

## 一、什么是 workflow

以请假为例，大多数公司的请假流程是这样的

员工打电话（或网聊）向上级提出请假申请——上级口头同意——上级将请假记录下来——月底将请假记录上交公司——公司将请假录入电脑

采用 workflow 技术的公司的请假流程是这样的

员工使用账户登录系统——点击请假——上级登录系统点击允许

就这样，一个请假流程就结束了

有人会问，那上级不用向公司提交请假记录？公司不用将记录录入电脑？答案是，用的。但是这一切的工作都会在上级的点击允许后自动运行！

这就是 workflow 技术。

接下来给出 workflow 的书面化概念：

workflow (Workflow)，就是“业务过程的部分或整体在计算机应用环境下的自动化”，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传递文档、信息或任务的过程自动进行，从而实现某个预期的业务目标，或者促使此目标的实现”。

下面也给出 workflow 管理系统的概念：

workflow 管理系统 (Workflow Management System, WfMS) 是一个软件系统，它完成工作量的定义和管理，并按照在系统中预先定义好的 workflow 逻辑进行 workflow 实例的执行。workflow 管理系统不是企业的业务系统，而是为企业的业务系统的运行提供了一个软件的支撑环境。

workflow 管理联盟 (WfMC, Workflow Management Coalition) 也给出了关于 workflow 管理系统的定义：

workflow 管理系统是一个软件系统，它通过执行经过计算的流程定义去支持一批专门设定的业务流程。workflow 管理系统被用来定义、管理和执行 workflow。

而 workflow 管理系统的目标为：

管理工作的流程以确保工作在正确的时间被期望的人员所执行——在自动化进行的业务过程中插入人工的执行和干预。

## 二、Activiti 介绍

Activiti5 是由 Alfresco 软件在 2010 年 5 月 17 日发布的业务流程管理 (BPM) 框架，它是覆盖了业务流程管理、workflow、服务协作等领域的一个开源的、灵活的、易扩展的可执行流程语

言框架。Activiti基于Apache许可的开源BPM平台，创始人Tom Baeyens是JBoss JBPM的项目架构师，它的特色是提供了eclipse插件，开发人员可以通过插件直接绘画出业务流程图。

### 三、运行Activiti官方例子

因为网上Activiti5的资料比较多，所以这里主要就说Activiti5

打开官网地址：<https://www.activiti.org/get-started> 下载Activiti5

#### Older Versions

##### Activiti 6.0

6.x Download  
6.x Developer Guide  
Getting Started  
Migration Guide 5.x to 6.x  
Javadocs

##### Activiti 5.0

5.x Download  
5.x Javadocs  
5.x Full Guide

下载后文件地址：C:\worksoft\ativiti6

解压下载的压缩包后得到：

名称	大小	压缩后大小	类型
..(上层目录)			
database	247.14 KB	68.85 KB	文件夹
docs	16.72 MB	8.61 MB	文件夹
libs	20.75 MB	18.74 MB	文件夹
wars	63.02 MB	62.84 MB	文件夹
license.txt	11.09 KB	3.85 KB	文本文档
notice.txt	8.82 KB	2.11 KB	文本文档
readme.html	227.65 KB	48.33 KB	Chrome HTML D...

数据库初始化，支持多种数据库  
开发文档/用户手册  
开发所需要的jar  
demo

将wars包中的war解压到一个tomcat的webapps中，如：C:\mywork\apache-tomcat-9.0.8-activiti5\webapps

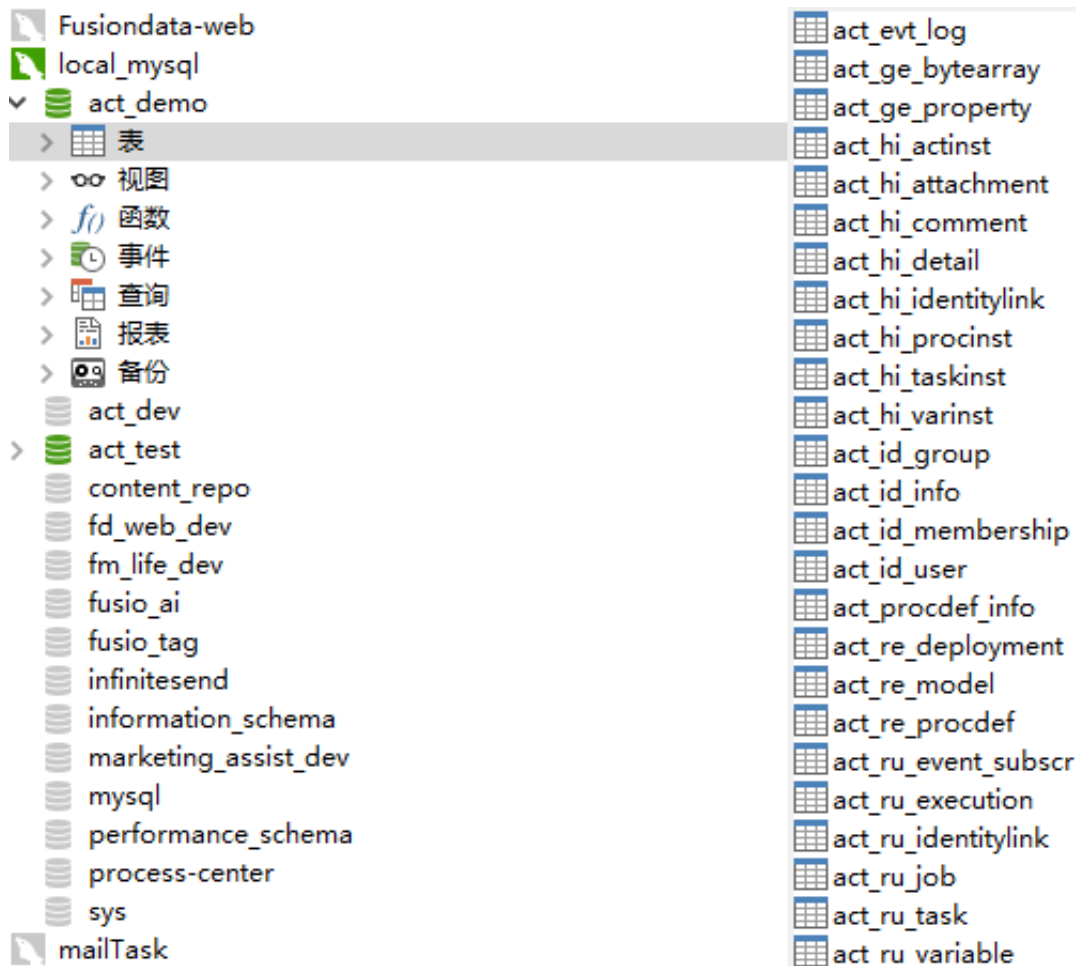
并修改\activiti-explorer\WEB-INF\classes中的db.properties并修改\activiti-explorer\WEB-

## INF\classes中的db.properties

```
1 db=mysql
2 jdbc.driver=com.mysql.jdbc.Driver
3 jdbc.url=jdbc:mysql://localhost:3306/act_demo?
  useUnicode=true&characterEncoding=UTF-
  8&allowMultiQueries=true&zeroDateTimeBehavior=convertToNull&useSSL=tru
  e
4 jdbc.username=root
5 jdbc.password=root
```

注意：需要负责mysql-connector-java jar包到lib中

启动tomcat，启动完后，连接的数据库会自动生成几张表



访问首页，打开浏览器输入<http://localhost:8080/activiti-explorer>，使用用户kermit，密码kermit登录

## 6)流程的使用

点击上方菜单“流程”，在左侧可以看到内置的示例流程，点击“Simple approval process”可以看到一个简单审批流程

在Simple approval process流程右上方，点击启动流程，跳转到一个表单，填写好表单，发起一个流程实例

然后就可以登录到下个处理人的帐号，从“任务”菜单中找到流程任务，如图，填写表单，并选择审核通过；然后任务会跳转到下任务。



点击菜单“管理”-》“管理”，再点左侧“已完成流程实例”，可以看到刚刚已经跑完成的流程  
同时也可在数据库，查看DB表数据，可以看到刚刚完成的两个任务数据

当然也自己创建属于自己的流程，点击菜单“流程”→“流程设计工作区”→“新建模型”

**注意：用户任务需要指定操作的人，不然找不到人完成任务，需要先部署**



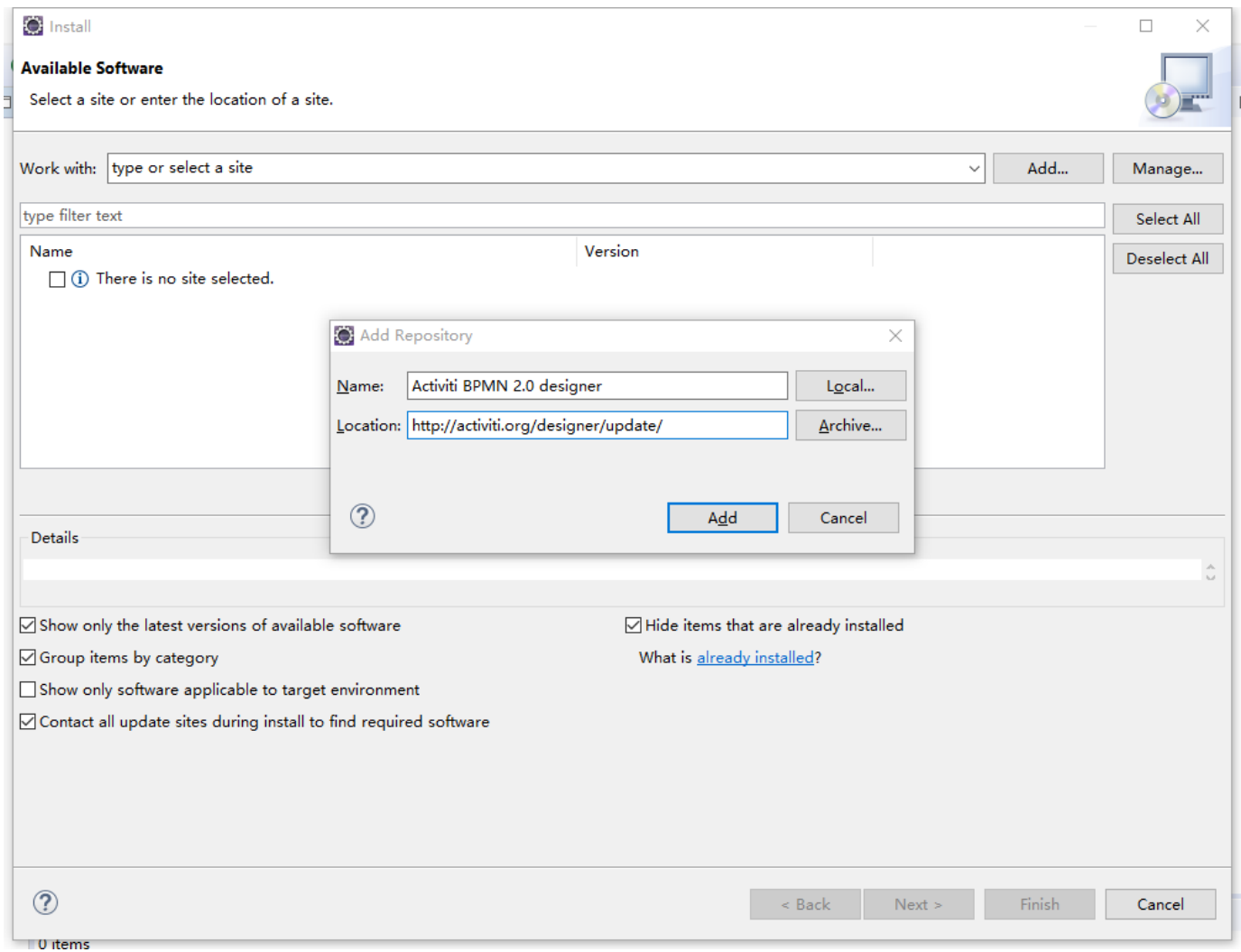
## 四、使用Eclipse开发一个简单的流程例子

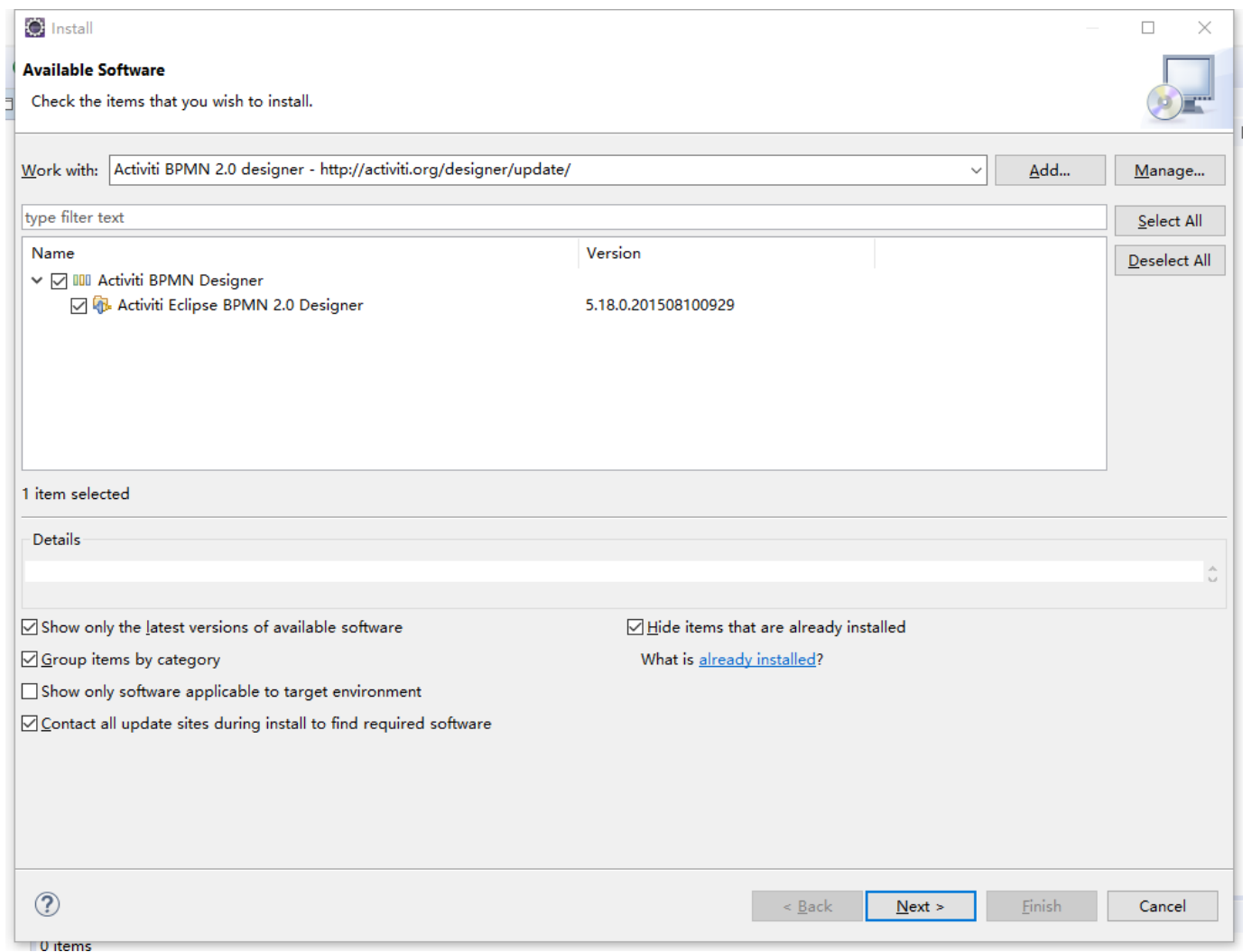
### 1、安装插件

打开eclipse，点击Help → Install New Software

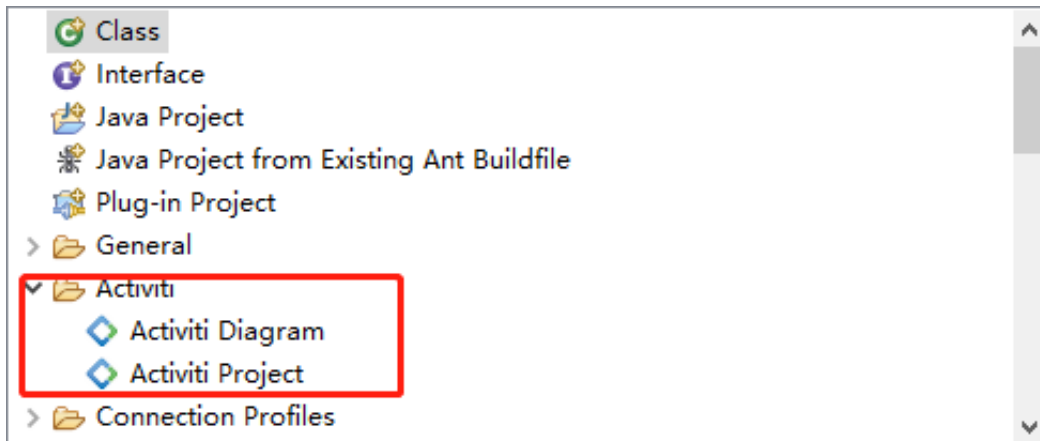
- **\*Name:\*Activiti BPMN 2.0 designer**

- \*Location:\*<http://activiti.org/designer/update/>





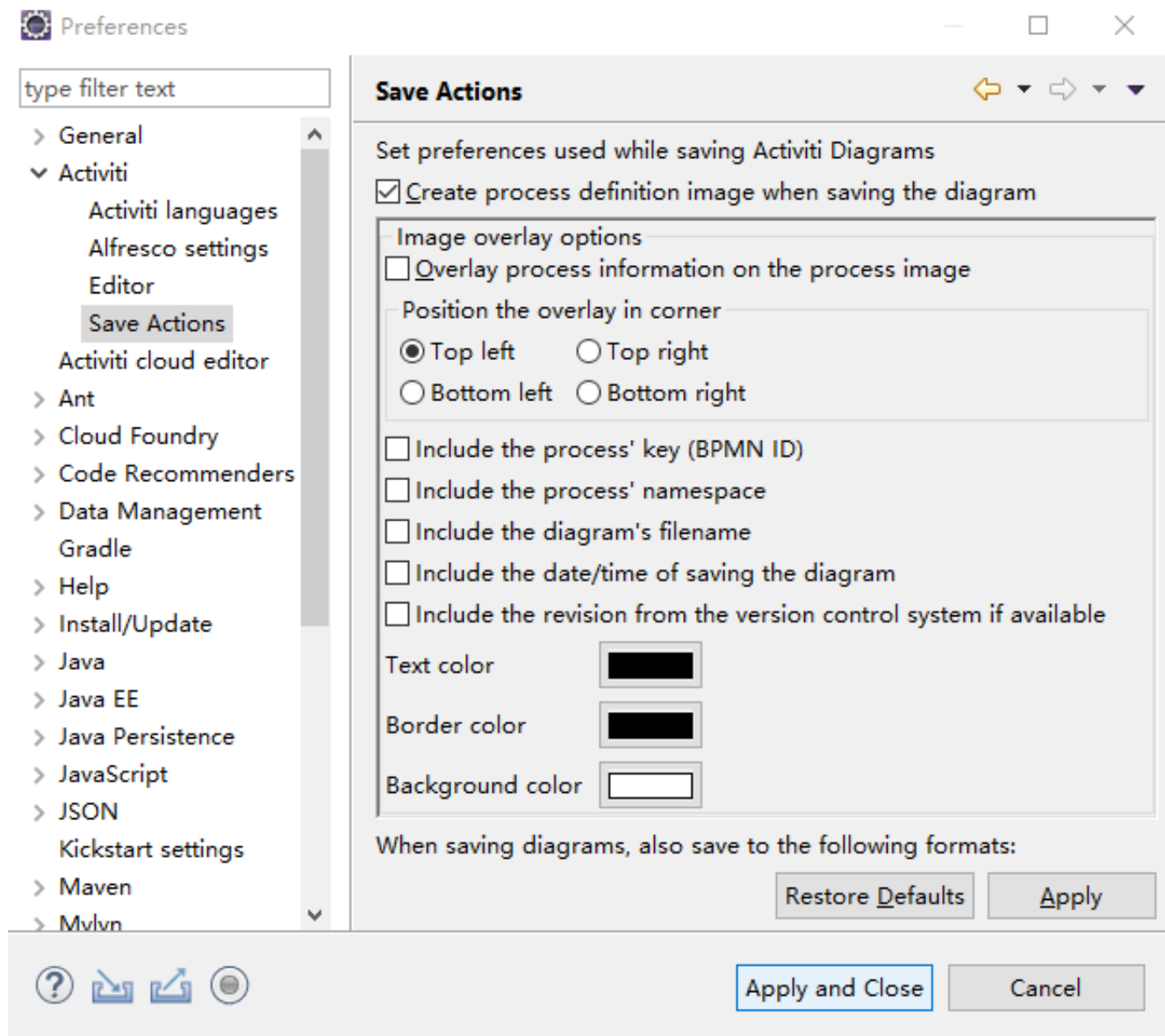
安装完后，重启eclipse



表示成功安装。

对流程设计器的使用说明

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



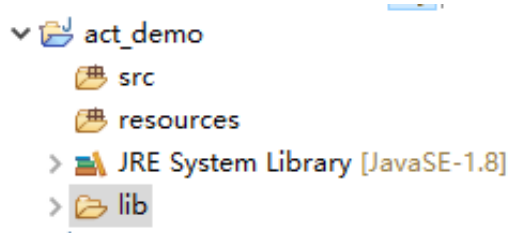
虽然流程引擎在单独部署bpmn文件时会自动生成图片，但在实际开发过程中，自动生成的图片会导致和BPMN中的坐标有出入，在实际项目中展示流程当前位置图会有问题。

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。

## 2、创建一个java程序

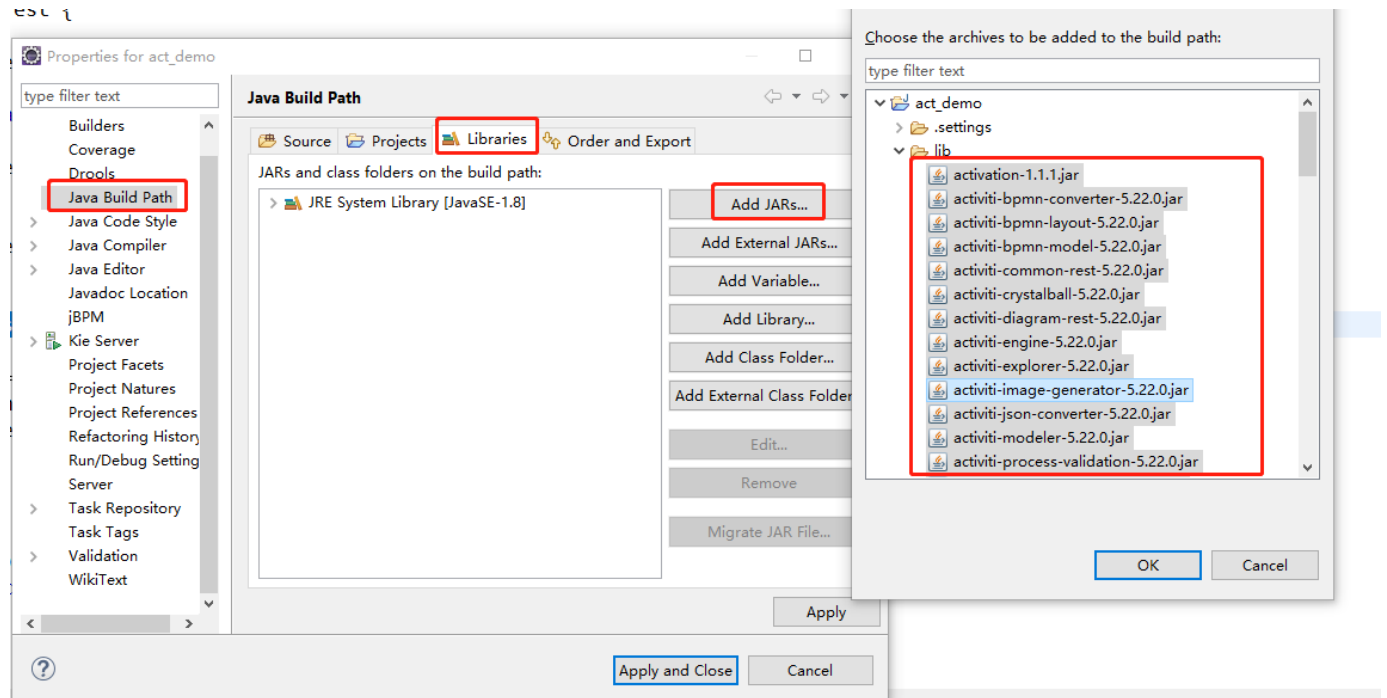
New-->Java Project-->act\_demo-->finish

建立resources和lib文件夹，如



将C:\mywork\apache-tomcat-9.0.8-activiti5\webapps\activiti-explorer\WEB-INF\lib中的jar包复制到lib中

然后将jar包导入到项目中



将activiti.cfg.xml文件复制到resources配置文件夹内

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xsi:schemaLocation="http://www.springframework.org/schema/beans
5 http://www.springframework.org/schema/beans/spring-beans.xsd">
6     <!-- 流程引擎配置的bean -->
7     <bean id="processEngineConfiguration"
8
9         class="org.activiti.engine.impl.cfg.StandaloneProcessEngineConfigurat
10 ion">
11         <property name="jdbcUrl"
12 value="jdbc:mysql://localhost:3306/act_demo?useSSL=true"/>
13         <property name="jdbcDriver" value="com.mysql.jdbc.Driver" />
14         <property name="jdbcUsername" value="root" />
15         <property name="jdbcPassword" value="root" />
16         <property name="databaseSchemaUpdate" value="true" />
17     </bean>
18 </beans>
```



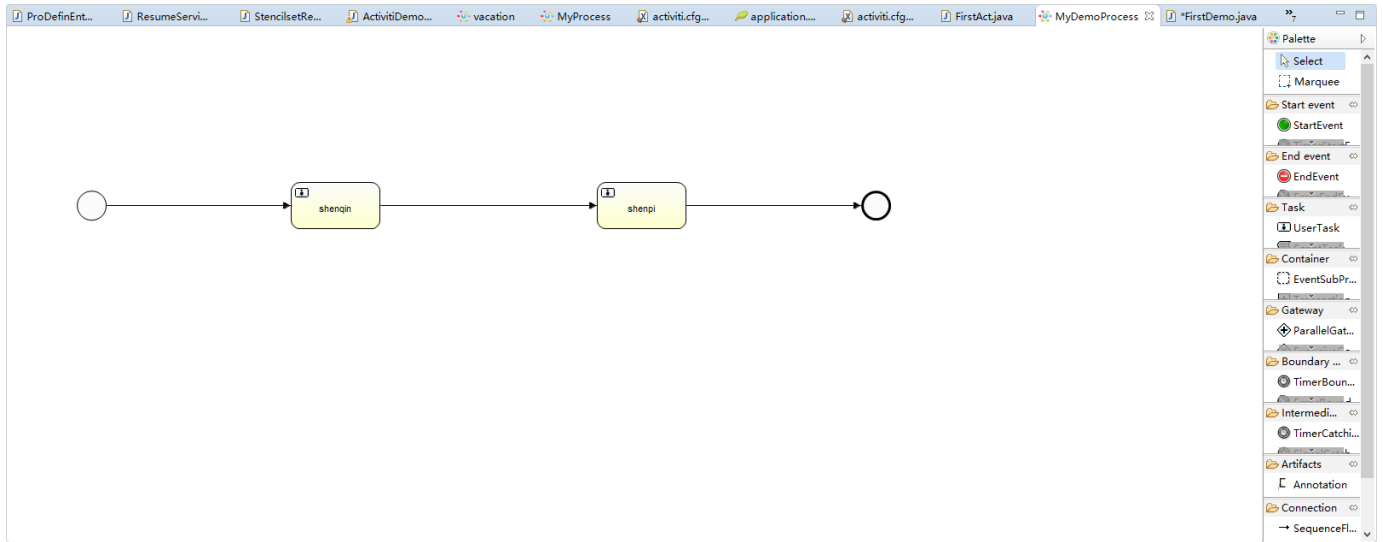
databaseSchemaUpdate: 用于设置流程引擎启动关闭时使用的数据库表结构控制策略。

false (默认): 当引擎启动时，检查数据库表结构的版本是否匹配库文件版本。版本不匹配时抛出异常。

true : 构建引擎时，检查并在需要时更新表结构。表结构不存在则会创建。

create-drop : 引擎创建时创建表结构，并在引擎关闭时删除表结构。

## 创建bpmn文件



## 创建java类:

```
1 public static void main(String[] args) throws Exception {
2     //创建流程引擎，调用ProcessEngines的getDefaultProceeEngine方法时
    会自动加载classpath下名为activiti.cfg.xml文件。
3     ProcessEngine engine =
    ProcessEngines.getDefaultProcessEngine();
4     //存储服务
5     RepositoryService rs = engine.getRepositoryService();
6     //运行时服务
7     RuntimeService runService = engine.getRuntimeService();
8     //任务服务
9     TaskService taskService = engine.getTaskService();
10    //添加部署
11
12    rs.createDeployment().addClasspathResource("MyDemoProcess.bpmn").depl
    oy();
13    //启动流程，可以启动多次，就是可以有多个个人请假
14    ProcessInstance pi =
```

```

runService.startProcessInstanceByKey("myDemoProcess");
14      //普通员工完成填写请假的任务
15      Task task =
taskService.createTaskQuery().processInstanceId(pi.getId()).singleResult();
16      System.out.println("当前流程节点: " + task.getName());
17      taskService.complete(task.getId());
18      //领导审核任务
19      task =
taskService.createTaskQuery().processInstanceId(pi.getId()).singleResult();
20      System.out.println("当前流程节点: " + task.getName());
21      taskService.complete(task.getId());
22      task =
taskService.createTaskQuery().processInstanceId(pi.getId()).singleResult();
23      System.out.println("流程结束了: " + task);
24      engine.close();
25      System.exit(0);
26
27  }

```

运行结果:

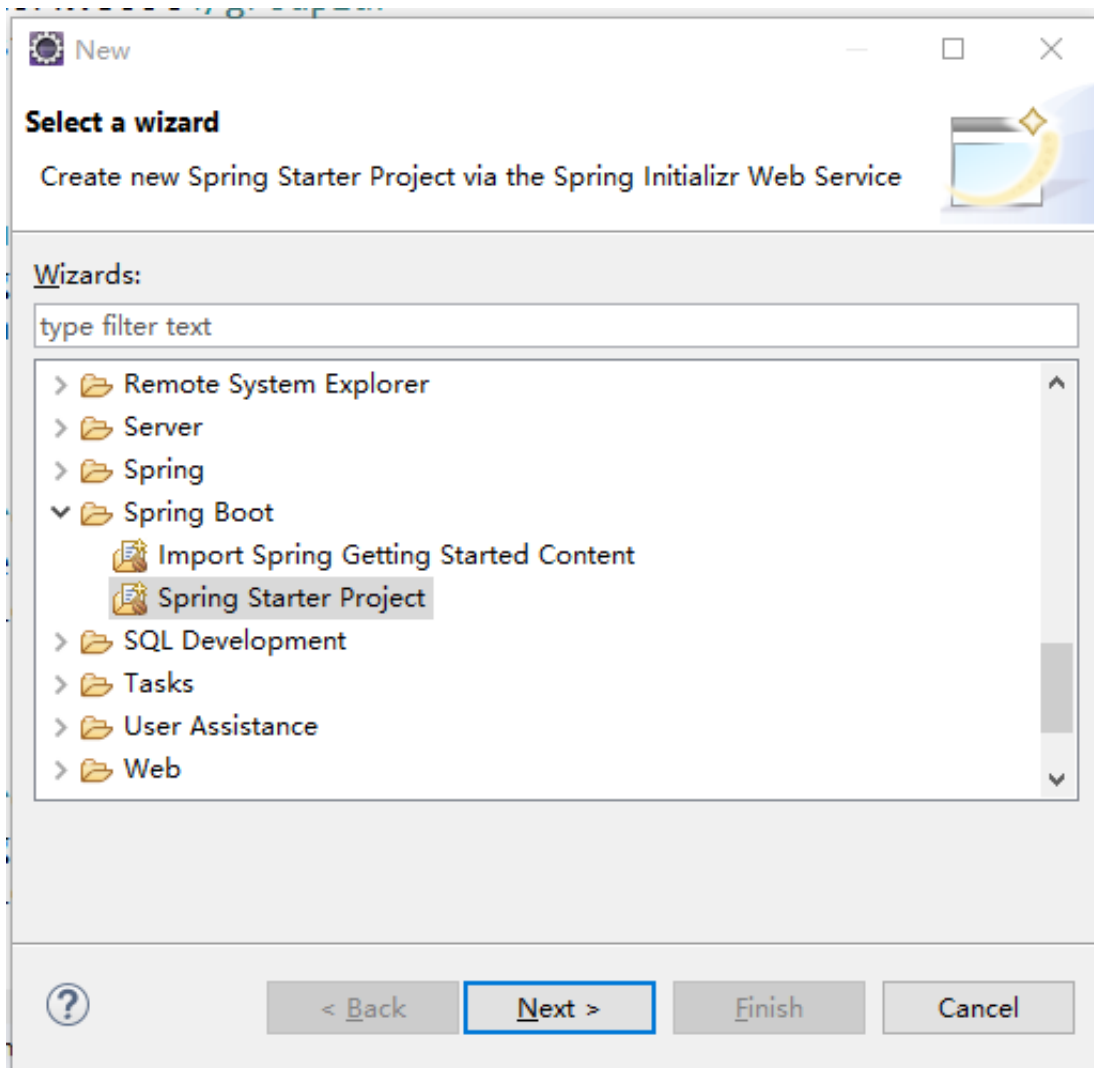
```

log4j:WARN No appenders could be found for logger (org.activiti.engine.Pr
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for
Fri Jul 13 15:33:38 GMT+08:00 2018 WARN: Establishing SSL connection with
当前流程节点: shenqin
当前流程节点: shenpi
流程结束了: null

```

## 五、Activiti整合到SpringBoot中

创建SpringBoot项目



1、添加以下依赖：

```
1 <dependencies>
2     <dependency>
3         <groupId>org.springframework.boot</groupId>
4         <artifactId>spring-boot-starter-web</artifactId>
5     </dependency>
6     <dependency>
7         <groupId>mysql</groupId>
8         <artifactId>mysql-connector-java</artifactId>
9         <scope>runtime</scope>
10    </dependency>
11    <dependency>
12        <groupId>org.springframework.boot</groupId>
13        <artifactId>spring-boot-starter-test</artifactId>
14        <scope>test</scope>
15    </dependency>
16    <dependency>
```

```

17         <groupId>org.activiti</groupId>
18         <artifactId>activiti-spring-boot-starter-basic</artifactId>
19         <version>5.22.0</version>
20     </dependency>
21 </dependencies>

```

## 2、数据源和activiti配置：

```

server:
  port: 8081

spring:
  datasource:
    url: jdbc:mysql://localhost:3306/act5?useSSL=true
    driver-class-name: com.mysql.jdbc.Driver
    username: root
    password: root

# activiti default configuration
activiti:
  database-schema-update: true
  check-process-definitions: true
  process-definition-location-prefix: classpath:/processes/
#   process-definition-location-suffixes:
#     - *.bpmn
#     - *.bpmn20.xml
  history-level: full

```

在activiti的默认配置中，process-definition-location-prefix 是指定activiti流程描述文件的前缀（即路径），默认为/processes/，**启动时，activiti就会去寻找此路径下的流程描述文件，并且自动部署**；suffix 是一个String数组，表示描述文件的默认后缀名，默认以上两种。

#保存历史数据级别设置为full最高级别，便于历史数据的追溯

spring.activiti.history-level=full

对于历史数据，保存到何种粒度，Activiti提供了history-level属性对其进行配置。history-level属性有点像log4j的日志输出级别，该属性有以下四个值：

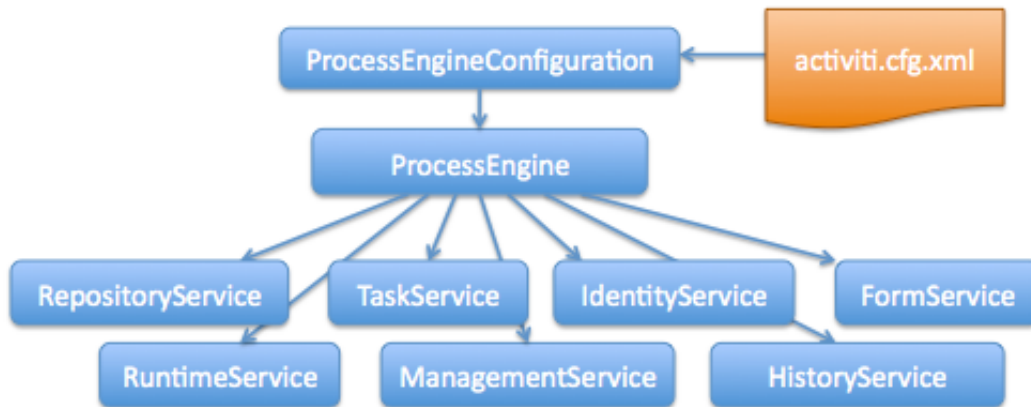
none：不保存任何的历史数据，因此，在流程执行过程中，这是最高效的。

activity：级别高于none，保存流程实例与流程行为，其他数据不保存。

audit: 除activity级别会保存的数据外，还会保存全部的流程任务及其属性。audit为history的默认值。

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

## 1、Process Engine API和服务



```
1 ProcessEngine processEngine =
  ProcessEngines.getDefaultProcessEngine();
2
3 RuntimeService runtimeService = processEngine.getRuntimeService();
4 RepositoryService repositoryService =
  processEngine.getRepositoryService();
5 TaskService taskService = processEngine.getTaskService();
6 ManagementService managementService =
  processEngine.getManagementService();
7 IdentityService identityService = processEngine.getIdentityService();
8 HistoryService historyService = processEngine.getHistoryService();
9 FormService formService = processEngine.getFormService();
10 DynamicBpmnService dynamicBpmnService =
  processEngine.getDynamicBpmnService();
```

**可以在方法中直接注入这几个service**

创建controller类

```

1  @RestController
2  public class HelloActivitiController {
3
4      @Autowired
5      private HelloActivitiService actService;
6
7      @RequestMapping("/startProcesses")
8      public String startProcesses(String processKey) {
9          return actService.startProcesses(processKey);
10     }
11
12     @RequestMapping("/qryCurrentTask")
13     public String qryCurrentTask(String instanceId) {
14         return actService.qryCurrentTask(instanceId);
15     }
16
17     @RequestMapping("/completeCurTask")
18     public String completeCurTask(String taskId) {
19         actService.completeCurTask(taskId);
20         return "OK";
21     }
22
23     @RequestMapping("/queryProImg")
24     public void queryProImg(String
processInstanceId, HttpServletResponse resp) throws Exception {
25         InputStream in = actService.queryProImg(processInstanceId);
26         try {
27             OutputStream out = resp.getOutputStream();
28             // 把图片的输入流写入response的输出流中
29             byte[] b = new byte[1024];
30             for (int len = -1; (len= in.read(b))!=-1; ) {
31                 out.write(b, 0, len);
32             }
33             // 关闭流
34             out.close();
35             in.close();
36         } catch (IOException e) {
37             e.printStackTrace();
38         }
39     }
40
41     @RequestMapping("/queryProHighLighted")

```

```

42     public void queryProHighLighted(String
processInstanceId,HttpServletResponse resp) throws Exception {
43         InputStream in =
actService.queryProHighLighted(processInstanceId);
44         try {
45             OutputStream out = resp.getOutputStream();
46             // 把图片的输入流写入response的输出流中
47             byte[] b = new byte[1024];
48             for (int len = -1; (len= in.read(b))!=-1; ) {
49                 out.write(b, 0, len);
50             }
51             // 关闭流
52             out.close();
53             in.close();
54         } catch (IOException e) {
55             e.printStackTrace();
56         }
57     }
58 }

```

service类:

```

1  @Service
2  public class HelloActivitiService {
3      @Autowired
4      private RuntimeService runtimeService;
5      @Autowired
6      private TaskService taskService;
7      @Autowired
8      private HistoryService historyService;
9      @Autowired
10     private RepositoryService repositoryService;
11     @Autowired
12     private ProcessEngineConfigurationImpl processEngineConfiguration;
13     /**
14      * 启动流程
15      *
16      * @param
17      */

```

```

18     public String startProcesses(String processKey) {
19         ProcessInstance pi =
runtimeService.startProcessInstanceByKey(processKey); // 流程图id
20         System.out.println("流程启动成功, 流程id:" + pi.getId());
21         return pi.getId();
22     }
23
24     /**
25      * 启动流程
26      *
27      * @param bizId 业务id
28      */
29     public String qryCurrentTask(String instanceId) {
30         Task currentTask =
taskService.createTaskQuery().processInstanceId(instanceId).singleResult();
31         System.out.println("任务id:" + currentTask.getId());
32         return currentTask.getId();
33     }
34     /**
35      *
36      * <p>
37      * 描述:处理当前任务（通过/拒接）
38      * </p>
39      *
40      */
41     public void completeCurTask(String taskId) {
42         taskService.complete(taskId);
43     }
44     /**
45      *
46      * <p>
47      * 描述：生成流程图 首先启动流程，获取processInstanceId，替换即可生成
48      * </p>
49      *
50      */
51     public InputStream queryProImg(String processInstanceId) throws
Exception {
52         // 获取历史流程实例
53         HistoricProcessInstance processInstance =
historyService.createHistoricProcessInstanceQuery()
54             .processInstanceId(processInstanceId).singleResult();

```



```

55         // 根据流程定义获取输入流
56         InputStream is =
repositoryService.getProcessDiagram(processInstance.getProcessDefiniti
onId());
57         List<Task> tasks =
taskService.createTaskQuery().taskCandidateUser("userId").list();
58         for (Task t : tasks) {
59             System.out.println(t.getName());
60         }
61         return is;
62     }
63     /**
64      * 流程图高亮显示 首先启动流程，获取processInstanceId，替换即可生成
65      *
66      * @throws Exception
67      */
68     public InputStream queryProHighLighted(String processInstanceId)
throws Exception {
69         // 获取历史流程实例
70         HistoricProcessInstance processInstance =
historyService.createHistoricProcessInstanceQuery()
71             .processInstanceId(processInstanceId).singleResult();
72         // 获取流程图
73         BpmnModel bpmnModel =
repositoryService.getBpmnModel(processInstance.getProcessDefinitionId(
));
74         ProcessDiagramGenerator diagramGenerator =
processEngineConfiguration.getProcessDiagramGenerator();
75         ProcessDefinitionEntity definitionEntity =
(ProcessDefinitionEntity) repositoryService
76             .getProcessDefinition(processInstance.getProcessDefinitionId());
77         List<HistoricActivityInstance> highLightedActivitList =
historyService.createHistoricActivityInstanceQuery()
78             .processInstanceId(processInstanceId).list();
79         // 高亮环节id集合
80         List<String> highLightedActivitis = new ArrayList<String>();
81         // 高亮线路id集合
82         List<String> highLightedFlows = getHighLightedFlows(bpmnModel,
definitionEntity, highLightedActivitList);
83         for (HistoricActivityInstance tempActivity :
highLightedActivitList) {

```

```

84         String activityId = tempActivity.getActivityId();
85         highLightedActivitis.add(activityId);
86     }
87     // 配置字体
88     InputStream imageStream =
diagramGenerator.generateDiagram(bpmnModel, "png",
highLightedActivitis,
89         highLightedFlows, "宋体", "微软雅黑", "黑体", null,
2.0);
90     System.out.println("图片生成成功");
91     return imageStream;
92 }
93 /**
94  * 获取需要高亮的线
95  *
96  * @param processDefinitionEntity
97  * @param historicActivityInstances
98  * @return
99  */
100 public List<String> getHighLightedFlows(BpmnModel bpmnModel,
ProcessDefinitionEntity processDefinitionEntity,
101     List<HistoricActivityInstance> historicActivityInstances)
{
102     SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss"); // 24小时制
103     List<String> highFlows = new ArrayList<String>(); // 用以保存高
亮的线flowId
104     for (int i = 0; i < historicActivityInstances.size() - 1; i++)
{
105         // 对历史流程节点进行遍历
106         // 得到节点定义的详细信息
107         FlowNode activityImpl = (FlowNode)
bpmnModel.getMainProcess()
108
.getFlowElement(historicActivityInstances.get(i).getActivityId());
109         List<FlowNode> sameStartTimeNodes = new
ArrayList<FlowNode>(); // 用以保存后续开始时间相同的节点
110         FlowNode sameActivityImpl1 = null;
111         HistoricActivityInstance activityImpl_ =
historicActivityInstances.get(i); // 第一个节点
112         HistoricActivityInstance activityImpl2_;
113         for (int k = i + 1; k <= historicActivityInstances.size())

```

```

- 1; k++) {
114         activityImp2_ = historicActivityInstances.get(k); // 后续第1个节点
115         if (activityImpl_.getActivityType().equals("userTask")
116             &&
activityImp2_.getActivityType().equals("userTask")
117             &&
df.format(activityImpl_.getStartTime()).equals(df.format(activityImp2_
.getStartTime()))) // 都是usertask, 且主节点与后续节点的开始时间相同, 说明
不是真实的后继节点
118         {
119         } else {
120             sameActivityImpl1 = (FlowNode)
bpmnModel.getMainProcess()
121
.getFlowElement(historicActivityInstances.get(k).getActivityId()); //
找到紧跟在后面的一个节点
122             break;
123         }
124     }
125     sameStartTimeNodes.add(sameActivityImpl1); // 将后面第一个
节点放在时间相同节点的集合里
126     for (int j = i + 1; j < historicActivityInstances.size() -
1; j++) {
127         HistoricActivityInstance activityImpl1 =
historicActivityInstances.get(j); // 后续第一个节点
128         HistoricActivityInstance activityImpl2 =
historicActivityInstances.get(j + 1); // 后续第二个节点
129         if
(df.format(activityImpl1.getStartTime()).equals(df.format(activityImpl
2.getStartTime()))) { // 如果第一个节点和第二个节点开始时间相同保存
130             FlowNode sameActivityImpl2 = (FlowNode)
bpmnModel.getMainProcess()
131
.getFlowElement(activityImpl2.getActivityId());
132             sameStartTimeNodes.add(sameActivityImpl2);
133         } else { // 有不相同跳出循环
134             break;
135         }
136     }
137     List<SequenceFlow> pvmTransitions =
activityImpl.getOutgoingFlows(); // 取出节点的所有出去的线

```

```

138         for (SequenceFlow pvmTransition : pvmTransitions) { // 对所
有的线进行遍历
139             FlowNode pvmActivityImpl = (FlowNode)
bpmnModel.getMainProcess()
140
.getFlowElement(pvmTransition.getTargetRef()); // 如果取出的线的目标节点
存在时间相同的节点里，保存该线的id，进行高亮显示
141             if (sameStartTimeNodes.contains(pvmActivityImpl)) {
142                 highFlows.add(pvmTransition.getId());
143             }
144         }
145     }
146     return highFlows;
147 }
148 }

```

启动流程：<http://localhost:8081/startProcesses?processKey=myDemoProcess>

对象 act_hi_procinst @act_demo ...					
开始事务 备注 筛选 排序 导入 导出					
ID_	PROC_INST_ID_	BUSINESS_KEY_	PROC_DEF_ID_	START_TIME_	ENC
▶ 5	5	(Null)	myDemoProcess:1:4	2018-07-13 16:56:58.261	(Nul

查看当前任务：<http://localhost:8081/qryCurrentTask?instanceId=5>

act\_ru\_task

完成当前任务：<http://localhost:8081/completeCurTask?taskId=8>

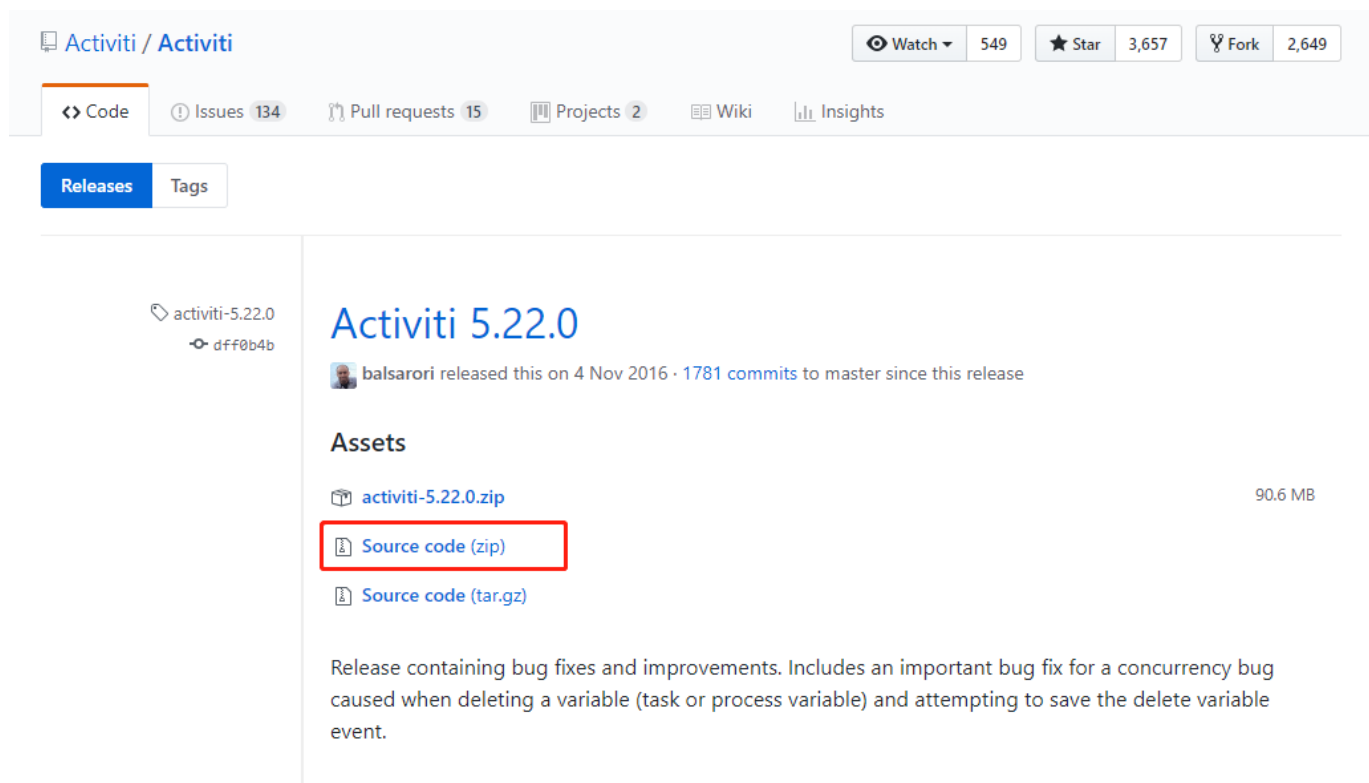
查看流程图：<http://localhost:8081/queryProImg>

查看流程图，已完成的标红色：<http://localhost:8081/queryProHighLighted>

## 六、SpringBoot集成Activiti Modeler

项目采用Springboot 1.5.8.RELEASE版本以及activiti 5.22.0版本

在acitiviti官网下载完整包<https://github.com/Activiti/Activiti/releases/tag/activiti-5.22.0>



Activiti / Activiti

Watch 549 Star 3,657 Fork 2,649

Code Issues 134 Pull requests 15 Projects 2 Wiki Insights

Releases Tags

activiti-5.22.0  
dffb0b4b

## Activiti 5.22.0

balsarori released this on 4 Nov 2016 · 1781 commits to master since this release

### Assets

activiti-5.22.0.zip 90.6 MB

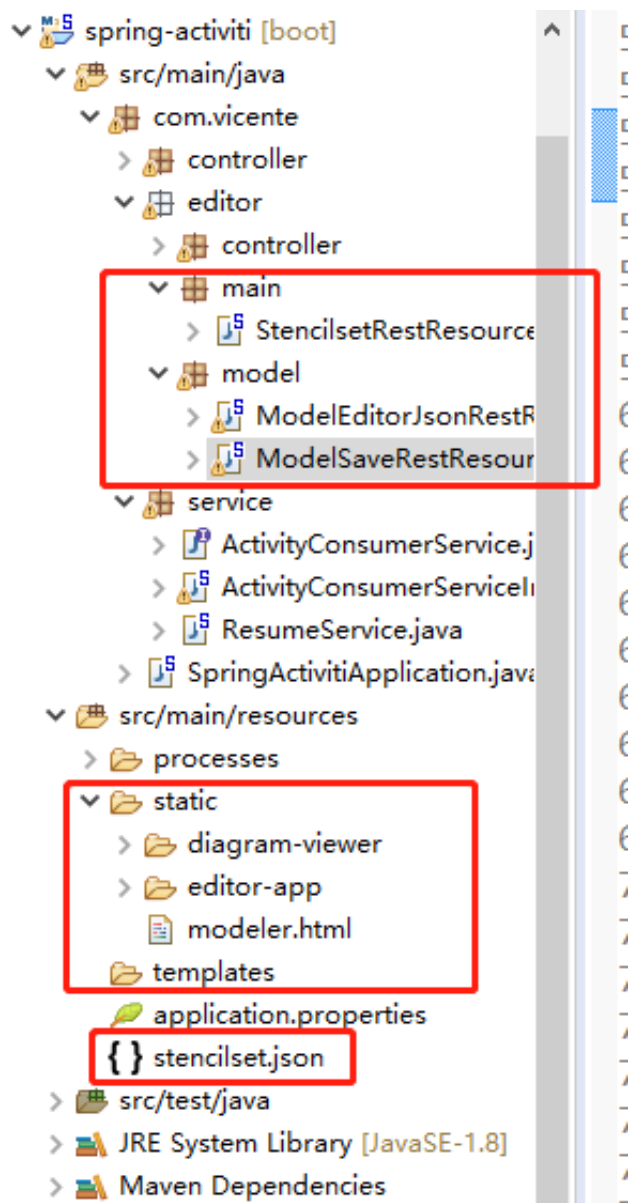
**Source code (zip)**

Source code (tar.gz)

Release containing bug fixes and improvements. Includes an important bug fix for a concurrency bug caused when deleting a variable (task or process variable) and attempting to save the delete variable event.

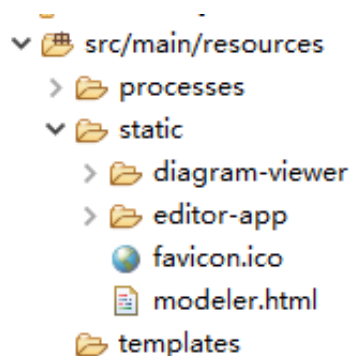
下载完成后,解压文件

- 1、将 Activiti-activiti-5.22.0\modules\activiti-webapp-explorer2\src\main\webapp 下的 diagram-viewer、editor-app以及modeler.html文件放置在项目resources\static文件夹下。
- 2、将 Activiti-activiti-5.22.0\modules\activiti-webapp-explorer2\src\main\resources 下的 stencilset.json放置在项目resources文件夹下。
- 3、将 Activiti-activiti-5.22.0\modules\activiti-modeler\src\main\java\org\activiti\rest\editor 下的main以及model中的java文件放置到项目mian\java目录下



### 3.编辑器前端部分

仅保留一些静态资源就行了，将这些文件放入项目的web目录下。



其中的editor-app就是编辑器，modeler.html是编辑器的入口页面。  
diagram-viewer是流程跟踪插件，虽然这次用不着，但之后会用到。

还有一个界面组件文件，在resource下，名称叫stencilset.json。本身是英文的，可以通过替换它来达到汉化的效果。

在editor-app/app-cfg.js中配置一下项目url。这个url是编辑器相关的后台服务的url。

```
ACTIVITI.CONFIG = {  
    'contextRoot' : '/service',  
};
```

我去掉了项目名。

## 4.后端部分

添加依赖，这三个都得有

```
1 <activiti.version>5.22.0</activiti.version>  
2  
3 <!-- activiti -->  
4 <dependency>  
5     <groupId>org.activiti</groupId>  
6     <artifactId>activiti-spring-boot-starter-basic</artifactId>  
7     <version>${activiti.version}</version>  
8 </dependency>  
9  
10 <!-- 引入两个activiti的模块，因为编辑器会用到这两个模块。 -->  
11 <dependency>  
12     <groupId>org.activiti</groupId>  
13     <artifactId>activiti-modeler</artifactId>  
14     <version>${activiti.version}</version>  
15 </dependency>  
16  
17 <dependency>  
18     <groupId>org.activiti</groupId>  
19     <artifactId>activiti-diagram-rest</artifactId>  
20     <version>${activiti.version}</version>  
21 </dependency>
```

其中需要将modeler模块的源代码放到src中，因为需要在其中做部分修改，主要是url的映射。

其中有3个类，都是Controller：

StencilsetRestResource #获取编辑器组件及配置项信息。

ModelEditorJsonRestResource #根据modelId获取model的节点信息，编辑器根据返回的json进行绘图。

ModelSaveRestResource #编辑器制图之后，将节点信息以json的形式提交给这个Controller，然后由其进行持久化操作。

需要修改的地方就三个，在每个Controller类上加上@RequestMapping注解，并指定值为"service"（对应前台app-cfg.js中配置的url）。

```
@RestController
@RequestMapping(value = "/service")
public class ModelSaveRestResource implements ModelDataJsonConstants {

    protected static final Logger LOGGER = LoggerFactory.getLogger(ModelSaveRestResource.class);

    @Autowired
    private RepositoryService repositoryService;

    @Autowired
    private ObjectMapper objectMapper;
```

然后修改resources\static\editor-app\app-cfg.js，如下图

```
) 'use strict';
)
| var ACTIVITI = ACTIVITI || {};
|
| ACTIVITI.CONFIG = {
|     'contextRoot' : '/service',
| };
;
```

修改ModelSaveRestResource.java（主要是因为集成后保存时候报400错误），

主要就是修改参数，如下：



```

@RequestMapping(value="/model/{modelId}/save", method = RequestMethod.PUT)
@ResponseStatus(value = HttpStatus.OK)
public void saveModel(@PathVariable String modelId, @RequestParam("name") String name, @RequestParam("json_xml") String json_xml,
    @RequestParam("svg_xml") String svg_xml, @RequestParam("description") String description) {
    try {
        Model model = repositoryService.getModel(modelId);

        ObjectNode modelJson = (ObjectNode) objectMapper.readTree(model.getMetaInfo());

        modelJson.put(MODEL_NAME, name);
        modelJson.put(MODEL_DESCRIPTION, description);
        model.setMetaInfo(modelJson.toString());
        model.setName(name);

        repositoryService.saveModel(model);

        repositoryService.addModelEditorSource(model.getId(), json_xml.getBytes("utf-8"));

        InputStream svgStream = new ByteArrayInputStream(svg_xml.getBytes("utf-8"));
        TranscoderInput input = new TranscoderInput(svgStream);
    }
}

```

#### 4.启动项目,访问modeler页面

http://localhost:8080/modeler.html

需要进行身份验证

http://localhost:8080

用户名

密码

登录 取消

#### 修改启动类，屏蔽登录功能

```

1 @SpringBootApplication
2 @EnableAutoConfiguration(exclude = {
3
4     org.springframework.boot.autoconfigure.security.SecurityAutoConfigurat
5     ion.class,
6     org.activiti.spring.boot.SecurityAutoConfiguration.class,
7 })
8 public class DemoApplication {

```

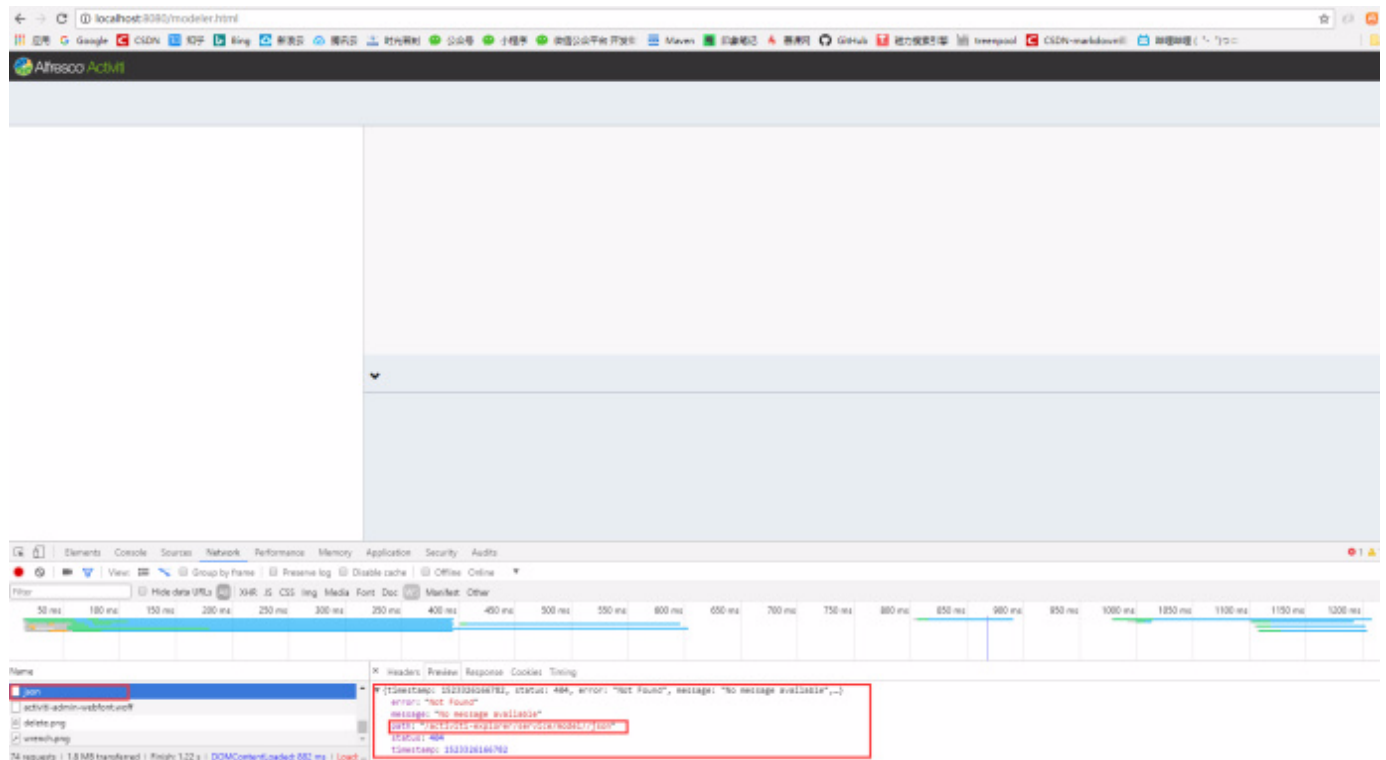
```

7     public static void main(String[] args) {
8         SpringApplication.run(DemoApplication.class, args);
9     }
10 }

```

或者可以在applicatio.properties文件中添加上: security.basic.enabled=false

再次访问modeler页面



处理 /activiti-explorer/service/model/json 请求的报错

该请求的控制类为ModelEditorJsonRestResource.java

`@RequestMapping(value="/model/{modelId}/json", method = RequestMethod.GET, produces = "application/json")`

在/public/editor-app/app-cfg.js文件中修改请求的地址

```

1  ACTIVITI.CONFIG = {
2      //'contextRoot' : '/activiti-explorer/service',

```

```
3     'contextRoot' : '/service',
4 };
```

再次访问modeler页面，发现页面未显示内容，这是因为目前还未创建任何model

报错GET http://localhost:8080/model//json 404 ()

这是因为我们没有已经建好的model模型，无法查看

## 5.新建model

```
1 @RequestMapping("model")
2 public class ModelController {
3
4     @Autowired
5     private ProcessEngine processEngine;
6     @Autowired
7     private ObjectMapper objectMapper;
8     @Autowired
9     private RepositoryService repositoryService;
10
11 @RequestMapping("create")
12     public void createModel(HttpServletRequest request,
13                             HttpServletResponse response){
14         try{
15             String name = "modelName";
16             String key = "modelKey";
17             String description = "description";
18             ObjectMapper objectMapper = new ObjectMapper();
19             ObjectNode editorNode = objectMapper.createObjectNode();
20             editorNode.put("id", "canvas");
21             editorNode.put("resourceId", "canvas");
22             ObjectNode stencilSetNode =
23             objectMapper.createObjectNode();
24             stencilSetNode.put("namespace",
25 "http://b3mn.org/stencilset/bpmn2.0#");
26             editorNode.put("stencilset", stencilSetNode);
27             Model modelData = repositoryService.newModel();
28             ObjectNode modelObjectNode =
29             objectMapper.createObjectNode();
30             modelObjectNode.put(ModelDataJsonConstants.MODEL_NAME,
```

```

name);
27         modelObjectNode.put(ModelDataJsonConstants.MODEL_REVISION,
1);
28
        modelObjectNode.put(ModelDataJsonConstants.MODEL_DESCRIPTION,
description);
29         modelData.setMetaInfo(modelObjectNode.toString());
30         modelData.setName(name);
31         modelData.setKey(key);
32         //保存模型
33         repositoryService.saveModel(modelData);
34         repositoryService.addModelEditorSource(modelData.getId(),
editorNode.toString().getBytes("utf-8"));
35         response.sendRedirect(request.getContextPath() +
"/modeler.html?modelId=" + modelData.getId());
36     }catch (Exception e){
37     }
38 }
39
40 /**
41  * 获取所有模型
42  *
43  * @return
44  */
45 @GetMapping("/list")
46 public List<Model> modelList() {
47     return
repositoryService.createModelQuery().orderByCreateTime().desc().list()
;
48 }
49 /**
50  * 发布模型为流程定义
51  */
52 @RequestMapping("/deploy")
53 @ResponseBody
54 public Object deploy(String modelId) throws Exception {
55
56     //获取模型
57     RepositoryService repositoryService =
processEngine.getRepositoryService();
58     Model modelData = repositoryService.getModel(modelId);
59     byte[] bytes =

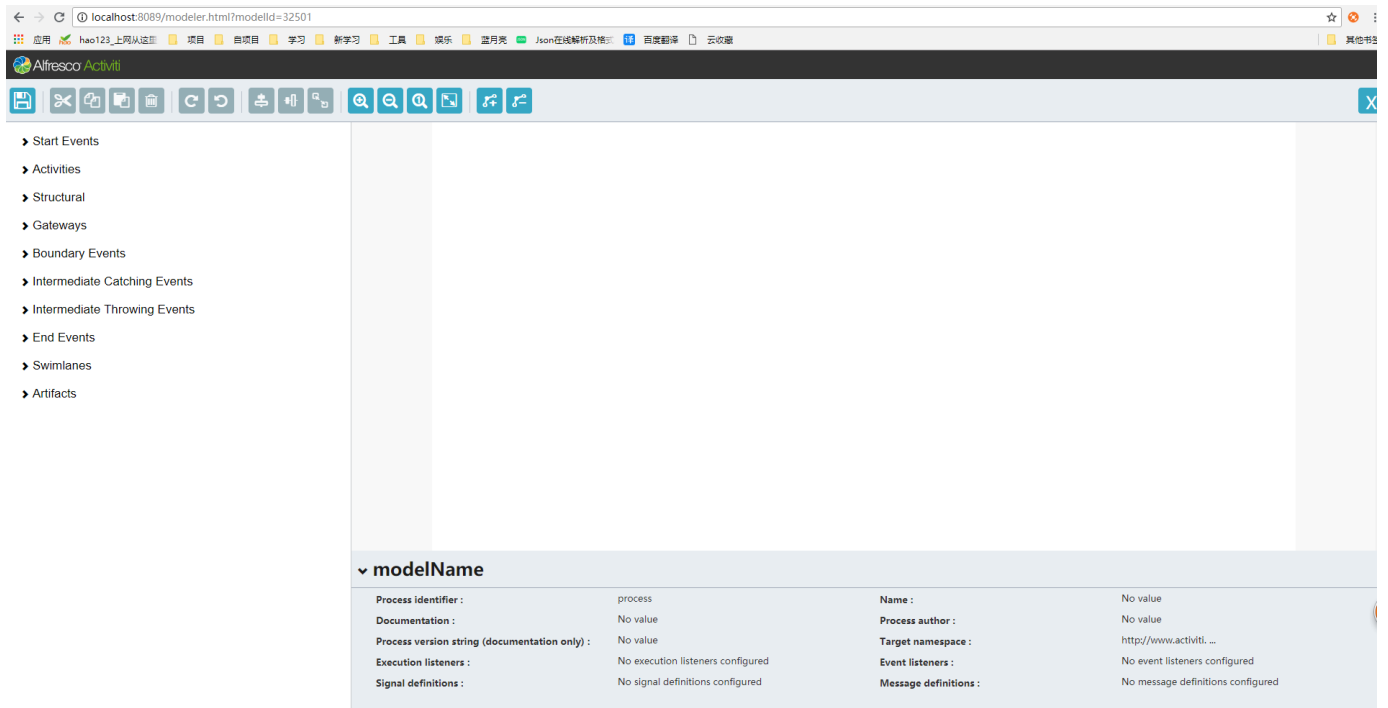
```

```

60 repositoryService.getModelEditorSource(modelData.getId());
61     if (bytes == null) {
62         return "模型数据为空，请先设计流程并成功保存，再进行发布。";
63     }
64
65     JsonNode modelNode = new ObjectMapper().readTree(bytes);
66
67     BpmnModel model = new
68 BpmnJsonConverter().convertToBpmnModel(modelNode);
69     if(model.getProcesses().size()==0){
70         return "数据模型不符合要求，请至少设计一条主线流程。";
71     }
72     byte[] bpmnBytes = new BpmnXMLConverter().convertToXML(model);
73
74     //发布流程
75     String processName = modelData.getName() + ".bpmn20.xml";
76     Deployment deployment = repositoryService.createDeployment()
77         .name(modelData.getName())
78         .addString(processName, new String(bpmnBytes, "UTF-
8""))
79         .deploy();
80     modelData.setDeploymentId(deployment.getId());
81     repositoryService.saveModel(modelData);
82
83     return "SUCCESS";
84 }
85
86 }

```

完成后打开页面：



创建model访问：<http://localhost:8089/model/create>

自动跳转到：<http://localhost:8089/modeler.html?modelId=20001>

获取所有的model：<http://localhost:8089/model/list>

发布模型为流程定义：<http://localhost:8089/model/20001/deployment>

查询所有的流程定义：<http://localhost:8089/activiti/listProcess>

根据Key创建流程实例：<http://localhost:8089/activiti/start?keyName=qjmo>

查看指定流程定义下的任务明细：<http://localhost:8089/activiti/listInst?keyName=qjmo>

根据流程实例ID认领并完成任务：<http://localhost:8089/activiti/run?processInstanceId=30016>

新建一个空模型--》画工作流程图---》发布模型为流程定义--》创建流程实例--》认领并完成任务1--》认领并完成任务2--》结束

act\_re\_model-->act\_re\_deployment-->act\_re\_procdef-->act\_ge\_bytearray--  
>act\_ru\_execution-->act\_ru\_task-->act\_hi\_procinst-->act\_hi\_taskinst-->act\_hi\_actinst