# 环境搭建

开发环境：

activiti 5.22.0

jdk 1.8

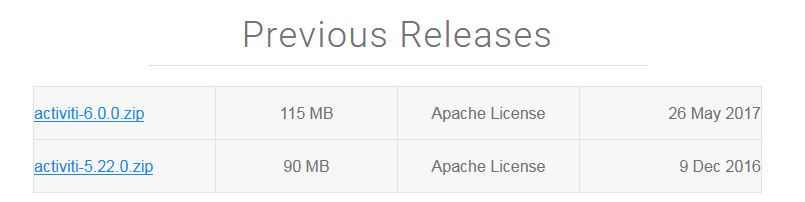
mysql 5.6

tomcat 8.5

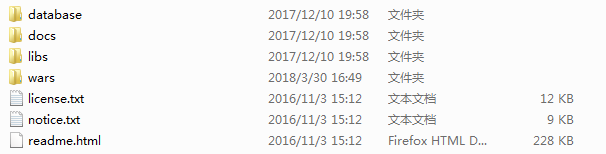
注意，activiti需要在数据库里创建表，可以在官网下载源码安装包，里面有数据库的表。或者通过java代码自动创建表。所以1.1和1.2节中的方法二选一即可。

## 下载源码包

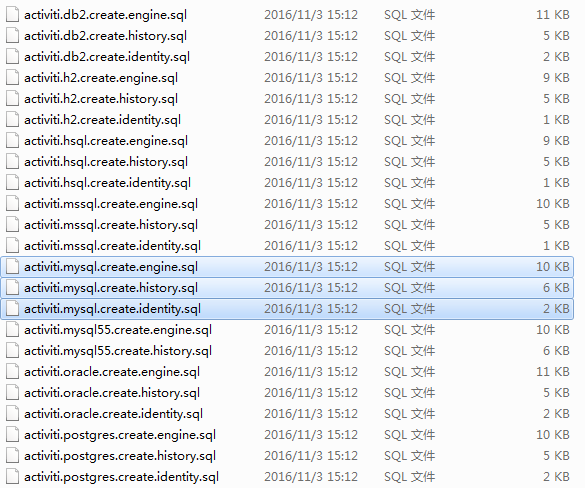
<https://www.activiti.org/download-links>



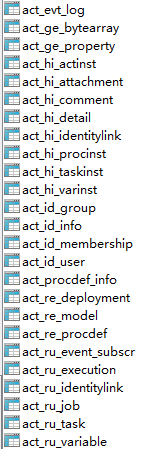
本教程使用的是5.22.0版本，解压后如下：



在database\create文件夹下有各种版本数据库使用的创建数据库的sql语句。5.22.0版本里有25张基础表。此处使用mysql5.6数据库，新建一个数据库，运行如下文件



运行后得到25张表：



数据库表说明：



## 通过java代码创建表

引入依赖的jar

<dependencies>  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.12</version>  
 </dependency>  
 n<dependency>  
 <groupId>org.activiti</groupId>  
 <artifactId>activiti-bpmn-converter</artifactId>  
 <version>5.22.0</version>  
 </dependency>  
 <dependency>  
 <groupId>org.activiti</groupId>  
 <artifactId>activiti-bpmn-model</artifactId>  
 <version>5.22.0</version>  
 </dependency>  
 <dependency>  
 <groupId>org.activiti</groupId>  
 <artifactId>activiti-image-generator</artifactId>  
 <version>5.22.0</version>  
 </dependency>  
 <dependency>  
 <groupId>org.activiti</groupId>  
 <artifactId>activiti-process-validation</artifactId>  
 <version>5.22.0</version>  
 </dependency>  
 <dependency>  
 <groupId>org.slf4j</groupId>  
 <artifactId>slf4j-log4j12</artifactId>  
 <version>1.6.1</version>  
 </dependency>  
 <dependency>  
 <groupId>org.slf4j</groupId>  
 <artifactId>slf4j-api</artifactId>  
 <version>1.6.1</version>  
 </dependency>  
 <dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <version>5.1.44</version>  
 </dependency>  
 <dependency>  
 <groupId>commons-dbcp</groupId>  
 <artifactId>commons-dbcp</artifactId>  
 <version>1.4</version>  
 </dependency>  
 <dependency>  
 <groupId>org.activiti</groupId>  
 <artifactId>activiti-root</artifactId>  
 <version>5.22.0</version>  
 </dependency>  
 <dependency>  
 <groupId>org.activiti</groupId>  
 <artifactId>activiti-spring</artifactId>  
 <version>5.22.0</version>  
 </dependency>  
</dependencies>

在resources目录下创建activiti.cfg.xml，配置数据源和默认引擎

<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/activiti2?characterEncoding=utf-8" />  
 <property name="username" value="root" />  
 <property name="password" value="" />  
</bean>  
  
<bean id="processEngineConfiguration" class="org.activiti.engine.impl.cfg.StandaloneProcessEngineConfiguration">  
 <property name="dataSource" ref="dataSource" />  
 <property name="databaseSchemaUpdate" value="true" />  
</bean>

其中databaseSchemaUpdate属性允许在进程引擎启动和关闭时设置策略来处理数据库模式。

1. false （默认）：当创建流程引擎时，检查数据库模式对库的版本，如果版本不匹配则抛出异常。
2. true：在构建流程引擎时，执行检查，如果需要，执行模式的更新。如果模式不存在，则创建它。
3. create-drop：在创建流程引擎时创建模式，并在流程引擎关闭时删除模式。

这里设置成true，运行程序的时候自动生成表。

activiti底层使用的是mybtais，可以通过log4j查看执行的语句：

# For all other servers: Comment out the Log4J listener in web.xml to activate Log4J.  
log4j.rootLogger=DEBUG, stdout  
log4j.appender.stdout=org.apache.log4j.ConsoleAppender  
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout  
log4j.appender.stdout.layout.ConversionPattern=%d %p [%c] - %m%n  
#begin  
#for normal test,delete when online  
log4j.logger.com.ibatis=DEBUG  
log4j.logger.com.ibatis.common.jdbc.SimpleDataSource=DEBUG  
log4j.logger.com.ibatis.common.jdbc.ScriptRunner=DEBUG  
log4j.logger.com.ibatis.sqlmap.engine.impl.SqlMapClientDelegate=DEBUG  
log4j.logger.java.sql.Connection=DEBUG  
log4j.logger.java.sql.Statement=DEBUG  
log4j.logger.java.sql.PreparedStatement=DEBUG  
log4j.logger.java.sql.ResultSet=DEBUG

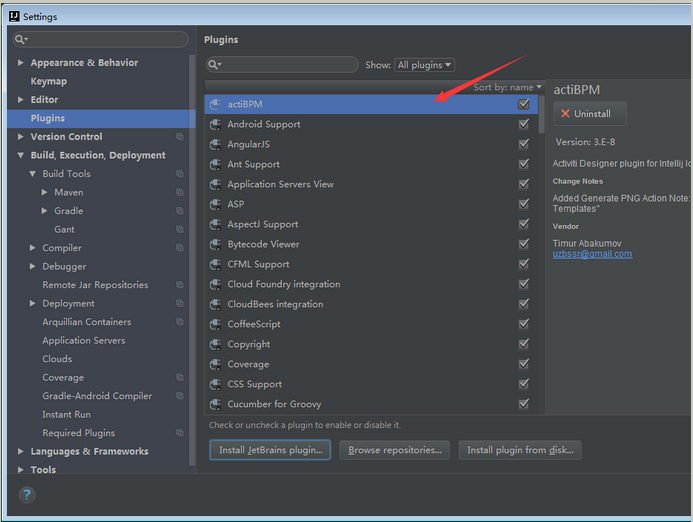
测试代码：

**@Test  
public void** createTable() **{** ProcessEngineConfiguration **configuration =** ProcessEngineConfiguration.createProcessEngineConfigurationFromResource(**"activiti.cfg.xml"**);  
 **//ProcessEngines.getDefaultProcessEngine();** ProcessEngine **processEngine = configuration**.**buildProcessEngine**();  
 **System**.***out***.println(**processEngine**);  
**}**

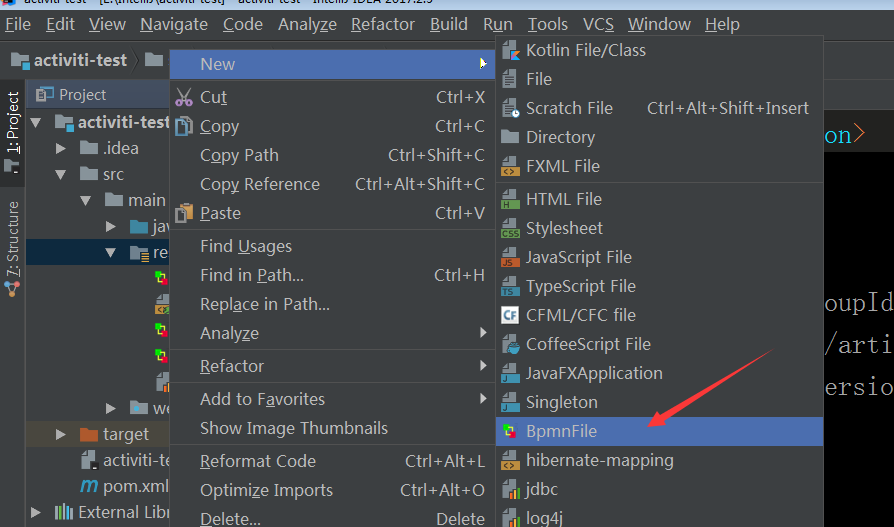
## idea安装activiti插件

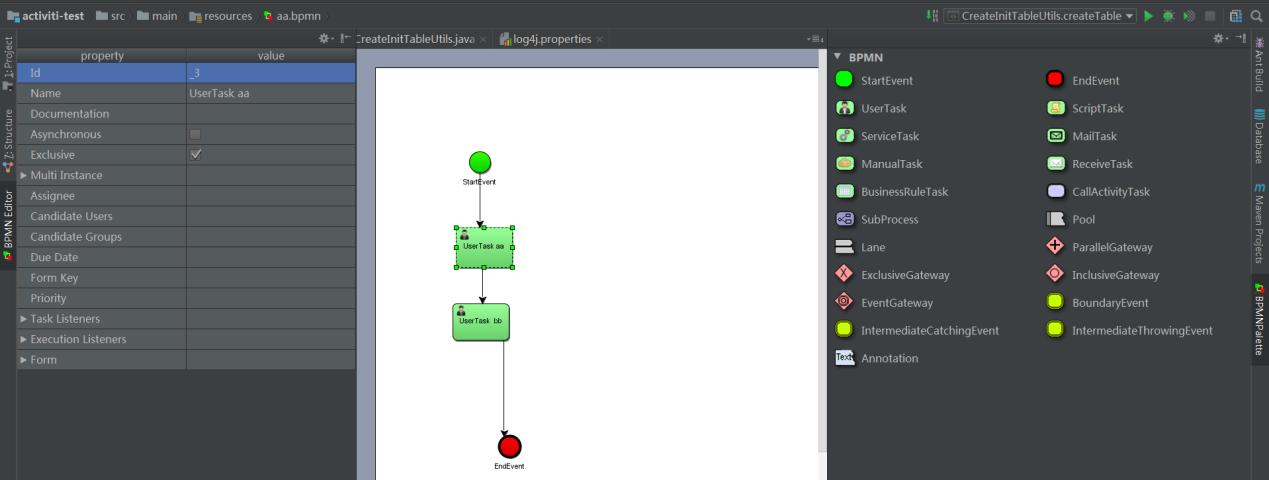
File->Settings->Plugins

搜索actiBPM，右侧选择Install，安装成功后重启Idea。

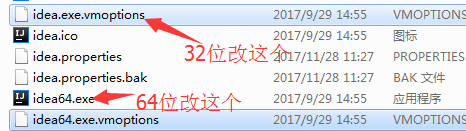


在工程中可以创建 bpmn文件了。bpmn规范会在后续课程中讲解。

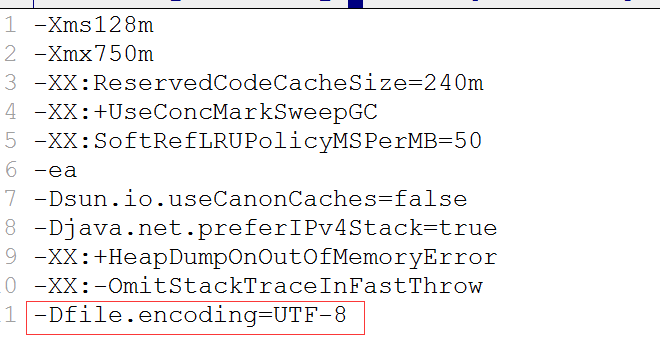




idea中绘制流程图的时候中文会乱码，在idea安装路径的bin目录下修改配置文件：

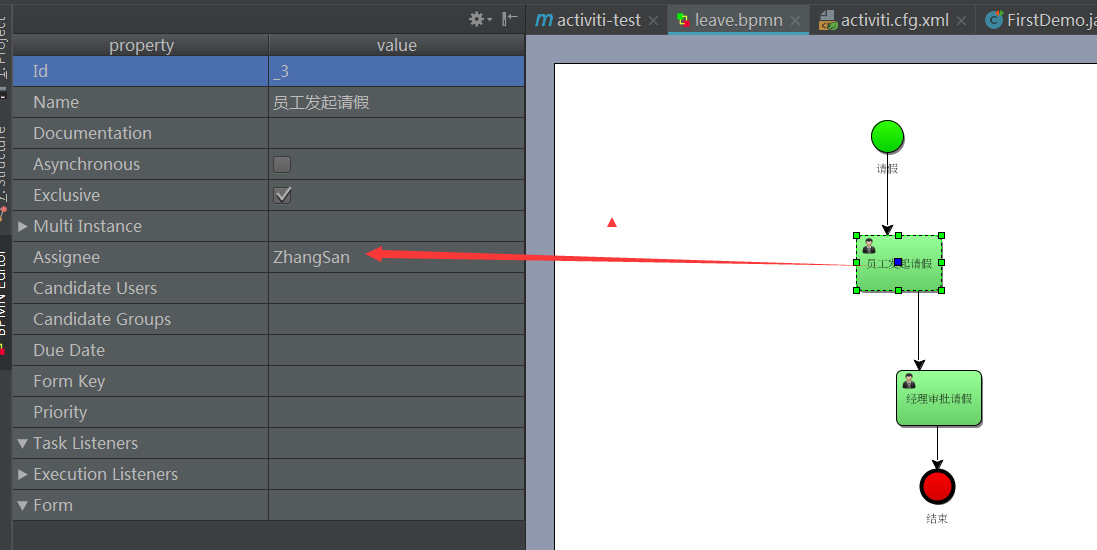


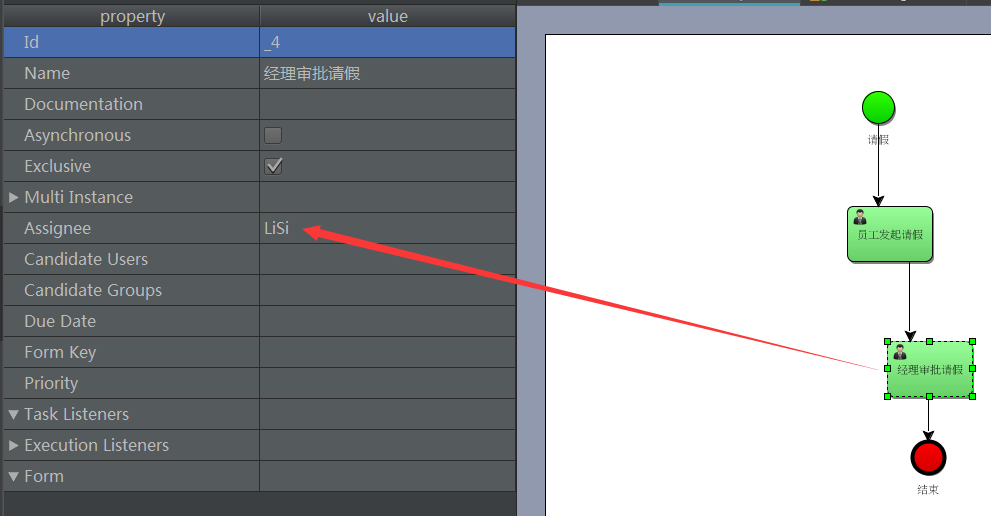
在文件末尾加一句，如图：-Dfile.encoding=UTF-8



## 示例工程

绘制流程图leave.bpmn：ZhangSan发起请假，LiSi审批请假

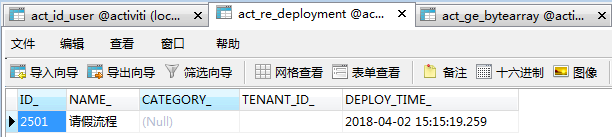


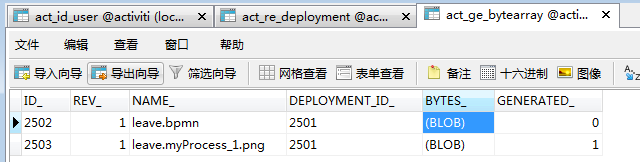


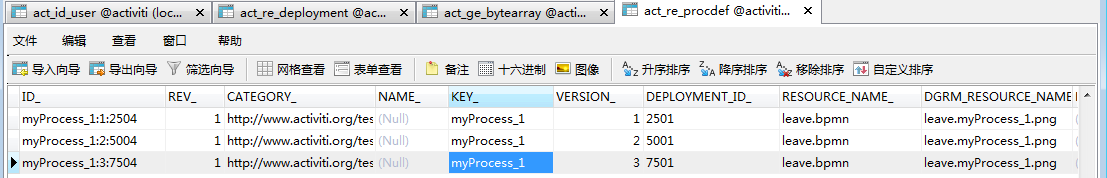
部署任务：

**@Test  
public void** addDeployment() **{  
 Deployment deployment = processEngine**.**getRepositoryService**() **// 与流程定义和部署对象相关的Service** .**createDeployment**() **// 创建一个部署对象** .**name**(**"请假流程"**) **// 设置对应流程的名称** .**addClasspathResource**(**"leave.bpmn"**)   
 .**deploy**(); **// 完成部署  
}**

数据库里已经插入了对应的数据







启动任务：

**@Test  
public void** startProcess() **{  
 RuntimeService runtimeService = processEngine**.**getRuntimeService**();  
 **//key是act\_re\_procdef中的KEY\_,bpmn的id  
 runtimeService**.**startProcessInstanceByKey**(**"myProcess\_1"**);  
**}**

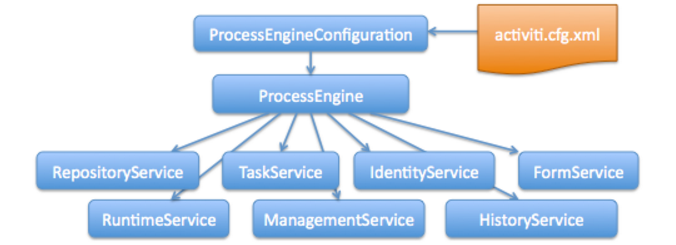
查看ZhangSan的任务并提交给下一个操作者。查看李四的任务并处理。(act\_ru\_task表)

**@Test  
public void** queryZhangSanTask() **{  
 String assignee = "ZhangSan"**;  
 **List<Task> taskList = processEngine**.**getTaskService**()**//获取任务service** .**createTaskQuery**()**//创建查询对象** .**taskAssignee**(**assignee**)**//指定查询人** .**list**();  
 **for** (**Task task : taskList**) **{  
 System**.***out***.println(**"代办任务ID:" + task**.**getId**());  
 **System**.***out***.println(**"代办任务name:" + task**.**getName**());  
 **System**.***out***.println(**"代办任务创建时间:" + task**.**getCreateTime**());  
 **System**.***out***.println(**"代办任务办理人:" + task**.**getAssignee**());  
 **System**.***out***.println(**"流程实例ID:" + task**.**getProcessInstanceId**());  
 **System**.***out***.println(**"执行对象ID:" + task**.**getExecutionId**());  
 **//提交任务到下一个代理人  
 engine**.**getTaskService**().**complete**(**task**.**getId**());  
 **}  
}  
@Test  
public void** queryLiSiTask() **{  
 String assignee = "LiSi"**;  
 **List<Task> taskList = processEngine**.**getTaskService**()**//获取任务service** .**createTaskQuery**()**//创建查询对象** .**taskAssignee**(**assignee**)**//指定查询人** .**list**();  
 **for** (**Task task : taskList**) **{  
 System**.***out***.println(**"代办任务ID:" + task**.**getId**());  
 **System**.***out***.println(**"代办任务name:" + task**.**getName**());  
 **System**.***out***.println(**"代办任务创建时间:" + task**.**getCreateTime**());  
 **System**.***out***.println(**"代办任务办理人:" + task**.**getAssignee**());  
 **System**.***out***.println(**"流程实例ID:" + task**.**getProcessInstanceId**());  
 **System**.***out***.println(**"执行对象ID:" + task**.**getExecutionId**());  
 **processEngine**.**getTaskService**().**complete**(**task**.**getId**());  
 **}  
}**

查看执行历史act\_hi\_taskinst表：

**@Test  
public void** viewHistory() **{  
 HistoryService historyService = processEngine**.**getHistoryService**();  
 **HistoricProcessInstance historicProcessInstance =  
 historyService** .**createHistoricProcessInstanceQuery**()  
 .**processInstanceId**(**"15001"**).**singleResult**();  
 **System**.***out***.println(**"开始时间：" + historicProcessInstance**.**getStartTime**());  
 **System**.***out***.println(**"结束时间：" + historicProcessInstance**.**getEndTime**());  
**}**

# Activiti核心配置



ProcessEngine流程引擎是Activiti的核心。Activiti默认读取activiti.cfg.xml文件，获得ProcessEngineConfiguration对象，通过ProcessEngineConfiguration创建ProcessEngine。

## ProcessEngineConfiguration配置

### 不使用配置文件

**@Test  
public void** createEngineWithoutXml() **{** ProcessEngineConfiguration **processEngineConfiguration =** ProcessEngineConfiguration.createStandaloneInMemProcessEngineConfiguration();  
 **processEngineConfiguration**.setJdbcDriver(**"com.mysql.jdbc.Driver"**);  
 **processEngineConfiguration**.setJdbcUrl(**"jdbc:mysql://localhost:3306/activiti?characterEncoding=utf-8"**);  
 **processEngineConfiguration**.setJdbcUsername(**"root"**);  
 **processEngineConfiguration**.setJdbcPassword(**""**);  
 **processEngineConfiguration**.setDatabaseSchemaUpdate(**"true"**);  
 **ProcessEngine processEngine = processEngineConfiguration**.**buildProcessEngine**();  
 **System**.***out***.println(**processEngine**);  
**}**

### 读取配置文件

配置文件activiti.cfg.xml，注意配置XML文件其实是一个spring的配置文件。 **但不是说Activiti只能用在Spring环境中！** 我们只是利用了Spring的解析和依赖注入功能 来构建引擎。

读取配置文件一共有五个方法：

1. ProcessEngineConfiguration.createProcessEngineConfigurationFromResourceDefault();
2. ProcessEngineConfiguration.createProcessEngineConfigurationFromResource(String resource);
3. ProcessEngineConfiguration.createProcessEngineConfigurationFromResource(String resource, String beanName);
4. ProcessEngineConfiguration.createProcessEngineConfigurationFromInputStream(InputStream inputStream);
5. ProcessEngineConfiguration.createProcessEngineConfigurationFromInputStream(InputStream inputStream, String beanName);

<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?characterEncoding=utf-8"/>  
 <property name="username" value="root"/>  
 <property name="password" value=""/>  
</bean>  
<!--带数据源-->  
<bean id="processEngineConfiguration" class="org.activiti.engine.impl.cfg.StandaloneProcessEngineConfiguration">  
 <property name="dataSource" ref="dataSource"/>  
 <property name="databaseSchemaUpdate" value="true"/>  
</bean>  
  
<!--不带数据源-->  
<!--<bean id="processEngineConfiguration" class="org.activiti.engine.impl.cfg.StandaloneProcessEngineConfiguration">-->  
 <!--<property name="jdbcUrl" value="com.mysql.jdbc.Driver" />-->  
 <!--<property name="jdbcDriver" value="jdbc:mysql://localhost:3306/activiti?characterEncoding=utf-8" />-->  
 <!--<property name="jdbcUsername" value="root" />-->  
 <!--<property name="jdbcPassword" value="" />-->  
 <!--<property name="databaseSchemaUpdate" value="true"/>-->  
<!--</bean>-->

**@Test  
 public void** createEngineWithXml() **{  
 //默认读取的是activiti.cfg.xml，并且bean的id必须是processEngineConfiguration  
// ProcessEngineConfiguration processEngineConfiguration =  
// ProcessEngineConfiguration.createProcessEngineConfigurationFromResourceDefault();** ProcessEngineConfiguration **processEngineConfiguration =** ProcessEngineConfiguration.createProcessEngineConfigurationFromResource(**"activiti.cfg.xml"**);  
 **ProcessEngine processEngine = processEngineConfiguration**.**buildProcessEngine**();  
 **System**.***out***.println(**processEngine**);  
 **}**

如果自定义bean的id，可以这样读取：

ProcessEngineConfiguration **processEngineConfiguration =** ProcessEngineConfiguration.createProcessEngineConfigurationFromResource(**"activiti.cfg.xml"**,**"configuration"**);

### ProcessEngineConfiguration子类

创建ProcessEngineConfiguration时，目前可以使用如下类，以后会更多：

1. **org.activiti.engine.impl.cfg.StandaloneProcessEngineConfiguration**: 单独运行的流程引擎。Activiti会自己处理事务。 默认，数据库只在引擎启动时检测 （如果没有Activiti的表或者表结构不正确就会抛出异常）。
2. **org.activiti.engine.impl.cfg.StandaloneInMemProcessEngineConfiguration**: 单元测试时的辅助类。Activiti会自己控制事务。 默认使用H2内存数据库。数据库表会在引擎启动时创建，关闭时删除。 使用它时，不需要其他配置（除非使用job执行器或邮件功能）。
3. **org.activiti.spring.SpringProcessEngineConfiguration**: 在Spring环境下使用流程引擎。
4. **org.activiti.engine.impl.cfg.JtaProcessEngineConfiguration**: 单独运行流程引擎，并使用JTA事务。

### 数据库配置

Activiti支持如下数据库：

|  |  |
| --- | --- |
| **数据库类型** | **url示例** |
| h2 | jdbc:h2:tcp://localhost/activiti |
| mysql | jdbc:mysql://localhost:3306/activiti?autoReconnect=true |
| oracle | jdbc:oracle:thin:@localhost:1521:xe |
| postgres | jdbc:postgresql://localhost:5432/activiti |
| db2 | jdbc:db2://localhost:50000/activiti |
| mssql | jdbc:sqlserver://localhost:1433;databaseName=activiti (jdbc.driver=com.microsoft.sqlserver.jdbc.SQLServerDriver) *OR* jdbc:jtds:sqlserver://localhost:1433/activiti (jdbc.driver=net.sourceforge.jtds.jdbc.Driver) |

Activiti可能使用两种方式配置数据库。 第一种方式是定义数据库配置参数，另一种是直接配置DataSource。

直接配置数据库参数，有如下参数：

* **jdbcUrl**: 数据库的JDBC URL。
* **jdbcDriver**: 对应不同数据库类型的驱动。
* **jdbcUsername**: 连接数据库的用户名。
* **jdbcPassword**: 连接数据库的密码。

基于JDBC参数配置的数据库连接 会使用默认的[MyBatis](http://www.mybatis.org/)连接池。 下面的参数可以用来配置连接池（来自MyBatis参数）：

* **jdbcMaxActiveConnections**: 连接池中处于被使用状态的连接的最大值。默认为10。
* **jdbcMaxIdleConnections**: 连接池中处于空闲状态的连接的最大值。
* **jdbcMaxCheckoutTime**: 连接被取出使用的最长时间，超过时间会被强制回收。 默认为20000（20秒）。
* **jdbcMaxWaitTime**: 这是一个底层配置，让连接池可以在长时间无法获得连接时， 打印一条日志，并重新尝试获取一个连接。（避免因为错误配置导致沉默的操作失败）。 默认为20000（20秒）。

无论你使用JDBC还是DataSource的方式，都可以设置下面的配置：

* **databaseType**: 一般不用设置，因为可以自动通过数据库连接的元数据获取。 只有自动检测失败时才需要设置。 可能的值有：{h2, mysql, oracle, postgres, mssql, db2}。这个配置会决定使用哪些创建/删除脚本和查询语句。
* **databaseSchemaUpdate**: 设置流程引擎启动和关闭时如何处理数据库表，可能值如下：
  1. 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。
  2. 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。
  3. 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。

使用数据源的配置参考各种DataSource的使用说明，ProcessEngineConfiguration的dataSource属性引用定义的数据源即可。如2.1.2中使用了DBCP的数据源。

## ProcessEngine创建

**@Test  
public void** createDefaultEngine() **{  
 //默认读取的是activiti.cfg.xml，并且bean的id必须是processEngineConfiguration  
 ProcessEngine processEngine=** ProcessEngines.getDefaultProcessEngine();  
 **System**.***out***.println(**processEngine**);  
**}**

ProcessEngines.getDefaultProcessEngine()方法读取的是activiti.cfg.xml，且bean的id必须是processEngineConfiguration。

使用ProcessEngineConfiguration创建：

ProcessEngineConfiguration **processEngineConfiguration =** ProcessEngineConfiguration.createProcessEngineConfigurationFromResource(**"activiti.cfg.xml"**);  
**ProcessEngine processEngine = processEngineConfiguration**.**buildProcessEngine**();  
**System**.***out***.println(**processEngine**);

## 数据库说明

Activiti的后台是有数据库的支持，所有的表都以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\_ID\_\*: 'ID'表示identity。 这些表包含身份信息，比如用户，组等等。

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\_\*: 通用数据， 用于不同场景下，如存放资源文件。

**资源库流程规则表**

1) act\_re\_deployment 部署信息表

2) act\_re\_model   流程设计模型部署表

3) act\_re\_procdef   流程定义数据表

4) act\_re\_event\_subscr 事件监听

**运行时数据库表**

1) act\_ru\_execution 运行时流程执行实例表

2) act\_ru\_identitylink 运行时流程人员表，主要存储任务节点与参与者的相关信息

3) act\_ru\_task 运行时任务节点表

4) act\_ru\_variable 运行时流程变量数据表

5) act\_ru\_job 异步作业

**历史数据库表**

1) act\_hi\_actinst 历史节点表

2) act\_hi\_attachment 历史附件表

3) act\_hi\_comment 历史意见表，评论

4) act\_hi\_identitylink 历史流程人员表

5) act\_hi\_detail 历史详情表，提供历史变量的查询

6) act\_hi\_procinst 历史流程实例表

7) act\_hi\_taskinst 历史任务实例表

8) act\_hi\_varinst 历史变量表

**组织机构表**

1) act\_id\_group 用户组信息表

2) act\_id\_info 用户扩展信息表

3) act\_id\_membership 用户与用户组对应信息表

4) act\_id\_user 用户信息表

这四张表很常见，基本的组织机构管理，关于用户认证方面建议还是自己开发一套，组件自带的功能太简单，使用中有很多需求难以满足

**通用数据表**

1) act\_ge\_bytearray 二进制数据表，所有二进制内容保存到这个表，比如bpmn文件。

2) act\_ge\_property 属性数据表存储整个流程引擎级别的数据,初始化表结构时，会默认插入三条记录，

**其它数据表**

1. act\_evt\_log 事件日志，默认不开启
2. act\_procdef\_info 流程定义的动态变更信息

## Activiti对象说明

1. 几个和流程相关的对象

Deployment：部署对象，和部署表(act\_re\_deployment)对应

ProcessDefinition：流程定义对象，和流程定义表(act\_re\_procdef)对应

ProcessInstance：流程实例对象，和流程实例表(act\_ru\_execution)对应

Task：任务对象，和任务表(act\_ru\_task)对应

1. 几个Service对象

RepositoryService：操作部署、流程定义等静态资源信息

RuntimeService：操作流程实例，启动流程实例、查询流程实例、删除流程实例等动态信息

TaskService：操作任务，查询任务、办理任务等和任务相关的信息

HistoryService：操作历史信息的，查询历史信息

IdentityService：操作用户和组

1. 几个Query对象

DeploymentQuery：对应查询部署表(act\_re\_deployment)

ProcessDefinitionQuery：对应查询流程定义表(act\_re\_procdef)

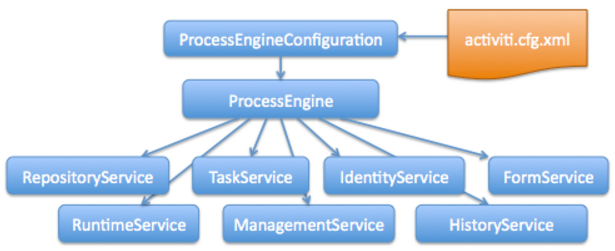
ProcessInstanceQuery：对应查询流程实例表(act\_ru\_execution)

TaskQuery：对应查询任务表(act\_ru\_task)

bpmn通过RepositoryService部署，引擎会把bpmn解析成可执行的东西，生成一个Deployment(部署)，对应的生成流程定义(ProcessDefinition)。

通过RuntimeService启动流程定义，得到一个流程实例。同时会生Task(任务)，任务是绑定到用户的，记录当前操作者是谁，任务状态是什么，用TaskService操作任务。

# Service服务



所有的Service都通过流程引擎获得。

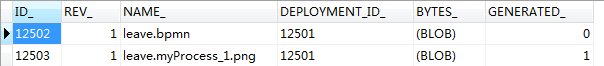
## RepositoryService

仓库服务是存储相关的服务，一般用来部署流程文件，获取流程文件（bpmn和图片），查询流程定义信息等操作，是引擎中的一个重要的服务。部署流程定义操作的数据库表有：部署表(act\_re\_deployment)、流程定义表(act\_re\_procdef)和二进制表(act\_ge\_bytearray)

**//获取仓库服务  
RepositoryService repositoryService= processEngine**.**getRepositoryService**();  
**//部署流程  
Deployment deployment =repositoryService**.**createDeployment**()  
 .**name**(**"请假流程"**)  
 .**addClasspathResource**(**"leave.bpmn"**)  
 .**deploy**(); **// 完成部署**

**ProcessDefinitionQuery query = repositoryService**.**createProcessDefinitionQuery**();  
**// 根据流程定义的key来过滤  
query**.**processDefinitionKey**(**"leave2"**);  
**// 添加排序条件  
query**.**orderByProcessDefinitionVersion**().**desc**();  
**//分页查询，从哪开始，查询几条  
query**.**listPage**(**0**, **2**);  
**// 查询的是所有的流程定义  
List<ProcessDefinition> list = query**.**list**();  
**for** (**ProcessDefinition pd : list**) **{  
 System**.***out***.println(**pd**.**getId**() **+ "-" + pd**.**getName**() **+ "-" + pd**.**getVersion**());  
**}  
//根据id删除流程  
repositoryService**.**deleteDeployment**(**"1005"**);

部署文件实际是一个bpmn文件和一个png图片



部署bpmn的时候，系统自动生成了一个png图片，可以下载查看：

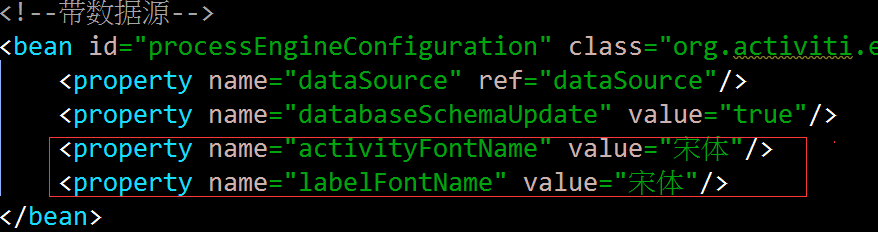
**RepositoryService repositoryService = processEngine**.**getRepositoryService**();  
**String deploymentId = "72501"**;**//部署id  
List<String> names = repositoryService**.**getDeploymentResourceNames**(**deploymentId**);  
**for** (**String name : names**) **{  
 System**.***out***.println(**name**);  
 **// 获得两个流程定义文件对应的输入流** InputStream **in = repositoryService**.**getResourceAsStream**(**deploymentId**, **name**);  
 **// 读取输入流写到指定的本地磁盘上** FileCopyUtils.copy(**in**, **new** FileOutputStream(**"F:/" + name**));  
 **in**.close();  
**}**

去F盘下可以看到两个文件：



自动生成的png图片中文会乱码，解决这个问题需要在processEngineConfiguration的bean中配置字体，如图：

<property name="activityFontName" value="宋体"/>  
<property name="labelFontName" value="宋体"/>



如果不想生成png图片，可以配置如下属性：

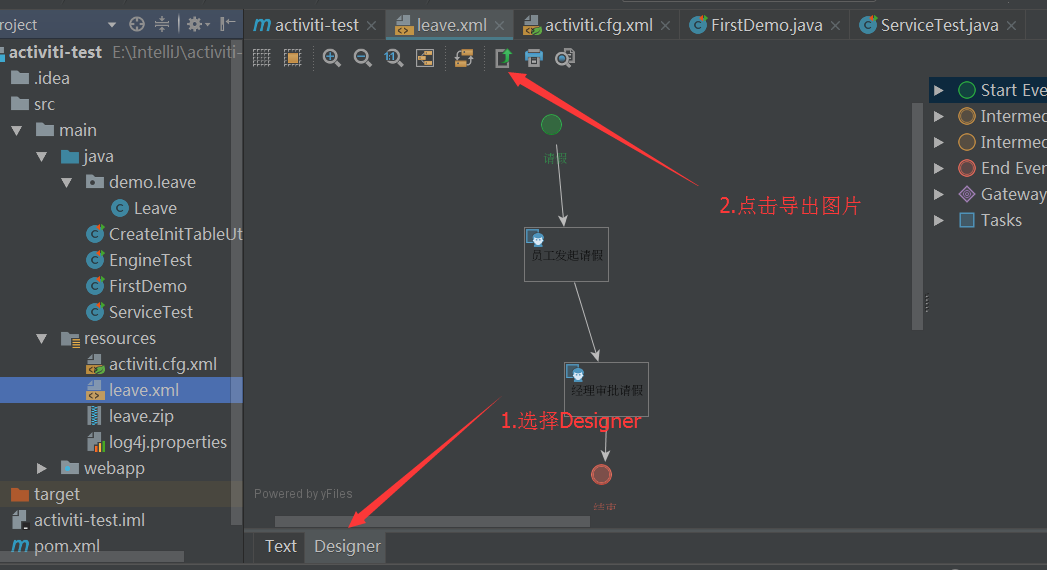
<property name="createDiagramOnDeploy" value="false" />

还可以将png和bpnm添加到zip压缩包中，部署zip压缩包。

**// 从类路径下读取leave.zip压缩文件，并把它包装成一个输入流  
ZipInputStream zipInputStream = new** ZipInputStream(**this**.getClass()  
 .getClassLoader().getResourceAsStream(**"leave.zip"**));  
**Deployment deployment = repositoryService**.**createDeployment**()  
 .**addZipInputStream**(**zipInputStream**)  
 .**deploy**(); **// 完成部署**

Activiti会把压缩包内的bpmn和png添加到数据库，即数据库里存的是解压后的两个文件而不是zip。

使用idea生成bpmn对应的png，需要把bpmn的文件后缀名改成xml



根据流程定义id获取文件流：

**String processDefinitionId = "leave2:5:62504"**; **// 流程定义id  
// 直接获得png图片的名称  
// 根据流程定义id查询流程定义对象  
ProcessDefinitionQuery query = repositoryService**.**createProcessDefinitionQuery**();  
**query**.**processDefinitionId**(**processDefinitionId**);  
**ProcessDefinition processDefinition = query**.**singleResult**();  
**// 根据流程定义对象获得png图片的名称,getResourceName()是获得bpmn  
String pngName = processDefinition**.**getDiagramResourceName**();  
**// 直接获得png图片对应的输入流**InputStream **pngStream = repositoryService**.**getProcessDiagram**(**processDefinitionId**);  
**// 读取输入流写到指定的本地磁盘上**FileCopyUtils.copy(**pngStream**, **new** FileOutputStream(**"F:/" + pngName**));  
**pngStream**.close();

## RuntimeService

流程运行时的流程实例，流程定义，流程版本，流程节点等信息，使用运行时服务操作，是引擎中的一个重要的服务，启动流程实例操作的数据表有流程实例表(act\_ru\_execution)、任务表(act\_ru\_task)。

**RuntimeService runtimeService = processEngine**.**getRuntimeService**();  
**//key是act\_re\_procdef中的KEY\_,即bpmn文件的id  
runtimeService**.**startProcessInstanceByKey**(**"leave2"**);  
**//或者根据流程定义id启动，id是act\_re\_procdef中的ID\_，推荐使用  
runtimeService**.**startProcessInstanceById**(**"leave2:5:62504"**);

一个KEY\_对应多个ID\_，即一个流程定义可以产生多个流程实例。

**//查询流程，操作的是流程实例表(act\_ru\_execution)  
ProcessInstanceQuery query = runtimeService**.**createProcessInstanceQuery**();  
**List<ProcessInstance> list = query**.**list**();

删除一个流程实例

**String processInstanceId = "1001"**; **// 流程实例id  
String deleteReason = "不请假了"**; **// 删除原因，任君写  
runtimeService**.**deleteProcessInstance**(**processInstanceId**, **deleteReason**);

## TaskService

对任务进行查询、接收、办理、完成等操作。查询任务操作的数据表是任务表(act\_ru\_task)。

**TaskService taskService = processEngine**.**getTaskService**();  
**//创建查询对象,指定查询人,进行查询  
List<Task> taskList =taskService**.**createTaskQuery**().**taskAssignee**(**"ZhangSan"**).**list**();

**//提交任务到下一个代理人  
taskService**.**complete**(**task**.**getId**());

listPage(offset,pageSize)用于分页查询

## IdentityService

流程运行过程中的一些用户信息，组信息等操作使用认证服务，但是认证服务一般只作为辅助，每一个系统都有一个比较完整的人员系统，创建用户和用户组等操作。一般不使用自带的认证。

**//添加用户组  
IdentityService identityService = processEngine**.**getIdentityService**();  
**Group groupEntity = identityService**.**newGroup**(**"1001"**);  
**groupEntity**.**setName**(**"超级管理员"**);  
**groupEntity**.**setType**(**"administrator"**);  
**identityService**.**saveGroup**(**groupEntity**);**//保存用户组  
//删除用户组  
identityService**.**deleteGroup**(**"1"**);  
  
**//添加用户  
User user = identityService**.**newUser**(**"10001"**);  
**user**.**setEmail**(**"admin@sina.com"**);  
**user**.**setFirstName**(**"zhang"**);  
**user**.**setLastName**(**"san"**);  
**user**.**setPassword**(**"admin"**);  
**identityService**.**saveUser**(**user**);**//保存用户  
identityService**.**deleteUser**(**"10001"**);**//删除用户**

## HistoryService

流程运行时和运行完成之后的一些历史信息，包括历史任务，历史节点等

**HistoryService historyService = processEngine**.**getHistoryService**();  
**//查询历史运行信息  
HistoricProcessInstance historicProcessInstance =  
 historyService** .**createHistoricProcessInstanceQuery**()  
 .**processInstanceId**(**"40001"**).**singleResult**();  
**System**.***out***.println(**"开始时间：" + historicProcessInstance**.**getStartTime**());  
**System**.***out***.println(**"结束时间：" + historicProcessInstance**.**getEndTime**());

## FormService

可选服务，任务表单管理

后续讲解

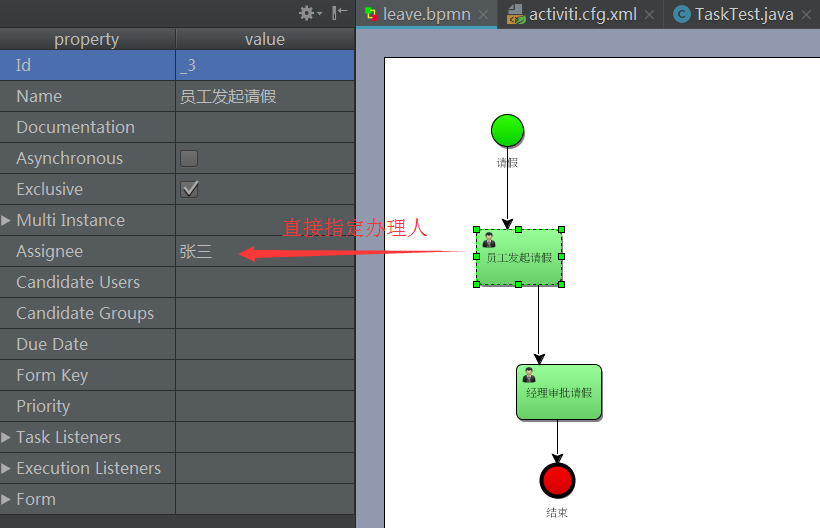
## ManagementService

Management Service 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。

**ManagementService managementService = processEngine**.**getManagementService**();  
**String taskTable = managementService**.**getTableName**(**Task**.**class**);  
**System**.***out***.println(**taskTable**);

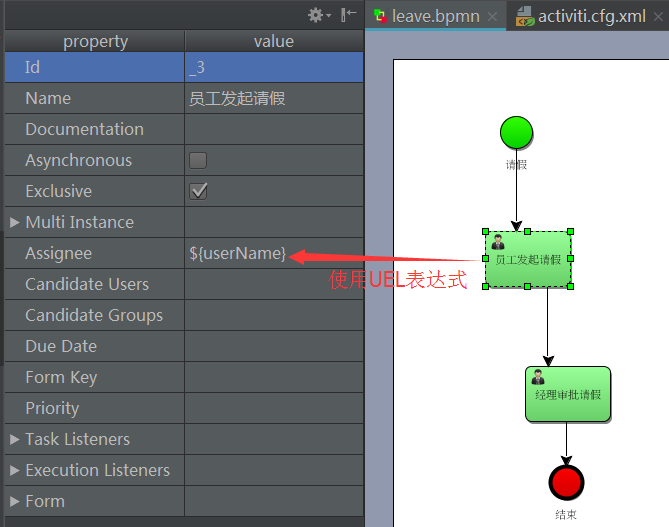
# 任务管理

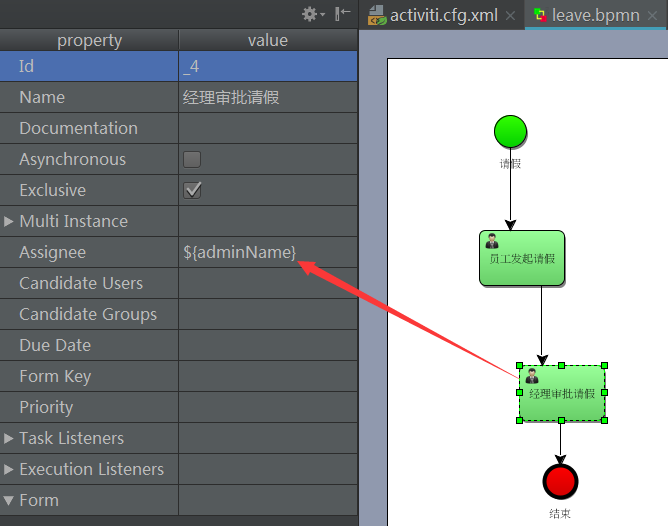
## 指定办理人



## 使用UEL

### UEL-VALUE



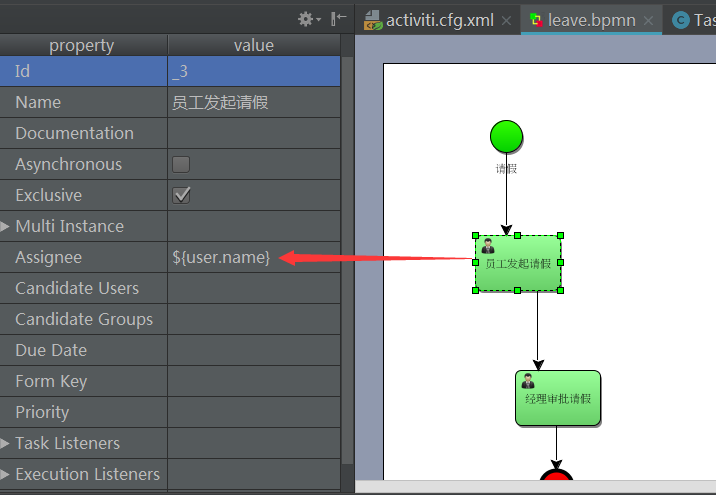


**Map<String**,**Object> variable = new** HashMap**<>**();  
**//指定当前节点的办理人是WangWu   
variable**.**put**(**"userName"**,**"WangWu"**);  
**runtimeService**.**startProcessInstanceById**(**"leave:5:110003"**,**variable**);

执行任务

**List<Task> taskList = taskService**.**createTaskQuery**().**taskAssignee**(**"WangWu"**).**list**();  
**for** (**Task task : taskList**) **{**  
 **//提交任务到下一个代理人  
 Map<String**,**Object> variable = new** HashMap**<>**();  
 **//指定处理人是ZhaoLiu  
 variable**.**put**(**"adminName"**,**"ZhaoLiu"**);  
 **taskService**.**complete**(**task**.**getId**(),**variable**);  
**}**

也可以绑定pojo实体类



注意实体类必须实现java.io.Serializable接口

**public class User implements Serializable {  
 private Integer id**;  
 **private String name**;  
 **//getter/setter方法略**  
**}**

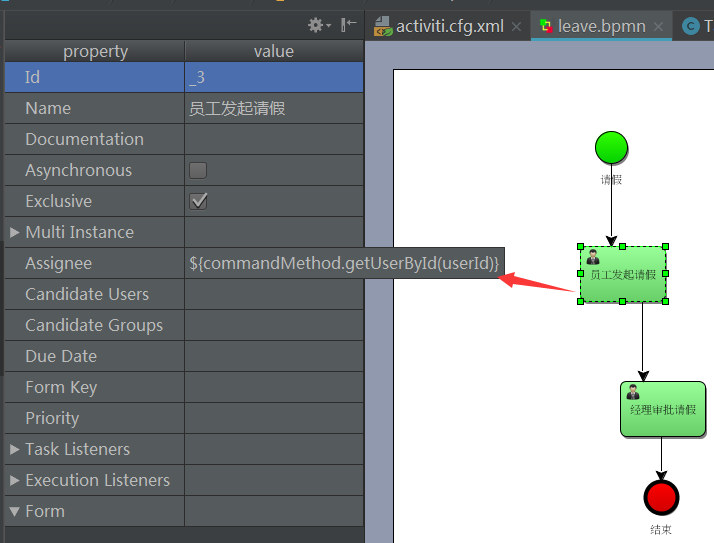
**Map<String**, **Object> variable = new** HashMap**<>**();  
**//绑定实体类属性，${user.name}就会调用user对象的getName()方法  
User user = new** User();  
**user**.setName(**"小明"**);  
**variable**.**put**(**"user"**, **user**);  
**runtimeService**.**startProcessInstanceById**(**"leave:6:127503"**, **variable**);

### UEL-METHOD

执行步骤

1. 设置节点的执行人为${bean.methodName(param)} ，其中bean方法是我们注入到spring中的一个bean的id，methodName是bean对应的类中的方法名。
2. 将method方法注入到activiti的processEngineConfiguration的bean中（在我们的activiti.cfg.xml中）
3. 启动一个流程设置全局变量param作为启动参数.

**package demo.leave**;  
  
**public class CommandMethod {  
 public String** getUserById(**int userId**)**{  
 //模拟数据库中查询的数据  
 return "activiti"+userId**;  
 **}  
}**

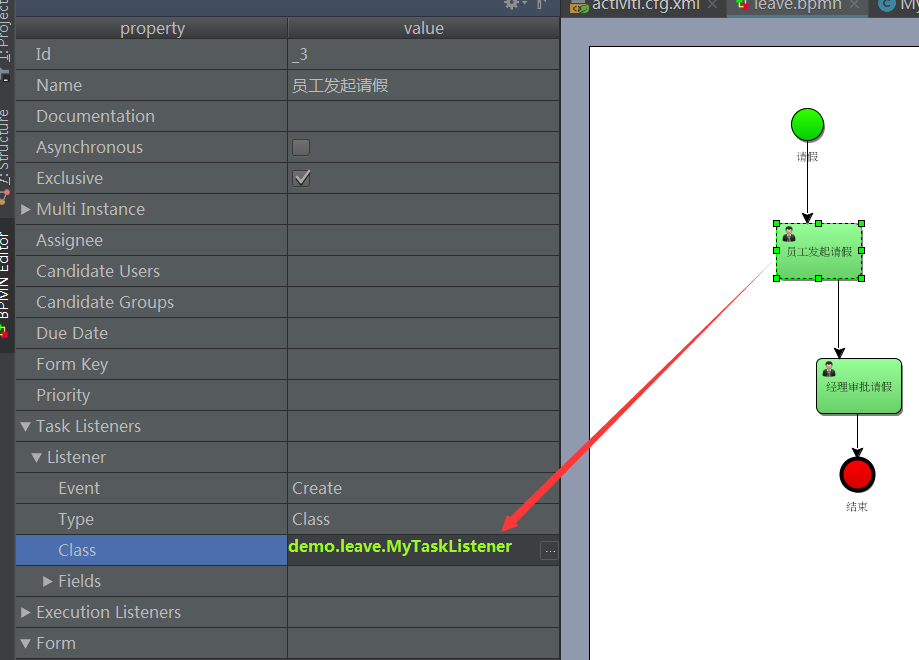


<bean id="commandMethod" class="demo.leave.CommandMethod"/>  
<!--带数据源-->  
<bean id="processEngineConfiguration" class="org.activiti.engine.impl.cfg.StandaloneProcessEngineConfiguration">  
 <property name="dataSource" ref="dataSource"/>  
 <property name="databaseSchemaUpdate" value="true"/>  
 <property name="beans">  
 <map>  
 <entry key="commandMethod" value-ref="commandMethod"></entry>  
 </map>  
 </property>  
</bean>

**Map<String**, **Object> variable = new** HashMap**<>**();  
**//绑定方法中参数  
variable**.**put**(**"userId"**, **123**);  
**runtimeService**.**startProcessInstanceById**(**"leave:10:167503"**, **variable**);

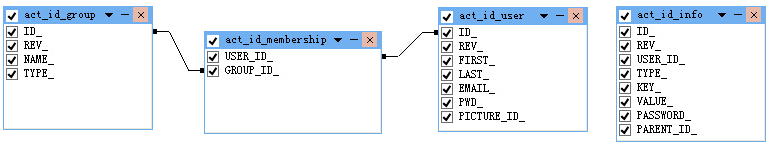
## 使用监听器

**package demo.leave**;  
  
**import org.activiti.engine.delegate.DelegateTask**;  
**import org.activiti.engine.delegate.TaskListener**;  
  
**public class MyTaskListener implements TaskListener {  
 //设置待办事宜的处理人  
 @Override  
 public void** notify(**DelegateTask delegateTask**) **{  
 delegateTask**.**setAssignee**(**"zhangsan"**);  
 **}  
}**



发布之后启动流程，会自动设定处理人。

# 用户与用户组



activiti的用户表和其它表没有外键关系，方便进行扩展。activiti的用户表提供了简单的信息，和其它框架整合的时候，可以使用自己定义的表。

## 用户组

**IdentityService identityService = processEngine**.**getIdentityService**();  
**GroupQuery groupQuery = identityService**.**createGroupQuery**();  
**List<Group> datas = groupQuery**.**list**();  
**//其他类似查询方法  
groupQuery**.**listPage**(**1**, **2**);**//分页返回查询结果；偏移量，每页几条  
groupQuery**.**count**();**//结果总数  
groupQuery**.**groupName**(**"超级管理员"**).**list**();**//根据name查询  
groupQuery**.**groupNameLike**(**"%超级A%"**).**list**();**//根据name模糊查询  
groupQuery**.**groupType**(**"typeA"**).**list**();**//根据type查询  
groupQuery**.**groupId**(**"1"**).**singleResult**();**//根据id查询，查询结果唯一  
groupQuery**.**orderByGroupId**().**asc**().**list**();**//结果升序  
groupQuery**.**orderByGroupName**().**desc**().**list**();**//结果降序**

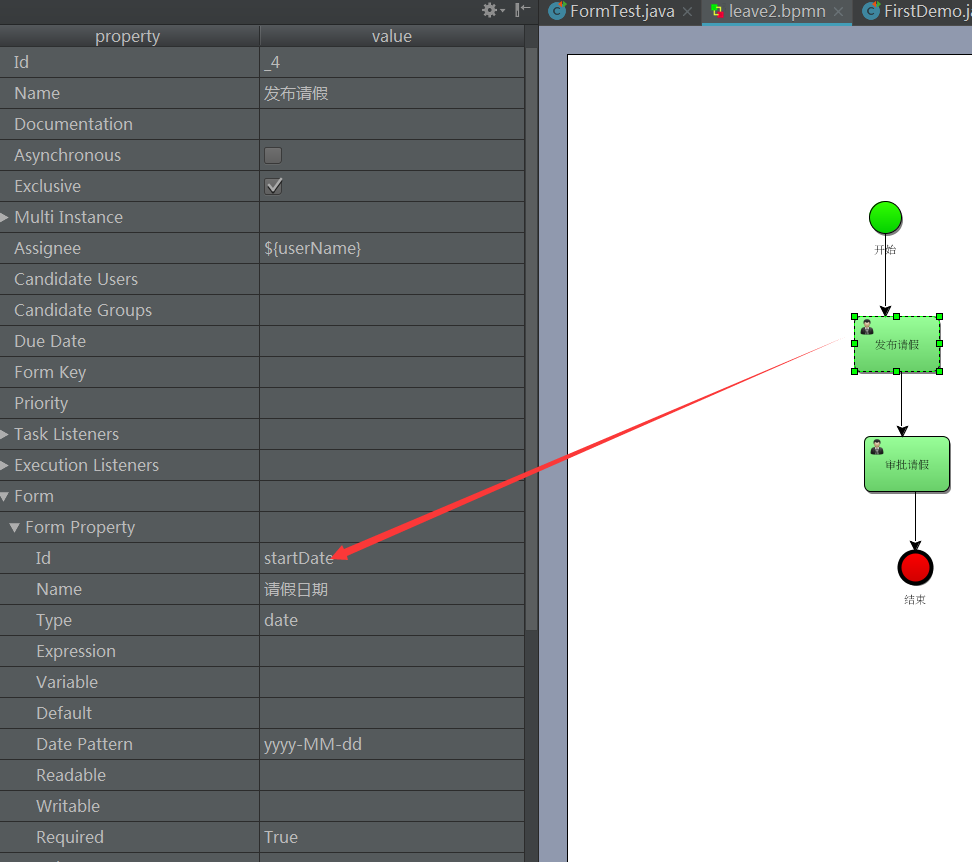
## 用户

**User user = identityService**.**newUser**(**"10001"**);  
**user**.**setEmail**(**"zhangsan@xx.com"**);  
**user**.**setFirstName**(**"zhang"**);  
**user**.**setLastName**(**"san"**);  
**user**.**setPassword**(**"admin"**);  
**identityService**.**saveUser**(**user**);**//保存用户  
identityService**.**deleteUser**(**"10001"**);**//删除用户  
//验证用户密码，参数用户id和密码  
boolean flag = identityService**.**checkPassword**(**"10001"**, **"123456"**);  
**//设置用户info信息  
identityService**.**setUserInfo**(**"用户id"**, **"INFO表KEY"**, **"INFO表VALUE"**);  
**//查询相关信息.  
identityService**.**getUserInfo**(**"INFO表USER\_ID"**, **"INFO表KEY"**);  
**identityService**.**createMembership**(**"用户ID"**, **"组ID"**);**//给用户添加分组  
identityService**.**deleteMembership**(**"用户ID"**, **"组ID"**);**//删除用户和组关系  
//查询某用户的分组  
List<Group> datas = identityService**.**createGroupQuery**().**groupMember**(**"userId"**).**list**();  
**//查询某组下的用户  
List<User> datas = identityService**.**createUserQuery**().**memberOfGroup**(**"groupId"**).**list**();

# 表单

除了使用Assignee保存任务操作者，每个任务节点可以保存一些简单的信息。表单能保存的数据较少，可以自己创建表保存数据。

## 内置表单



<userTask activiti:assignee="${userName}" activiti:exclusive="true" id="\_4" name="发布请假">  
 <extensionElements>  
 <activiti:formProperty datePattern="yyyy-MM-dd" id="startDate" name="请假日期" required="true" type="date"/>  
 <activiti:formProperty id="reason" name="请假原因" required="true" type="string"/>  
 </extensionElements>  
</userTask>  
<userTask activiti:assignee="${adminName}" activiti:exclusive="true" id="\_5" name="审批请假">  
 <extensionElements>  
 <activiti:formProperty id="advise" name="意见" type="enum">  
 <activiti:value id="true" name="同意"/>  
 <activiti:value id="false" name="拒绝"/>  
 </activiti:formProperty>  
 </extensionElements>  
</userTask>

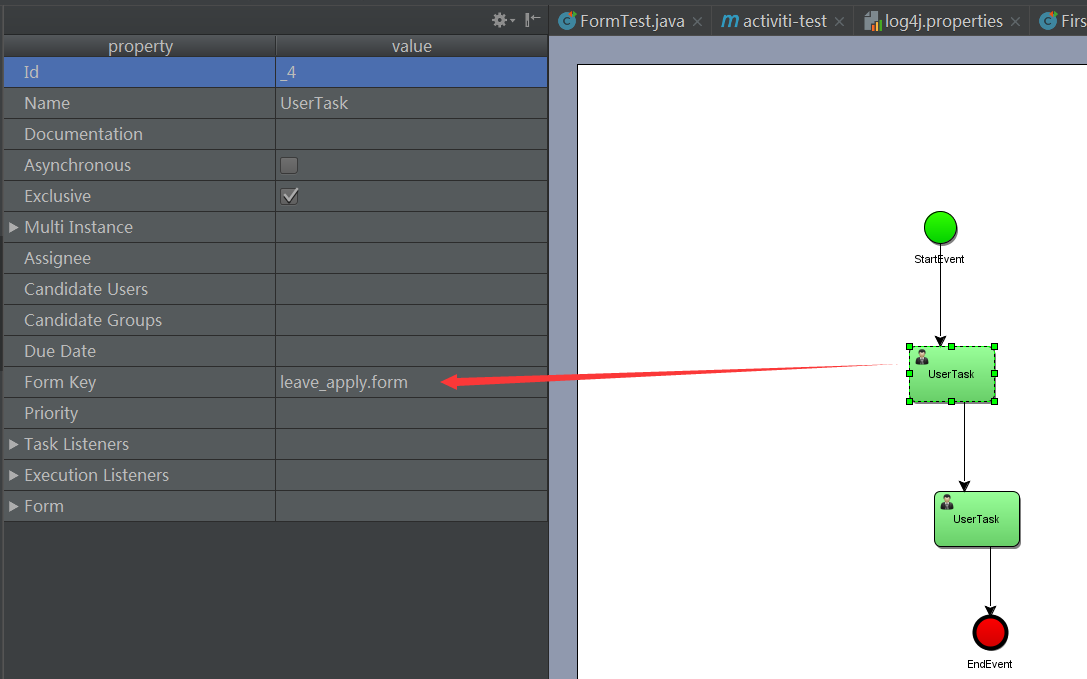
操作表单：

**ProcessDefinition definition = processEngine**.**getRepositoryService**()  
 .**createProcessDefinitionQuery**().**processDefinitionKey**(**"leave\_mybatis"**).**latestVersion**().**singleResult**();  
**FormService formService = processEngine**.**getFormService**();  
**TaskService taskService = processEngine**.**getTaskService**();  
**RuntimeService runtimeService = processEngine**.**getRuntimeService**();  
**Map<String**, **String> param1 = new** HashMap**<>**();  
**param1**.**put**(**"userName"**, **"user1"**);  
**//模拟当前请假条的id,保证task能查出唯一的任务。当然也可以查出一个task list批量处理  
String businessKey = System**.currentTimeMillis() **+ ""**;  
**//启动任务  
ProcessInstance processInstance = formService**.**submitStartFormData**(**definition**.**getId**(), **businessKey**, **param1**);  
**//查询任务  
Task task = taskService**.**createTaskQuery**().**taskAssignee**(**"user1"**).**processDefinitionKey**(**"leave\_mybatis"**).**processInstanceBusinessKey**(**businessKey**).**singleResult**();  
**//查询任务中的表单信息  
TaskFormData formData = formService**.**getTaskFormData**(**task**.**getId**());  
**for** (**FormProperty property : formData**.**getFormProperties**()) **{  
 System**.***out***.println(**property**.**getName**() **+ "=" + property**.**getType**());  
**}  
Map<String**, **String> variable = new** HashMap**<>**();  
**variable**.**put**(**"reason"**, **"有事"**);  
**variable**.**put**(**"startDate"**, **"2018-01-01"**);  
**variable**.**put**(**"adminName"**, **"admin1"**);  
**//提交表单(完成当前节点的任务)  
formService**.**submitTaskFormData**(**task**.**getId**(), **variable**);  
  
**//查询出下一节点的任务  
Task task2 = taskService**.**createTaskQuery**().**taskAssignee**(**"admin1"**).**processDefinitionKey**(**"leave\_mybatis"**).**processInstanceBusinessKey**(**businessKey**).**singleResult**();  
**//查询出任务中的参数  
Map<String**, **Object> variables = runtimeService**.**getVariables**(**processInstance**.**getId**());  
**for** (**String key : variables**.**keySet**()) **{  
 System**.***out***.println(**key + "=" + variables**.**get**(**key**));  
**}  
//完成任务  
Map<String**, **String> variable2 = new** HashMap**<>**();  
**variable2**.**put**(**"advise"**, **"true"**);  
**formService**.**submitTaskFormData**(**task2**.**getId**(), **variable2**);

## 外置表单

外置表单可以自定义一个代码片段，如html片段，另存为leave\_apply.form：

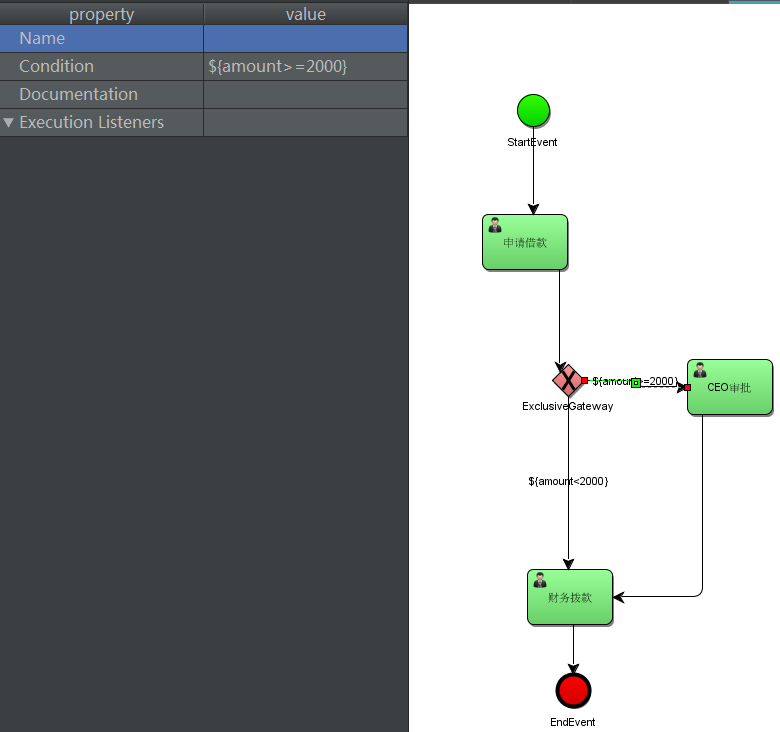
<div class="control-group">  
 <label class="control-label" for="startDate">开始时间：</label>  
 <div class="controls">  
 <input type="text" id="startDate" name="startDate" class="datepicker" data-date-format="yyyy-mm-dd" required />  
 </div>  
</div>  
<div class="control-group">  
 <label class="control-label" for="reason">请假原因：</label>  
 <div class="controls">  
 <textarea id="reason" name="reason" required></textarea>  
 </div>  
</div>



**Deployment deployment = processEngine**.**getRepositoryService**() **// 与流程定义和部署对象相关的Service** .**createDeployment**() **// 创建一个部署对象** .**name**(**"1801请假流程-外置表单"**) **// 设置对应流程的名称** .**addClasspathResource**(**"leave7.bpmn"**).**addClasspathResource**(**"leave\_apply.form"**)  
 .**deploy**(); **// 完成部署**

**Map<String**, **String> param1 = new** HashMap**<>**();  
**param1**.**put**(**"userName"**, **"user1"**);  
**//模拟当前请假条的id,保证task能查出唯一的任务。当然也可以查出一个task list批量处理  
String businessKey = System**.currentTimeMillis() **+ ""**;  
**//启动任务  
ProcessInstance processInstance = formService**.**submitStartFormData**(**definition**.**getId**(), **businessKey**, **param1**);  
**//查询任务  
Task task = taskService**.**createTaskQuery**().**taskAssignee**(**"user1"**).**processDefinitionKey**(**"myProcess\_1"**).**processInstanceBusinessKey**(**businessKey**).**singleResult**(); **//查询出html片段，可以显示在页面上  
Object renderedStartForm = formService**.**getRenderedTaskForm**(**task**.**getId**());  
**System**.***out***.println(**renderedStartForm**);

# 网关



网关类似于程序中的if判断，当满足某一条件时，任务进入下一个指定的节点。

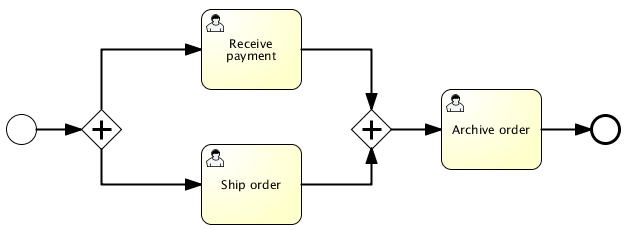
网关有：排他网关、并行网关、包含网关、基于事件网关。

## 排他网关

排他网关只会选择一条顺序流执行。 就是说，虽然多个顺序流的条件结果为true， 那么XML中的第一个顺序流（也只有这一条）会被选中，并用来继续运行流程。 如果没有选中任何顺序流，会抛出一个异常。排他网关类似于if-elseif

**ProcessDefinition definition = processEngine**.**getRepositoryService**()  
 .**createProcessDefinitionQuery**().**processDefinitionKey**(**"loan\_gateway"**).**latestVersion**().**singleResult**();  
**Map<String**, **Object> param1 = new** HashMap**<>**();  
**param1**.**put**(**"amount"**, **"5500"**);  
**//模拟当前借款单的id,保证task能查出唯一的任务。当然也可以查出一个task list批量处理  
String businessKey = System**.currentTimeMillis() **+ ""**;  
**//启动任务  
ProcessInstance processInstance =  
 runtimeService**.**startProcessInstanceByKey**(**"loan\_gateway"**, **businessKey**, **param1**);  
**//查询任务  
Task task = taskService**.**createTaskQuery**()  
 .**processDefinitionKey**(**"loan\_gateway"**)  
 .**processInstanceBusinessKey**(**businessKey**).**singleResult**();  
**taskService**.**complete**(**task**.**getId**());

## 并行网关



网关也可以表示流程中的并行情况。最简单的并行网关是 **并行网关**，它允许将流程 分成多条分支，也可以把多条分支 汇聚到一起。

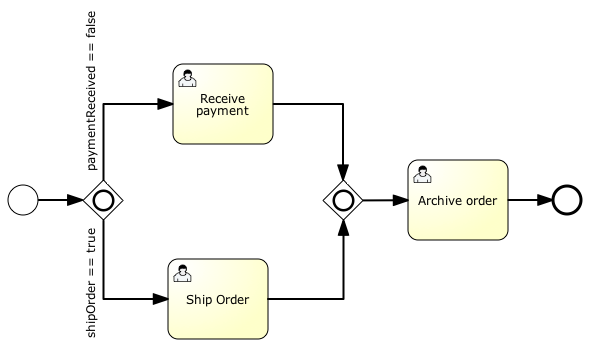
并行网关的功能是基于进入和外出的顺序流的：

* **分支：** 并行后的所有外出顺序流，为每个顺序流都创建一个并发分支。
* **汇聚：** 所有到达并行网关，在此等待的进入分支， 直到所有进入顺序流的分支都到达以后， 流程就会通过汇聚网关。

注意，如果同一个并行网关有多个进入和多个外出顺序流， 它就同时具有**分支和汇聚功能**。 这时，网关会先汇聚所有进入的顺序流，然后再切分成多个并行分支。

**与其他网关的主要区别是，并行网关不会解析条件。 即使顺序流中定义了条件，也会被忽略。**

## 包含网关



**包含网关**可以看做是排他网关和并行网关的结合体。 和排他网关一样，你可以在外出顺序流上定义条件，包含网关会解析它们。 但是主要的区别是包含网关可以选择多于一条顺序流，这和并行网关一样。

包含网关的功能是基于进入和外出顺序流的：

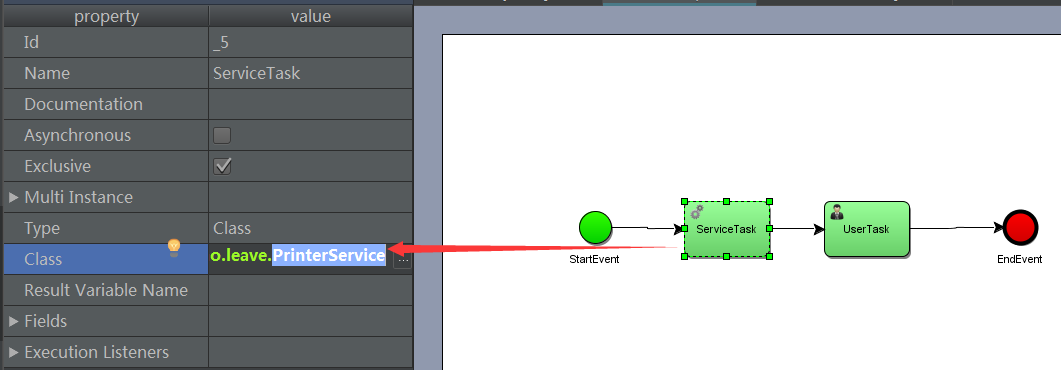
* **分支：** 所有外出顺序流的条件都会被解析，结果为true的顺序流会以并行方式继续执行， 会为每个顺序流创建一个分支。
* **汇聚：** 所有并行分支到达包含网关，会进入等待章台， 直到每个包含流程token的进入顺序流的分支都到达。 这是与并行网关的最大不同。换句话说，包含网关只会等待被选中执行了的进入顺序流。 在汇聚之后，流程会穿过包含网关继续执行。

注意，如果同一个包含节点拥有多个进入和外出顺序流， 它就会同时**含有分支和汇聚功能**。 这时，网关会先汇聚所有拥有流程token的进入顺序流， 再根据条件判断结果为true的外出顺序流，为它们生成多条并行分支。

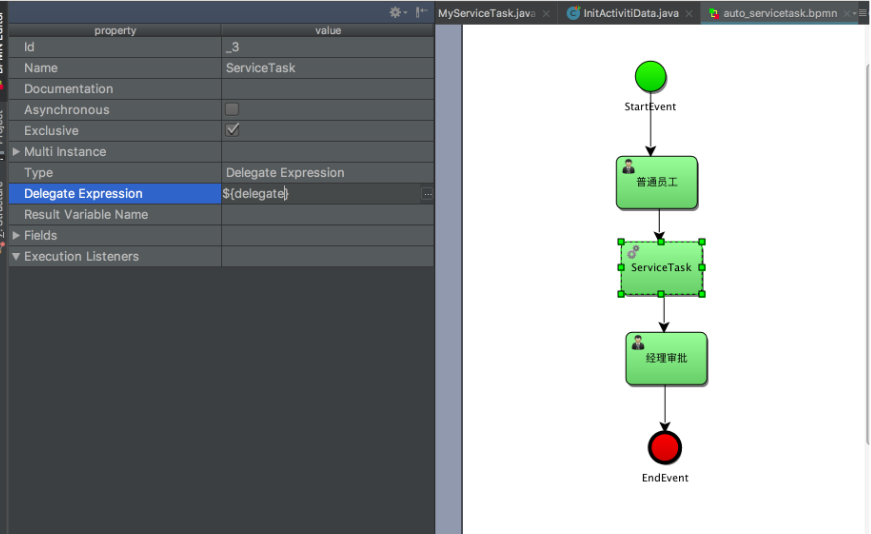
# ServiceTask

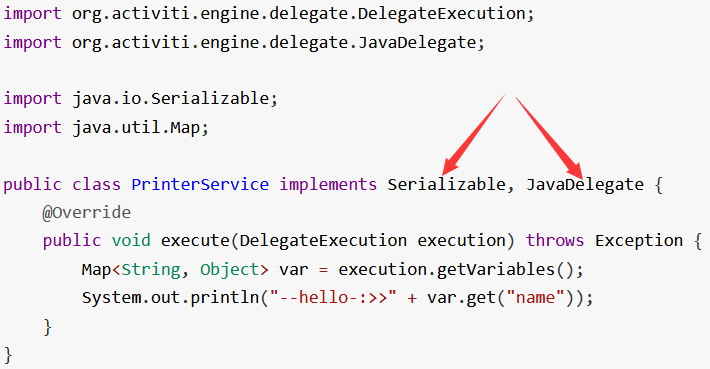
定义Service类。

**import org.activiti.engine.delegate.DelegateExecution**;  
**import org.activiti.engine.delegate.JavaDelegate**;  
  
**import java.util.Map**;  
  
**public class PrinterService implements JavaDelegate {  
 @Override  
 public void** execute(**DelegateExecution delegateExecution**) **throws Exception {  
 Map<String**, **Object> variable = delegateExecution**.**getVariables**();  
 **System**.***out***.println(**variable**.**get**(**"name"**));  
 **}  
}**



启动任务时，PrinterService会自动执行。



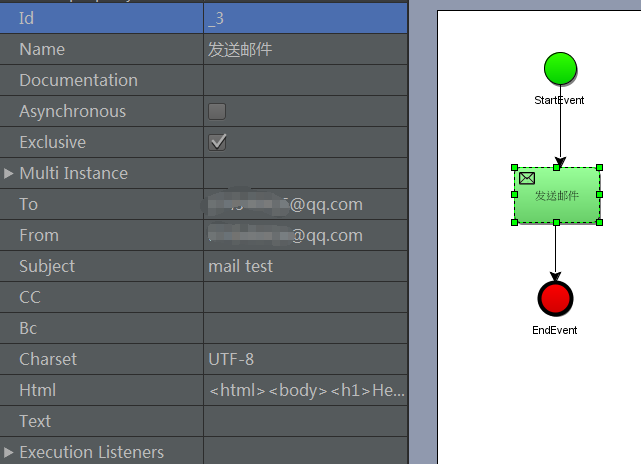


**Map<String**, **Object> param1 = new** HashMap**<>**();  
**param1**.**put**(**"delegate"**,**new** PrinterService());  
**param1**.**put**(**"name"**, **"abcd"**);  
**param1**.**put**(**"userId"**, **"123"**);  
**ProcessInstance processInstance =  
 runtimeService**.**startProcessInstanceByKey**(**"ServiceTask"**,**param1**);

# MailTask邮件任务

使用QQ邮箱测试邮件服务，默认QQ邮箱是不开启的，





<bean id="processEngineConfiguration" class="org.activiti.engine.impl.cfg.StandaloneProcessEngineConfiguration">  
 <property name="dataSource" ref="dataSource"/>  
 <property name="databaseSchemaUpdate" value="true"/>  
 <property name="mailServerHost" value="smtp.qq.com"/>  
 <property name="mailServerPort" value="465"/>  
 <property name="mailServerDefaultFrom" value="xxxx@qq.com"/>  
 <property name="mailServerUsername" value="xxxx@qq.com"/>  
 <property name="mailServerPassword" value="xxxxx"/>  
 <property name="mailServerUseSSL" value="true"/>  
</bean>

其中的mailServerPassword是QQ邮箱验证成功后生成的授权码

# BPMN规范

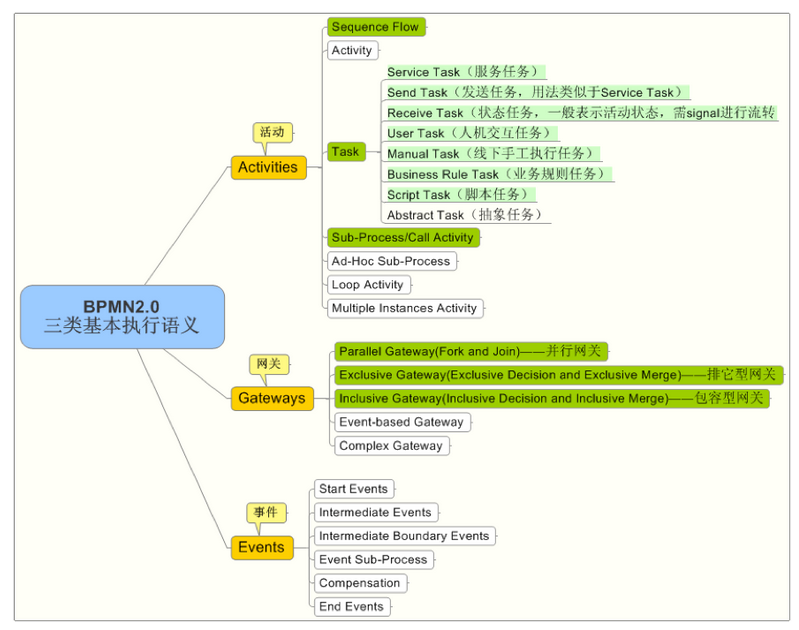
业务流程模型注解（Business Process Modeling Notation - BPMN）是业务流程模型的一种标准图形注解。这个标准 是由对象管理组（Object Management Group - OMG）维护的。包含以下部分：

流对象：事件、活动、网关

连接对象：序列流、消息流、关联

泳道：池、道

人工制品：数据对象、组、注释



## 事件

启动事件：

<startEvent id="start" name="myStart" />

结束事件：

<endEvent id="end" name="myEnd" />

## 网关

包含网关:

**<inclusiveGateway id="inclusiveGatewaySplit" default="flow3"/>**

排他网关：

<exclusiveGateway id="decision" name="decideBasedOnAmountAndBankType" **default="myFlow"**/>

并行网关：

<parallelGateway id="myParallelGateway" name="My Parallel Gateway" />

## 任务

人工任务：

<userTask id="myTask" name="My task" />

服务任务：

<serviceTask id="MyServiceTask" name="My service task"

implementation="Other" operationRef="myOperation" />

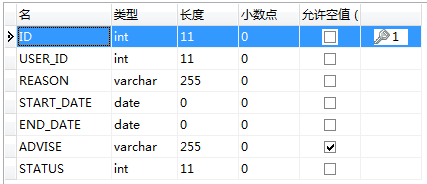
# 与Spring和Mybatis整合

<bean id="activitiDataSource" class="org.apache.commons.dbcp.BasicDataSource">  
 <property name="driverClassName" value="com.mysql.jdbc.Driver"/>  
 <property name="url" value="jdbc:mysql://localhost:3306/activiti?characterEncoding=utf-8"/>  
 <property name="username" value="root"/>  
 <property name="password" value=""/>  
</bean>  
  
<!--配置事务管理器-->  
<bean id="activitiTransactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">  
 <property name="dataSource" ref="activitiDataSource"/>  
</bean>  
  
<!--流程引擎配置-->  
<bean id="processEngineConfiguration" class="org.activiti.spring.SpringProcessEngineConfiguration">  
 <property name="dataSource" ref="activitiDataSource"/>  
 <property name="transactionManager" ref="activitiTransactionManager"/>  
 <property name="databaseSchemaUpdate" value="true"/>  
 <property name="jobExecutorActivate" value="false"/>  
</bean>  
  
<bean id="processEngine" class="org.activiti.spring.ProcessEngineFactoryBean">  
 <property name="processEngineConfiguration" ref="processEngineConfiguration"/>  
</bean>  
  
<!--获取各种服务-->  
<bean id="repositoryService" factory-bean="processEngine" factory-method="getRepositoryService"/>  
<bean id="runtimeService" factory-bean="processEngine" factory-method="getRuntimeService"/>  
<bean id="taskService" factory-bean="processEngine" factory-method="getTaskService"/>  
<bean id="historyService" factory-bean="processEngine" factory-method="getHistoryService"/>  
<bean id="managementService" factory-bean="processEngine" factory-method="getManagementService"/>

整合思路

比如我们想把请假条保存到自己创建的表中，又想使用activiti操作请假流程，

请假条表LEAVE\_APPLY



添加一个请假条之后，获得唯一标识符（如id）。

在启动activiti任务的时候，可以传入一个BUSINESS\_ID,可以把请假条的id当成BUSINESS\_ID传递给activiti。

**runtimeService**.**startProcessInstanceByKey**(**"ProcessDefinitionKey"**, **"BUSINESS\_KEY"**, **variable**);

在act\_ru\_task表中保存了EXECUTION\_ID\_和ASSIGNEE\_（处理人）

act\_ru\_execution表中保存了BUSINESS\_KEY\_

如果处理人指定的是任务组，那么在act\_ru\_identitylink中保存了USER\_ID\_(处理人)和TASK\_ID\_(任务id)与act\_ru\_task表关联，其中处理人与我们自己设计的用户表关联。

如查询某个人要处理的请假条：

select L.ID,L.USER\_ID,L.START\_DATE,L.END\_DATE,L.STATUS,L.REASON,L.ADVISE,L.TASK\_ID,U.REAL\_NAME  
from activiti.ACT\_RU\_TASK RES  
inner join activiti.ACT\_RU\_IDENTITYLINK I on I.TASK\_ID\_ = RES.ID\_  
INNER JOIN activiti.act\_ru\_execution RU ON RES.EXECUTION\_ID\_=RU.ID\_  
INNER JOIN test.leave\_apply L ON RU.BUSINESS\_KEY\_=L.ID  
inner join ADMIN\_USER U on L.USER\_ID=U.ID  
WHERE RES.ASSIGNEE\_ =#{userId}  
or(I.TYPE\_ = 'candidate' and I.USER\_ID\_ = #{userId} )

其中#{userId}是任务的代办者