PDF-XChange Editor Click to BUY NOW! tracker-software.com

SaaS-IHRM 项目-Activiti7

工作流引擎

传智播客-研究院

第1章 什么是工作流

1.1 工作流介绍

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。

1.2.1 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。

1.2.2 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。

1.3 工作流实现方式

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。

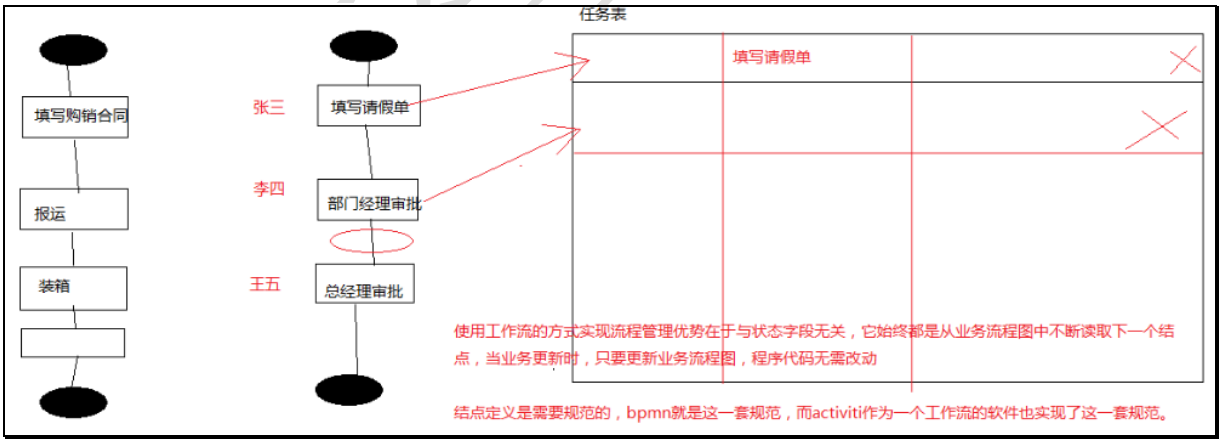
针对有权限可以查看的记录，当前用户根据自己的角色来决定审批是否合格的操作。如果合格将状态字段设置一个值，来代表合格；当然如果不合格也需要设置一个值来代表不合格的情况。

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。

1.4 工作流实现原理分析

如何可以做到我们在业务流程发生变更后，我们的业务系统代码可以不发生改变？此时我们就来分析一下原理。
具体分析过程如下图所示：



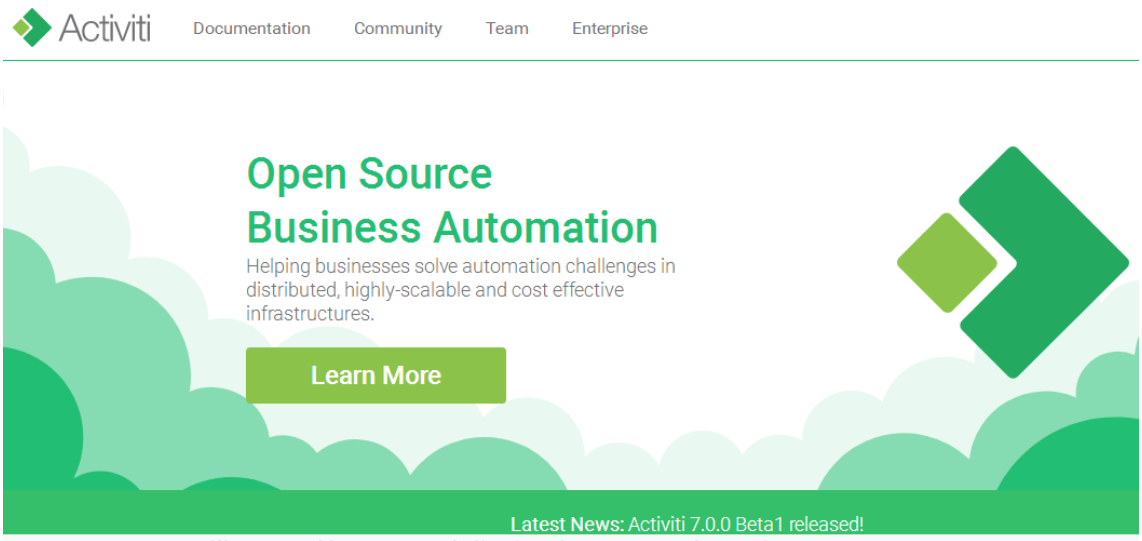
第2章 什么是Activiti7

2.1Activiti 介绍

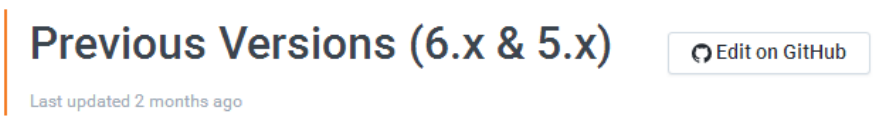
Alfresco 软件在 2010 年 5 月 17 日宣布 Activiti 业务流程管理（BPM）开源项目的正式启动，其首席架构师由业务流程管理 BPM 的专家 Tom Baeyens 担任，Tom Baeyens 就是原来 jbpm 的架构师，而 jbpm 是一个非常有名的工作流引擎，当然 activiti 也是一个工作流引擎。

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，业务系统按照预先定义的流程进行执行，实现了业务系统的业务流程由 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/



经历的版本:



目前最新版本：Activiti7.0.0.Beta

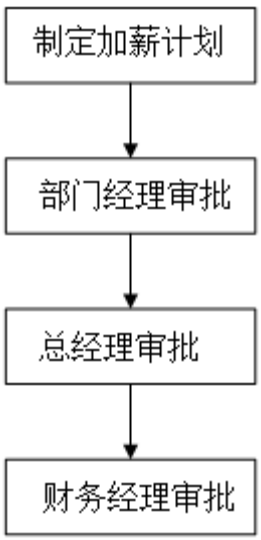
2.1.1 BPM

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系统化方法，常见商业管理教育如 EMBA、MBA 等均将 BPM 包含在内。

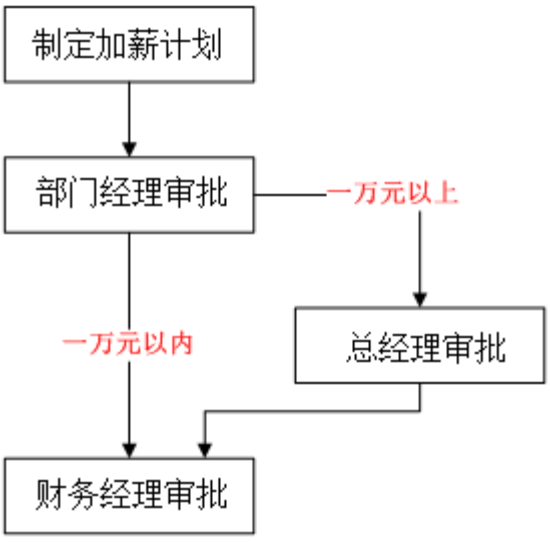
企业流程管理主要是对企业内部改革，改变企业职能管理机构重叠、中间层次多、流程不闭环等，做到机构不重叠、业务不重复，达到缩短流程周期、节约运作资本、提高企业效益的作用。

比较下边的两个人事加薪流程哪个效率更高？

流程一：



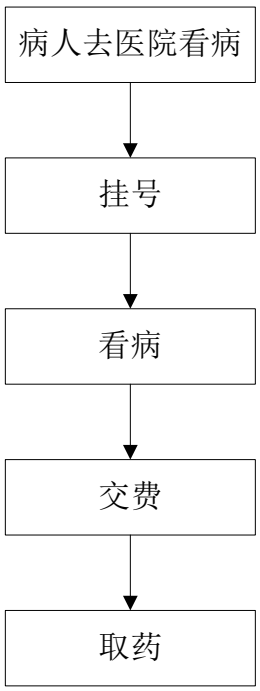
流程二：



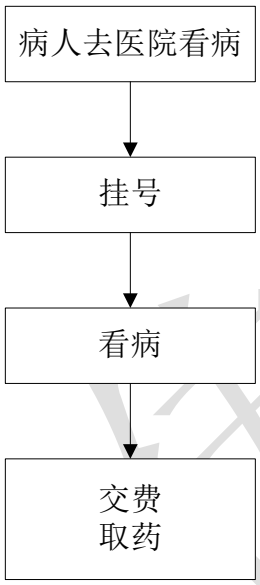
上边两个流程的区别在于第二个流程在执行时，如果本次加薪金额在一万元以内不再由总经理审批将比第一个流程缩短流程周期，从而提交效率。

再比较下边的例子，哪个效率更高？

流程一：



流程二：



上边两个流程的区别在于第二个流程将交费和取药放在一起进行，这样导致的结果是此窗口的工作人员必须具备财务、药学专业知识，岗位强度加大，人员培训难度加大从而导致人员不易扩展，工作效率低下。

2.1.2 BPM 软件

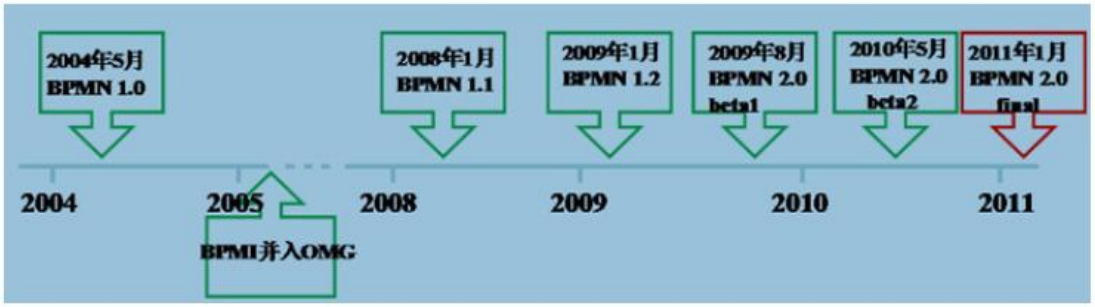
BPM 软件就是根据企业中业务环境的变化，推进人与人之间、人与系统之间以及系统与系统之间的整合及调整的经营方法与解决方案的 IT 工具。 通常以 Internet 方式实现信息传递、数据同步、业务监控和企业业务流程的持续升级优化，从而实现跨应用、跨部门、跨合作伙伴与客户的企业运作。通过 BPM 软件对企业内部及外部的业务流程的整个生命周期进行建模、自动化、管理监控和优化，使企业成本降低，利润得以大幅提升。

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。

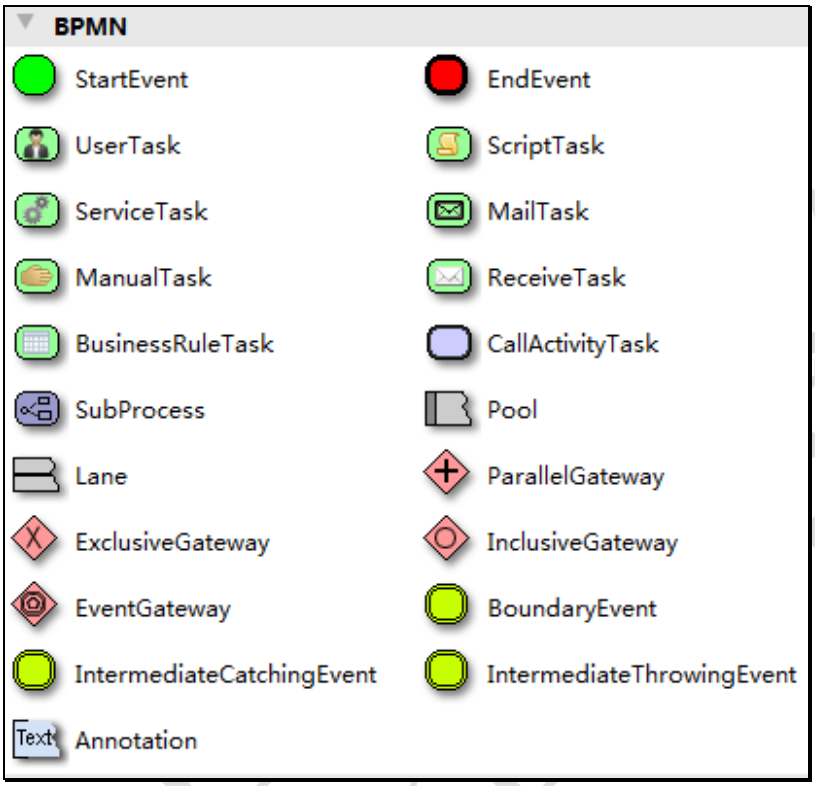
具体发展历史如下:



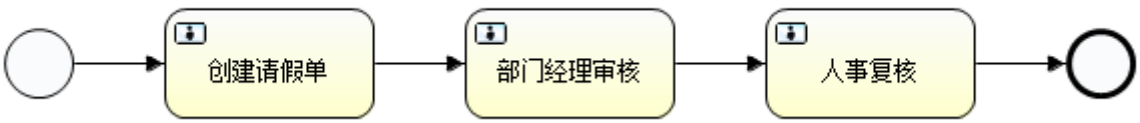
BPMN 是目前被各 BPM 厂商广泛接受的 BPM 标准。Activiti 就是使用 BPMN 2.0 进行流程建模、流程执行管理，它包括很多的建模符号，比如：

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



一个 bpmn 图形的例子：
首先当事人发起一个请假单；
其次他所在部门的经理对请假单进行审核；
然后人事经理进行复核并进行备案；
最后请假流程结束。



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：

```
<?xml version="1.0" encoding="UTF-8"?>
<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">
      <process id="myProcess" name="My process" isExecutable="true">
        <startEvent id="startevent1" name="Start"></startEvent>
        <userTask id="usertask1" name="创建请假单"></userTask>
        <sequenceFlow              id="flow1"              sourceRef="startevent1"
targetRef="usertask1"></sequenceFlow>
        <userTask id="usertask2" name="部门经理审核"></userTask>
        <sequenceFlow              id="flow2"              sourceRef="usertask1"
targetRef="usertask2"></sequenceFlow>
        <userTask id="usertask3" name="人事复核"></userTask>
        <sequenceFlow              id="flow3"              sourceRef="usertask2"
targetRef="usertask3"></sequenceFlow>
        <endEvent id="endevent1" name="End"></endEvent>
        <sequenceFlow              id="flow4"              sourceRef="usertask3"
targetRef="endevent1"></sequenceFlow>
      </process>
      <bpmndi:BPMNDiagram id="BPMNDiagram_myProcess">
        <bpmndi:BPMNPlane bpmnElement="myProcess" id="BPMNPlane_myProcess">
          <bpmndi:BPMNShape bpmnElement="startevent1" id="BPMNShape_startevent1">
            <omgdc:Bounds        height="35.0"        width="35.0"        x="130.0"
y="160.0"></omgdc:Bounds>
          </bpmndi:BPMNShape>
          <bpmndi:BPMNShape bpmnElement="usertask1" id="BPMNShape_usertask1">
            <omgdc:Bounds        height="55.0"        width="105.0"        x="210.0"
y="150.0"></omgdc:Bounds>
          </bpmndi:BPMNShape>
          <bpmndi:BPMNShape bpmnElement="usertask2" id="BPMNShape_usertask2">
            <omgdc:Bounds        height="55.0"        width="105.0"        x="360.0"
y="150.0"></omgdc:Bounds>
          </bpmndi:BPMNShape>
          <bpmndi:BPMNShape bpmnElement="usertask3" id="BPMNShape_usertask3">
            <omgdc:Bounds        height="55.0"        width="105.0"        x="510.0"
```

```
y="150.0"></omgdc:Bounds>
        </bpmndi:BPMNShape>
        <bpmndi:BPMNShape bpmnElement="endevent1" id="BPMNShape_endevent1">
          <omgdc:Bounds         height="35.0"         width="35.0"         x="660.0"
y="160.0"></omgdc:Bounds>
        </bpmndi:BPMNShape>
        <bpmndi:BPMNEdge bpmnElement="flow1" id="BPMNEdge_flow1">
          <omgdi:waypoint x="165.0" y="177.0"></omgdi:waypoint>
          <omgdi:waypoint x="210.0" y="177.0"></omgdi:waypoint>
        </bpmndi:BPMNEdge>
        <bpmndi:BPMNEdge bpmnElement="flow2" id="BPMNEdge_flow2">
          <omgdi:waypoint x="315.0" y="177.0"></omgdi:waypoint>
          <omgdi:waypoint x="360.0" y="177.0"></omgdi:waypoint>
        </bpmndi:BPMNEdge>
        <bpmndi:BPMNEdge bpmnElement="flow3" id="BPMNEdge_flow3">
          <omgdi:waypoint x="465.0" y="177.0"></omgdi:waypoint>
          <omgdi:waypoint x="510.0" y="177.0"></omgdi:waypoint>
        </bpmndi:BPMNEdge>
        <bpmndi:BPMNEdge bpmnElement="flow4" id="BPMNEdge_flow4">
          <omgdi:waypoint x="615.0" y="177.0"></omgdi:waypoint>
          <omgdi:waypoint x="660.0" y="177.0"></omgdi:waypoint>
        </bpmndi:BPMNEdge>
      </bpmndi:BPMNPlane>
    </bpmndi:BPMNDiagram>
  </definitions>
```

2.2 Activit 如何使用

1) 部署 activiti

Activiti 是一个工作流引擎（其实就是一堆 jar 包 API），业务系统使用 activiti 来对系统的业务流程进行自动化管理，为了方便业务系统访问(操作)activiti 的接口或功能，通常将 activiti 环境与业务系统的环境集成在一起。

2) 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。

如果使用其它公司开发的工作作引擎一般都提供了可视化的建模工具(Process Designer)用于生成流程定义文件，建模工具操作直观，一般都支持图形化拖拽方式、多窗口的用户界面、丰富的过程图形元素、过程元素拷贝、粘贴、删除等功能。

3) 流程定义部署

向activiti部署业务流程定义（.bpmn文件）。

使用activiti提供的api向activiti中部署.bpmn文件（一般情况还需要一块儿部署业务流程的图片.png）

4) 启动一个流程实例（ProcessInstance）

启动一个流程实例表示开始一次业务流程的运行，比如员工请假流程部署完成，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，就好比定义一个java类，实例化两个对象一样，部署的流程就好比java类，启动一个流程实例就好比new一个java对象。

5) 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，而不像上边需要我们在sql语句中的where条件中指定当前查询的状态值是多少。

6) 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，不需要我们在代码中硬编码指定下一个任务办理人了。

7) 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。

第3章 环境准备

3.1 三个环境

第一个环境：没有加入工作流SaaS-IHRM系统

作用：主要是为activiti工作流引擎的引入提供场景

第二个环境：activiti测试环境

作用：用于测试activiti的api，提供各种service接口。

需要创建一个数据库：

仅仅有activiti的数据表

第三个环境：activiti 应用环境，加入工作流的 SaaS-IHRM 系统
需要创建一个数据库：
包括 activiti 的数据表和业务表（SaaS-IHRM 系统的表）

3.2 开发环境

3.2.1 Java 环境

Jdk1.8 或以上版本。

3.2.2 数据库

Mysql 5 及以上的版本
本教程采用 5.5 版本

3.2.3 Web 容器

本项目采用的是 Tomcat8.5

3.2.4 开发工具

Mysql 客户端连接工具，Sqlyog 或其它
文本编辑器 EditPlus 或其它
Java 开发工具：IDEA 或 Eclipse 工具

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。

3.3 Activiti 环境

Activiti7.0.0.Beta1
默认支持 spring5

3.3.1 下载 activiti7

Activiti 下载地址：http://activiti.org/download.html

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.activiti</groupId>
            <artifactId>activiti-dependencies</artifactId>
            <version>7.0.0.Beta1</version>
            <scope>import</scope>
            <type>pom</type>
        </dependency>
    </dependencies>
</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 等，该目录存放 activiti 的建表脚本。

2) Docs

Activiti 的帮助文档。

3) Wars

官方自带的示例工程。

3.3.2 Activiti Designer 流程设计器(Eclipse 工具)

本教程使用 Activiti –Designer-eclipse-plugin（activiti 流程设计器插件）完成流程的制作。

下面介绍了 activiti designer 设计器插件的安装方式，本教程使用的插件安装方式详细参考“activiti 开发环境配置.docx”文档的“Eclipse 插件安装”章节。

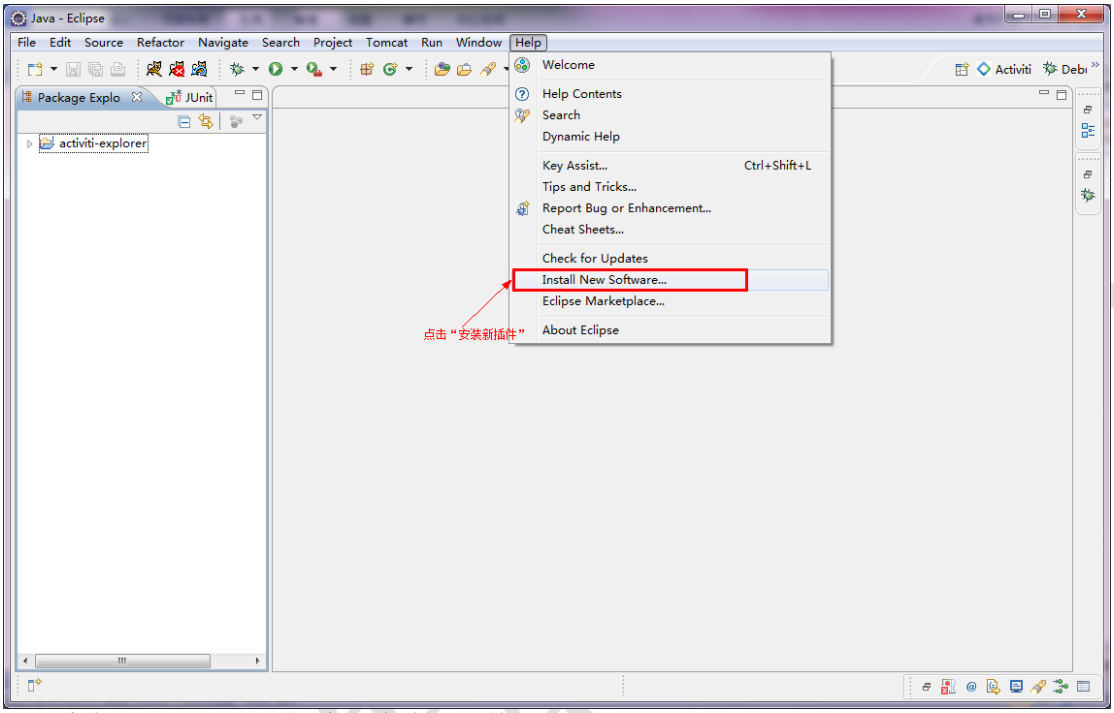
3.3.2.1 Eclispe 工具下插件安装方式 1

参数文档开发工具目录下的“activiti 开发环境配置.docx”中“eclipse 插件安装”，其中包括了 Activiti 插件。

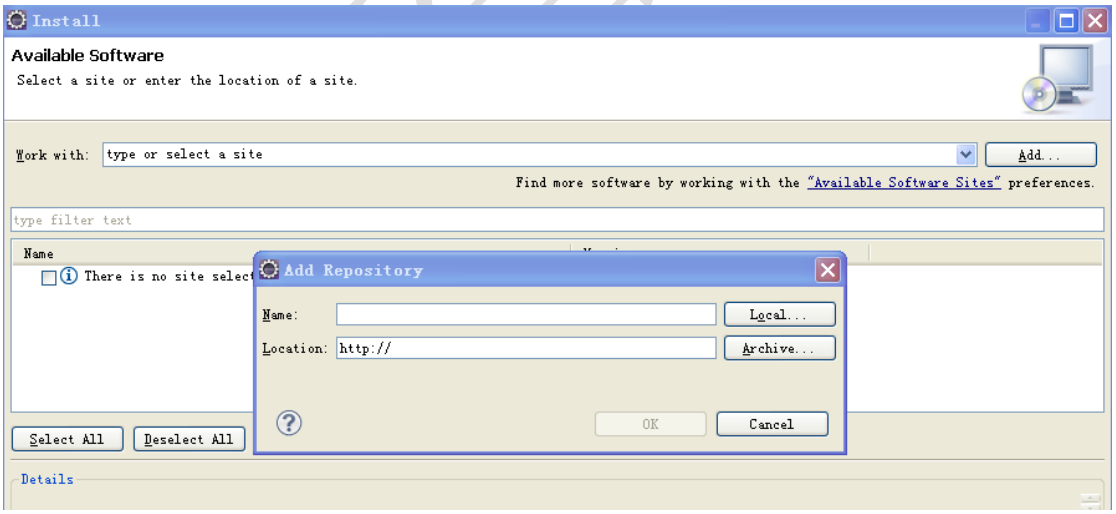
3.3.2.2 Eclispe 工具下插件安装方式 2

网络在线安装：

1） 打开 Help -> Install New Software. 在如下面板中：



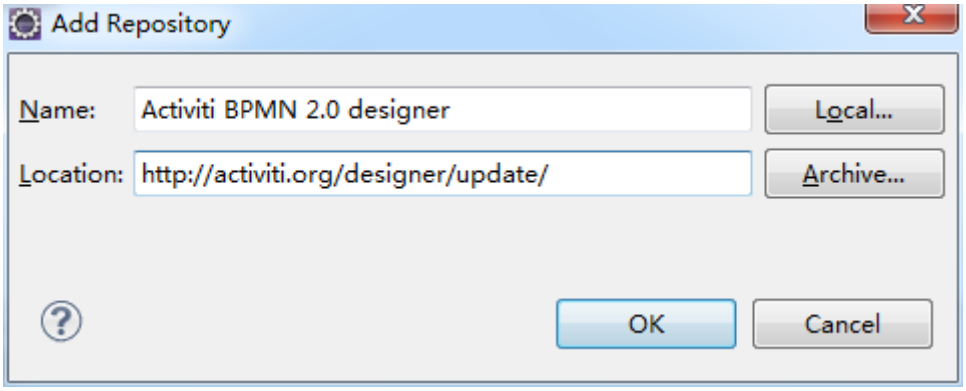
2） 在如下 Install 界面板中，点击 Add 按钮：



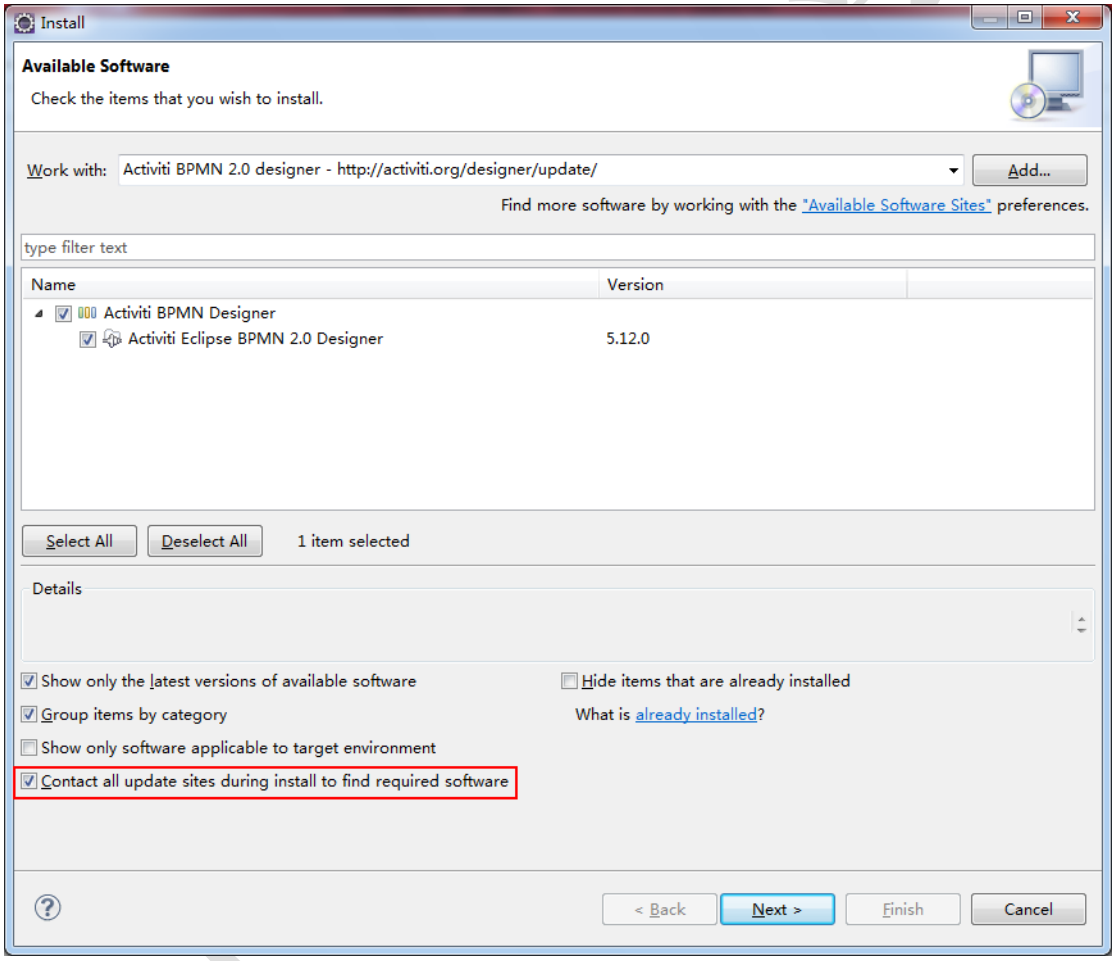
配置新装插件的地址和名称

3） 然后填入下列字段

Name: Activiti BPMN 2.0 designer
Location: http://activiti.org/designer/update/



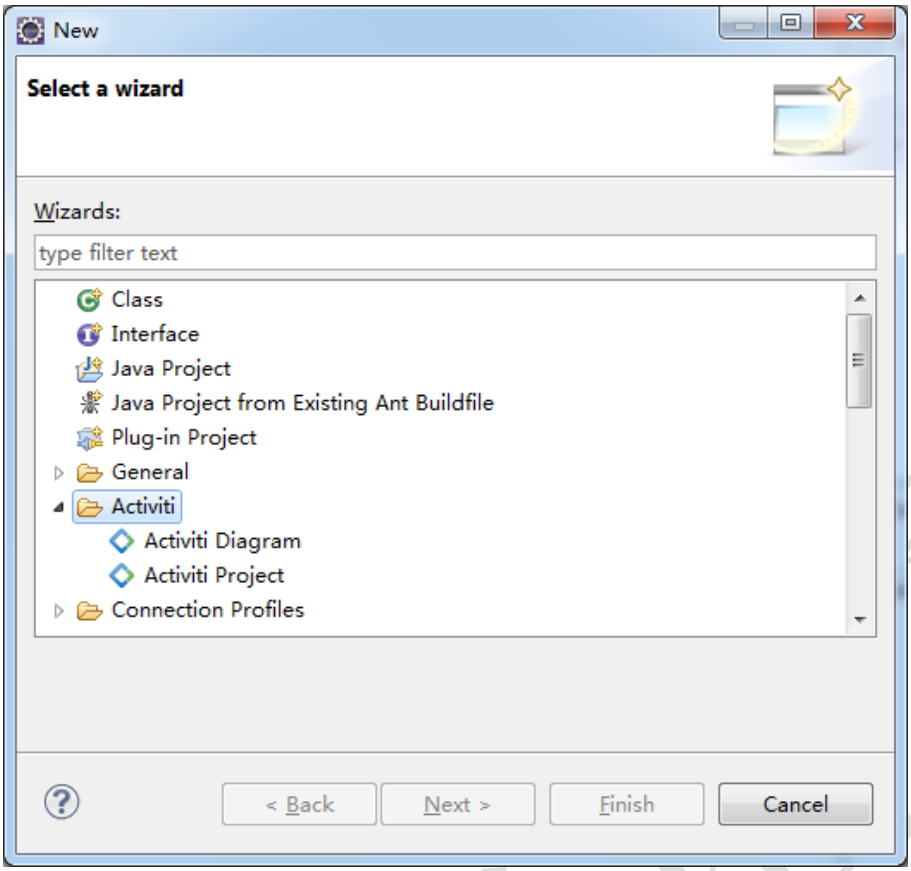
4) 回到 Install 界面，在面板正中列表中把所有展示出来的项目都勾上：



5) 点击复选框

在 Detail 部分记得选中 "Contact all updates sites.." ，因为它会检查所有当前安装所需要的插件并可以被 Eclipse 下载.

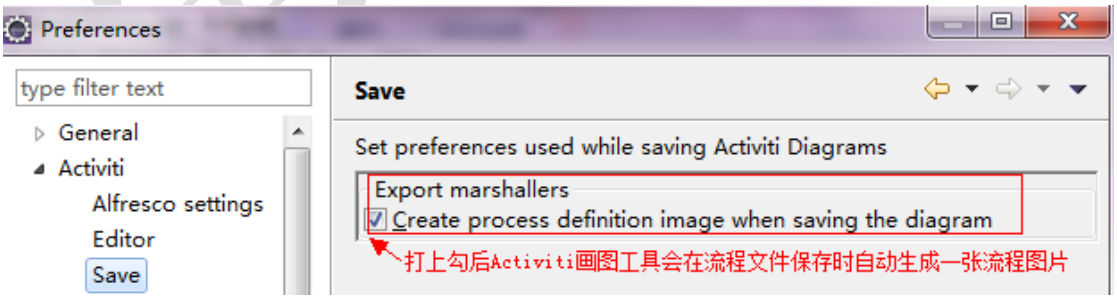
6) 安装完以后，点击新建工程 new->Other…打开面板，如果看到下图内容：



说明安装成功了。

3.3.2.3 补充说明

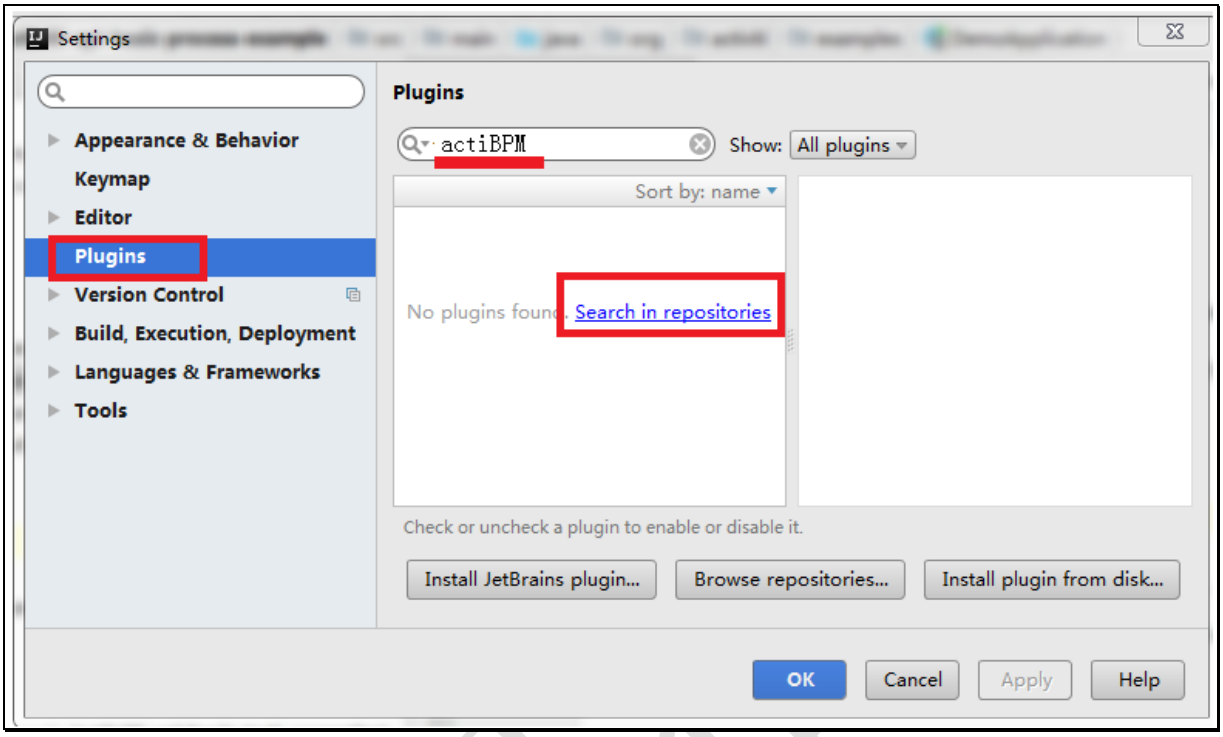
打开菜单 Windows->Preferences->Activiti->Save 下流程图片的生成方式:



虽然流程引擎在单独部署 bpmn 文件时会自动生成图片，但在实际开发过程中，自动生成的图片会导致和 BPMN 中的坐标有出入，在实际项目中展示流程当前位置图会有问题。
所在完成以上配置后，会由我们自己来管理流程图片。在发布流程时把流程规则文件和流程图片一起上传就行了。

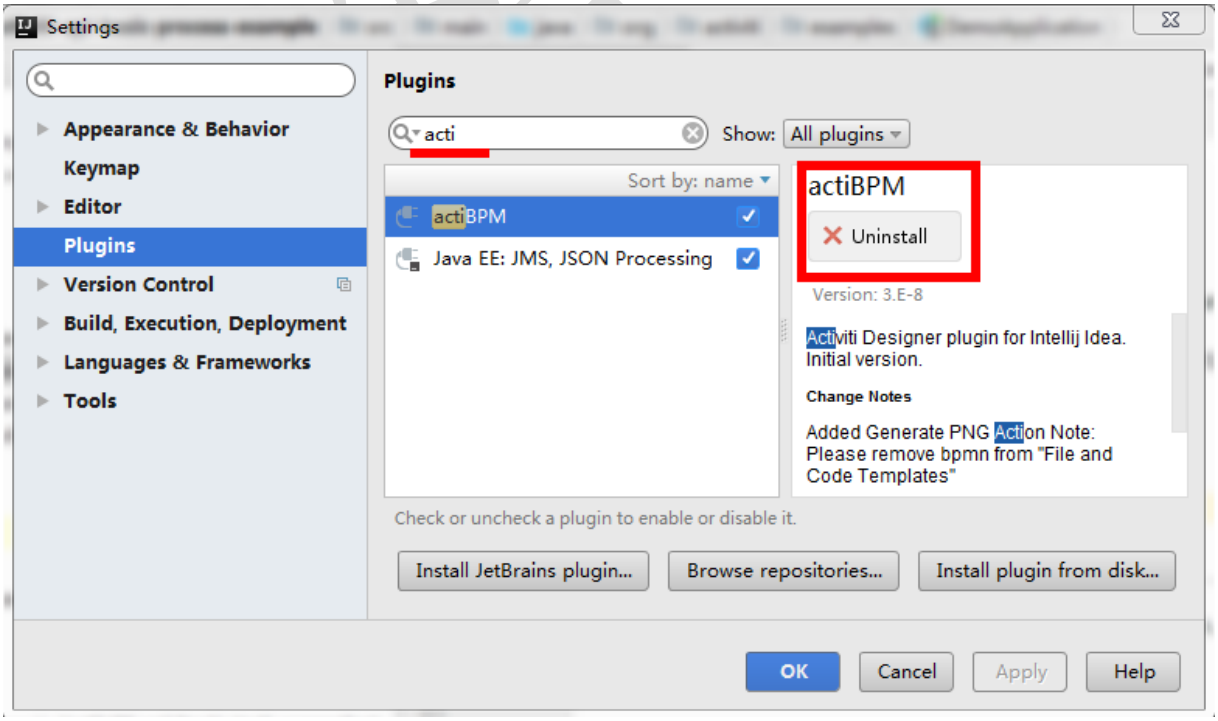
3.3.3 Activiti Designer 流程设计器(IDEA 工具)

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。

安装好后，页面如下：



3.3.4 Activiti 支持的数据库

Activiti 的运行需要数据库支撑，需要安装 activiti 数据库，支持如下版本：

Activiti 数据库类型	测试版本	JDBC URL 实例	备注
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	

3.3.5 创建 mysql 数据库

本教程使用 mysql 数据库。

创建 mysql 数据库 activiti（名字任意）：

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

3.3.6 创建表方式

通过运行 java 程序创建表。

3.3.6.1 创建 java 工程

使用 eclipse 或 idea 创建 maven 的 java 工程。

3.3.6.2 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) 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。

```
<properties>
    <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>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.40</version>
</dependency>

<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
</dependency>

<!-- log start -->
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>${log4j.version}</version>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>${slf4j.version}</version>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
    <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>
```

3.3.6.3 log4j.properties

```
# 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

# LOGFILE is set to be a File appender using a PatternLayout.
log4j.appender.LOGFILE=org.apache.log4j.FileAppender
log4j.appender.LOGFILE.File=d:\axis.log
log4j.appender.LOGFILE.Append=true
log4j.appender.LOGFILE.layout=org.apache.log4j.PatternLayout
log4j.appender.LOGFILE.layout.ConversionPattern=%d{ISO8601}                    %-6r
[%15.15t] %-5p %30.30c %x - %m\n
```

3.3.6.4 activiti.cfg.xml

在 classpath 下创建 activiti.cgf.xml 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"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
```

```
http://www.springframework.org/schema/beans/spring-beans.xsd
                http://www.springframework.org/schema/contex
http://www.springframework.org/schema/context/spring-context.xsd
                http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx.xsd">



</beans>
```

在 activiti.cfg.xml 中配置*数据源*和 *processEngineConfiguration*

1) 数据源

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <property          name="url"          value="jdbc:mysql://localhost:3306/
itcast0711activiti" />
        <property name="username" value="root" />
        <property name="password" value="root" />
        <property name=" maxActive" value="3" />
        <property name="maxIdle" value="1" />
    </bean>
```

2) processEngineConfiguration

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

```
    <bean id="processEngineConfiguration"
    class="org.activiti.engine.impl.cfg.StandaloneProcessEngineConfiguration">
        <!-- 数据源 -->
        <property name="dataSource" ref="dataSource" />
        <!-- activiti数据库表处理策略 -->
        <property name="databaseSchemaUpdate" value="true"/>
    </bean>
```

关于 *processEngineConfiguration* 中的 **databaseSchemaUpdate 参数，**通过此参数设计 activiti 数据表的处理策略，参数如下：

false（默认）：检查数据库表的版本和依赖库的版本， 如果版本不匹配就抛出异常。
true: 构建流程引擎时，执行检查，如果需要就执行更新。 如果表不存在，就创建。
create-drop: 构建流程引擎时创建数据库表， 关闭流程引擎时删除这些表。
drop-create：先删除表再创建表。
create: 构建流程引擎时创建数据库表， 关闭流程引擎时不删除这些表。

注意：在 activiti.cfg.xml 配置文件中的 dataSource 和 processEngineConfiguration 也可以使用一次性配置出来。

```
<bean
id="processEngineConfiguration"
    class="org.activiti.engine.impl.cfg.StandaloneProcessEngineConfiguration">
        <property name="jdbcDriver" value="com.mysql.jdbc.Driver"/>
        <property                                                    name="jdbcUrl"
    value="jdbc:mysql://localhost:3306/itcast0711activiti"/>
        <property name="jdbcUsername" value="root"/>
        <property name="jdbcPassword" value="root"/>
        <property name="databaseSchemaUpdate" value="true"/>
    </bean>
```

3.3.6.5 编写程序

创建 ProcessEngineConfiguration，通过 ProcessEngineConfiguration 创建 ProcessEngine，在创建 ProcessEngine 时会自动创建数据库。

```
//创建ProcessEngineConfiguration
        ProcessEngineConfiguration           configuration           =
ProcessEngineConfiguration
    .createProcessEngineConfigurationFromResource("activiti.cfg.xml")
        //通过ProcessEngineConfiguration创建ProcessEngine，此时会创建数据
库
        ProcessEngine                    processEngine                   =
configuration.buildProcessEngine();
        System.out.println(processEngine);
```

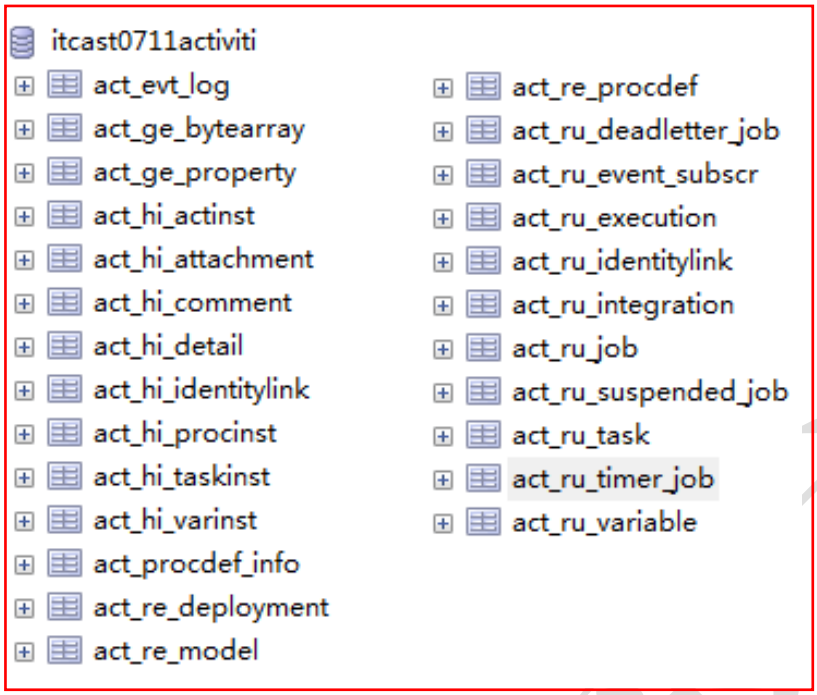
说明：

1、运行以上程序段即可完成 activiti 数据库创建，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 *createProcessEngineConfigurationFromResource* 在执行时在 *activiti.cfg.xml* 中找固定的名称 *processEngineConfiguration*
也可以使用重载方法调用，这时可以不用限定 *processEngineConfiguration* 名称

```
createProcessEngineConfigurationFromResource(String resource, String beanName)
```

此时我们查看数据库，创建了25张表，结果如下：

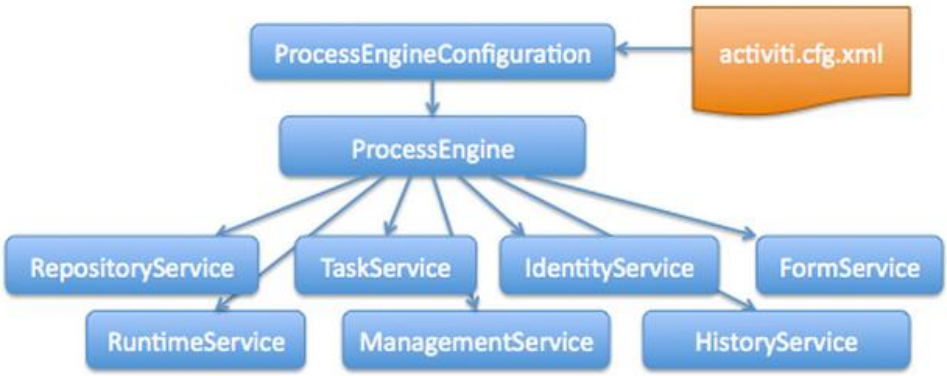


3.3.7 数据库表的命名规则

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。

第4章 Activiti 服务架构图



在新版本中，我们通过实验可以发现 IdentityService，FormService 两个 Serivce 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。

4.1 activiti.cfg.xml

activiti 的引擎配置文件，包括：ProcessEngineConfiguration 的定义、数据源定义、事务管理器等，此文件其实就是一个 spring 配置文件，下面是一个基本的配置只配置了 ProcessEngineConfiguration 和数据源：

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:tx="http://www.springframework.org/schema/tx"
    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-3.1.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.1.xsd
    http://www.springframework.org/schema/tx
     http://www.springframework.org/schema/tx/spring-tx-3.1.xsd">



    <!--数据库连接池 -->
    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <property name="url" value="jdbc:mysql://localhost:3306/activiti" />
        <property name="username" value="root" />
        <property name="password" value="mysql" />
    </bean>
    <bean id="processEngineConfiguration"
```

```
    class="org.activiti.engine.impl.cfg.StandaloneProcessEngineConfiguration">
        <!-- 数据源 -->
        <property name="dataSource" ref="dataSource" />
        <!-- 数据库策略 -->
        <property name="databaseSchemaUpdate" value="true"/>
    </bean>

</beans>
```

4.2 ProcessEngineConfiguration：

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

4.2.1 StandaloneProcessEngineConfiguration

通过 org.activiti.engine.impl.cfg.StandaloneProcessEngineConfiguration

Activiti 可以单独运行，使用它创建的 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，这里会使用 spring 的依赖注入来构建引擎。
方法如下：

```
<bean id="processEngineConfiguration"

    class="org.activiti.engine.impl.cfg.StandaloneProcessEngineConfig
uration">
        <!-- 数据源 -->
        <property name="dataSource" ref="dataSource" />
        <!-- 数据库策略 -->
        <property name="databaseSchemaUpdate" value="true"/>
    </bean>
```

4.2.2 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（名称不固定）

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc-3.1.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.1.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-3.1.xsd
        http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-3.1.xsd ">

    <!-- 工作流引擎配置bean -->
    <bean                                    id="processEngineConfiguration"
class="org.activiti.spring.SpringProcessEngineConfiguration">
        <!-- 数据源 -->
        <property name="dataSource" ref="dataSource" />
        <!-- 使用spring事务管理器 -->
        <property name="transactionManager" ref="transactionManager" />
        <!-- 数据库策略 -->
        <property name="databaseSchemaUpdate" value="drop-create" />
        <!-- activiti的定时任务关闭 -->
        <property name="jobExecutorActivate" value="false" />
    </bean>


    <!-- 流程引擎 -->
    <bean                                              id="processEngine"
class="org.activiti.spring.ProcessEngineFactoryBean">
```

```
        <property                                    name="processEngineConfiguration"
ref="processEngineConfiguration" />
    </bean>
    <!-- 资源服务service -->
    <bean id="repositoryService" factory-bean="processEngine"
        factory-method="getRepositoryService" />
    <!-- 流程运行service -->
    <bean id="runtimeService" factory-bean="processEngine"
        factory-method="getRuntimeService" />
    <!-- 任务管理service -->
    <bean id="taskService" factory-bean="processEngine"
        factory-method="getTaskService" />
    <!-- 历史管理service -->
    <bean id="historyService" factory-bean="processEngine"
        factory-method="getHistoryService" />
    <!-- 用户管理service -->
    <bean id="identityService" factory-bean="processEngine"
        factory-method="getIdentityService" />
    <!-- 引擎管理service -->
    <bean id="managementService" factory-bean="processEngine"
        factory-method="getManagementService" />


    <!-- 数据源 -->
    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <property name="url" value="jdbc:mysql://localhost:3306/activiti" />
        <property name="username" value="root" />
        <property name="password" value="mysql" />
        <property name="maxActive" value="3" />
        <property name="maxIdle" value="1" />
    </bean>
    <!-- 事务管理器 -->
    <bean id="transactionManager"

    class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource" />
    </bean>

    <!-- 通知 -->
    <tx:advice id="txAdvice" transaction-manager="transactionManager">
        <tx:attributes>
            <!-- 传播行为 -->
            <tx:method name="save*" propagation="REQUIRED" />
```

```
            <tx:method name="insert*" propagation="REQUIRED" />
            <tx:method name="delete*" propagation="REQUIRED" />
            <tx:method name="update*" propagation="REQUIRED" />
            <tx:method name="find*" propagation="SUPPORTS" read-only="true" />
            <tx:method name="get*" propagation="SUPPORTS" read-only="true" />
        </tx:attributes>
    </tx:advice>

    <!-- 切面，根据具体项目修改切点配置 -->
    <aop:config proxy-target-class="true">
        <aop:advisor advice-ref="txAdvice"
            pointcut="execution(* com.itheima.ihrm.service.impl.*.*(..))" />
    </aop:config>
</beans>
```

4.2.3 创建processEngineConfiguration

```
        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：

```
ProcessEngineConfiguration.createProcessEngineConfigurationFromResou
rce(String resource, String beanName);
```

4.3 ProcessEngine

工作流引擎，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。

4.3.1 一般创建方式

```
//通过ProcessEngineConfiguration创建ProcessEngine
        ProcessEngine processEngine =
processEngineConfiguration.buildProcessEngine();
```

4.3.2 简单创建方式

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:

```
//使用classpath下的activiti.cfg.xml中的配置创建processEngine
      ProcessEngine                    processEngine                    =
ProcessEngines.getDefaultProcessEngine();
      System.out.println(processEngine);
```

4.4 Service

4.4.1 Service 创建方式

通过 ProcessEngine 创建 Service，Service 是工作流引擎提供用于进行工作流部署、执行、管理的服务接口。
方式如下：

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

4.4.2 Service 总览

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

注：红色标注为常用 service。

4.4.3 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。使用工作流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。

4.4.4 RuntimeService

它是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

4.4.5 TaskService

是activiti的任务管理类。可以从这个类中获取任务的信息。

4.4.6 HistoryService

是 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。

4.4.7 ManagementService

是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。

第5章 Activiti 入门体验

5.1 流程定义

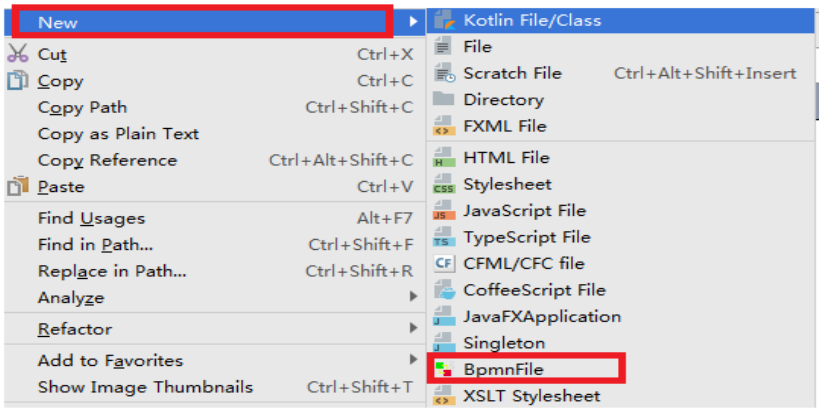
5.1.1 Activiti-Designer 使用

5.1.1.1 Palette（画板）

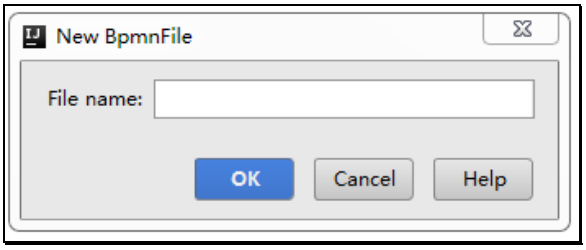
在 eclipse 或 idea 中安装 activiti-designer 插件即可使用，画板中包括以下结点：
Connection—连接
Event---事件
Task---任务
Gateway---网关
Container—容器
Boundary event—边界事件
Intermediate event- -中间事件
流程图设计完毕保存生成.bpmn 文件。

5.1.1.2 新建流程(IDEA 工具)

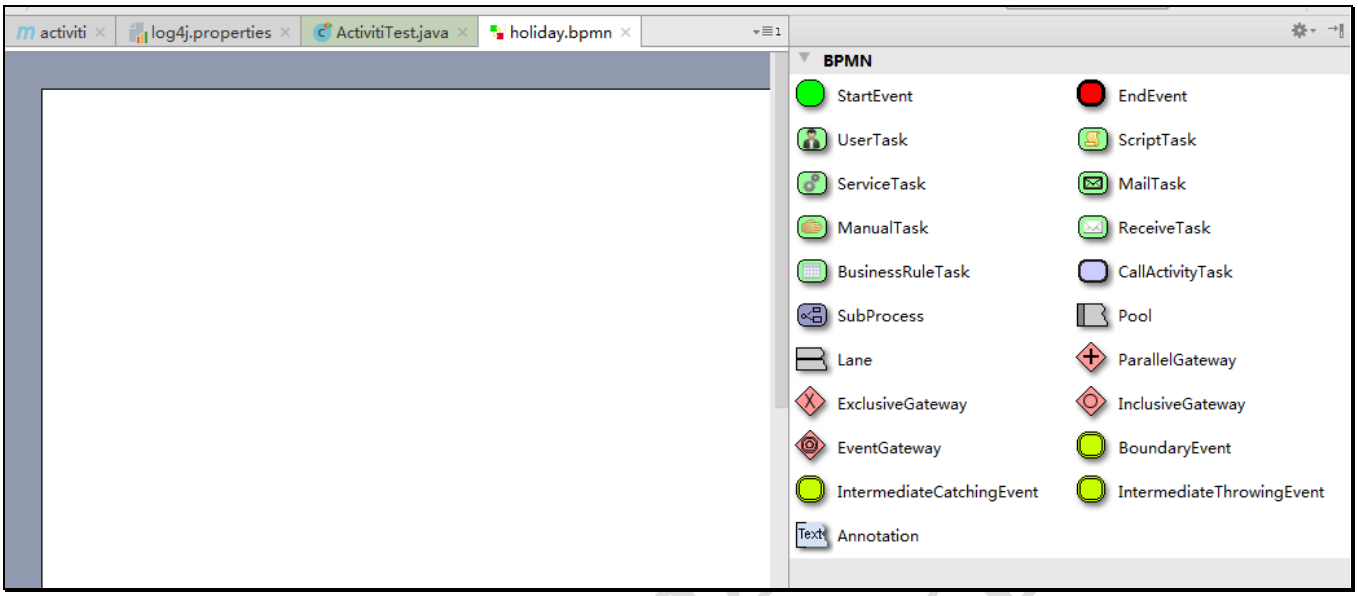
首先选中存放图形的目录(本次我们选择 resources 下的 bpmn 目录)，点击菜单：New-BpmnFile，如下图所示：



弹出如下图所示框：

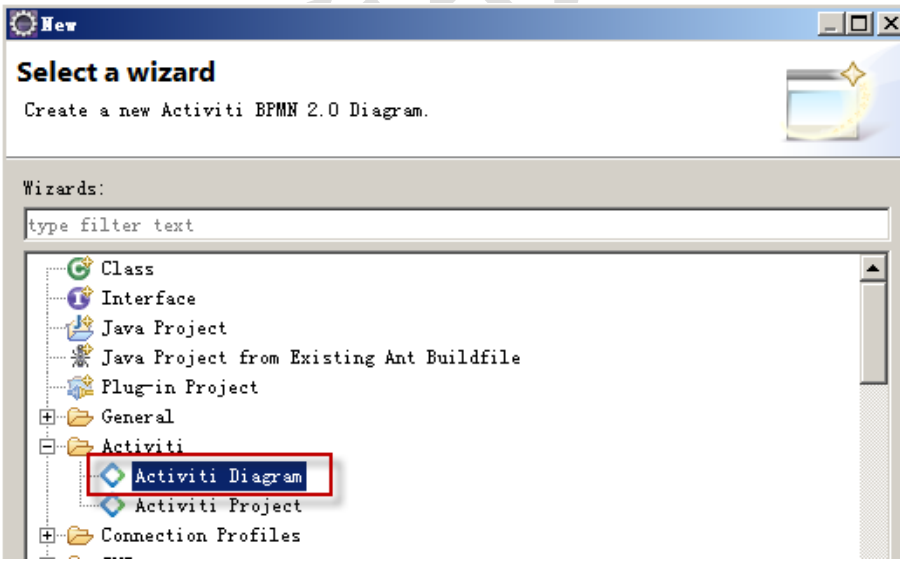


起完名字 holiday 后（默认扩展名为 bpmn），就可以看到进入了流程设计页面，如图所示：

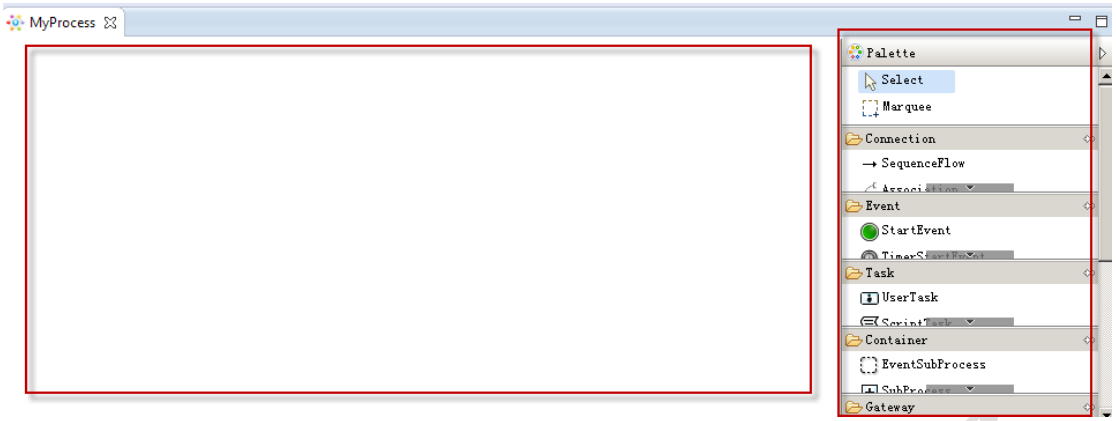


5.1.1.3 新建流程(Eclipse 工具)

首先选中存放图形的目录(本次我们选择 resources 下的 bpmn 目录)， File-New-Other 菜单，打开如下窗口。

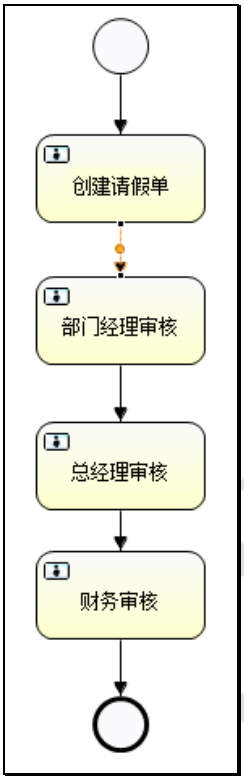


创建成功：



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。

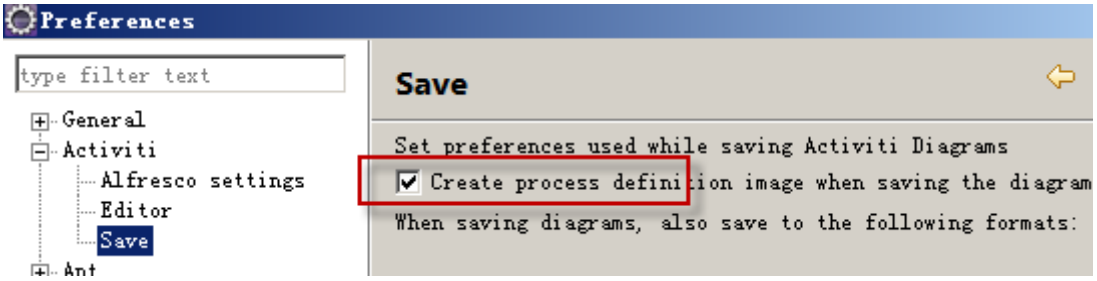
5.1.2 绘制流程



图形绘制好后会生成两个文件：

purchasingflow01.bpmn
purchasingflow01.png

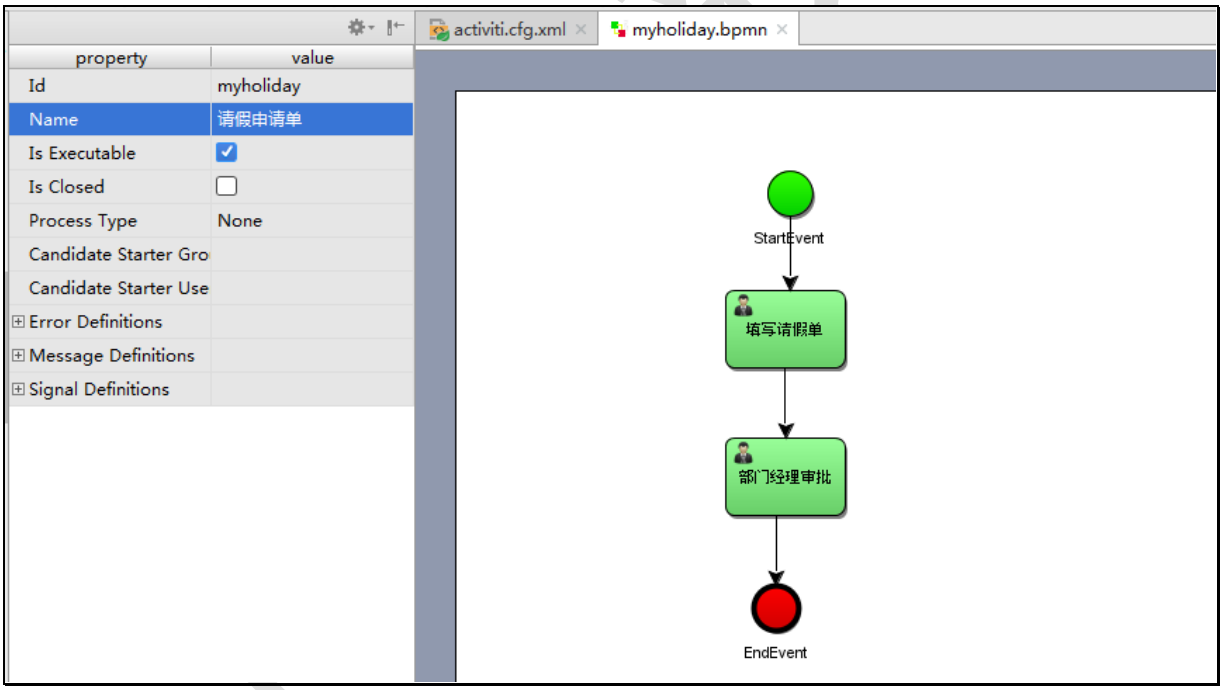
自动生成图形，需要设置 eclipse:



5.1.3 指定流程定义 key

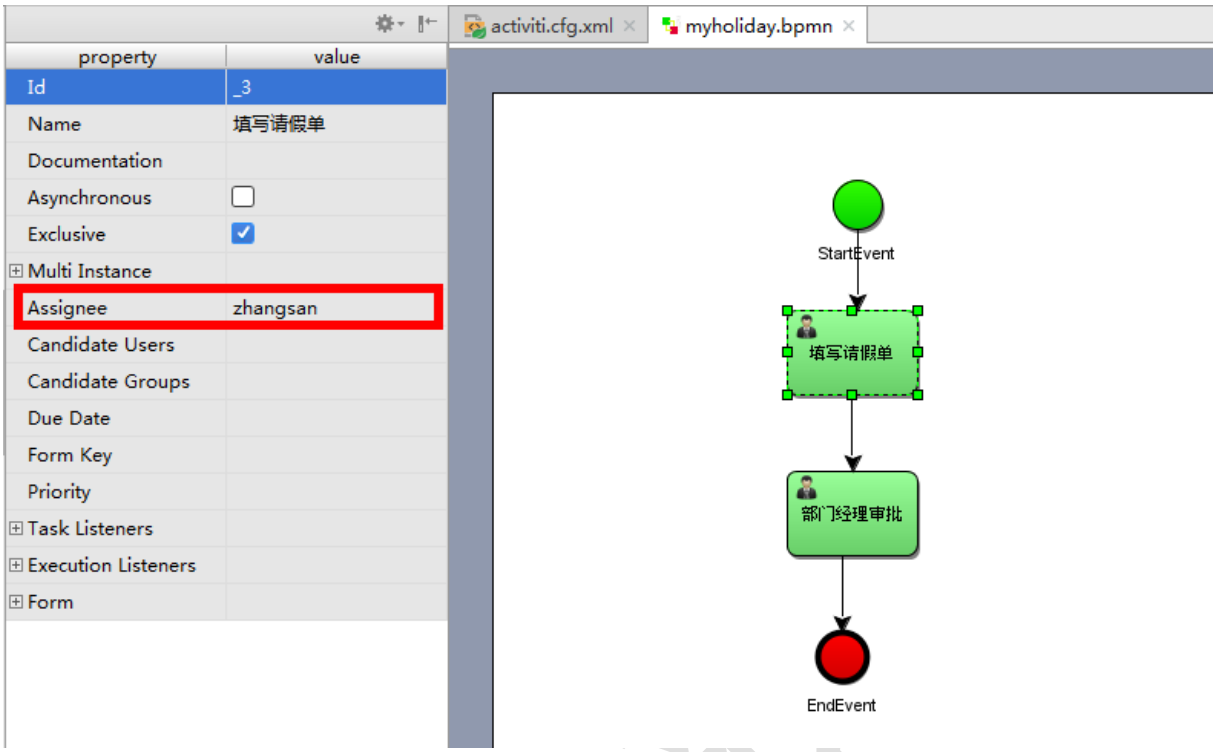
流程定义 key 即流程定义的标识，在 eclipse 中通过 properties 视图查看流程的 key

建议：相同的业务流程，流程定义的 key 名字定义一样，比如，如果需要创建新的业务流程，该业务流程则使用新的 key。



5.1.4 指定任务负责人

在 properties 视图指定每个任务结点的负责人，
比如下边是填写请假单的负责人为 zhangsan



5.2 部署流程定义

部署流程定义就是要将上边绘制的图形即流程定义（.bpmn）部署在工作流程引擎 activiti 中，方法如下：

使用 ProcessEngine 创建 RepositoryService，代码如下：

```
// 获取repositoryService
	RepositoryService repositoryService = processEngine
		.getRepositoryService();
	//部署对象
	Deployment deployment = repositoryService.createDeployment()
		.addClasspathResource("diagram/myholiday.bpmn")// bpmn文件
		.addClasspathResource("diagram/myholiday.png")// 图片文件
		.name("请假申请流程")
		.deploy();
	System.out.println("流程部署id:" + deployment.getId());
	System.out.println("流程部署名称:" + deployment.getName());
```

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。

5.3 启动一个流程实例

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请假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请假申请单，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。对于请假申请流程，张三发起一个请假申请单需要启动一个流程实例，李四发起一个请假单也需要启动一个流程实例。

代码如下：

```
// 启动一个流程实例
    @Test
    public void startProcessInstance() {

        // 获取RunTimeService
        RuntimeService                runtimeService                =
processEngine.getRuntimeService();
        // 根据流程定义key启动流程
        ProcessInstance processInstance = runtimeService
              .startProcessInstanceByKey("myholiday01");

        System.out.println("    流    程    定    义    id    :    "    +
processInstance.getProcessDefinitionId());
        System.out.println("流程实例id: " + processInstance.getId());
        System.out.println("    当    前    活    动    Id    :    "    +
processInstance.getActivityId());

    }
```

5.4 任务查询

流程启动后，各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。

```
// 查询当前个人待执行的任务
    @Test
    public void findPersonalTaskList() {
        // 任务负责人
        String assignee = "zhangsan";
```

```
        // 创建TaskService
        TaskService taskService = processEngine.getTaskService();
        List<Task> list = taskService.createTaskQuery()//
                .processDefinitionKey("myholiday01")//
                .taskAssignee(assignee)//只查询该任务负责人的任务
                .list();

        for (Task task : list) {

            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());

        }
    }
```

5.5 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。

```
// 完成任务
    @Test
    public void completTask() {
        //任务id
        String taskId = "8305";
        // 创建TaskService
        TaskService taskService = processEngine.getTaskService();
        //完成任务
        taskService.complete(taskId);
        System.out.println("完成任务id="+taskId);
    }
```

第6章 流程定义

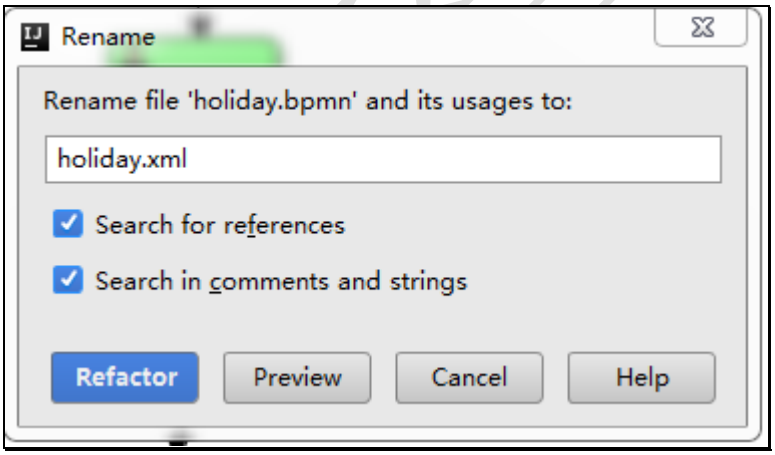
6.1 流程定义

6.1.1 什么是流程定义

流程定义是线下按照 bpmn2.0 标准去描述 业务流程，通常使用 activiti-explorer（web 控制台）或 activiti-eclipse-designer 插件对业务流程进行建模，这两种方式都遵循 bpmn2.0 标准。本教程使用 activiti-eclipse-designer 插件完成流程建模。使用 designer 设计器绘制流程，会生成两个文件：.bpmn 和.png

6.1.2 .bpmn 文件

使用 activiti-desinger 设计业务流程，会生成.bpmn 文件，
首先将 holiday.bpmn 文件改名为 holiday.xml，如下图：



.bpmn 内容如下：

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<definitions                    xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
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"
xmlns:tns="http://www.activiti.org/test"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
expressionLanguage="http://www.w3.org/1999/XPath"     id="m1539820628606"     name=""
targetNamespace="http://www.activiti.org/test"
typeLanguage="http://www.w3.org/2001/XMLSchema">
  <process id="myProcess_1" isClosed="false" isExecutable="true" processType="None">
    <startEvent id="_2" name="StartEvent"/>
    <userTask activiti:exclusive="true" id="_3" name="UserTask"/>
    <sequenceFlow id="_4" sourceRef="_2" targetRef="_3"/>
    <userTask activiti:exclusive="true" id="_5" name="UserTask"/>
    <sequenceFlow id="_6" sourceRef="_3" targetRef="_5"/>
  </process>
  <bpmndi:BPMNDiagram
documentation="background=#FFFFFF;count=1;horizontalcount=1;orientation=0;width=842
.4;height=1195.2;imageableWidth=832.4;imageableHeight=1185.2;imageableX=5.0;imageab
leY=5.0" id="Diagram-_1" name="New Diagram">
    <bpmndi:BPMNPlane bpmnElement="myProcess_1">
      <bpmndi:BPMNShape bpmnElement="_2" id="Shape-_2">
        <omgdc:Bounds height="32.0" width="32.0" x="200.0" y="70.0"/>
        <bpmndi:BPMNLabel>
          <omgdc:Bounds height="32.0" width="32.0" x="0.0" y="0.0"/>
        </bpmndi:BPMNLabel>
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape bpmnElement="_3" id="Shape-_3">
        <omgdc:Bounds height="55.0" width="85.0" x="165.0" y="175.0"/>
        <bpmndi:BPMNLabel>
          <omgdc:Bounds height="55.0" width="85.0" x="0.0" y="0.0"/>
        </bpmndi:BPMNLabel>
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape bpmnElement="_5" id="Shape-_5">
        <omgdc:Bounds height="55.0" width="85.0" x="160.0" y="295.0"/>
        <bpmndi:BPMNLabel>
          <omgdc:Bounds height="55.0" width="85.0" x="0.0" y="0.0"/>
        </bpmndi:BPMNLabel>
      </bpmndi:BPMNShape>
      <bpmndi:BPMNEdge     bpmnElement="_4"     id="BPMNEdge__4"     sourceElement="_2"
targetElement="_3">
        <omgdi:waypoint x="216.0" y="102.0"/>
        <omgdi:waypoint x="216.0" y="175.0"/>
        <bpmndi:BPMNLabel>
          <omgdc:Bounds height="0.0" width="0.0" x="0.0" y="0.0"/>
```

```
      </bpmndi:BPMNLabel>
    </bpmndi:BPMNEdge>
    <bpmndi:BPMNEdge    bpmnElement="_6"    id="BPMNEdge__6"    sourceElement="_3"
targetElement="_5">
      <omgdi:waypoint x="205.0" y="230.0"/>
      <omgdi:waypoint x="205.0" y="295.0"/>
      <bpmndi:BPMNLabel>
        <omgdc:Bounds height="0.0" width="0.0" x="0.0" y="0.0"/>
      </bpmndi:BPMNLabel>
    </bpmndi:BPMNEdge>
  </bpmndi:BPMNPlane>
</bpmndi:BPMNDiagram>
</definitions>
```

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。

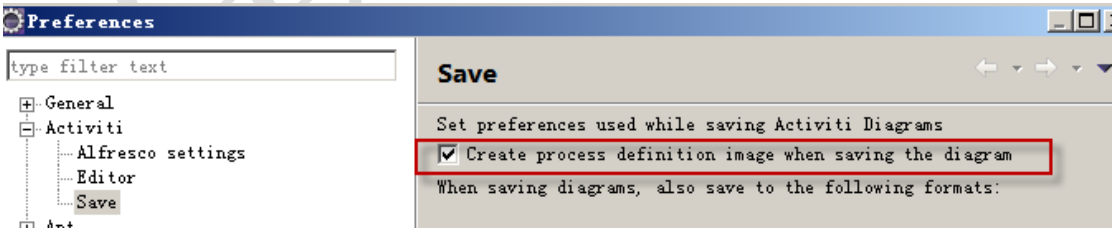
6.1.3 .png 图片文件

Eclipse 工具中的操作

流程图片生成的两种方式：

- 使用 activiti-designer 设计流程图时自动生成

需在 eclipse 中进行配置：



使用 designer 设计流程图的同时自动生成与 bpmn 文件同名的图片文件(.png)

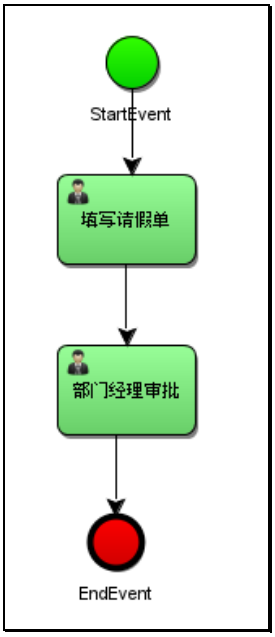
- 由 activiti 自动生成图形

流程图设计完毕向 activiti 中部署流程定义文件 bpmn，部署时由 activiti 自动生成流程图片。(流程部署在下面的章节讲解)

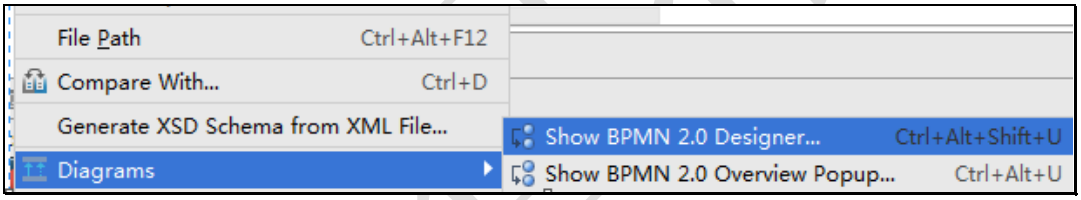
注意：此方法生成时如果图形中有中文生成的图片上显示乱码，且.bpmn 中的坐标和图片显示错位。

IDEA 工具中的操作方式

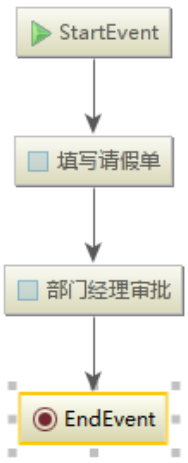
第一步：将 holiday.bpmn 文件改为扩展名 xml 的文件名称：holiday.xml
修改前的 bpmn 文件，效果如下：



第二步：在 holiday.xml 文件上面，点右键并选择 Diagrams 菜单，再选择 Show BPMN2.0 Designer…



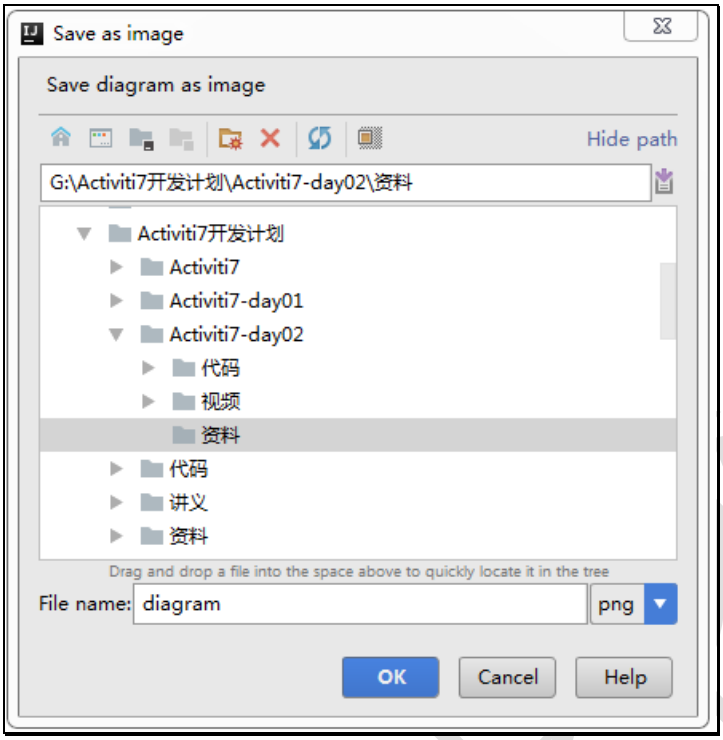
第三步：打开后的效果图如下：



第四步：

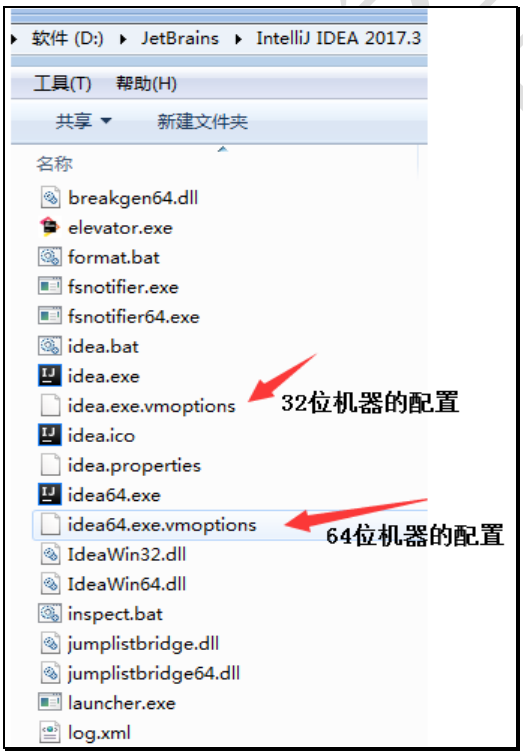
点击 Export To File 的小图标，如下：

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：

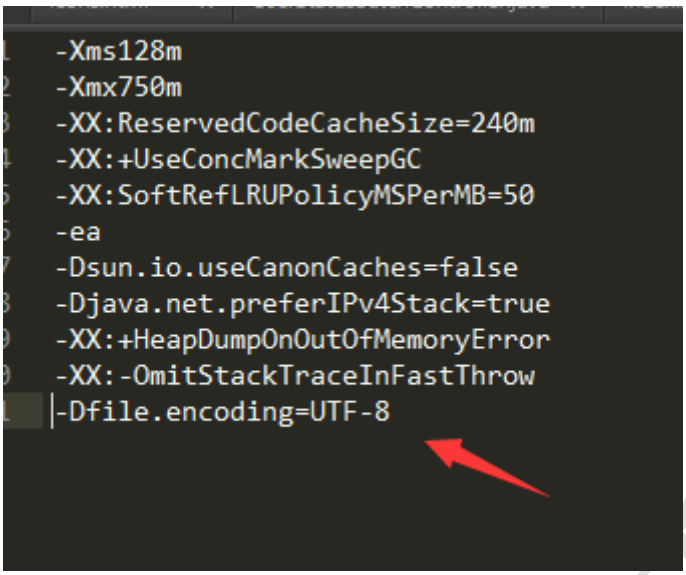


第五步：中文乱码的解决

1.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：



一定注意，不要有空格，否则重启 IDEA 时会打不开，然后 重启 IDEA，把原来的 png 图片删掉，再重新生成，即可解决乱码问题

6.2 流程定义部署

6.2.1 什么是流程定义部署

将线下定义的流程部署到 activiti 数据库中，这就是流程定义部署，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。

6.2.2 单个文件部署方式

分别将 bpmn 文件和 png 图片文件部署。

```
@Test
    public void deployProcess() {
        // 获取repositoryService
        RepositoryService repositoryService = processEngine
```

```
                .getRepositoryService();
        // bpmn输入流
        InputStream inputStream_bpmn = this
                .getClass()
                .getClassLoader()
                .getResourceAsStream(
                        "diagram/holiday.bpmn");
        // 图片输入流
        InputStream inputStream_png = this
                .getClass()
                .getClassLoader()
                .getResourceAsStream(
                        " diagram/holiday.png");
        // 流程部署对象
        Deployment deployment = repositoryService.createDeployment()
                .addInputStream("holiday.bpmn", inputStream_bpmn)
                .addInputStream("holiday.png", inputStream_png)
                .deploy();
        System.out.println("流程部署id: " + deployment.getId());
        System.out.println("流程部署名称: " + deployment.getName());
    }
```

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。

6.2.3 压缩包部署方式

将 holiday.bpmn 和 holiday.png 压缩成 zip 包。

```
@Test
    public void deployProcessByZip() {
        // 定义zip输入流
        InputStream inputStream = this
                .getClass()
                .getClassLoader()
                .getResourceAsStream(
                        " diagram/holiday.zip");
        ZipInputStream zipInputStream = new ZipInputStream(inputStream);
        // 获取repositoryService
        RepositoryService repositoryService = processEngine
                .getRepositoryService();
        // 流程部署
        Deployment deployment = repositoryService.createDeployment()//
                .addZipInputStream(zipInputStream)//
```

```
                .deploy();
        System.out.println("流程部署id: " + deployment.getId());
        System.out.println("流程部署名称: " + deployment.getName());
    }
```

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。

6.2.4 操作数据表

流程定义部署后操作 activiti 数据表如下：

SELECT * FROM act_re_deployment #流程定义部署表，记录流程部署信息
SELECT * FROM act_re_procdef #流程定义表，记录流程定义信息
SELECT * FROM act_ge_bytearray #资源表

说明：
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的关系，方便读取流程部署及流程定义信息。

6.3 流程定义查询

查询部署的流程定义。

```
// 流程定义查询
    @Test
    public void queryProceccDefinition() {
        // 流程定义key
        String processDefinitionKey = "holiday";
        // 获取repositoryService
        RepositoryService repositoryService = processEngine
                .getRepositoryService();
        // 查询流程定义
        ProcessDefinitionQuery processDefinitionQuery = repositoryService
                .createProcessDefinitionQuery();
        //遍历查询结果
        List<ProcessDefinition> list = processDefinitionQuery
                .processDefinitionKey(processDefinitionKey)
```

```
                .orderByProcessDefinitionVersion().desc().list();
        for (ProcessDefinition processDefinition : list) {
            System.out.println("------------------------");
            System.out.println("    流    程    部    署    id    :    "    +
processDefinition.getDeploymentId());
            System.out.println("流程定义id: " + processDefinition.getId());
            System.out.println("流程定义名称: " + processDefinition.getName());
            System.out.println("流程定义key: " + processDefinition.getKey());
            System.out.println("流程定义版本: " + processDefinition.getVersion());
        }

    }
```

6.4 流程定义删除

删除已经部署成功的流程定义。

```
public void deleteDeployment() {
        // 流程部署id
        String deploymentId = "8801";
        // 通过流程引擎获取repositoryService
        RepositoryService repositoryService = processEngine
                .getRepositoryService();
        //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
        repositoryService.deleteDeployment(deploymentId);
        //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非级别删除方式，如果流程
        //repositoryService.deleteDeployment(deploymentId, true);
    }
```

说明：

1) 使用 repositoryService 删除流程定义

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。

3) 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。项目开发中使用级联删除的情况比较多，删除操作一般只开放给超级管理员使用。

6.5 流程定义资源查询

6.5.1 方式 1

通过流程定义对象获取流程定义资源，获取 bpmn 和 png。

```
@Test
    public void getProcessResources() throws IOException {

        // 流程定义id
        String processDefinitionId = "";
        // 获取repositoryService
        RepositoryService repositoryService = processEngine
                .getRepositoryService();
        // 流程定义对象
        ProcessDefinition processDefinition = repositoryService
                .createProcessDefinitionQuery()

    .processDefinitionId(processDefinitionId).singleResult();
        //获取bpmn
        String resource_bpmn = processDefinition.getResourceName();
        //获取png
        String                    resource_png                    =
processDefinition.getDiagramResourceName();

        // 资源信息
        System.out.println("bpmn: " + resource_bpmn);
        System.out.println("png: " + resource_png);

        File file_png = new File("d:/purchasingflow01.png");
        File file_bpmn = new File("d:/purchasingflow01.bpmn");

        // 输出bpmn
        InputStream resourceAsStream = null;
        resourceAsStream = repositoryService.getResourceAsStream(
                processDefinition.getDeploymentId(), resource_bpmn);

        FileOutputStream        fileOutputStream        =        new
FileOutputStream(file_bpmn);
```

```
        byte[] b = new byte[1024];
        int len = -1;
        while ((len = resourceAsStream.read(b, 0, 1024)) != -1) {
            fileOutputStream.write(b, 0, len);
        }

        // 输出图片
        resourceAsStream = repositoryService.getResourceAsStream(
                processDefinition.getDeploymentId(), resource_png);

        fileOutputStream = new FileOutputStream(file_png);
        // byte[] b = new byte[1024];
        // int len = -1;
        while ((len = resourceAsStream.read(b, 0, 1024)) != -1) {
            fileOutputStream.write(b, 0, len);
        }

    }
```

6.5.2 方式 2

通过查询流程部署信息获取流程定义资源。

```
// 获取流程定义图片资源
    @Test
    public void getProcessResources() throws IOException {
        //流程部署id
        String deploymentId = "9001";
        // 通过流程引擎获取repositoryService
        RepositoryService repositoryService = processEngine
                .getRepositoryService();
        //读取资源名称
        List<String>                    resources                    =
repositoryService.getDeploymentResourceNames(deploymentId);
        String resource_image = null;
        //获取图片
        for(String resource_name :resources){
            if(resource_name.indexOf(".png")>=0){
                resource_image = resource_name;
            }
        }
```

```
        //图片输入流
        InputStream                    inputStream                    =
repositoryService.getResourceAsStream(deploymentId, resource_image);

        File exportFile = new File("d:/holiday.png");

        FileOutputStream        fileOutputStream        =        new
FileOutputStream(exportFile);
        byte[] buffer = new byte[1024];
        int len = -1;
        //输出图片
        while((len = inputStream.read(buffer))!=-1){
            fileOutputStream.write(buffer, 0, len);
        }
        inputStream.close();
        fileOutputStream.close();
    }
```

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得所有文件的名称

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

5) 最后的将输入流中的图片资源进行输出。

6.6 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。所以我们还是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。

```
    public void testHistoric01(){
        HistoryService historyService = pe.getHistoryService();
        HistoricActivityInstanceQuery query =
historyService.createHistoricActivityInstanceQuery();
        query.processInstanceId("1501");

        List<HistoricActivityInstance> list = query.list();
        for(HistoricActivityInstance ai :list){
            System.out.println(ai.getActivityId());
            System.out.println(ai.getActivityName());
```

```
            System.out.println(ai.getProcessDefinitionId());
            System.out.println(ai.getProcessInstanceId());
            System.out.println("==============================");
        }
    }
```