pngOut);
39         IOUtils.copy(bpmnInput, bpmnOut);
40     //      8、关闭流
41         pngOut.close();
42         bpmnOut.close();
43         pngInput.close();
44         bpmnInput.close();
45     }
46
47 }

```

说明:

- 1) deploymentId为流程部署ID
- 2) resource_name为act_ge_bytearray表中NAME_列的值
- 3) 使用repositoryService的getDeploymentResourceNames方法可以获取指定部署下所有文件的名称

4) 使用repositoryService的getResourceAsStream方法传入部署ID和资源图片名称可以获取部署下指定名称文件的输入流

最后的将输入流中的图片资源进行输出。

6.9 流程历史信息的查看

即使流程定义已经删除了，流程执行的历史信息通过前面的分析，依然保存在activiti的act_hi_*相关的表中。所以我们可以查询流程执行的历史信息，可以通过HistoryService来查看相关的历史记录。

```
1      /**
2      * 查看历史信息
3      */
4      @Test
5      public void findHistoryInfo(){
6          // 获取引擎
7          ProcessEngine processEngine = ProcessEngines.getDefaultProcessEngine();
8          // 获取HistoryService
9          HistoryService historyService = processEngine.getHistoryService();
10         // 获取 actinst表的查询对象
11         HistoricActivityInstanceQuery instanceQuery =
            historyService.createHistoricActivityInstanceQuery();
12         // 查询 actinst表, 条件: 根据 InstanceId 查询
13         // instanceQuery.processInstanceId("2501");
14         // 查询 actinst表, 条件: 根据 DefinitionId 查询
15         instanceQuery.processDefinitionId("myEvection:1:4");
16         // 增加排序操作, orderByHistoricActivityInstanceStartTime 根据开始时间排序 asc 升序
17         instanceQuery.orderByHistoricActivityInstanceStartTime().asc();
18         // 查询所有内容
19         List<HistoricActivityInstance> activityInstanceList = instanceQuery.list();
20         // 输出
21         for (HistoricActivityInstance hi : activityInstanceList) {
22             System.out.println(hi.getActivityId());
23             System.out.println(hi.getActivityName());
24             System.out.println(hi.getProcessDefinitionId());
25             System.out.println(hi.getProcessInstanceId());
26             System.out.println("<=====");
27         }
28     }
```