一、Activiti工作流概述

1.1 什么是工作流

工作流(Workflow)：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“使再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”，如：请假审批流程、报销审批流程、出差审批流程、合同审批流程等。

工作流引擎：主要是为了实现流程自动化控制。

工作流管理系统：一个软件系统，它完成工作量的定义和管理，并按照在系统中预定义好的工作流规则进行工作流实例的执行。

1.2 工作流应用场景

- 业务类：合同审批流程、订单处理流程、出入库审批流程等。
- 行政类：请假流程、出差流程、用车流程、办公用品申请流程等。
- 财务类：报销流程、支付流程等。
- 客户服务类：售后跟踪、客户投诉等。

1.3 什么是Activiti

官网：**https://www.activiti.org**

- Activiti 是由 jBPM （BPM，Business Process Management 即业务流程管理） 的创建者 Tom Baeyens 离开JBoss 之后建立的项目，构建在开发 jBPM 版本 1 到4 时积累的多年经验的基础之上，旨在创建下一代的 BPM 解决方案。
- Activiti 是一个开源的工作流引擎，它实现了BPMN 2.0规范，可以发布设计好的流程定义，并通过api进行流程调度。
- Activiti 作为一个遵从 Apache 许可的工作流和业务流程管理开源平台，其核心是基于Java的超快速、超稳定的BPMN2.0 流程引擎，强调流程服务的可嵌入性和可扩展性，同时更加强调面向业务人员。
- Activiti 流程引擎重点关注在系统开发的易用性和轻量性上。每一项 BPM 业务功能 Activiti 流程引擎都以服务的形式提供给开发人员。通过使用这些服务，开发人员能够构建出功能丰富、轻便且高效的 BPM 应用程序。
- Activiti是一个针对企业用户、开发人员、系统管理员的轻量级工作流业务管理平台，其核心是使用Java开发的快速、稳定的BPMN 2.0流程引擎。Activiti是在ApacheV2许可下发布的，可以运行在任何类型的Java程序中，例如服务器、集群、云服务等。Activiti可以完美地与Spring集成。同时，基于简约思想的设计使Activiti非常轻量级。

1.4 Activiti开发流程

1. 画流程定义模型：

   遵守BPMN的流程规范，使用BPMN的流程定义工具，通过流程符号把整个业务流程定义出来，可以将流程定义文件字节流保存在模型数据表中（Model）。

2. 部署流程定义：

   加载画好的流程定义文件，将它转换成流程定义数据（ProcessDefinition），保存到流程定义数据表中。

3. 启动流程（提交流程申请）：

   生成流程实例数据（ProcessInstance），生成第1节点任务数据（Task）。

4. 处理人审批流程节点任务：

   完成任务审批，生成审批结果， 生成下一节点任务数据。

1.5 BPMN2.0规范是什么

- 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。
- 标准的早期版本（1.2版以及之前）仅仅限制在模型上， 目标是在所有的利益相关者之间形成通用的理解， 在文档，讨论和实现业务流程之上。 BPMN标准证明了它自己，现在市场上许多建模工具都使用了BPMN标准中的元素和结构。
- BPMN规范的2.0版本，当前已经处于最终阶段了， 允许添加精确的技术细节在BPMN的图形和元素中， 同时制定BPMN元素的执行语法。 通过使用XML语言来指定业务流程的可执行语法， BPMN规范已经演变为业务流程的语言， 可以执行在任何兼容BPMN2的流程引擎中，同时依然可以使用强大的图形注解。
- 目前BPMN2.0是最新的版本，它用于在BPM上下文中进行布局和可视化的沟通。BPMN 2.0是使用一些符号来明确业务流程设计流程图的一整套符号规范，它能增进业务建模时的沟通效率。

1.6 BPMN2.0基本流程符号

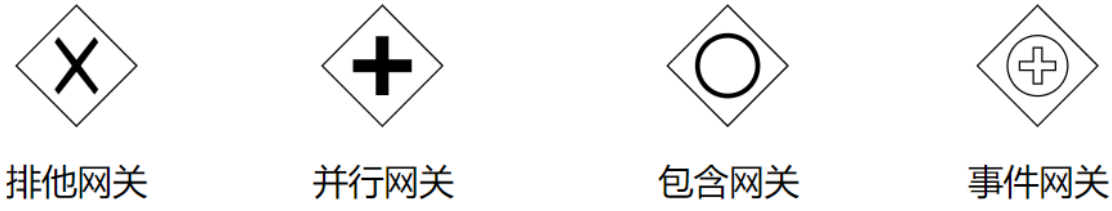
1.6.1 事件Event

1.6.2 活动Activity

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

1.6.3 网关Gateway

网关用来处理决策



排他网关(x)

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

- 如果多条线路计算结果都是true，则会执行第一个值为true的线路。如果所有网关计算结果没有true，则引擎会抛异常；
- 排他网关需要和条件顺序流结合使用，default属性指定默认顺序流，当所有的条件不满足时，会执行默认顺序流。

并行网关（+）

所有路径会被同时选择

- 分支：并行执行所有输出顺序流，为每一条顺序流创建一个并行执行路线。
- 汇聚：所有从并行网关拆分并执行完成的线路均在此等候，直到所有的线路都执行完成才继续向下执行。

包容网关（o）

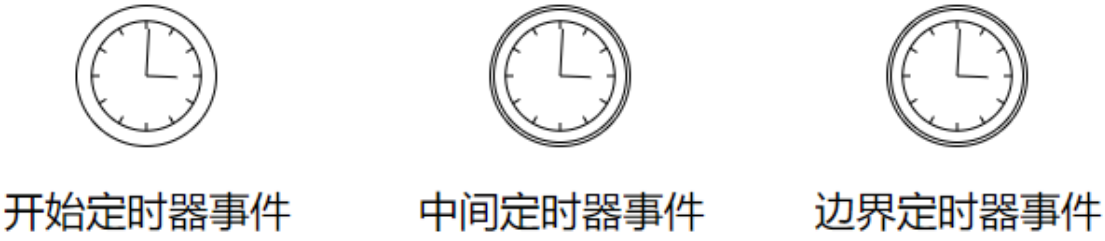
可以同时执行多条线路，也可以在网关上设置条件

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

事件网关（o+）

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

1.6.4 定时器事件



- **开始定时器事件**：可以设置时间，定时开始启动流程实例。
- **中间定时器事件**：设定延迟时间，当完成任务1后，到达延时时间，流程才会走向任务2。
- **边界定时器事件**：用于向某节点上添加边界定时时间，在设定时间内没有完成，流程实例则自动走向下一节点。

二、Activiti环境搭建和流程基础入门

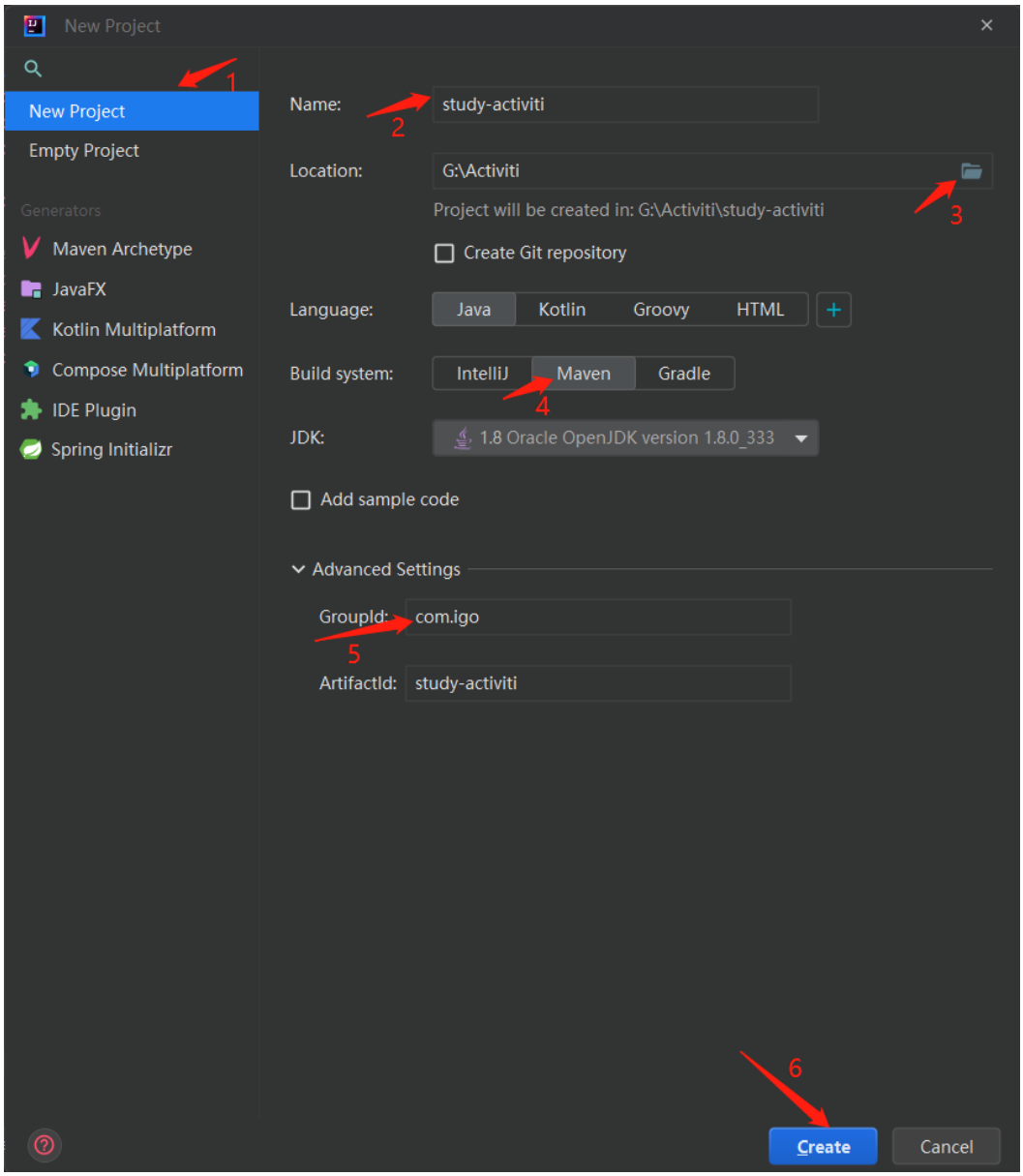
2.1 开发环境

环境参数	版本	环境参数	版本
IDEA	2022.1	maven	3.8.6
Activiti	7.1.0-M6	MySQL	8.0.29
jdk	1.8.0_333	Navicat	16.0.13

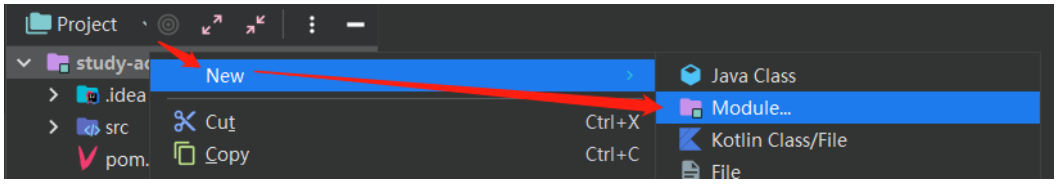
2.2 基于maven构建项目

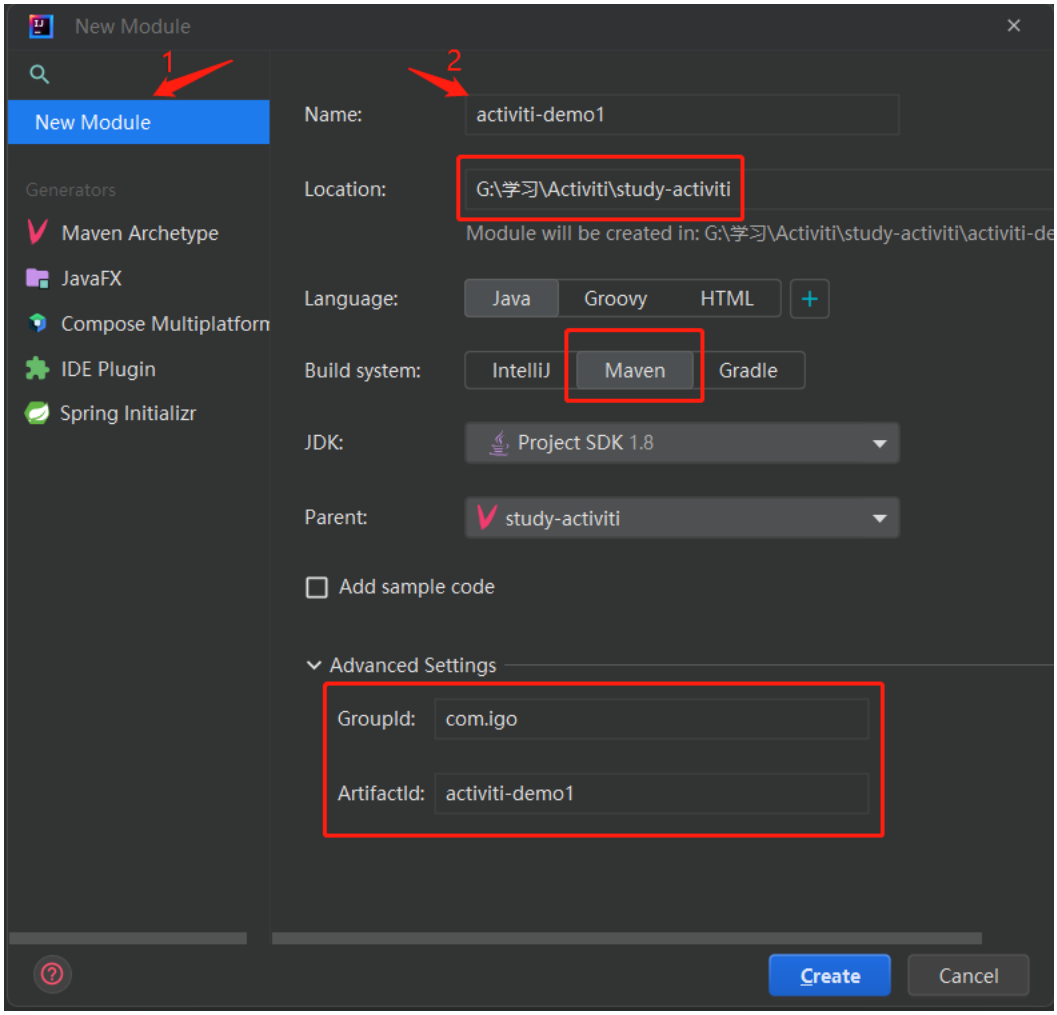
2.2.1 构建项目

1. 使用idea创建一个空工程：`study-activiti`



2. 创建基于maven的项目，项目名：`activiti-demo1`





3. pom.xml添加依赖坐标

参考： **https://www.activiti.org/userguide/#getting.started.including.libs**

最新版本activiti： **https://search.maven.org/search?q=activiti-dependencies**

在pom.xml中添加的依赖：

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

    <groupId>com.igo</groupId>
    <artifactId>activiti-demo1</artifactId>
    <version>1.0-SNAPSHOT</version>

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

    <dependencies>
        <!-- activiti核心依赖 -->
        <dependency>
            <groupId>org.activiti</groupId>
            <artifactId>activiti-engine</artifactId>
            <version>7.1.0.M6</version>
```

```
        </dependency>
        <!-- mysql驱动 -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.29</version>
        </dependency>
        <!-- mybatis -->
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
            <version>3.5.10</version>
        </dependency>
        <!-- 日志 -->
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-log4j12</artifactId>
            <version>1.7.36</version>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>jcl-over-slf4j</artifactId>
            <version>1.7.36</version>
        </dependency>
        <!-- 单元测试 -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13.2</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>commons-io</groupId>
            <artifactId>commons-io</artifactId>
            <version>2.11.0</version>
        </dependency>
    </dependencies>
</project>
```

4. activiti.cfg.xml核心配置文件

   Activiti流程引擎通过名为XML文件进行配置activiti.cfg.xml

   在resources目录下创建activiti.cfg.xml文件

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

    <!-- 默认方式下bean的id值必须是processEngineConfiguration -->
    <bean id="processEngineConfiguration"

 class="org.activiti.engine.impl.cfg.StandaloneProcessEngineConfiguration">
        <!-- 配置数据库 -->
```

```
        <property name="jdbcUrl"
value="jdbc:mysql://192.168.80.129:3306/activiti01?
useUnicode=true&amp;nullCatalogMeansCurrent=true&amp;characterEncoding=UTF-
8&amp;userSSL=false&amp;serverTimezone=GMT%2B8" />
        <property name="jdbcDriver" value="com.mysql.cj.jdbc.Driver" />
        <property name="jdbcUsername" value="root" />
        <property name="jdbcPassword" value="123456" />

        <!-- activiti数据库表生成策略 -->
        <!--
            自动更新数据库结构
            true：使用开发环境，默认值。activiti会对数据库中所有表进行更新操作。如果表
不存在，则自动创建
            false：使用生产环境。activiti在启动时，对比数据表中保存的版本，如果没有表
或者版本不匹配，将抛出异常
            create_drop：在activiti启动时创建表，在关闭时删除表（必须手动关闭引擎，才
能删除表）
            drop_create：在activiti启动时删除原来的旧表，然后在创建新表（不需要手动关
闭引擎）
        -->
        <property name="databaseSchemaUpdate" value="true" />
    </bean>
</beans>
```

> 1、防止插入中文数据乱码，要加上字符集 `characterEncoding=UTF-8`
>
> 2、如果报错：`Table 'activiti01.act_act_ge_property' doesn't exist`，jdbcUrl后面加上 `nullCatalogMeansCurrent=true`

5. log4j.properties配置日志

   如果想在控制台打印执行的sql日志，需要配置日志输出格式等，在resources目录下创建log4j.properties文件。

```
log4j.rootCategory=debug, CONSOLE, LOGFILE
log4j.logger.org.apache.axis.enterprise=FATAL, CONSOLE

log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender
log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout
log4j.appender.CONSOLE.layout.ConversionPattern=%d{HH:mm:ss.SSS} %p [%t]
%C.%M(%L) | %m%n

# LOGFILE is set to be a File appender using a PatternLayout.
log4j.appender.LOGFILE=org.apache.log4j.FileAppender
log4j.appender.LOGFILE.File=G:/logs/activiti.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
```

6. 创建ProcessEngine流程引擎实例和数据表

   加载类路径上的activiti.cfg.xml，并根据该文件中的配置构造一个流程引擎和创建数据表，方式如下：

```
package com.igo.test;
```

```
import org.activiti.engine.ProcessEngine;
import org.activiti.engine.ProcessEngineConfiguration;
import org.activiti.engine.ProcessEngines;
import org.junit.Test;

public class ActivitiTest01 {
    /** 创建ProcessEngine流程引擎，自动创建activiti数据表 */
    @Test
    public void getProcessEngine() {
        /**
         * 方式一：使用activiti提供的工具类ProcessEngines，
         * 调用getDefaultProcessEngine会默认读取resource下的activiti.cfg.xml文
件，
         * 并创建Activiti流程引擎和创建数据库表
         */
        ProcessEngine processEngine =
ProcessEngines.getDefaultProcessEngine();
        System.out.println(processEngine);

        // 1.等同于方式一
        // ProcessEngineConfiguration configuration =
ProcessEngineConfiguration.createProcessEngineConfigurationFromResourceDefau
lt();
        // 2.自定义配置路径和文件名，但流程引擎bean的id要等于
processEngineConfiguration
        // ProcessEngineConfiguration configuration =
ProcessEngineConfiguration.createProcessEngineConfigurationFromResource("act
iviti.cfg.xml");
        // 3.自定义：配置路径、文件名和流程引擎bean的id
        // ProcessEngineConfiguration configuration =
ProcessEngineConfiguration.createProcessEngineConfigurationFromResource("act
iviti.cfg.xml", "processEngineConfiguration");
        // ProcessEngine processEngine = configuration.buildProcessEngine();
    }
}
```

2.2.2 Activiti 25张数据表分析

- 数据库表命名：Activiti数据库中表的命名都是以 `act_` 开头，第二部分是一个两个字符用例表的标识。此用例大体与服务API是匹配的。
- 第二部分字符说明：
  - `act_ge_*`： `ge` 表示general，通用数据，各种情况都使用的数据，如存放资源文件（图片、规则等）
  - `act_hi_*`： `hi` 表示history，就是这些表包含着历史的相关数据，如结束的流程实例（变量、任务等）
  - `act_re_*`： `re` 表示repository，带此前缀的表包含的是静态信息，如流程定义、流程的资源（图片、规则等）。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致很小速度很快
  - `act_ru_*`： `ru` 表示runtime，这是运行时的表存储着流程变量，用户任务、变量、职责等运行时的数据。Activiti只存储实例执行期间的运行时数据，当流程实例结束时，将删除这些记录。这就保证了这些运行时的表小且快

    - `act_evt_*`： `evt`表示EVENT，流程引擎的通用事件日志记录表，方便管理员跟踪处理
- 详细说明：

表分类	表名	说明

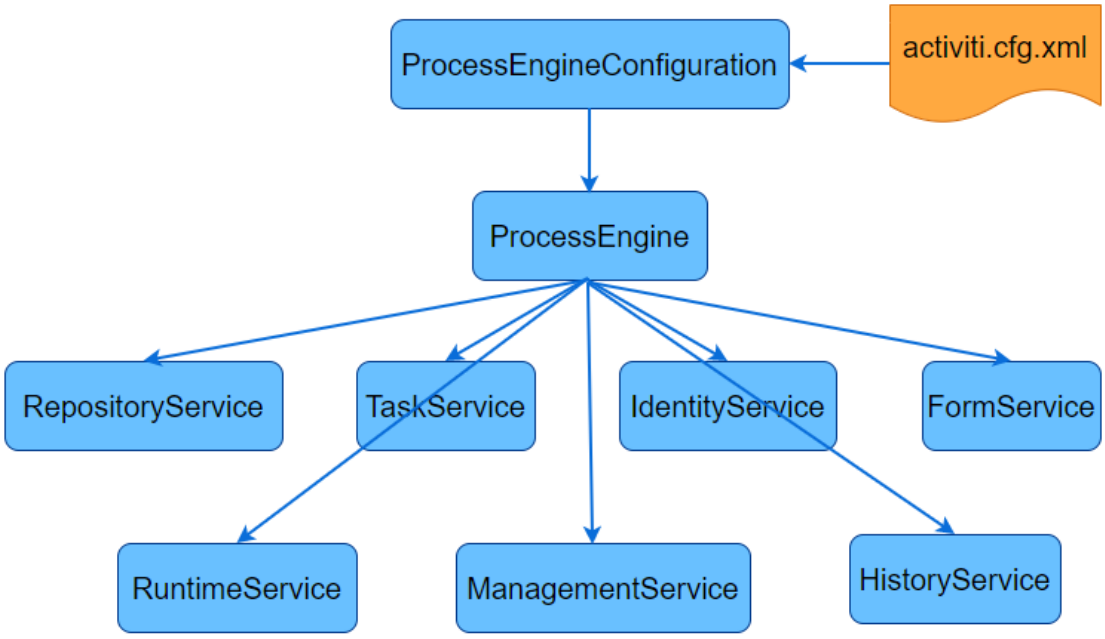
表分类	表名	说明
通用数据		
	act_ge_bytearray	二进制数据表（流程图）
	act_ge_property	属性数据表，存储整个流程引擎级别的数据，初始化表结构时，会插入版本号信息等
历史数据		
	act_hi_actinst	历史节点表
	act_hi_attachment	历史附件表
	act_hi_comment	历史意见表
	act_hi_detail	历史详情表，提供历史变量的查询
	act_hi_identitylink	历史流程人员表，主要存储任务节点与参与者的相关信息
	act_hi_procinst	历史流程实例表
	act_hi_taskinst	历史任务实例表
	act_hi_varinst	历史变量表
流程定义表		
	act_re_deployment	部署信息表
	act_re_model	流程设计模型表
	act_re_procdef	流程定义数据表
流程运行数据表		
	act_ru_deadletter_job	作业死亡信息表，如果作业失败超过重试次数，则写入到此表
	act_ru_event_subscr	throwEvent、chartEvent时间监听信息表
	act_ru_execution	运行时流程执行实例表
	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
	act_ru_integration	运行时积分表
	act_ru_job	定时异步任务数据表

表分类	表名	说明
	act_ru_suspended_job	运行时作业暂停表，比如流程中有一个定时任务，如果把这个任务停止工作了，这个任务写入到此表中
	act_ru_task	运行时任务节点表
	act_ru_timer_job	运行时定时器作业表
	act_ru_variable	运行时流程变量数据表
其他表		
	act_procdef_info	流程定义的动态变更信息
	act_evt_log	流程引擎的通用事件日志记录表

> Activiti7的M4以上版本，部署流程定义时，报错如下：
>
> 报错：`MySQLSyntaxErrorException: Unknown column 'VERSION_' in 'field list'`
>
> 解决：因为 `act_re_deployment` 表缺少 `VERSION_` 和 `PROJECT_RELEASE_VERSION_` 字段，执行一下语句添加：
>
> ```
> ALTER TABLE ACT_RE_DEPLOYMENT ADD COLUMN VERSION_ VARCHAR(255);
> ALTER TABLE ACT_RE_DEPLOYMENT ADD COLUMN PROJECT_RELEASE_VERSION_
> VARCHAR(255);
> ```

2.3 Activiti API服务接口

2.3.1 Process Engine API和服务



> - ProcessEngine和服务对象是线程安全的
> - activiti7没有IdentityService和FormService接口

2.3.2 Activiti7的Service核心接口

1. Service管理接口说明

Service接口	说明
RuntimeService	运行时 Service，可以处理所有正在运行状态的流程实例和任务等
RepositoryService	流程仓库 Service，主要用于管理流程仓库，比如流程定义的控制管理（部 署、删除、挂起、激活....）
DynamicBpmnService	RepositoryService可以用来部署流程定义（使用xml形式定义好的），一旦 部署到Activiti（解析后保存到DB），那么流程定义就不会再变了，除了修改 xml定义文件内容；而DynamicBpmnService就允许我们在程序运行过程中 去修改流程定义，例如：修改流程定义中的分配角色、优先级、流程流转的 条件...
TaskService	任务 Service，用于管理和查询任务，例如：签收、办理等
HistoryService	历史 Service，可以查询所有历史数据，例如：流程实例信息、参与者信 息、完成时间...
ManagementService	引擎管理Service，和具体业务无关，主要用于对Activiti流程引擎的管理和维 护

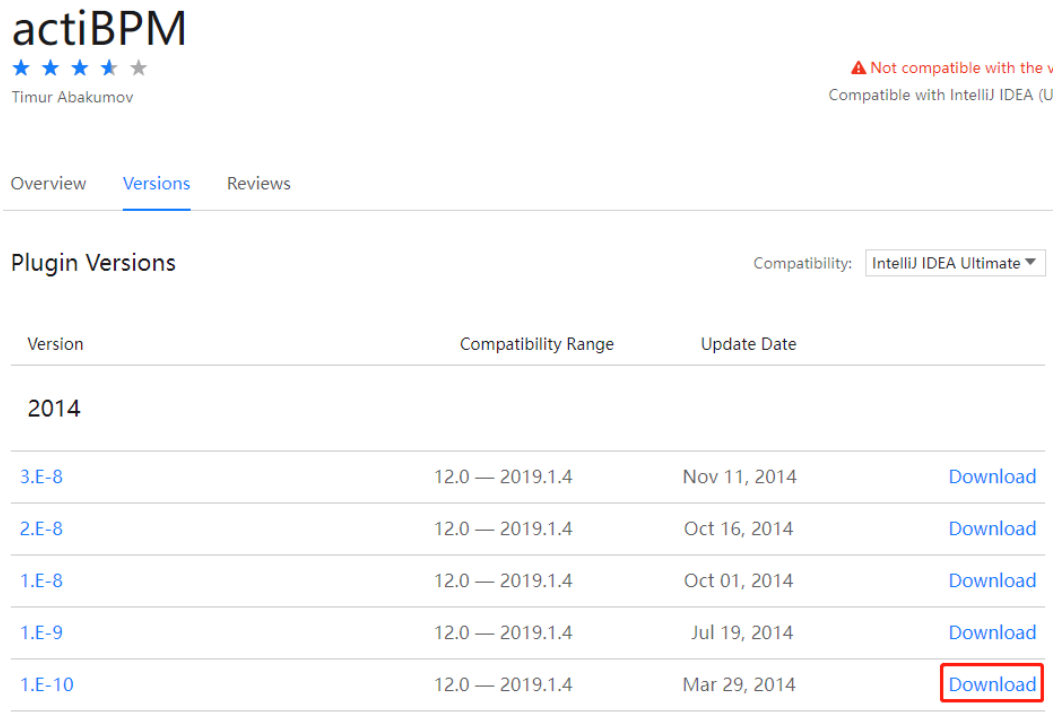
2. 核心Service接口实例获取方式如下：

```
// 动态修改流程管理类
DynamicBpmnService dynamicBpmnService =
processEngine.getDynamicBpmnService();

// 流程运行管理类
RuntimeService runtimeService = processEngine.getRuntimeService();

// 流程仓库管理类
RepositoryService repositoryService = processEngine.getRepositoryService();

// 任务管理类
TaskService taskService = processEngine.getTaskService();

// 历史管理类
HistoryService historyService = processEngine.getHistoryService();

// activiti 7 没有IdentityService和FormService接口
// IdentityService identityService = processEngine.getIdentityService();
// FormService formService = processEngine.getFormService();
```

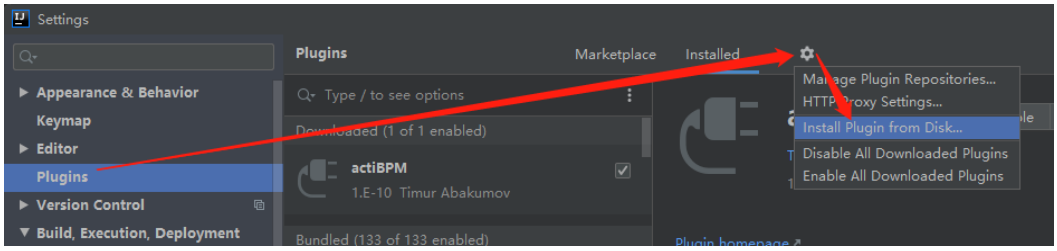
三、Activiti流程实操入门

3.1 IDEA安装actiBPM插件

1. IDEA2019版本搜索不到actiBPM，需要从idea官网插件库手动下载：**https://plugins.jetbrains.com/plugin/7429-actibpm/versions**



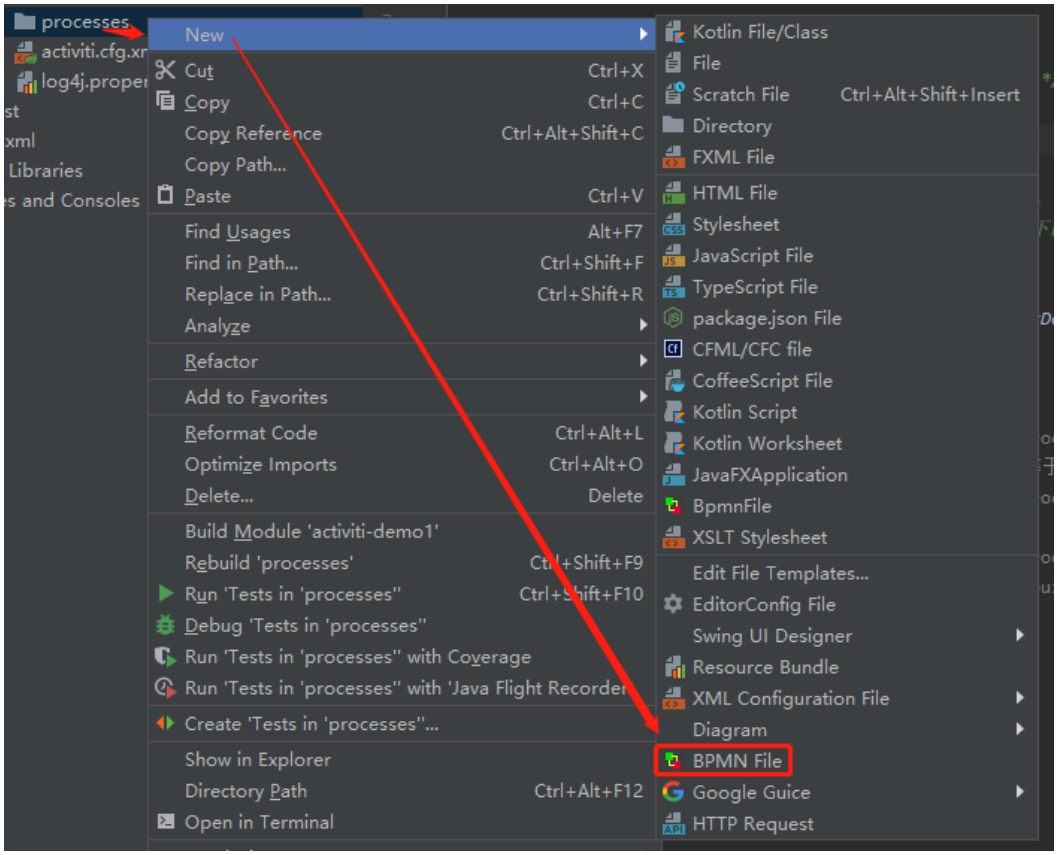
2. 指定本地actBPM.jar插件进行安装，如下从本地安装插件



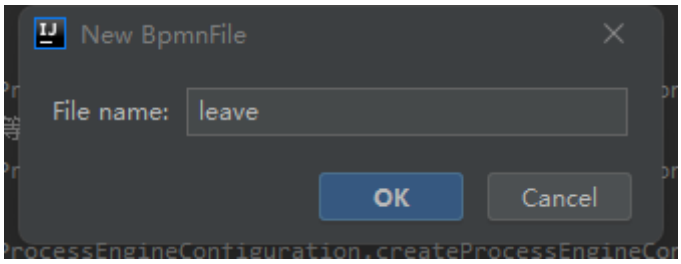
3. 点击Restart IDE重启idea

3.2 绘制流程定义模型

1. 在/resources目录下创建processes目录，用于存放流程图
2. 创建BPMN文件：右击processes目录，点击【New】-->【BpmnFile】



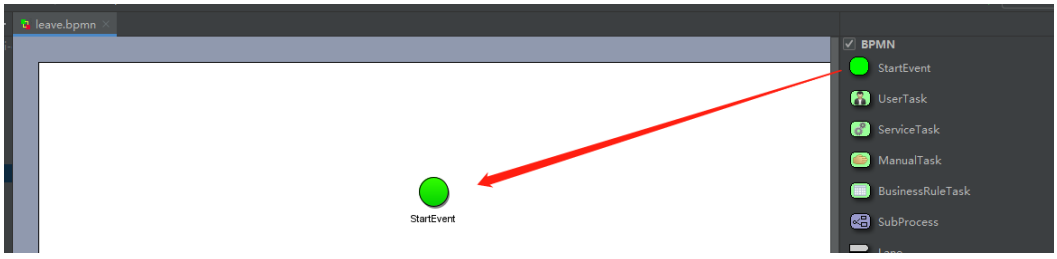
3. 输入文件名称 `leave` ，点击【OK】按钮



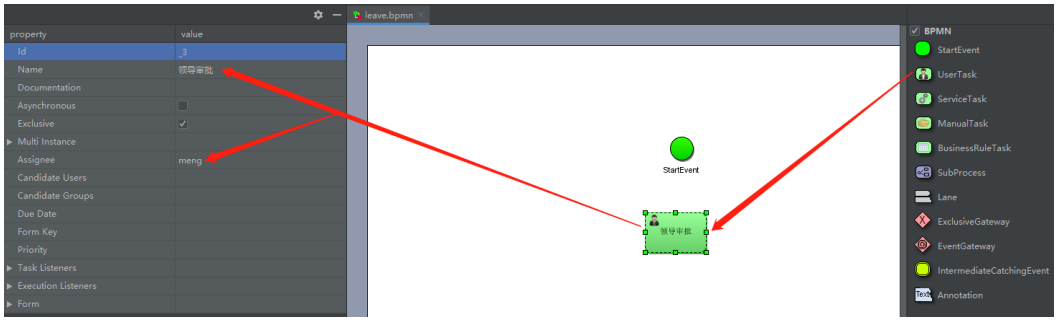
3.3 绘制请假申请流程图

将xxx.bpmn文件放在/resources/processes/目录下，即/resources/processes/leave.bpmn

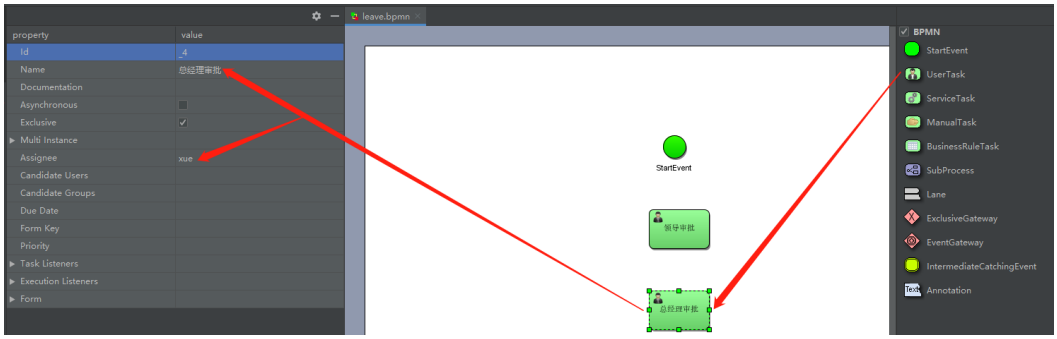
1. 鼠标左键拖拽右侧**开始事件**符号，将其拖下左侧界面上，同样的方式在拖拽其他图标



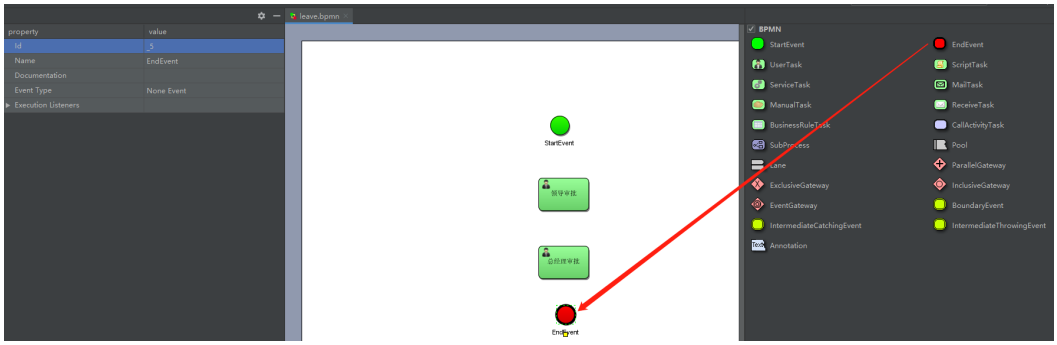
2. 添加人工任务符号，修改Name节点名称“领导审批”，和Assignee节点处理：meng



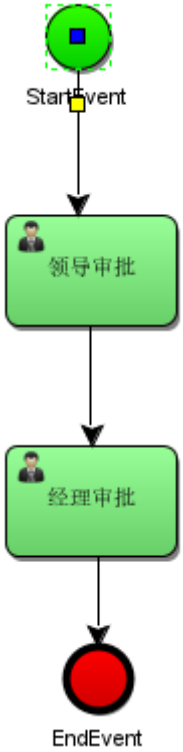
3. 添加**人工任务**符号：修改 Name 节点名称：总经理审批，和 Assignee 节点处理人：xue



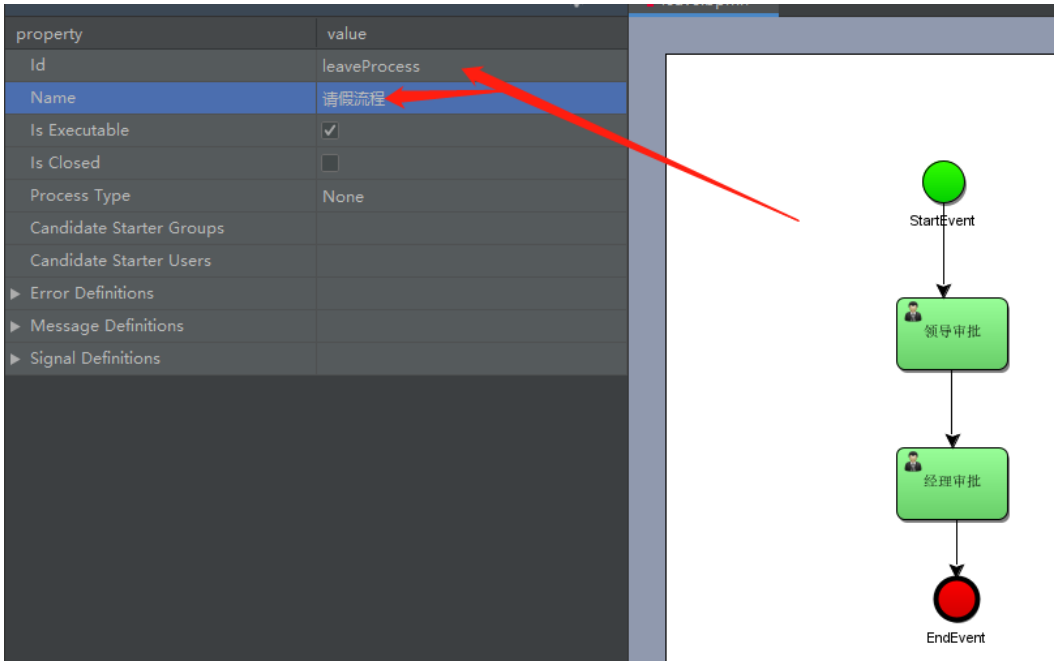
4. 添加**结束事件**符号



5. 流程连线：点击图标的中心，会变成方块，拖拽到另一图标，即可连接

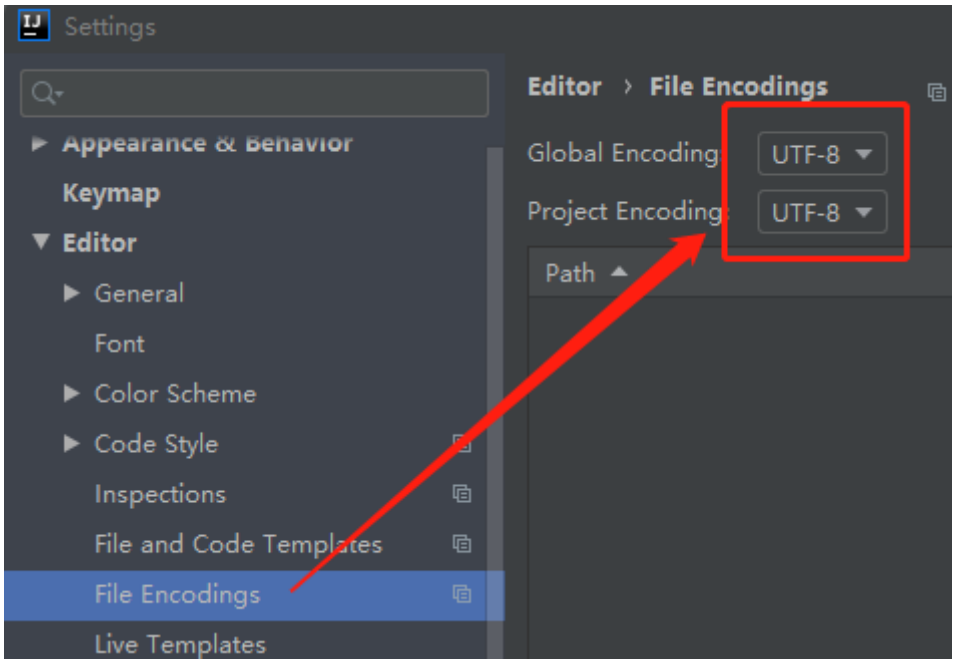


6. 点击流程图的空白处（不是点击流程符号）：设置当前流程唯一标签ID（也称为：KEY）：leaveProcess ，流程名称：请假流程。



3.4 解决中文乱码问题

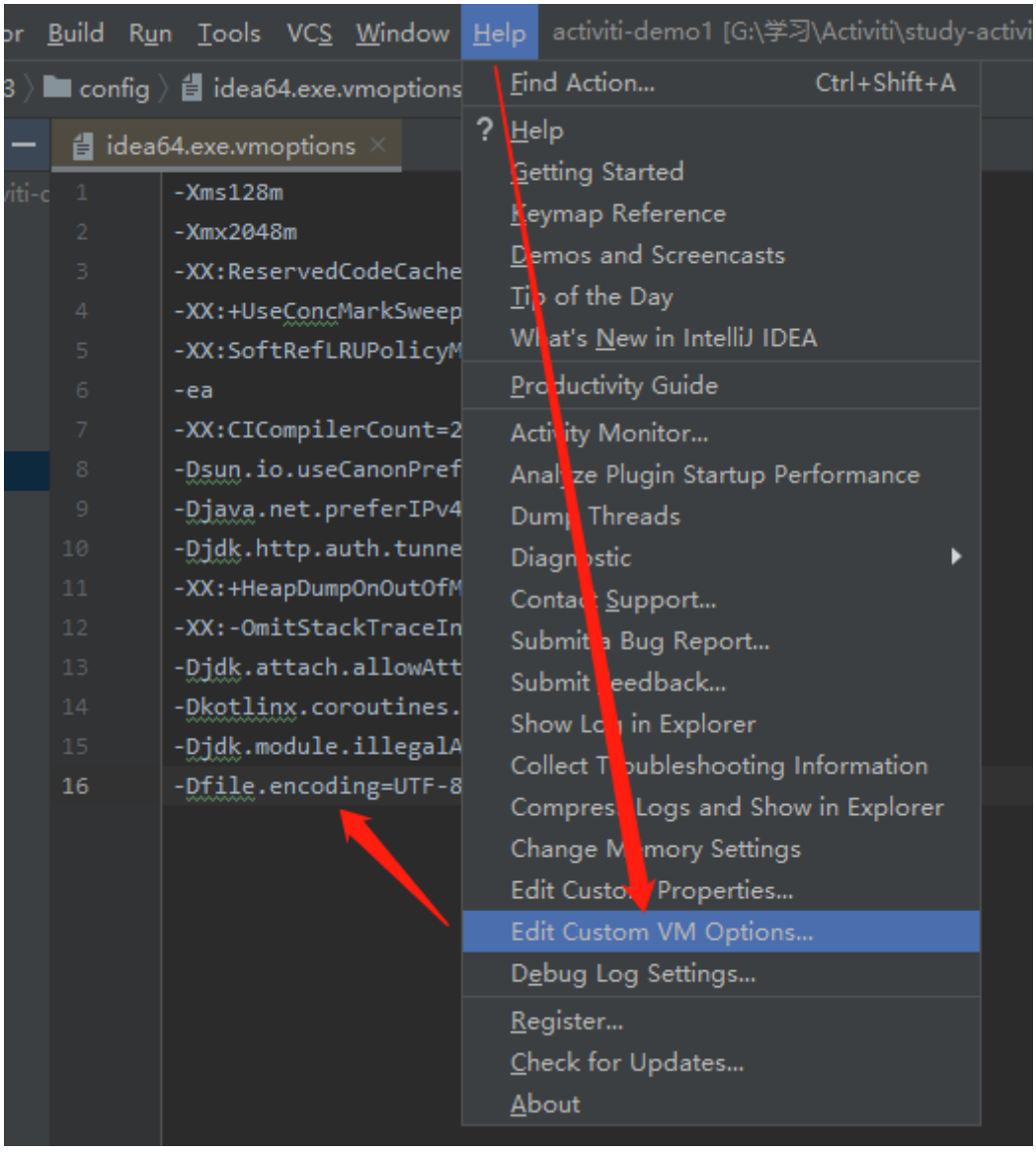
1. 打开Setting窗口，点击Editor > File Encodings > 将字符编码均设置为UTF-8



2. 打开idea安装目录，在 idea.exe.vmoptions （32位系统）和 idea64.exe.vmoptions （64位系统）文件最后追加一行命令

```
-Dfile.encoding=UTF-8
```

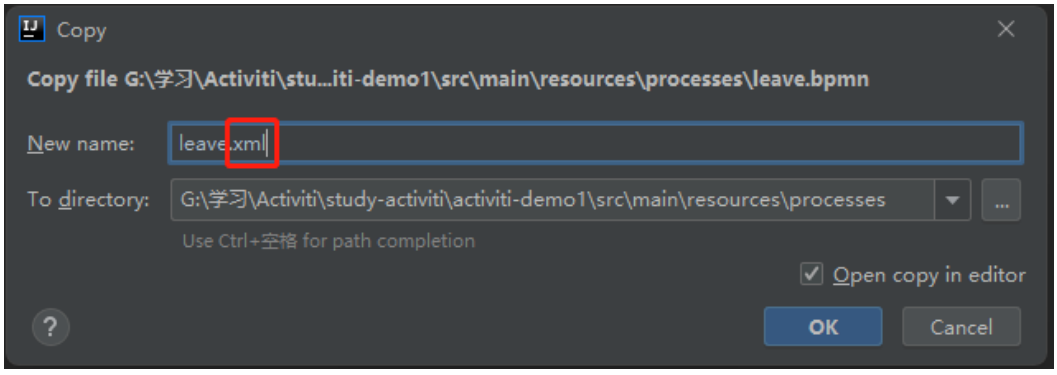
3. 如果上面修改后，重启idea重新打开bpmn文件，还是乱码，则：



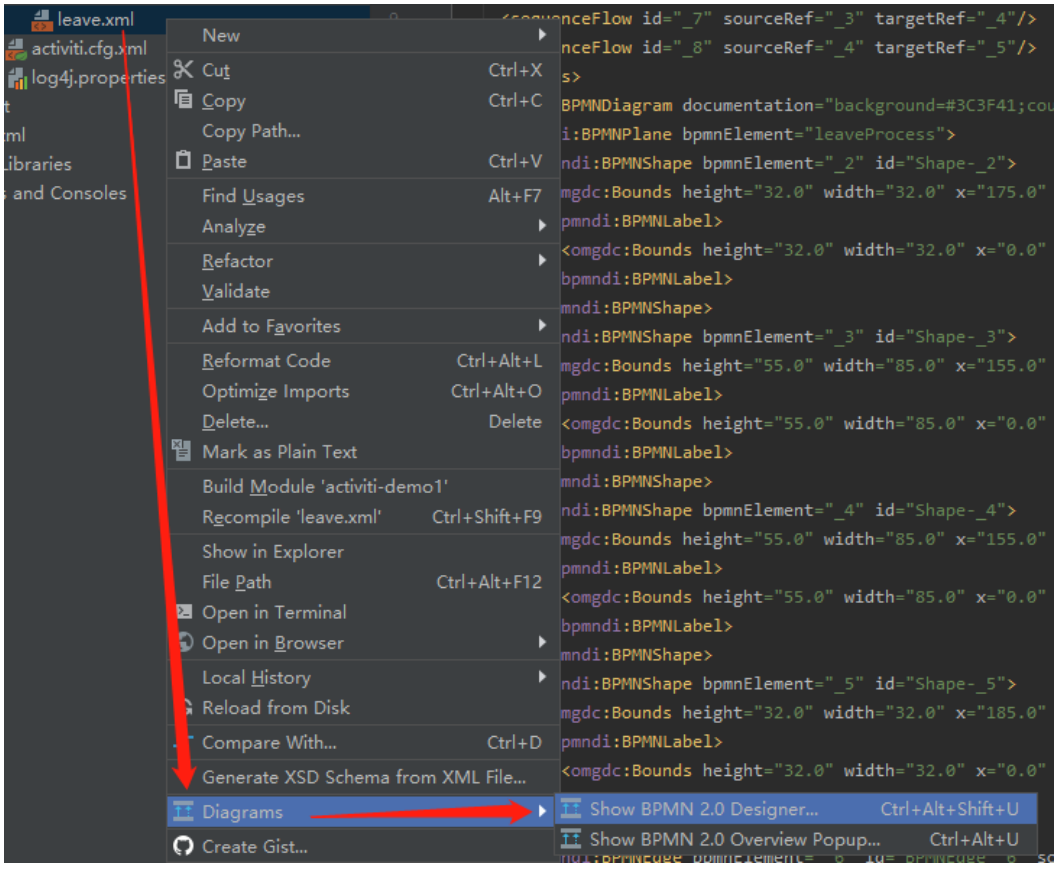
3.5 生成png图片

将 bpmn 文件不方便随处打开提供给用户看，为了方便将流程图转成图片格式，方便随处打开演示；并在部署流程时也需要png图片。

1. 将 leave.bpmn 文件复制一份为 leave.xml

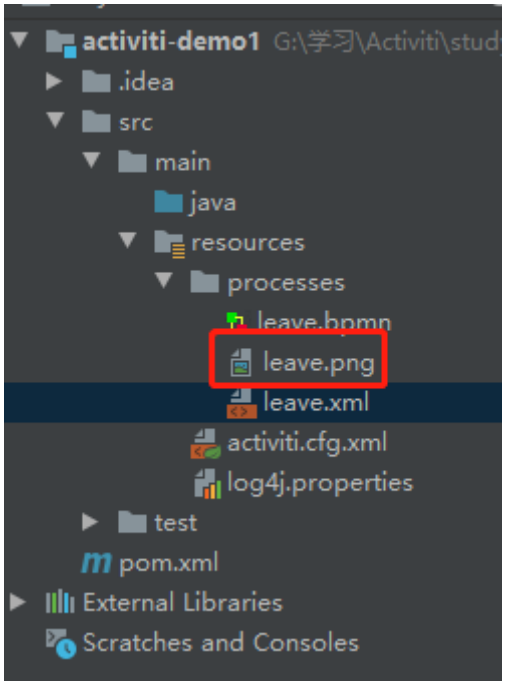


2. 右击 leave.xml 文件，选择【Diagrams】-->【Show BPMN 2.0 Diagrams...】



3. 打开后，点击上方导出按钮，保存即可

4. 将生成的 leave.png 图片拷贝到 resources/processes 目录下



3.6 部署流程定义

将上面在设计器中定义的流程部署到activiti数据库中，就是流程定义部署。通过调用activiti的api将流程定义的 .bpm 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。

3.6.1 .bpmn流程定义文件部署

```
/**
 * 部署流程：
 * 1.ACT_RE_DEPLOYMENT 流程部署表，每执行一次部署，会插入一条数据
 * 2.ACT_RE_PROCDEF 生成流程定义信息
 *  ID组成：流程唯一标识key:版本号:随机标识（如leaveProcess:1:4）
 *  每次部署，针对相同的流程定义key，对应的version会自增1
 *  其中ACT_RE_DEPLOYMENT与ACT_RE_PROCDEF表示一对多的关系，
 *  ACT_RE_PROCDEF每条记录对应一个流程的定义信息(如：小梦、小谷请假申请）
 * 3.ACT_GE_BYTEARRAY 流程资源表，插入资源数据，当前插入两条记录(.bpmn和.png资源)
 */
@Test
public void deployByFile() {
    // 1.实例化流程引擎实例
    ProcessEngine processEngine = ProcessEngines.getDefaultProcessEngine();

    // 2.获取流程定义和部署对象相关的Service
    RepositoryService repositoryService = processEngine.getRepositoryService();

    // 3.创建部署对象进行流程的部署，定义一个流程的名字，把.bpmn和.png部署到数据库中
    Deployment deployment = repositoryService.createDeployment()
        .name("请假申请流程")
        .addClasspathResource("processes/leave.bpmn")
        .addClasspathResource("processes/leave.png")
        .deploy();

    // 4.输出部署信息
    System.out.println("部署ID: " + deployment.getId());
    System.out.println("部署名称: " + deployment.getName());
```

```
}
```

3.6.2 .zip流程定义压缩部署包

```
/** 通过zip压缩包部署流程定义 */
@Test
public void deployByZip() {
    // 1.实例化流程引擎实例
    ProcessEngine processEngine = ProcessEngines.getDefaultProcessEngine();

    // 2.获取流程定义和部署对象相关的Service
    RepositoryService repositoryService = processEngine.getRepositoryService();

    // 3.流程部署
    // 读取zip资源包，构成InputStream输入流
    InputStream inputStream =
ReflectUtil.getResourceAsStream("processes/leave.zip");
    // 封装ZipInputStream输入流进行流程部署
    ZipInputStream zipInputStream = new ZipInputStream(inputStream);
    Deployment deployment = repositoryService.createDeployment()
        .addZipInputStream(zipInputStream)
        .name("请假申请流程-压缩包")
        .deploy();

    // 4.输出部署信息
    System.out.println("部署ID: " + deployment.getId());
    System.out.println("部署名称: " + deployment.getName());
}
```

3.6.4 部署涉及的数据表

- act_re_deployment 流程定义部署表，每部署一次增加一条记录
- act_re_procdef 流程定义表，部署每个新的流程定义都会在这张表中增加一条记录
  - 注意：表中字段 KEY 是不同流程定义的唯一标识
- act_ge_bytearray 流程资源表，bpmn的 xml 串和 流程图片,每次增加对应资源数据

> act_re_deployment 和 act_re_procdef 一对多关系，一次部署在流程部署表生成一条记录，但一次部署可以部署多个流程定义，每个流程定义在流程定义表生成一条记录。每一个流程定义在 act_ge_bytearray 会存在两个资源记录，bpmn 和 png
>
> **建议**：一次部署一个流程，这样部署表和流程定义表是一对一有关系，方便读取流程部署及流程定义信息

3.6.5 查询流程定义

查询部署的流程定义数据 ACT_RE_PROCDEF

```
/** 查询部署的流程定义数据 */
@Test
public void getProcessDefinitionList() {
    // 1.实例化流程引擎实例
    ProcessEngine processEngine = ProcessEngines.getDefaultProcessEngine();

    // 2.获取流程定义和部署对象相关的Service
```

```
    RepositoryService repositoryService = processEngine.getRepositoryService();

    // 3.获取ProcessDefinitionQuery
    ProcessDefinitionQuery query =
repositoryService.createProcessDefinitionQuery();
    List<ProcessDefinition> definitionList =
query.processDefinitionKey("leaveProcess")
        .orderByProcessDefinitionVersion()  // 按版本号排序
        .desc()  // 降序
        .list();

    for (ProcessDefinition pd : definitionList) {
        System.out.println("流程部署ID: " + pd.getDeploymentId());
        System.out.println("流程定义ID: " + pd.getId());
        System.out.println("流程定义Key: " + pd.getKey());
        System.out.println("流程定义名称: " + pd.getName());
        System.out.println("流程定义版本号: " + pd.getVersion());
    }
}
```

```
程部署ID: 1
流程定义ID: leaveProcess:1:4
流程定义Key: leaveProcess
流程定义名称: 请假流程
流程定义版本号: 1
```

3.6.6 启动流程实例(提交申请)

流程定义部署后，然后可以通过 activiti 工作流管理业务流程了。例如上面部署好了请假流程，可以申请请假了。针对部署好的流程定义，每次用户发起一个新的请假申请，就对应的启动一个新的请假流程实例；类似于 java 类与 java 对象（实例）的关系，定义好类后，使用 new 创建一个对象（实例）使用，当然可以 new多个对象（实例）。
例如：请假流程，小梦发起一个请假申请单，对应的就启动一个针对小梦的新流程实例；小王发起一个请假申请单，也启动针对小王的新流程实例。

```
/** 启动流程实例（提交申请） */
@Test
public void startProcessInstance() {
    // 1.实例化流程引擎实例
    ProcessEngine processEngine = ProcessEngines.getDefaultProcessEngine();

    // 2.获取RuntimeService
    RuntimeService runtimeService = processEngine.getRuntimeService();

    // 3.开启流程实例（流程设计图唯一标识key)
    ProcessInstance processInstance =
runtimeService.startProcessInstanceByKey("leaveProcess");

    System.out.println("流程定义id: " + processInstance.getProcessDefinitionId());
    System.out.println("流程实例id: " + processInstance.getId());
}
```

```
流程定义id: leaveProcess:1:4
流程实例id: 2501
```

3.6.7 流程实例涉及的数据表

- act_hi_actinst 流程实例执行的节点历史信息
- act_hi_identitylink 流程的参与用户历史信息
- act_hi_procinst 流程实例历史信息
- act_hi_taskinst 流程实例的任务历史信息
- act_ru_execution 流程运行中执行信息
- act_ru_identitylink 流程运行中参与用户信息
- act_ru_task 流程运行中任务信息

3.6.8 查询办理人待办任务

```
/** 查询指定人员的待办任务 */
@Test
public void taskListByAssignee() {
    // 1.实例化流程引擎实例
    ProcessEngine processEngine = ProcessEngines.getDefaultProcessEngine();

    // 2.获取TaskService
    TaskService taskService = processEngine.getTaskService();

    // 3.根据流程唯一标识key和任务办理人查询任务
    List<Task> taskList = taskService.createTaskQuery()
        .processDefinitionKey("leaveProcess")
        .taskAssignee("meng") // 查询meng的任务
        .list();

    for (Task task : taskList) {
        System.out.println("流程实例id: " + task.getProcessInstanceId());
        System.out.println("任务id: " + task.getId());
        System.out.println("任务名称: " + task.getName());
        System.out.println("任务办理人: " + task.getAssignee());
    }
}
```

```
流程实例id: 2501
任务id: 2505
任务负责人: meng
任务名称: 领导审批
```

3.6.9 完成任务

1. 先查询 meng 办理人任务，然后完成任务
2. 再查询 xue 办理人任务，然后完成任务
3. 这样此流程实例就完成结束

```
/** 完成待办任务 */
@Test
public void completeTask() {
    // 1.实例化流程引擎实例
    ProcessEngine processEngine = ProcessEngines.getDefaultProcessEngine();

    // 2.获取TaskService
    TaskService taskService = processEngine.getTaskService();
```

```
    // 3.根据流程唯一标识key和任务办理人查询任务
    Task task = taskService.createTaskQuery()
        .processDefinitionKey("leaveProcess")
        // .taskAssignee("meng") // 查询meng的任务
        .taskAssignee("xue")
        .singleResult();  // 目前只有一条任务，则可以只获取一条

    // 4.完成任务(任务id)
    taskService.complete(task.getId());
}
```

3.6.10 查询流程实例历史节点信息

查询流程办理历史信息，通过 HistoryService 历史数据对象来获取 HistoricActivityInstanceQuery 历史节点查询对象

```
/** 查看流程办理历史信息 */
@Test
public void historyInfo() {
    // 1.实例化流程引擎实例
    ProcessEngine processEngine = ProcessEngines.getDefaultProcessEngine();

    // 2.获取HistoryService
    HistoryService historyService = processEngine.getHistoryService();

    // 3.获取节点历史记录查询对象ACT_HI_ACTINST表
    HistoricActivityInstanceQuery query =
historyService.createHistoricActivityInstanceQuery();
    // 实例id，表act_hi_procinst
    String processInstanceId = "5001";
    List<HistoricActivityInstance> list =
query.processInstanceId(processInstanceId)
        .orderByHistoricActivityInstanceStartTime()  // 根据开始时间排序asc升序
        .asc()
        .list();

    for (HistoricActivityInstance hi : list) {
        System.out.print("流程定义ID: " + hi.getProcessDefinitionId());
        System.out.print(", 流程实例ID: " + hi.getProcessInstanceId());
        System.out.print(", 节点ID: " + hi.getActivityId());
        System.out.print(", 节点名称: " + hi.getActivityName());
        System.out.print(", 任务办理人: " + hi.getAssignee());
        System.out.print(", 开始时间: " + hi.getStartTime());
        System.out.println("结束时间: " + hi.getEndTime());
    }
}
```

四、SpringBoot与Activiti7整合

4.1 pom.xml依赖坐标

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.7.2</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.igo</groupId>
    <artifactId>activiti-boot</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>activiti-boot</name>
    <description>activiti-boot</description>
    <properties>
        <java.version>1.8</java.version>
        <activiti.version>7.1.0.M6</activiti.version>
        <mybatis-plus.version>3.5.2</mybatis-plus.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-test</artifactId>
            <scope>test</scope>
        </dependency>

        <!-- Activiti -->
        <dependency>
            <groupId>org.activiti</groupId>
            <artifactId>activiti-spring-boot-starter</artifactId>
            <version>${activiti.version}</version>
        </dependency>
        <!-- java代码绘activiti流程图 -->
        <dependency>
            <groupId>org.activiti</groupId>
            <artifactId>activiti-image-generator</artifactId>
            <version>${activiti.version}</version>
```

```
        </dependency>
        <!-- activiti json转化器 -->
        <dependency>
            <groupId>org.activiti</groupId>
            <artifactId>activiti-json-converter</artifactId>
            <version>${activiti.version}</version>
        </dependency>
        <!-- svn转换png图片工具 -->
        <dependency>
            <groupId>org.apache.xmlgraphics</groupId>
            <artifactId>batik-all</artifactId>
            <version>1.10</version>
        </dependency>
        <!-- mybatis-plus -->
        <dependency>
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus-boot-starter</artifactId>
            <version>${mybatis-plus.version}</version>
        </dependency>
        <!-- MySQL驱动 -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.29</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <version>2.7.2</version>
            </plugin>
        </plugins>
    </build>

</project>
```

4.2 application.yml配置文件

```
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://192.168.80.129:3306/activiti?
useUnicode=true&nullCatalogMeansCurrent=true&characterEncoding=UTF-
8&&userSSL=false&serverTimezone=GMT%2B8&allowMultiQueries=true
    username: root
    password: 123456

  # activiti配置
  activiti:
    # 自动更新数据库结构
    # true：适用开发环境，默认值。activiti会对数据库中所有表进行更新操作。如果表不存在，则自动
创建
```

```
    # false：适用生产环境。activiti在启动时，对比数据库表中保存的版本，如果没有表或版本不匹配，将抛出异常
    # create_drop：在activiti启动时创建表，在关闭时删除表（必须手动关闭引擎，才能删除表）
    # drop-create：在activiti启动时删除原来的旧表，然后在创建新表(不需要手动关闭引擎)
    database-schema-update: true
    # activiti7与springboot整合后默认不创建历史表，需要手动开启
    db-history-used: true
    # 记录历史等级，可配置的历史级别有none，activity，audit，full
    # 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级别会保护的数据外，还会保存全部的流程任务及其属性。
    # 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。
    history-level: full
    # 是否自动检查resources下的processes目录的流程定义文件
    check-process-definitions: false
    # 关闭不自动添加部署数据 SpringAutoDeployment
    # deployment-mode: never-fail

# 日志级别是debug才能显示SQL日志
logging:
  level:
    org.activiti.engine.impl.persistence.entity: debug
```

4.3 整合SpringSecurity

```
package com.igo.workflow.config;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfi
gurerAdapter;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.provisioning.InMemoryUserDetailsManager;
import org.springframework.security.provisioning.UserDetailsManager;

import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

@Configuration
public class SpringSecurityConfig extends WebSecurityConfigurerAdapter {
    private Logger logger = LoggerFactory.getLogger(SpringSecurityConfig.class);

    /** 内存UserDetailsManager */
    @Bean
    public UserDetailsService myUserDetailsService() {
```

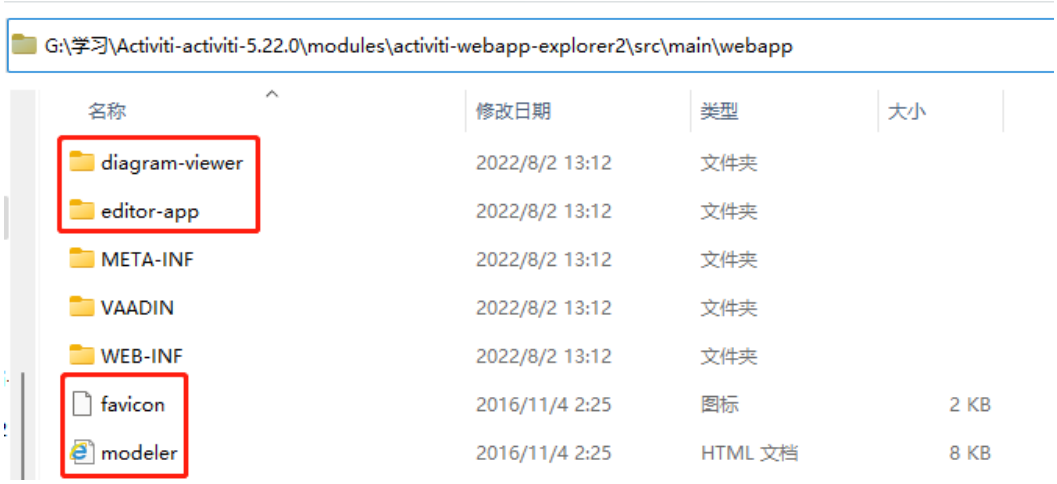
```
        InMemoryUserDetailsManager inMemoryUserDetailsManager = new
InMemoryUserDetailsManager();
        // 初始化账号角色数据
        addGroupAndRoles(inMemoryUserDetailsManager);
        return inMemoryUserDetailsManager;
    }

    private void addGroupAndRoles(UserDetailsManager userDetailsManager) {
        // 注意：后面流程办理人，必须是当前存在的用户username
        String[][] usersGroupsAndRoles = {
                {"meng", "123456", "ROLE_ACTIVITI_USER", "GROUP_activitiTeam"},
                {"xue", "123456", "ROLE_ACTIVITI_USER", "GROUP_activitiTeam"},
                {"gu", "123456", "ROLE_ACTIVITI_USER", "GROUP_activitiTeam"},
                {"小梦", "123456", "ROLE_ACTIVITI_ADMIN", "GROUP_otherTeam"},
                {"小学", "123456", "ROLE_ACTIVITI_ADMIN", "GROUP_otherTeam"},
                {"小谷", "123456", "ROLE_ACTIVITI_ADMIN", "GROUP_otherTeam"}
        };

        for (String[] user : usersGroupsAndRoles) {
            List<String> authoritiesStrings =
Arrays.asList(Arrays.copyOfRange(user, 2, user.length));
            logger.info("> Registering new user: " + user[0] + " with the
following Authorities[" + authoritiesStrings + "]");
            userDetailsManager.createUser(new User(user[0],
passwordEncoder().encode(user[1]), authoritiesStrings.stream().map(s -> new
SimpleGrantedAuthority(s)).collect(Collectors.toList())));
        }
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable();
    }

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

> activiti7 任务办理人，必须是当前可查询到的用户名

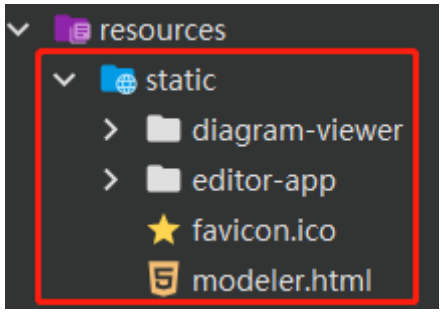
五、整合流程模型设计器Activiti Modeler

5.1 拷贝源码

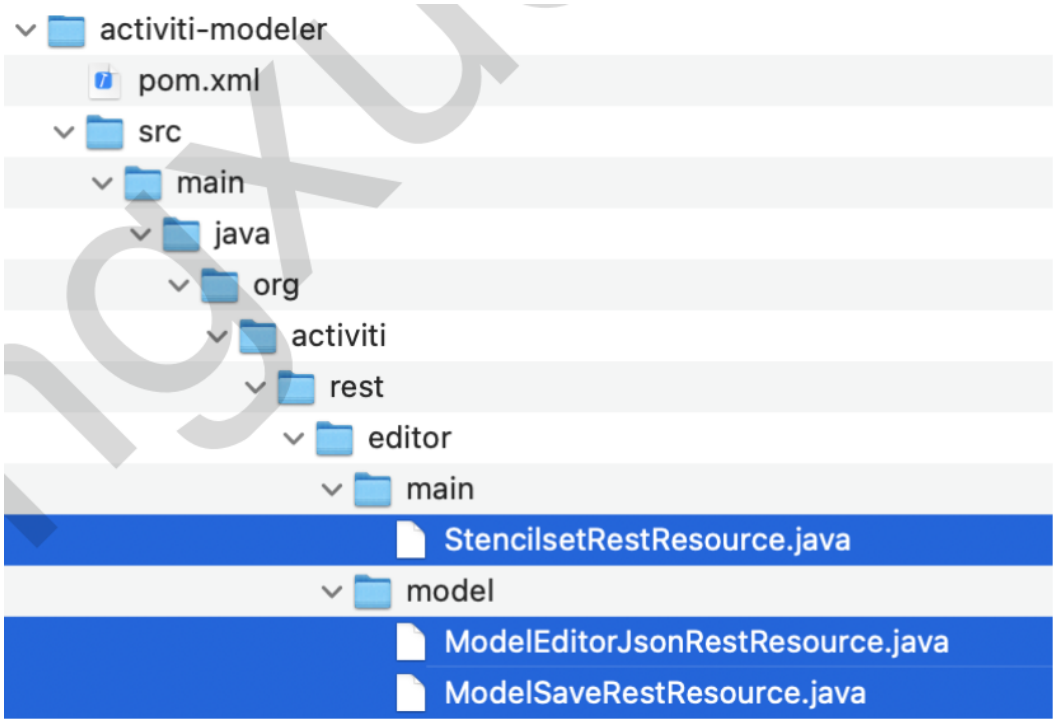
1. 下载zip格式的压缩包： **https://github.com/Activiti/Activiti/archive/activiti-5.22.0.zip**
2. 解压 Activiti-activiti-5.22.0.zip ，然后进入 Activiti-activiti-5.22.0/modules 目录
3. 复制 activiti-webapp-explorer2 工程中如下图画红框的文件夹和文件



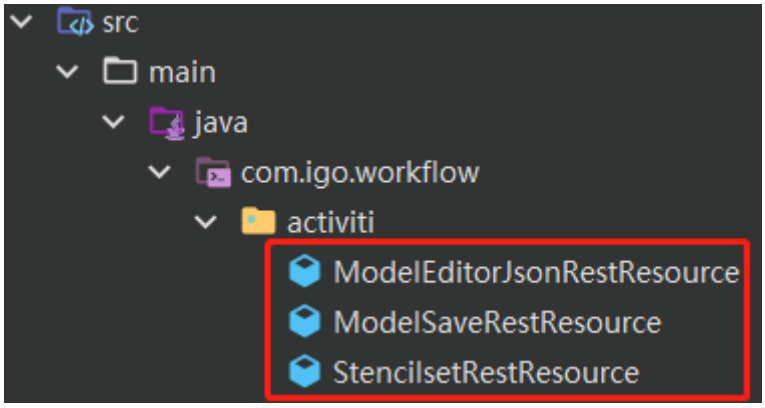
粘贴到activiti-boot工程的resources/static目录下(static自建)



4. 找到 activiti-modeler 工程下的 3个类



拷贝到 com.igo.workflow.activiti 包下面



5. 找到 static/editor-app/app-cfg.js ，修改文件中的项目上下文路径，这样才能请求到上面3个接口

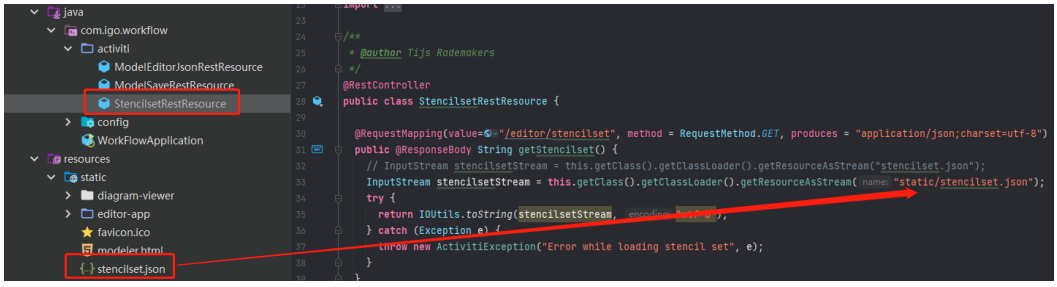
> - 上下文路径对应 application.yml 文件中配置的 server.servlet.context-path=/workflow
> - 在前端请求接口路径配置文件可参见：static/editor-app/configuration/url-config.js

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

5.2 汉化Activiti Modeler

1. 汉化页面文字，在 `resources/static/` 目录下添加 `stencilset.json` 文件
2. 需要修改 StencilsetRestResource.java 类中 stencilset.json 为 static/stencilset.json （最前面不要有 / ）

```
InputStream stencilsetStream =
this.getClass().getClassLoader().getResourceAsStream("static/stencilset.json
");
```

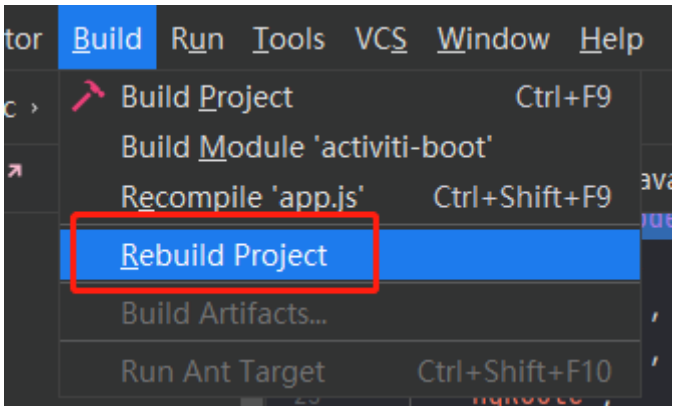


3. 在 `resources/static/editor-app/i18n` 目录下添加 `zh-CN.json` 文件

4. 修改 `resources/static/editor-app/app.js` 文件，将第51行的 `$translateProvider.preferredLanguage('en');` 替换为以下内容

```
// 支持多语言
if("zh-CN" == navigator.language) {
    $translateProvider.preferredLanguage('zh-CN');
}else {
    $translateProvider.preferredLanguage('en');
}
```

5. 如下图重新编译下，然后再启动项目



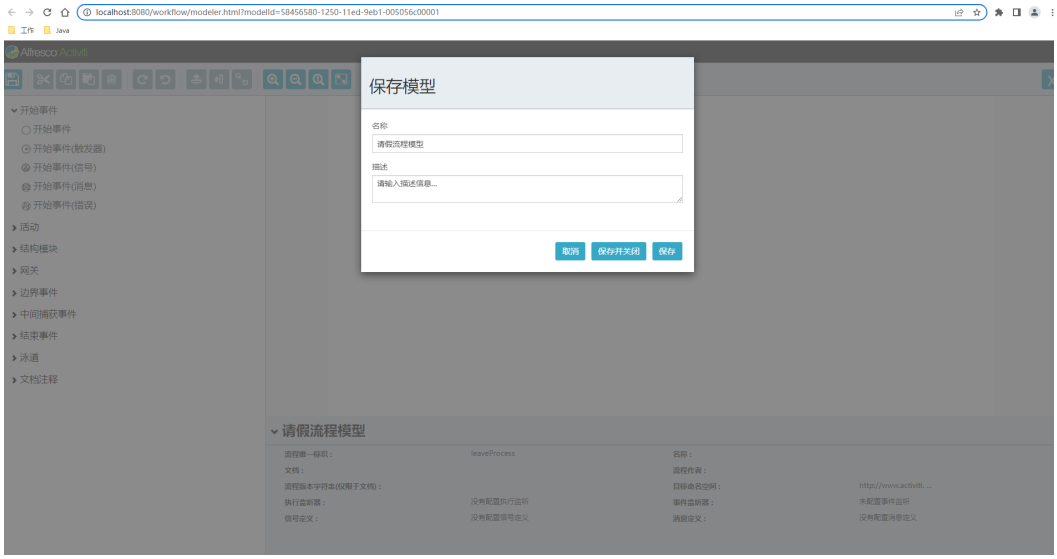
5.3 创建空的流程模型

1. 创建一个 ModelController 模型控制器，用于创建空流程模型和跳转到模型设计页面 modeler.html

```
package com.igo.workflow.controller;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.node.ObjectNode;
import org.activiti.editor.constants.ModelDataJsonConstants;
import org.activiti.engine.RepositoryService;
import org.activiti.engine.repository.Model;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * 模型设计控制层
 */
@RestController
@RequestMapping("/model")
public class ModelController {

    @Autowired
    RepositoryService repositoryService;

    @Autowired
    ObjectMapper objectMapper;
```

```
    @GetMapping("/create") // /workflow/model/create
    public void create(HttpServletRequest request, HttpServletResponse
response) {
        try {
            String name = "请假流程模型";
            String key = "leaveProcess";
            String desc = "请输入描述信息……";
            int version = 1;

            // 1. 初始空的模型
            Model model = repositoryService.newModel();
            model.setName(name);
            model.setKey(key);
            model.setVersion(version);

            // 封装模型json对象
            ObjectNode objectNode  = objectMapper.createObjectNode();
            objectNode.put(ModelDataJsonConstants.MODEL_NAME, name);
            objectNode.put(ModelDataJsonConstants.MODEL_REVISION, version);
            objectNode.put(ModelDataJsonConstants.MODEL_DESCRIPTION, desc);
            model.setMetaInfo(objectNode.toString());
            // 保存初始化的模型基本信息数据
            repositoryService.saveModel(model);

            // 封装模型对象基础数据json串
            // {"id":"canvas","resourceId":"canvas","stencilset":
{"namespace":"http://b3mn.org/stencilset/bpmn2.0#"},"properties":
{"process_id":"未定义"}}
            ObjectNode editorNode = objectMapper.createObjectNode();
            ObjectNode stencilSetNode = objectMapper.createObjectNode();
            stencilSetNode.put("namespace",
"http://b3mn.org/stencilset/bpmn2.0#");
            editorNode.replace("stencilset", stencilSetNode);
            // 标识key
            ObjectNode propertiesNode = objectMapper.createObjectNode();
            propertiesNode.put("process_id", key);
            editorNode.replace("properties", propertiesNode);

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

            response.sendRedirect(request.getContextPath() + "/modeler.html?
modelId=" + model.getId());
        }catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

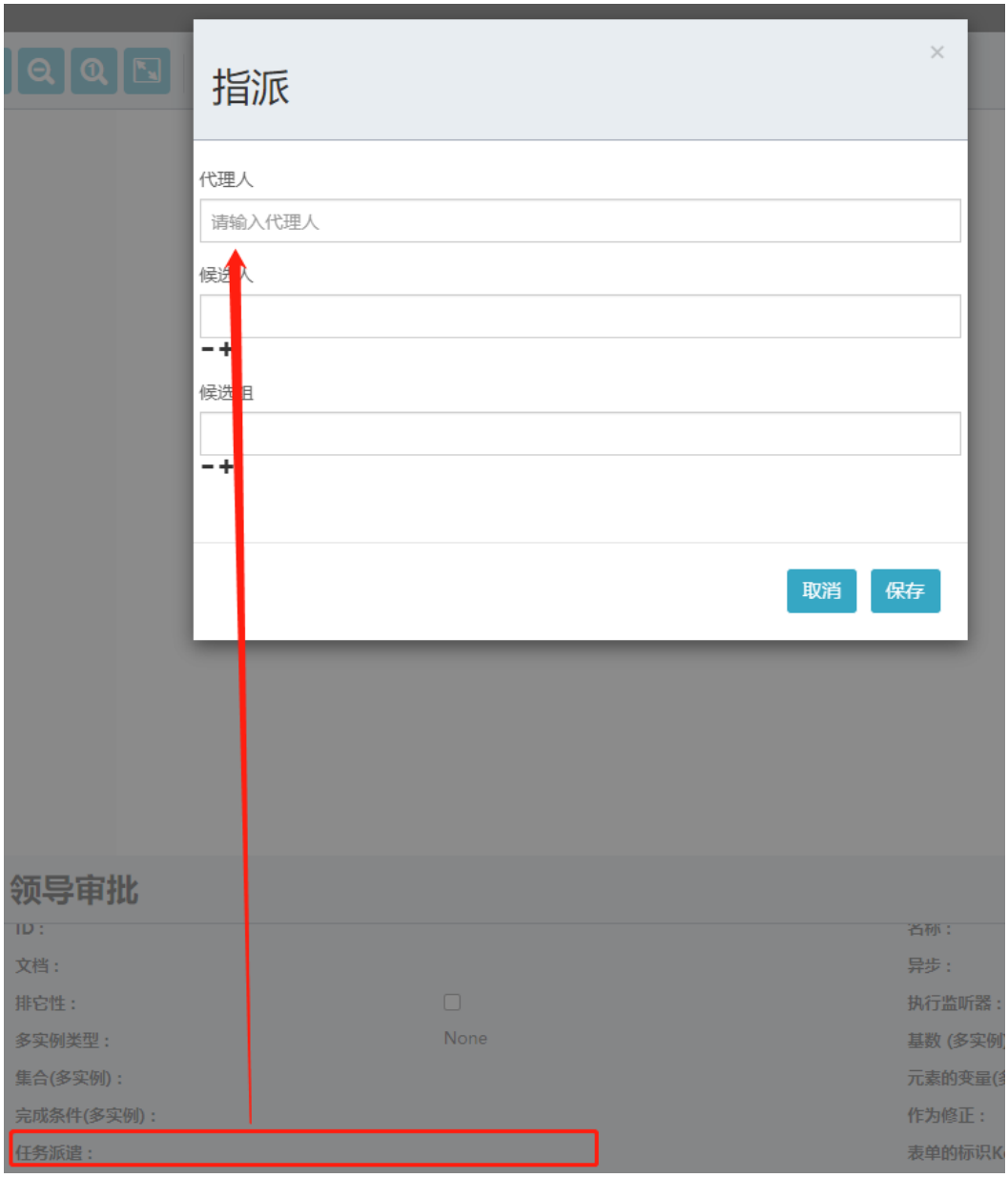
2. 重启activiti-boot项目，访问：**http://localhost:8080/workflow/model/create**



3. 绘制流程定义模型涉及表
    - ACT_RE_MODEL流程模型基本信息表
    - ACT_GE_BYTEARRAY流程模型描述json串（注意不是xml串）和流程图字节码

5.4 绘制请假流程定义模型

1. 绘制请假流程


指派
代理人
请输入代理人
候选人
候选组
取消
保存
领导审批
文档：
排它性：
多实例类型：
None
集合(多实例)：
完成条件(多实例)：
任务派遣：
异步：
执行监听器：

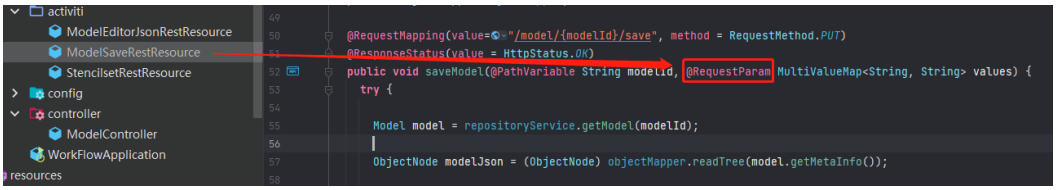
领导审批
ID：
文档：
排它性：
多实例类型：
None
集合(多实例)：
完成条件(多实例)：
任务派遣：
代理人 meng

开始事件
开始事件
开始事件(触发器)
开始事件(信号)
开始事件(消息)
开始事件(错误)
活动
人工任务
服务任务
保存模型
名称
请假流程模型
描述
取消
保存并关闭
保存

2. 上面点击保存，会报400错误

   原因：`ModelSaveRestResource`类中使用`@RequestBody MultiValueMap`接收参数无法接收到

   解决：将`@RequestBody MultiValueMap`改为`@RequestParam MultiValueMap`



六、流程定义模型管理Model

6.1 查询所有流程模型

```
@Autowired
private RepositoryService repositoryService;

/** 查询所有流程模型 ACT_RE_MODEL */
@GetMapping("/modelList")
public List<Map<String, Object>> modelList() {
    // 获取模型查询对象
    ModelQuery query = repositoryService.createModelQuery();

    // 按模型创建时间降序排序排列
    List<Model> list = query.orderByCreateTime().desc().list();

    return list.stream().map(model -> {
        Map<String, Object> map = new HashMap<>();
        map.put("模型id", model.getId());
        map.put("模型名称", model.getName());
        map.put("模型描述", model.getMetaInfo());
        map.put("模型标识key", model.getKey());
        map.put("模型版本号", model.getVersion());
        map.put("创建时间", model.getCreateTime());
        map.put("更新时间", model.getLastUpdateTime());
        return map;
    }).collect(Collectors.toList());
}
```

结果：

```
[
    {
        "模型id": "729b1dd4-1316-11ed-89b2-005056c00001",
        "模型标识key": "leaveProcess",
        "模型描述": "{\"name\":\"请假流程模型\",\"revision\":1,\"description\":\"请假流程模型描述信息……\"}",
        "模型版本号": 1,
        "创建时间": "2022-08-03T10:24:25.133+00:00",
        "模型名称": "请假流程模型",
        "更新时间": "2022-08-03T10:25:57.111+00:00"
    }
]
```

6.2 删除流程模型

```
@Autowired
private RepositoryService repositoryService;

/**
 * 删除模型：
 * 涉及表：ACT_RE_MODEL、ACT_GE_BYTEARRAY
 */
@DeleteMapping("/delete")
public String deleteModel() {
    // 模型id
    String id = "c2235252-12c7-11ed-8804-005056c00001";
    repositoryService.deleteModel(id);
    return "删除成功";
}
```

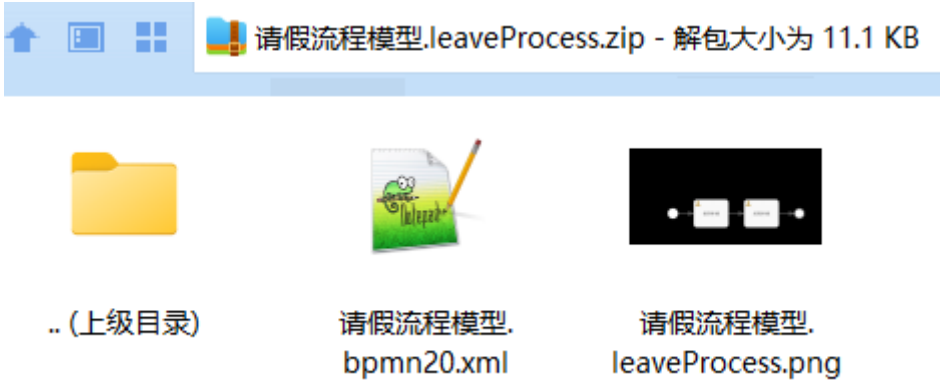
6.3 导出下载模型图zip压缩包

导出下载模型图zip压缩包，压缩包中有 `.bpmn20.xml` 流程描述和 `.png图片` 资源，在流程部署时，可以使用上传流程模型图zip压缩包进行部署

```
@Autowired
private RepositoryService repositoryService;

@Autowired
private ObjectMapper objectMapper;

/** 导出下载模型图zip压缩包(.bpmn20.xml流程描述和.png图片资源) */
@PostMapping("/exportZip")
public String exportZip() throws Exception {
    // 模型id
    String id = "729b1dd4-1316-11ed-89b2-005056c00001";

    // 查询模型信息
    Model model = repositoryService.getModel(id);
    if (model != null) {
        // 获取流程图json字节码
        byte[] bpmnJsonBytes = repositoryService.getModelEditorSource(id);
        // 流程图json字节码转xml字节码
        byte[] xmlBytes = bpmnJsonToXmlBytes(bpmnJsonBytes);
        if (xmlBytes == null) {
            return "模型数据为空，请先设计完整流程，再导出";
        } else {
            // 压缩包文件名
            String zipName = model.getName() + "." + model.getKey() + ".zip";
            // 文件输出流
            File file = new File("D:/" + zipName);
            FileOutputStream outputStream = new FileOutputStream(file);

            // 实例化zip压缩对象输出流
            ZipOutputStream zipOutputStream = new ZipOutputStream(outputStream);
```

```
            // 指定压缩包里的name.bpmn20.xml文件名
            zipOutputStream.putNextEntry(new ZipEntry(model.getName() +
".bpmn20.xml"));
            // 将xml写入压缩流
            zipOutputStream.write(xmlBytes);

            // 查询png图片
            byte[] pngBytes = repositoryService.getModelEditorSourceExtra(id);
            if (pngBytes != null) {
                // 指定压缩包里的name.key.png文件名
                zipOutputStream.putNextEntry(new ZipEntry(model.getName() + "." +
model.getKey() + ".png"));
                // 图片文件写到压缩包中
                zipOutputStream.write(pngBytes);
            }
            zipOutputStream.closeEntry();
            zipOutputStream.close();
            return "导出成功";
        }
    } else {
        return "模型不存在";
    }
}

/** 流程图保存的时候是json串，引擎认识的却是符合bpmn2.0规范的xml，json字节码转xml字节码 */
private byte[] bpmnJsonToXmlBytes(byte[] jsonBytes) throws IOException {
    if (jsonBytes == null) {
        return null;
    }
    // json转回BpmnModel对象
    JsonNode modelNode = objectMapper.readTree(jsonBytes);
    BpmnModel bpmnModel = new BpmnJsonConverter().convertToBpmnModel(modelNode);
    if (bpmnModel.getProcesses().size() == 0) {
        return null;
    }

    // BpmnModel对象转xml字节数组
    return new BpmnXMLConverter().convertToXML(bpmnModel);
}
```

压缩包如下：



6.4 导出下载模型xml文件

在流程部署时，可以只上传流程模型图xml文件进行部署

```
@Autowired
private RepositoryService repositoryService;

/** 导出下载模型xml文件 */
@PostMapping("exportXml")
public String exportXml() throws Exception {
    // 模型id
    String id = "729b1dd4-1316-11ed-89b2-005056c00001";

    ByteArrayInputStream in = null;
    // 获取流程图json字节码
    byte[] bytes = repositoryService.getModelEditorSource(id);

    // json转xml字节数组
    String fileName = null;
    if (bytes != null) {
        JsonNode modelNode = objectMapper.readTree(bytes);
        BpmnModel bpmnModel = new
BpmnJsonConverter().convertToBpmnModel(modelNode);
        if (bpmnModel.getProcesses().size() != 0) {
            // 转xml字节数组
            byte[] bpmnBytes = new BpmnXMLConverter().convertToXML(bpmnModel);
            in = new ByteArrayInputStream(bpmnBytes);
            // 如果流程名称为空，则取流程定义key
            fileName = StringUtils.isEmpty(bpmnModel.getMainProcess().getName())
? bpmnModel.getMainProcess().getId() : bpmnModel.getMainProcess().getName();
        }
    }

    if (fileName == null) {
        fileName = "模型数据为空，请先设计完整流程，再导出";
        in = new ByteArrayInputStream(fileName.getBytes(StandardCharsets.UTF_8));
    }

    // 文件输出流
    FileOutputStream out = new FileOutputStream(new File("D:/" + fileName +
".bpmn20.xml"));

    // 输入流，输出流的转换
    IOUtils.copy(in, out);

    // 关闭流
    out.close();
    in.close();

    return "下载模型xml文件成功";
}
```

运行后，生成文件：`请假申请流程.bpmn20.xml`

6.5 通过模型数据部署流程定义

```
@Autowired
private RepositoryService repositoryService;

@Autowired
private ObjectMapper objectMapper;

/**
 * 通过流程模型进行流程定义部署
 * 流程图保存的时候是json串，引擎认识的却是符合bpmn2.0规范xml，
 * 所以在首次的部署的时候要将json串转换为BpmnModel，
 * 再将BpmnModel转换成xml保存进数据库，以后每次使用就直接将xml转换成BpmnModel，
 * 这套操作确实有点啰嗦，实际项目中如果不用activiti自带的设计器，可以考虑用插件，直接生成的是xml
 * 或者自己开发的设计器，在后端生成节点及其属性，引擎有现成的节点实体，如：开始节点StartEvent，SequenceFlow等
 * 涉及表：
 *      ACT_RE_PROCDEF 新增数据：流程定义数据
 *      ACT_RE_DEPLOYMENT 新增数据：流程部署数据
 *      ACT_GE_BYTEARRAY 新增数据：将当前流程图绑定到此流程定义部署数据上
 *      ACT_RE_MODEL 更新部署id
 */
@PostMapping("/deploy")
public String deploy() throws Exception {
    // 模型id
    String id = "729b1dd4-1316-11ed-89b2-005056c00001";

    // 获取流程图json字节码
    byte[] jsonBytes = repositoryService.getModelEditorSource(id);
    if (jsonBytes == null) {
        return "模型数据为空，请先设计流程并成功保存，再进行发布。";
    }

    // 转xml字节数组
    byte[] xmlBytes = bpmnJsonToXmlBytes(jsonBytes);
    if (xmlBytes == null) {
        return "数据模型不符要求，请至少涉及一条主线流程";
    }

    // 流程图片字节码
    byte[] pngBytes = repositoryService.getModelEditorSource(id);

    // 获取模型
    Model model = repositoryService.getModel(id);

    // 流程定义xml名称
    String processName = model.getName() + "bpmn20.xml";
    // 流程定义png名称
    String pngName = model.getName() + "." + model.getKey() + ".png";
    // 流程部署
    Deployment deployment = repositoryService.createDeployment()
        .name(model.getName())
        .addString(processName, new String(xmlBytes, "UTF-8"))  // xml文件
        .addBytes(pngName, pngBytes)  // 图片
        .deploy();
```

```
    // 更新部署id到模型对象（将模型与部署数据绑定）
    model.setDeploymentId(deployment.getId());
    repositoryService.saveModel(model);

    return "部署完成";
}

/** 流程图保存的时候是json串，引擎认识的却是符合bpmn2.0规范的xml，json字节码转xml字节码 */
private byte[] bpmnJsonToXmlBytes(byte[] jsonBytes) throws IOException {
    if (jsonBytes == null) {
        return null;
    }
    // json转回BpmnModel对象
    JsonNode modelNode = objectMapper.readTree(jsonBytes);
    BpmnModel bpmnModel = new BpmnJsonConverter().convertToBpmnModel(modelNode);
    if (bpmnModel.getProcesses().size() == 0) {
        return null;
    }

    // BpmnModel对象转xml字节数组
    return new BpmnXMLConverter().convertToXML(bpmnModel);
}
```

> - 每个流程定义模型可以多次流程定义部署，activiti通过流程定义模型中的标识key来判断是否为同一流程模型，相同标识key则视为同一流程定义模型
> - 相同的标识key流程定义模型，每部署一次对应的新增一条流程定义数据，对应流程定义版本号会基于之前的加1

部署流程定义涉及表：

```
ACT_RE_PROCDEF     新增数据：流程定义数据
ACT_RE_DEPLOYMENT 新增数据：流程部署数据
ACT_GE_BYTEARRAY  新增数据：流程定义 xml 和 png 保存下来，对应绑定到此流程定义数据上
ACT_RE_MODEL       更新部署id
```

部署流程报错：

1. 报错：元素'sequenceFlow'中必须包含属性'targetRef'

   解决：流程唯一标识`不能以数字开头`，以字母开头

2. 元素 'sequenceFlow' 中必须包含属性 'sourceRef'

   解决：两个图标箭头连线，箭头来源和目标多拉一点到图标上(Ctrl+A全选流程设计图，移动一下哪根连线没动说明这根有问题)

七、流程定义部署管理Deployment

7.1 概要

通过流程定义模型，进行部署流程定义，部署后会生成流程定义数据（相当于java类），此时生成的流程定义数据主要用于生成流程实例（相当于java对象），一个流程定义Java类对应的可以创建无数个java流程实例对象

7.2 通过.zip压缩包部署流程定义

```
@Autowired
private RepositoryService repositoryService;

@PostMapping("/deployByZip")
public String deployByZip() throws FileNotFoundException {
    File file = new File("D:/请假流程模型.leaveProcess.zip");
    String filename = file.getName();
    // 压缩包输入流
    ZipInputStream zipInputStream = new ZipInputStream(new
FileInputStream(file));

    // 创建部署实例
    DeploymentBuilder deploymentBuilder = repositoryService.createDeployment();
    // 添加zip流
    deploymentBuilder.addZipInputStream(zipInputStream);
    // 部署名称
    deploymentBuilder.name(filename.substring(0, filename.indexOf(".")));
    // 执行部署流程定义
    deploymentBuilder.deploy();

    return "zip压缩包方式部署流程定义完成";
}
```

部署流程定义涉及表：

```
ACT_RE_PROCDEF     新增数据： 流程定义数据
ACT_RE_DEPLOYMENT 新增数据： 流程部署数据
ACT_GE_BYTEARRAY   新增数据：将当前流程图绑定到此流程定义部署数据上
ACT_RE_MODEL       更新部署id
```

7.3 通过.bpmn或.bpmn.xml文件部署流程定义

```
@Autowired
private RepositoryService repositoryService;

@PostMapping("/deployByBpmnFile")
public String deployByBpmnFile() throws FileNotFoundException {
    // .bpm文件
    // File file = new File("D:/leave.bpmn");
    // .bpmn20.xml文件
    File file = new File("D:/请假申请流程.bpmn20.xml");

    String filename = file.getName();
    // 输入流
    FileInputStream inputStream = new FileInputStream(file);

    // 创建部署实例
    DeploymentBuilder deployment = repositoryService.createDeployment();
    // bpmn20.xml或.bpmn(activiti5.10版本以上支持)
    deployment.addInputStream(filename, inputStream);
    // 部署名称
    // deployment.name(filename.substring(0, filename.indexOf(".")));
    // 执行流程定义部署
```

```
    deployment.deploy();

    return "通过.bpmn或.bpmn20.xml部署完成";
}
```

7.4 删除流程定义部署信息

```
@Autowired
private RepositoryService repositoryService;

/**
 * 根据部署id删除流程定义部署信息：
 * ACT_GE_BYTEARRAY、ACT_RE_DEPLOYMENT、ACT_RU_IDENTITYLINK、ACT_RE_PROCDEF、
ACT_RU_EVENT_SUBSCR
 */
@DeleteMapping("/delete")
public String delete() {
    try {
        // 部署ID
        String deploymentId = "d323c400-1393-11ed-b44d-005056c00001";
        // 不带级联的删除：如果有正在执行的流程，则删除失败抛出异常；不会删除ACT_HI_和历史表数
据
        repositoryService.deleteDeployment(deploymentId);

        // 级联删除：不管流程是否启动，都能可以删除；并删除历史表数据
        // repositoryService.deleteDeployment(deploymentId, true);
        return "删除流程定义部署信息成功";
    } catch (Exception e) {
        e.printStackTrace();
        if (e.getMessage().indexOf("a foreign key constraint fails") > 0) {
            return "有正在执行的流程，不允许删除";
        } else {
            return "删除失败，原因：" + e.getMessage();
        }
    }
}
```

八、流程定义管理ProcessDefinition

8.1 概要

部署好流程定义后，则可以进行查询、激活(启动)、挂起(暂停)、删除流程定义数据(上面讲的删除流程定义部署信息)，下载流程定义对应的xml文件和png文件

8.2 分页条件查询流程定义

```
@Autowired
private RepositoryService repositoryService;

@PostMapping("/getProcessDefinitionList")
public Map<String, Object> getProcessDefinitionList() {
    // 1.获取ProcessDefinitionQuery
```

```
    ProcessDefinitionQuery query =
repositoryService.createProcessDefinitionQuery();
    // 条件查询
    query.processDefinitionNameLike("%请假%");
    // 有多个相同标识key的流程时，只查询其最新版本
    query.latestVersion();
    // 按流程定义key升序排序
    query.orderByProcessDefinitionKey().asc();
    // 当前查询第几页
    int current = 1;
    // 每页显示多少条数据
    int size = 5;
    // 当前页第1条数据下标
    int firstResult = (current - 1) * size;
    // 开始分页查询
    List<ProcessDefinition> definitionList = query.listPage(firstResult, size);

    List<Map<String, Object>> list =
definitionList.stream().map(processDefinition -> {
        Map<String, Object> map = new HashMap<>();
        map.put("流程部署ID：", processDefinition.getDeploymentId());
        map.put("流程定义ID：", processDefinition.getId());
        map.put("流程定义Key：", processDefinition.getKey());
        map.put("流程定义名称：", processDefinition.getName());
        map.put("流程定义版本号：", processDefinition.getVersion());
        map.put("状态：", (processDefinition.isSuspended() ? "挂起(暂停)" : "激活(开
启)"));
        return map;
    }).collect(Collectors.toList());

    Map<String, Object> map = new HashMap<>();
    map.put("list", list);
    map.put("total", query.count());  // 满足条件的流程定义总记录数

    return map;
}
```

8.3 激活或挂起流程定义

- 流程定义被挂起：此流程定义下的所有流程实例不允许继续后流转了，就被停止了
- 流程定义被激活：此流程定义下的所有流程实例允许继续往后流转
- 为什么会被挂起？
  - 可能当前公司的请假流程发现了一些不合理的地方，然后就把此流程定义挂起
  - 流程不合理解决方法：
    - 方法一：可以先挂起流程定义，然后更新流程定义，然后激活流程定义
    - 方法二：挂起了就不激活了，重新创建一个新的请假流程定义

```
@Autowired
private RepositoryService repositoryService;

/** 通过流程定义id，挂起或激活流程定义 */
@PostMapping("/updateProcessDefinitionState")
public String updateProcessDefinitionState() {
```

```
    // 流程定义ID
    String definitionId = "leaveProcess:1:d337c133-1393-11ed-b44d-005056c00001";

    // 流程定义对象
    ProcessDefinition processDefinition =
repositoryService.createProcessDefinitionQuery()
        .processDefinitionId(definitionId)
        .singleResult();

    // 获取当前状态是否为：挂起
    boolean suspended = processDefinition.isSuspended();
    if (suspended) {
        // 如果状态是：挂起，将状态更新为：激活
        // 参数1：流程定义id；参数2：是否级联激活该流程定义下的流程实例；参数3：设置什么时间激
活这个流程定义，如果null则立即激活
        repositoryService.activateProcessDefinitionById(definitionId, true,
null);
        return "流程定义激活成功";
    } else {
        // 如果状态是：激活，将状态更新为：挂起
        // 参数（流程定义id，是否挂起，激活时间）
        repositoryService.suspendProcessDefinitionById(definitionId, true, null);
        return "流程定义挂起成功";
    }
}
```

对应act_re_procdef表中的SUSPENSION_STATE_字段，1是激活，2是挂起

8.4 下载流程定义的xml和png文件

下载流程定义的xml和png文件，方便用户浏览当前流程定义是怎样的

```
/** 导出下载刘恒定义文件(.bpmn20.xml流程描述或.png图片资源) */
@PostMapping("/exportProcessDefinitionFile")
public String exportProcessDefinitionFile() throws IOException {
    // 流程定义ID
    String definitionId = "leaveProcess:1:d337c133-1393-11ed-b44d-005056c00001";

    // 查询流程定义数据
    ProcessDefinition processDefinition =
repositoryService.getProcessDefinition(definitionId);

    // xml文件名
    // String fileName = processDefinition.getResourceName();

    // png图片名
    String fileName = processDefinition.getDiagramResourceName();

    // 获取对应文件输入流
    InputStream in =
repositoryService.getResourceAsStream(processDefinition.getDeploymentId(),
fileName);

    // 创建输出流
    File file = new File("D:/igo/" + fileName);
    FileOutputStream output = new FileOutputStream(file);
```

```
    // 流拷贝
    IOUtils.copy(in, output);

    // 关闭流
    in.close();
    output.close();

    return "导出成功";
}
```

九、流程实例管理ProcessInstance

9.1 概要

通过流程定义模型，进行部署流程定义，部署后会生成流程定义数据（相当于java类），此时生成的流程定义数据主要用于启动流程实例，一个流程定义Java类对应的可以创建无数个java流程实例对象

例如：部署好了请假流程定义，用户就可以发送请假流程了，即启动对应用户的请假流程实例，就可以开始对这个用户进行流程审批了

- 张三提交清酒流程（启动一个新的流程实例）
- 李四提交请假流程（启动一个新的流程实例）
- 第N个人提交请求流程（启动一个新的流程实例）

9.2 启动流程实例（提交申请-关联业务ID businessKey）

启动流程实例，实际上就是提交流程申请

例如：提交请假申请流程，要先填写请假表单数据保存数据库，此请假单ID对应的就是businessKey，在启动流程实例时将此businessKey与此流程实例绑定，办理此流程实例任务，其实就是对此请假表单进行办理

开启流程实例指定的businessKey，在流程实例的执行表act_ru_execution存储businessKey

```
@Autowired
private RuntimeService runtimeService;

/** 开启流程实例(提交申请，启动流程申请) */
@PostMapping("/startProcessApply")
public String startProcessApply() {
    // 流程定义唯一标识KEY
    String processKey = "leaveProcess";
    // 业务唯一标识值
    String bussinessKey = "10000";

    // 启动流程用户（保存在act_hi_procinst的start_user_id）字段
    Authentication.setAuthenticatedUserId("igo");

    // 开启流程实例(流程定义标识key，业务key),会采用指定：流程key的最新版本流程定义
    ProcessInstance processInstance =
runtimeService.startProcessInstanceByKey(processKey, bussinessKey);

    // 将流程定义名称作为流程实例名称
```

```
    runtimeService.setProcessInstanceName(processInstance.getProcessInstanceId(),
processInstance.getProcessDefinitionName());

    return "创建流程实例（提交请假申请）完成：" +
processInstance.getProcessInstanceId();
}
```

9.3 查询正在运行中的流程实例

```
@Autowired
private RuntimeService runtimeService;

/** 查询正在运行中的流程实例 */
@PostMapping("/getProcInstListRunning")
public List<Map<String, Object>> getProcInstListRunning() {
    List<ProcessInstance> instanceList =
runtimeService.createProcessInstanceQuery().list();

    return instanceList.stream().map(instance -> {
        Map<String, Object> map = new HashMap<>();
        map.put("流程业务唯一key", instance.getBusinessKey());
        map.put("流程实例id", instance.getProcessInstanceId());
        map.put("流程实例名称", instance.getName());
        map.put("流程定义Key", instance.getProcessDefinitionKey());
        map.put("流程定义版本号", instance.getProcessDefinitionVersion());
        map.put("流程发起人", instance.getStartUserId());
        map.put("流程状态", instance.isSuspended() ? "已挂起" : "已激活");
        return map;
    }).collect(Collectors.toList());
}
```

```
[
    {
        "流程业务唯一key": "10000",
        "流程发起人": "igo",
        "流程实例id": "eee9684d-1559-11ed-8da2-283a4d3b4979",
        "流程状态": "已激活",
        "流程实例名称": "请假申请流程",
        "流程定义Key": "leaveProcess",
        "流程定义版本号": 1
    }
]
```

9.4 挂起或激活流程实例

流程定义：

- 流程定义被挂起：此流程定义下的所有流程实例不允许继续往后流转了，就被停止了
- 流程定义被激活：此流程定义下的所有流程实例允许继续往后流转

流程实例：

- 流程实例被挂起：针对指定单个流程实例不允许继续往后流转了，被停止了
- 流程实例被激活：针对指定单个流程实例允许继续往后流转

```
@Autowired
```

```
private RuntimeService runtimeService;

/** 挂起或激活单个流程实例 */
@PostMapping("/updateProcInstState")
public String updateProcInstState() {
    // 流程实例ID
    String procInstId = "eee9684d-1559-11ed-8da2-283a4d3b4979";

    // 查询流程实例对象
    ProcessInstance processInstance =
runtimeService.createProcessInstanceQuery().processDefinitionId(procInstId).singl
eResult();

    // 获取当前流程实例状态是否：挂起（暂停）
    boolean suspended = processInstance.isSuspended();

    // 判断
    if (suspended) {
        // 如果状态是：挂起，将状态更新为：激活（启动）
        runtimeService.activateProcessInstanceById(procInstId);
        return "流程实例激活成功";
    } else {
        // 如果状态是：激活，将状态更新为：挂起（暂停）
        runtimeService.suspendProcessInstanceById(procInstId);
        return "流程实例挂起成功";
    }
}
```

9.4 删除（作废）流程实例

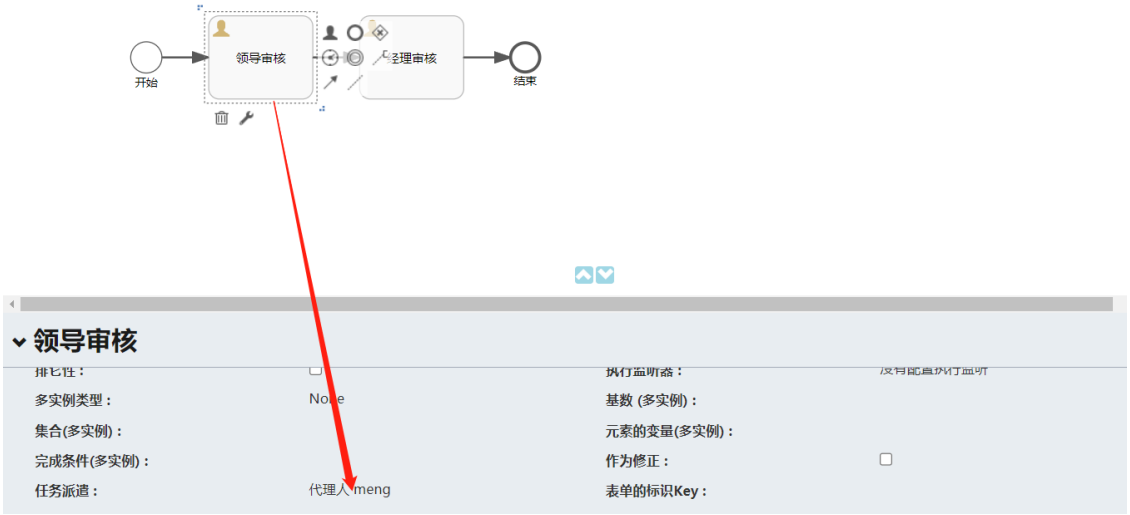
```
@Autowired
private RuntimeService runtimeService;

/** 删除（作废）流程实例，不会删除历史记录 */
@DeleteMapping("/deleteProcInst")
public String deleteProcInst() {
    // 流程实例ID
    String procInstId = "eee9684d-1559-11ed-8da2-283a4d3b4979";

    // 查询流程实例对象
    ProcessInstance processInstance =
runtimeService.createProcessInstanceQuery().processDefinitionId(procInstId).singl
eResult();

    if (processInstance == null) {
        return "流程实例不存在";
    }

    // 删除流程实例（流程实例ID，删除原因)
    runtimeService.deleteProcessInstance(procInstId, "xxx作废了当前流程申请");
    return "删除流程实例成功";
}
```

十、流程任务管理TaskService

10.1 固定分配-任务办理人

在绘制流程定义模型时，针对节点固定分配办理人进行办理此节点任务

如下图：针对领导审批节点填写任务派遣（任务处理人）为meng，也就是指定meng为此节点办理人



10.2 查询办理人或候选人的待办任务

```
@Autowired
    private TaskService taskService;

    /** 查询指定办理人的待办任务 */
    @PostMapping("findWaitTask")
    public List<Map<String, Object>> findWaitTask() {
        // 办理人
        String assignee = "meng";

        // 指定个人任务查询
        List<Task> taskList = taskService.createTaskQuery()
                .taskAssignee(assignee)
                //.taskCandidateOrAssigned(assignee)  // 作为办理人或候选人
                .orderByTaskCreateTime().desc()
                .list();

        return taskList.stream().map(task -> {
            Map<String, Object> map = new HashMap<>();
            map.put("任务ID", task.getId());
            map.put("任务名称", task.getName());
            map.put("流程状态", (task.isSuspended() ? "已挂起" : "已激活"));
            map.put("任务的创建时间", task.getCreateTime());
            map.put("任务的办理人", task.getAssignee());
            return map;
        }).collect(Collectors.toList());
    }
```

10.3 表达式分配-任务办理人

由于固定分配任务办理人，执行到每一个任务，按照的是绘制bpmn流程定义模型中配置的去分配任务负责人，即每个节点任务都是固定办理人。

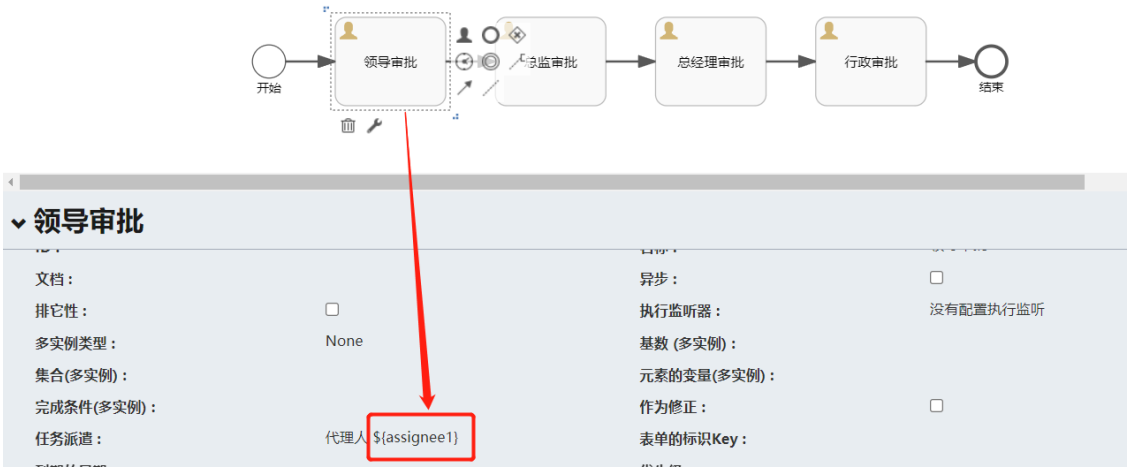
也可以使用UEL表达式动态分配办理人，UEL(Unified Expression Language)即统一表达式语言，UEL是java EE6 规范的一部分

activiti支持两种UEL表达式：UEL-value和UEL-method

10.4 UEL-Value流程变量表达式

10.4.1 指定属性名

语法：`${属性名}`，如下图：针对**领导审批**节点指定办理人`${assignee1}`，assignee1就是一个流程变量，在启动流程实例时，可以动态指定办理人



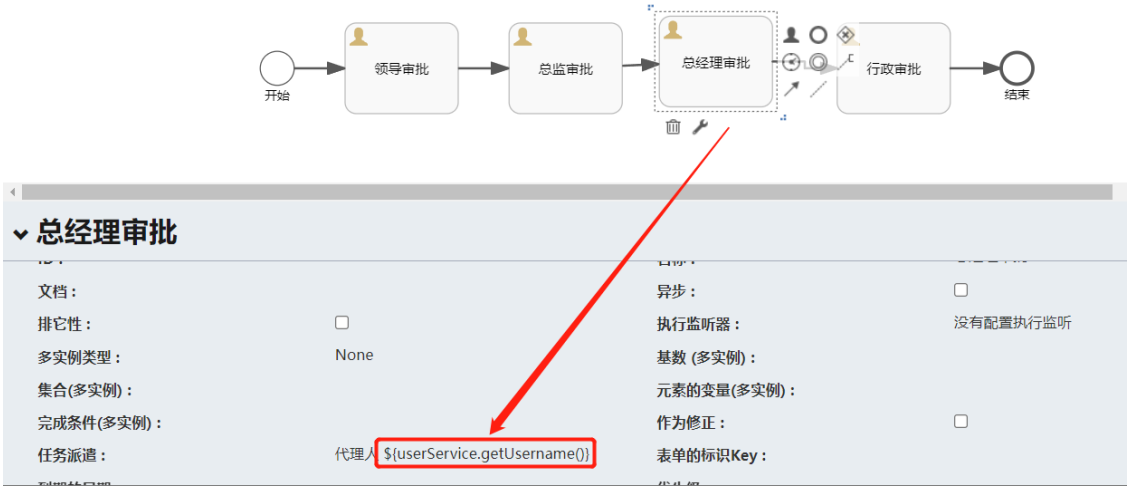
10.4.2 指定对象属性名

语法：`${对象.属性名}`，如下图：针对总监审批节点指定办理人`${user.username}`，user是对象名，也是一个流程变量，通过调用user对象中的getUsername方法获取到动态值。

在启动流程实例时，可以动态指定办理人。

10.5 UEL-Method方法表达式

针对**总经理审批**节点指定办理人 `${userService.getUsername()}` ， `userService` 是Spring容器中的一个bean实例，对应调用 `userService` 的 `getUsername()` 实例方法



10.6 UEL-Method和UEL-Value组合使用

通过用户id查询上级领导作为任务办理人：

针对行政审批节点指定办理人 `${deptService.findManagerByUserId(userId)}`

- `deptService` 是Spring容器的一个bean实例；
- `findManagerByUserId` 是 `deptService` 实例方法
- `userId` 是流程变量，将 `userId` 作为参数传到 `deptService.findManagerByUserId` 方法中

10.7 表达式其他支持格式

表达式支持解析：基本数据类型、JavaBean、Map集合、List集合、Array数组等

也可作为条件判断（**一般用于网关**），如： `${leave.duration > 3 && leave.duratuon < 10}`

10.8 注意事项

当使用了UEL流程变量表达式时，在执行到所需要流程变量的任务时，必须保证在此任务前流程变量存在，否则activiti会抛出异常，比如：某个节点任务设置了表达式 `${leave.duration > 3 && leave.duratuon < 10}`，当执行到此任务时必须保证 `leave` 在流程变量中存在

10.9 代码实现UEL动态分配办理人

1. 创建 `com.igo.service.UserService`，并在类上加上 `@Service` 注解

   在getUsername方法上打断点，观察：总监审批完成任务时，会调用getUsername获取总经理审批节点办理人

   ```
   @Service  // 不要少
   public class UserService {
       public String getUsername() {
           System.out.println("UserService.getUsername获取用户信息");
           return "gu";
       }
   }
   ```

2. 创建 `com.igo.service.DeptService`，并在类上加上 `@Service` 注解，在 `findManagerByUserId` 方法上打断点，观察： `总经理审批` 完成任务时，会调用findManagerByUserId获取 `行政审批` 节点办理人

   ```
   @Service  // 不要少
   public class DeptService {
       public String findManagerByUserId(String userId) {
           System.out.println("DeptService.findManagerByUserId查询userId=" +
   userId + "上级领导作为办理人");
           return "小梦";
       }
   }
   ```

3. 部署新的请假流程定义：触发接口 **http://localhost:8080/workflow/model/deploy**，注意：模型ID

4. 启动流程实例时：分配流程定义的流程变量

   ```
   @Autowired
   private RuntimeService runtimeService;

   /** 启动流程实例：分配流程定义中的流程变量值（节点任务就可以获取到办理人） */
   @PostMapping("/start")
   public String startProcessSetAssigneeUEL() {
       // 流程变量
       Map<String, Object> variables = new HashMap<>();
       variables.put("assignee1", "meng");

       // 使用Map作为User对象
       Map<String, Object> userMap = new HashMap<>();
       userMap.put("username", "xue");
       variables.put("user", userMap);

       // 设置方法参数userId流程变量：${deptService.findManagerByUserId(userId)}
       variables.put("userId", "123");
   ```

```
    // 启动流程实例（流程定义key，业务id，流程变量）
    // ProcessInstance processInstance =
runtimeService.startProcessInstanceByKey("leaveProcess", variables);
    ProcessInstance processInstance =
runtimeService.startProcessInstanceByKey("leaveProcess", "9999", variables);

    return "流程实例ID：" + processInstance.getProcessInstanceId();
}
```

启动流程实例成功后，在ACT_RU_VARIABLE表会插入variables流程变量数据

NAME_	EXECUTION_ID_	PROC_INST_ID_	TASK_ID_	BYTEARRAY_ID_	DOUBLE_	LONG_	TEXT_	TEXT2_
user	2597a0de-1642-11ed-b56c-283a4d3b4979	2597a0de-1642-11ed-b	(Null)	(Null)	(Null)	(Null)	{"username":"xue"}	java.util.HashMap
userId	2597a0de-1642-11ed-b56c-283a4d3b4979	2597a0de-1642-11ed-b	(Null)	(Null)	(Null)	(Null)	123	(Null)
assignee1	2597a0de-1642-11ed-b56c-283a4d3b4979	2597a0de-1642-11ed-b	(Null)	(Null)	(Null)	(Null)	meng	(Null)
user	25999cb5-1642-11ed-b56c-283a4d3b4979	2597a0de-1642-11ed-b	259c34c8-1642-11ed-b56c-283a4d3b4979	(Null)	(Null)	(Null)	{"username":"xue"}	java.util.LinkedHashMap
userId	25999cb5-1642-11ed-b56c-283a4d3b4979	2597a0de-1642-11ed-b	259c34c8-1642-11ed-b56c-283a4d3b4979	(Null)	(Null)	(Null)	123	(Null)
assignee1	25999cb5-1642-11ed-b56c-283a4d3b4979	2597a0de-1642-11ed-b	259c34c8-1642-11ed-b56c-283a4d3b4979	(Null)	(Null)	(Null)	meng	(Null)
user	a39ad8a7-1642-11ed-a53d-283a4d3b4979	a39ad8a7-1642-11ed-a	(Null)	(Null)	(Null)	(Null)	{"username":"xue"}	java.util.HashMap
userId	a39ad8a7-1642-11ed-a53d-283a4d3b4979	a39ad8a7-1642-11ed-a	(Null)	(Null)	(Null)	(Null)	123	(Null)
assignee1	a39ad8a7-1642-11ed-a53d-283a4d3b4979	a39ad8a7-1642-11ed-a	(Null)	(Null)	(Null)	(Null)	meng	(Null)
user	a39cfb8e-1642-11ed-a53d-283a4d3b4979	a39ad8a7-1642-11ed-a	a39fbab1-1642-11ed-a53d-283a4d3b4979	(Null)	(Null)	(Null)	{"username":"xue"}	java.util.LinkedHashMap
userId	a39cfb8e-1642-11ed-a53d-283a4d3b4979	a39ad8a7-1642-11ed-a	a39fbab1-1642-11ed-a53d-283a4d3b4979	(Null)	(Null)	(Null)	123	(Null)
assignee1	a39cfb8e-1642-11ed-a53d-283a4d3b4979	a39ad8a7-1642-11ed-a	a39fbab1-1642-11ed-a53d-283a4d3b4979	(Null)	(Null)	(Null)	meng	(Null)

上图后面三条数据是针对第一个节点任务的变量，说明会把流程变量带到每个任务上

5. 查询办理人meng的待办任务，触发接口：**http://localhost:8080/workflow/task/findWaitTask**

```
[
    {
        "流程状态": "已激活",
        "任务ID": "a39fbab1-1642-11ed-a53d-283a4d3b4979",
        "任务的创建时间": "2022-08-07T11:18:18.707+00:00",
        "任务的办理人": "meng",
        "任务名称": "领导审批"
    },
    {
        "流程状态": "已激活",
        "任务ID": "259c34c8-1642-11ed-b56c-283a4d3b4979",
        "任务的创建时间": "2022-08-07T11:14:47.292+00:00",
        "任务的办理人": "meng",
        "任务名称": "领导审批"
    },
    {
        "流程状态": "已激活",
        "任务ID": "eeed38e2-1559-11ed-8da2-283a4d3b4979",
        "任务的创建时间": "2022-08-06T07:32:32.304+00:00",
        "任务的办理人": "meng",
        "任务名称": "领导审核"
    }
]
```

进行完成第一条任务

```
@Autowired
private TaskService taskService;

/** 完成待办任务 */
@PostMapping("/complete")
public String completeTask() {
    // 完成任务(任务id)
    taskService.complete("a39fbab1-1642-11ed-a53d-283a4d3b4979");
    return "完成";
}
```

10.10 TaskListener任务监听器

TaskListener任务监听器可用于很多流程业务功能，比如：任务创建后；分配任务办理人；任务完成后；记录日志；发送提醒。

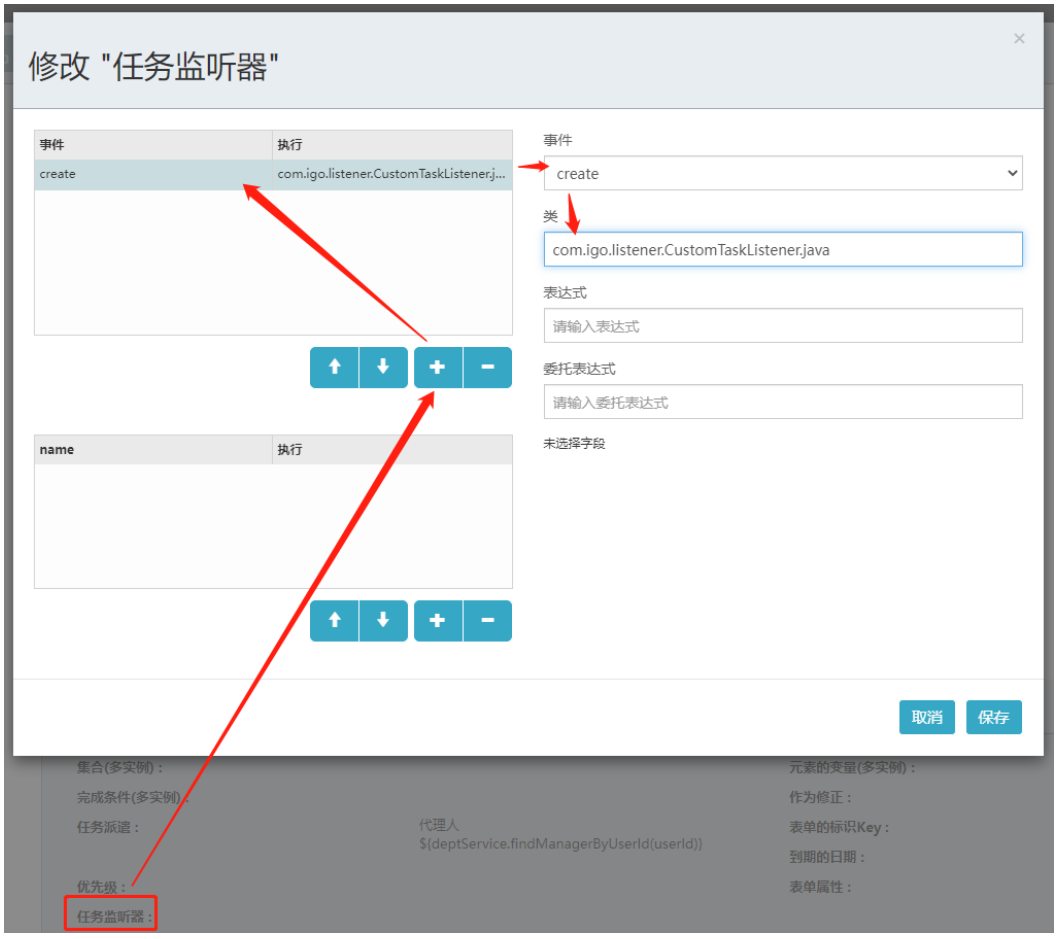
任务监听器相关触发事件名：

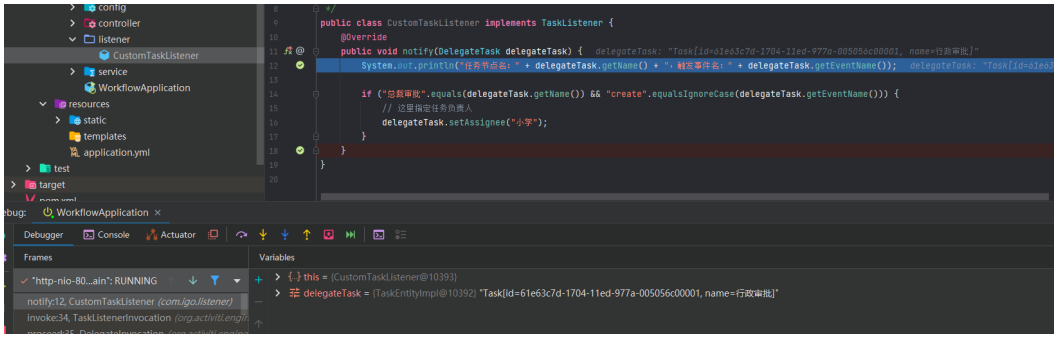
```
create: 任务创建后触发
assigment: 任务分配办理人后触发
complete: 任务完成后触发
delete: 任务删除后触发
```

1. 创建TaskListener实现类，用于指定处理人

```
package com.igo.listener;

import org.activiti.engine.delegate.DelegateTask;
import org.activiti.engine.delegate.TaskListener;

/**
 * 自定义任务监听器
 */
public class CustomTaskListener implements TaskListener {
    @Override
    public void notify(DelegateTask delegateTask) {
        System.out.println("任务节点名: " + delegateTask.getName() + ", 触发事
件名: " + delegateTask.getEventName());

        if ("行政审批".equals(delegateTask.getName()) &&
"create".equalsIgnoreCase(delegateTask.getEventName())) {
            // 这里指定任务负责人
            delegateTask.setAssignee("小学");
        }
    }
}
```

2. 针对流程模型添加一个行政审批节点，其中添加任务监听器，指定任务监听器create事件处理类，如下：





> 类 输入框不带“.java”，否则会报错

3. 测试
   - 部署流程定义：**http://localhost:8080/workflow/model/deploy**
   - 启动流程实例：**http://localhost:8080/workflow/assignee/start**
   - 完成节点任务：**http://localhost:8080/workflow/assignee/complete**（注意：任务id），在完成总裁审批的后，会创建行政审批节点任务，创建任务后会执行CustomTaskListener监听器，然后设置行政审批节点的处理人。

10.11 获取当前任务的下一节点信息

获取当前任务的下一节点信息，判断下一节点是用户任务UserTask则获取，获取后方便动态设置一下节点任务办理人

```
@Autowired
private TaskService taskService;
@Autowired
private RepositoryService repositoryService;
```

```java
/** 获取当前任务的下一节点用户任务信息，为了动态设置下一节点任务办理人 */
@PostMapping("/nextNodeInfo")
public List<Map<String, Object>> getNextNodeInfo() {
    String taskId = "38200436-1709-11ed-83f3-005056c00001";
    Task task = taskService.createTaskQuery().taskId(taskId).singleResult();
    if (task == null) {
        return new ArrayList<>();
    }

    // 获取当前模型
    BpmnModel bpmnModel =
repositoryService.getBpmnModel(task.getProcessDefinitionId());
    // 根据任务节点id获取当前节点
    FlowElement flowElement =
bpmnModel.getFlowElement(task.getTaskDefinitionKey());

    // 获取当前节点的连线信息(可能会有多条分支，所以返回值是集合)
    List<SequenceFlow> outgoingFlows =
((FlowNode)flowElement).getOutgoingFlows();
    List<Map<String, Object>> dataList = new ArrayList<>();
    outgoingFlows.stream().forEach(outgoingFlow -> {
        // 下一节点
        FlowElement nextFlowElement = outgoingFlow.getTargetFlowElement();
        if (nextFlowElement instanceof UserTask) {
            // 用户任务，获取节点id和名称
            Map<String, Object> map = new HashMap<>();
            map.put("节点id", nextFlowElement.getId());
            map.put("节点名称", nextFlowElement.getName());
            dataList.add(map);
        }
    });

    return dataList;
}
```

```
"节点id": "sid-B4ACF6EB-47BB-48B5-8A52-CAB47CA7A7D5",
"节点名称": "总监审批"
```

10.12 任务完成后设置一下任务办理人

不基于UEL表达式，当前任务完成后，查询下一任务，使用java代码动态指定下节点任务办理人，这样更加灵活

```java
/** 完成当前任务，设置下一节点任务处理人 */
@PostMapping("/completeTaskSetNextAssignee")
public String completeTaskSetNextAssignee() {
    String taskId = "4a427c90-17a4-11ed-b02b-005056c00001";
    // 查询任务
    Task task = taskService.createTaskQuery()
        .taskId(taskId)
        .singleResult();

    // 完成任务
    taskService.complete(taskId);
```

```
    // 查询下一个任务
    List<Task> taskList =
taskService.createTaskQuery().processInstanceId(task.getProcessInstanceId()).list
();

    // 有下节点任务
    if (!CollectionUtils.isEmpty(taskList)) {
        // 针对每个任务分配审批人
        for (Task t : taskList) {
            // 当前任务有审批人，则不设置新的审批人
            if (StringUtils.isNotEmpty(t.getAssignee())) {
                continue;
            }
            // 分配任务办理人
            String assignees = "小谷";
            taskService.setAssignee(t.getId(), assignees);
        }
    }

    return "设置下一节点处理完成";
}
```

10.13 候选人任务

10.13.1 需求

一个电商平台，每天订单量都很大，在处理订单的时候分配了user1，user2，user3三个员工，这时候就可以用上这个候选人功能了，在一个任务里配置上可能会参与这个任务的候选人，这样候选人员工就可以通过查询候选人任务知道自己可以领取哪些任务，从而达到员工自动领取任务的功能

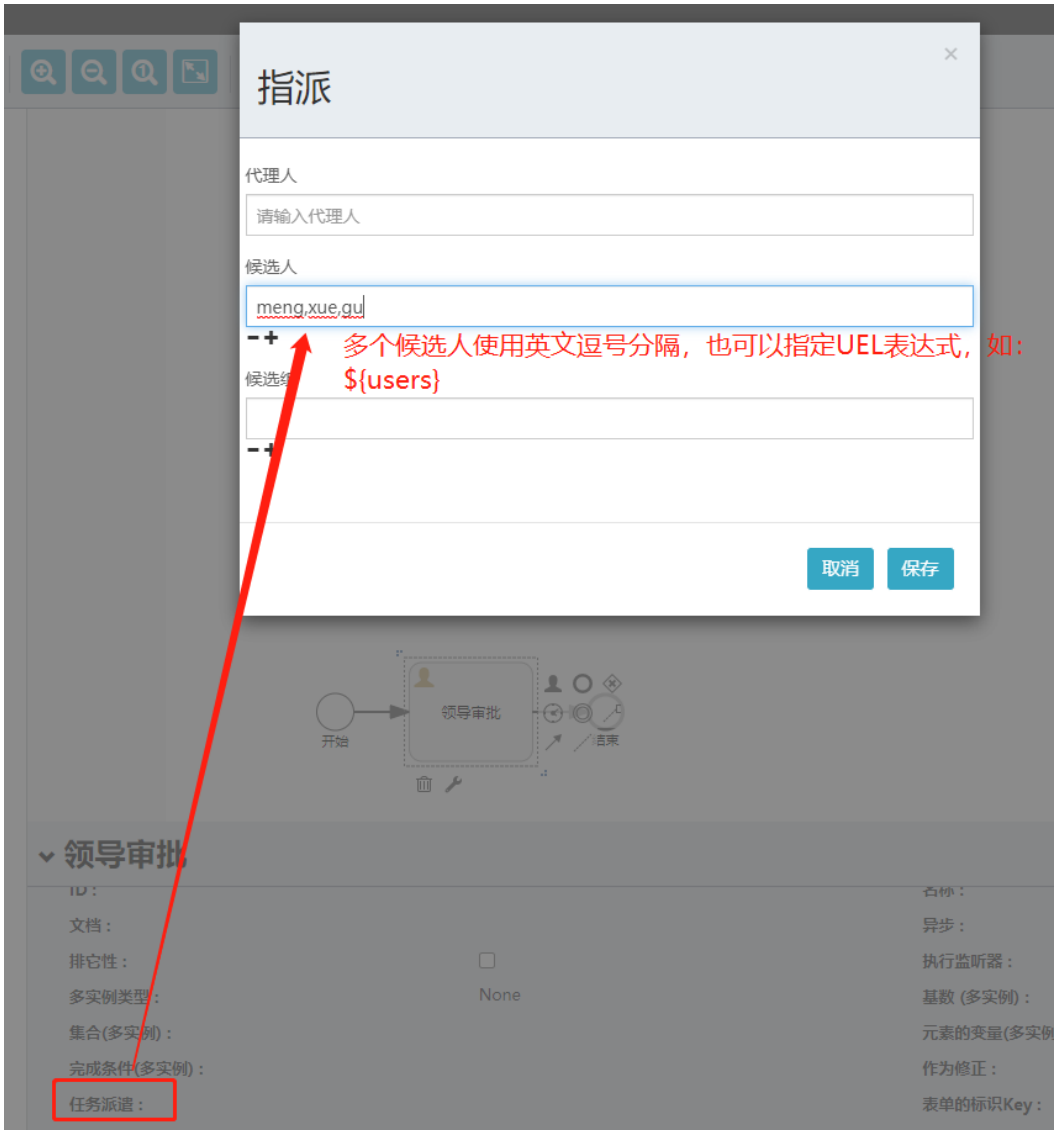
10.13.2 绘制候选人流程模型

1. 创建新流程模型：`testGroupTask`

   访问：**http://localhost:8080/workflow/model/create**

2. 指定任务多个候选人

   点击领导审批，指定此节点任务一组候选人`meng,xue,gu`，使用英文逗号分隔，也可以指定UEL表达式



3. 候选任务办理步骤
    1. **查询候选任务**

       候选人不能立即办理任务，要先查询指定候选人的待办任务
    2. **拾取任务(claim)**

       该任务的所有候选人都能拾取，现将候选人的组任务，变成个人任务，对应这个候选人就变成该任务的办理人
       - **拾取后如果不想办理该任务?**

         将个人任务变成了组任务，就是将已经拾取的个人任务归还到组里边
       - **如何拾取后转接给别人办理任务?**

         直接针对任务设置目标办理人
    3. **查询个人任务**

       查询方式同个人任务部分，根据assignee查询用户负责的个人任务
    4. **办理个人任务**
4. 部署流程定义和开启流程实例
    - 部署流程定义：**http://localhost:8080/workflow/model/deploy**，注意模型`id`
    - 开启流程实例

```
@Autowired
private RuntimeService runtimeService;
```

```
/** 启动流程实例 */
@PostMapping("/start")
public String startProcess() {
    // 流程定义唯一标识KEY
    String processKey = "testGroupTask";
    // 业务唯一标识值
    String bussinessKey = "4444";
    // 启动流程实例
    ProcessInstance processInstance =
runtimeService.startProcessInstanceByKey(processKey, bussinessKey);

    return "创建流程实例（候选人任务）完成：" +
processInstance.getProcessInstanceId();
}
```

5. 查询组任务

   根据候选人查询组任务，现在查询就不能用assignee来查询了，因为 `assignee` 是办理人，现在这条任务还没有指定办理人，要用 `candidateUser` 候选人查询

```
@Autowired
private TaskService taskService;

/** 查询候选任务 */
@PostMapping("/getGroupTaskList")
public List<Map<String, Object>> getGroupTaskList() {
    // 流程定义key
    String processDefinitionKey = "testGroupTask";

    // 任务候选人
    String candidateUser = "meng";

    // 查询组任务
    List<Task> list = taskService.createTaskQuery()
        .processDefinitionKey(processDefinitionKey)
        .taskCandidateUser(candidateUser)  // 根据候选人查询
        .list();

    return list.stream().map(task -> {
        Map<String, Object> map = new HashMap<>();
        map.put("任务id", task.getId());
        map.put("任务办理人（是候选人所以null）", task.getAssignee() == null ?
"null" : task.getAssignee());
        map.put("任务名称", task.getName());
        return map;
    }).collect(Collectors.toList());
}
```

```
"任务id": "dba5abef-1c5c-11ed-8bae-509a4c165bd9",
"任务办理人（是候选人所以null）": "null",
"任务名称": "领导审批"
```

6. 拾取组任务

   拾取组任务也可以是候选人自己领取的任务，就是指定任务的一个办理人，候选人员拾取组任务后该任务变为自己的个人任务

```java
@Autowired
private TaskService taskService;

/** 拾取候选任务 */
@PostMapping("/claimTask")
public String claimTask() {
    // 任务
    String taskId = "dba5abef-1c5c-11ed-8bae-509a4c165bd9";
    // 候选人id
    String userId = "meng";
    // 拾取候选任务，将候选人meng变为个人任务
    taskService.claim(taskId, userId);
    return userId + "任务拾取成功";
}
```

7. 查询个人代办任务

   再使用 `assignee` 来查询，就可以查询到了

```java
@Autowired
private TaskService taskService;

/** 查询个人代办任务 */
@PostMapping("/getUserTask")
public List<Map<String, Object>> getUserTask() {
    // 流程定义key
    String processDefinitionKey = "testGroupTask";

    // 任务办理人
    String assignee = "meng";

    // 查询代办任务
    List<Task> list = taskService.createTaskQuery()
        .processDefinitionKey(processDefinitionKey)
        .taskAssignee(assignee)
        .list();
    return list.stream().map(task -> {
        Map<String, Object> map = new HashMap<>();
        map.put("任务id", task.getId());
        map.put("代理人", task.getAssignee());
        map.put("任务名", task.getName());
        return map;
    }).collect(Collectors.toList());
}
```

```
"代理人": "meng",
"任务名": "领导审批",
"任务id": "dba5abef-1c5c-11ed-8bae-509a4c165bd9"
```

   可以看到代理人已经不是null了，而是被设置成了meng，这时候使用xue，gu去查询候选人任务就没有组任务了

8. 归还任务

   如果个人不想办理该组任务，可以归还组任务，归还后该用户不再是该任务的负责人，这时所有候选人就都可以拾取任务

   建议：归还任务前校验该用户是否是该任务的负责人

```java
@Autowired
private TaskService taskService;

/** 归还组任务 */
@PostMapping("/assigneeToGroupTask")
public String assigneeToGroupTask() {
    // 当前任务
    String taskId = "dba5abef-1c5c-11ed-8bae-509a4c165bd9";
    // 任务办理人
    String assignee = "meng";

    // 校验assignee是否为该任务的负责人，如果是负责人才可以归还组任务
    Task task = taskService.createTaskQuery()
        .taskId(taskId)
        .taskAssignee(assignee)
        .singleResult();

    if (task != null) {
        // 第2个参数代理人置为null，则归还组任务
        taskService.setAssignee(taskId, null);
    }

    return "归还任务完成：" + taskId;
}
```

9. 转办任务

   任务办理人将任务转交给其它候选人办理该任务，最好转换前先查询当前任务是指定办理人的任务，是再转接

```java
@Autowired
private TaskService taskService;

/** 转办任务 */
@PostMapping("/turnTask")
public String turnTask() {
    // 当前代办任务
    String taskId = "dba5abef-1c5c-11ed-8bae-509a4c165bd9";
    // 任务办理人
    String assignee = "meng";
    // 转接的目标候选人
    String candidateUser = "xue";

    // 校验assignee是否为该任务的负责人，如果是负责人才可以归还组任务
    Task task = taskService.createTaskQuery()
        .taskId(taskId)
        .taskAssignee(assignee)
        .singleResult();

    if (task != null) {
        // 第2个参数为转接其他代理人
        taskService.setAssignee(taskId, candidateUser);
    }

    return "任务转接给：" + candidateUser;
}
```

10. 办理个人任务

```
@Autowired
private TaskService taskService;

@PostMapping("/completeTask")
public String completeTask() {
    // 任务id
    String taskId = "dba5abef-1c5c-11ed-8bae-509a4c165bd9";
    taskService.complete(taskId);
    return "完成任务: " + taskId;
}
```

11. 数据库操作
    1. 查询当前任务执行表，当前是组任务，所有 assignee 为空，当拾取任务后该字段就是拾取用户的 id

       ```
       SELECT * FROM act_ru_task
       ```

    2. 查询任务参与者，记录当前参考任务用户或组，当前任务如果设置了候选人，会向该表插入候选人记录，有几个候选人就插入几条记录

       ```
       SELECT * FROM act_ru_identitylink
       ```

       注意：当act_ru_identitylink插入记录的同时，也会向act_hi_identitylink历史表插入记录

十一、历史任务管理HistoryService

11.1 查询指定用户的已处理任务

```
@Autowired
private HistoryService historyService;

/** 查询当前用户的已处理任务 */
@PostMapping("/findCompleteTask")
public List<Map<String, Object>> findCompleteTask() {
    // 办理人（当前用户）
    String assignee = "meng";
    List<HistoricTaskInstance> taskList =
historyService.createHistoricTaskInstanceQuery()
        .taskAssignee(assignee)  // 办理人
        .orderByTaskCreateTime().desc()
        .finished()
        .list();

    return taskList.stream().map(task -> {
        Map<String, Object> map = new HashMap<>();
        map.put("任务ID", task.getId());
        map.put("任务名称", task.getName());
        map.put("任务的开始时间", task.getStartTime());
        map.put("任务的结束时间", task.getEndTime());
        map.put("任务的办理人", task.getAssignee());
        map.put("流程实例ID", task.getProcessInstanceId());
```

```
        map.put("流程定义ID", task.getProcessDefinitionId());
        map.put("业务唯一标识", task.getBusinessKey());
        return map;
    }).collect(Collectors.toList());
}
```


```
{
    "任务的结束时间": "2022-08-16T01:19:44.628+00:00",
    "流程实例ID": "dba0f0fb-1c5c-11ed-8bae-509a4c165bd9",
    "任务的开始时间": "2022-08-15T05:41:06.599+00:00",
    "流程定义ID": "testGroupTask:1:c896979a-1c5a-11ed-b0d3-509a4c165bd9",
    "任务ID": "dba5abef-1c5c-11ed-8bae-509a4c165bd9",
    "任务的办理人": "meng",
    "任务名称": "领导审批"
},
{
    "任务的结束时间": "2022-08-09T05:29:50.503+00:00",
    "流程实例ID": "381bbe6c-1709-11ed-83f3-005056c00001",
    "任务的开始时间": "2022-08-08T10:59:48.212+00:00",
    "流程定义ID": "leaveProcess:1:f79dad6f-1703-11ed-9563-005056c00001",
    "任务ID": "38200436-1709-11ed-83f3-005056c00001",
    "任务的办理人": "meng",
    "任务名称": "领导审批"
},
{
    "任务的结束时间": "2022-08-08T10:24:00.068+00:00",
    "流程实例ID": "05f05d50-1704-11ed-9563-005056c00001",
    "任务的开始时间": "2022-08-08T10:22:36.560+00:00",
    "流程定义ID": "leaveProcess:1:f79dad6f-1703-11ed-9563-005056c00001",
    "任务ID": "05f4f13a-1704-11ed-9563-005056c00001",
    "任务的办理人": "meng",
    "任务名称": "领导审批"
}
```


11.2 查询流程实例办理历史信息

```
@Autowired
private HistoryService historyService;

/** 查询流程实例办理历史信息 */
@PostMapping("/historyInfo")
public List<Map<String, Object>> historyInfo() {
    // 获取历史节点查询对象 ACT_HI_ACTINST表
    HistoricActivityInstanceQuery query =
historyService.createHistoricActivityInstanceQuery();

    // 实例id
    String processInstanceId = "dba0f0fb-1c5c-11ed-8bae-509a4c165bd9";
    List<HistoricActivityInstance> list =
query.processInstanceId(processInstanceId)
        .orderByHistoricActivityInstanceStartTime().asc()  // 根据开始时间排序asc升序
        .list();
    return list.stream().map(hi -> {
        Map<String, Object> map = new HashMap<>();
        map.put("流程定义ID", hi.getProcessDefinitionId());
```

```
        map.put("流程实例ID", hi.getProcessInstanceId());
        map.put("节点ID", hi.getActivityId());
        map.put("节点名称", hi.getActivityName());
        map.put("任务办理人", hi.getAssignee());
        map.put("开始时间", hi.getStartTime());
        map.put("结束时间", hi.getEndTime());
        return map;
    }).collect(Collectors.toList());
}
```


```
[
    {
        "开始时间": "2022-08-15T05:41:06.576+00:00",
        "流程实例ID": "dba0f0fb-1c5c-11ed-8bae-509a4c165bd9",
        "节点ID": "sid-8B8F7A34-568C-4D20-A5D5-98B003FAE942",
        "流程定义ID": "testGroupTask:1:c896979a-1c5a-11ed-b0d3-509a4c165bd9",
        "结束时间": "2022-08-15T05:41:06.579+00:00",
        "节点名称": "开始"
    },
    {
        "任务办理人": "meng",
        "开始时间": "2022-08-15T05:41:06.583+00:00",
        "流程实例ID": "dba0f0fb-1c5c-11ed-8bae-509a4c165bd9",
        "节点ID": "sid-875A02AF-4742-45FF-87E0-CB3096FCCBD5",
        "流程定义ID": "testGroupTask:1:c896979a-1c5a-11ed-b0d3-509a4c165bd9",
        "结束时间": "2022-08-16T01:19:44.682+00:00",
        "节点名称": "领导审批"
    },
    {
        "开始时间": "2022-08-16T01:19:44.692+00:00",
        "流程实例ID": "dba0f0fb-1c5c-11ed-8bae-509a4c165bd9",
        "节点ID": "sid-42B4D51E-3958-4A76-BDCE-F5ECA335FE7B",
        "流程定义ID": "testGroupTask:1:c896979a-1c5a-11ed-b0d3-509a4c165bd9",
        "结束时间": "2022-08-16T01:19:44.695+00:00",
        "节点名称": "结束"
    }
]
```


11.3 查询已结束的流程实例

```
@Autowired
private HistoryService historyService;

/** 查询已结束的流程实例 */
@PostMapping("/getProcInstListFinish")
public List<Map<String, Object>> getProcInstListFinish() {
    List<HistoricProcessInstance> instanceList =
historyService.createHistoricProcessInstanceQuery()
        .orderByProcessInstanceEndTime()
        .desc()
        .finished()
        .list();

    return instanceList.stream().map(instance -> {
        Map<String, Object> map = new HashMap<>();
```

```
        map.put("流程实例id", instance.getId());
        map.put("流程实例名称", instance.getName());
        map.put("流程唯一key", instance.getProcessDefinitionKey());
        map.put("流程定义版本号", instance.getProcessDefinitionVersion());
        map.put("流程发起人", instance.getStartUserId());
        map.put("业务唯一标识", instance.getBusinessKey());
        map.put("开始时间", instance.getStartTime());
        map.put("结束时间", instance.getEndTime());
        map.put("原因详情", instance.getDeleteReason());
        return map;
    }).collect(Collectors.toList());
}
```

```
"开始时间": "2022-08-15T05:41:06.568+00:00",
"流程实例id": "dba0f0fb-1c5c-11ed-8bae-509a4c165bd9",
"业务唯一标识": "4444",
"结束时间": "2022-08-16T01:19:44.775+00:00",
"流程唯一key": "testGroupTask",
"流程定义版本号": 1
```

11.4 删除已结束流程实例历史记录

```
/**
  * 删除已结束流程实例历史记录
  * ACT_HI_DETAIL
  * ACT_HI_VARINST
  * ACT_HI_TASKINST
  * ACT_HI_PROCINST
  * ACT_HI_ACTINST
  * ACT_HI_IDENTITYLINK
  * ACT_HI_COMMENT
  */
@PostMapping("/deleteFinishProcInst")
public String deleteFinishProcInst() {
    String procInstId = "dba0f0fb-1c5c-11ed-8bae-509a4c165bd9";
    // 1.查询流程实例是否已结束
    HistoricProcessInstance instance =
historyService.createHistoricProcessInstanceQuery()
        .processInstanceId(procInstId)
        .finished()
        .singleResult();

    if (instance == null) {
        return "流程实例未结束或不存在";
    }

    // 2. 删除已结束流程实例（如果实例未结束，会抛出异常）
    historyService.deleteHistoricProcessInstance(procInstId);
    return "删除流程实例成功";
}
```

十二、流程网关管理

网关主要控制Activiti流程分支的流向

12.1 互斥网关ExclusiveGateway

概念：**互斥网关**用于对流程中的决策进行建模，当流程执行到达此网关时，所有分支都将按定义顺序进行判断，选择判断条件结果为真(true)的分支继续处理

注意：互斥网关时只选择一个条件值为true的分支进行处理，如果多个分支的条件计算结果都为true，互斥网关会选择id值较小的一条分支去执行。**简而言之，就是传递流程变量，排它网关根据流程变量决定该走哪条线路**

12.1.1 流程定义模型

12.1.2 需求

`直接领导审批后，如果小于等于3天的由行政审批；大于3天的由总经理审批，总经理审批完再由行政审批`

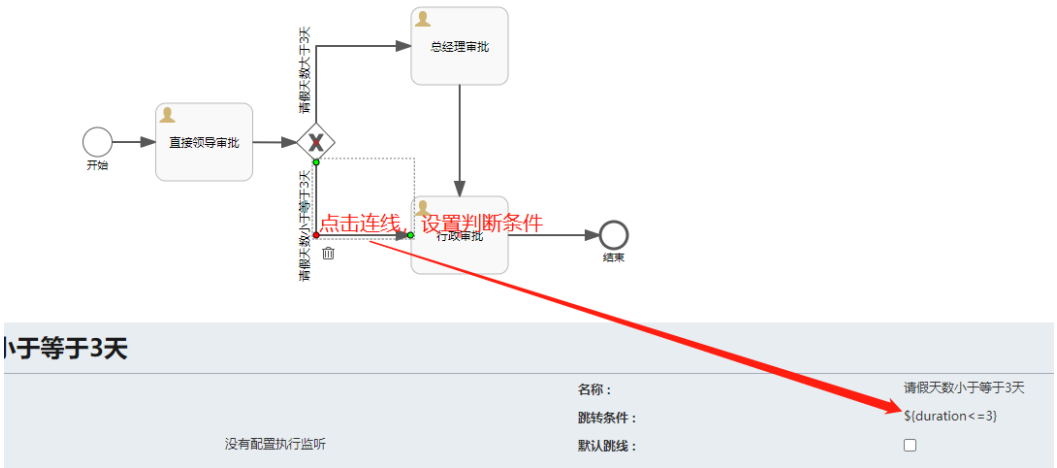
说明：**http://localhost:8080/workflow/model/create**

画一个流程定义模型：

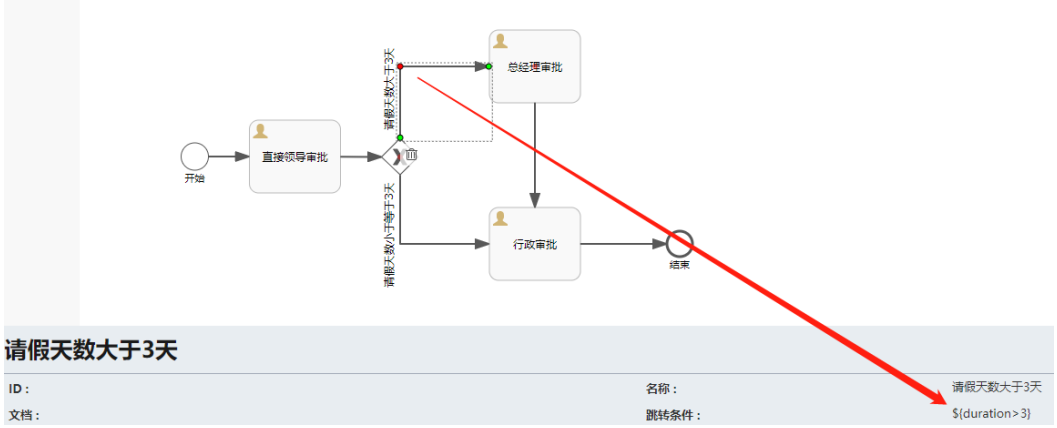
- 流程唯一标识：testExclusiveGateway
- 流程名：互斥网关流程模型
- 流程定义里用户任务节点的办理人分别是：
  - 直接领导审批 `meng`
  - 行政审批 `xue`
  - 总经理 `gu`

12.1.3 画流程模型

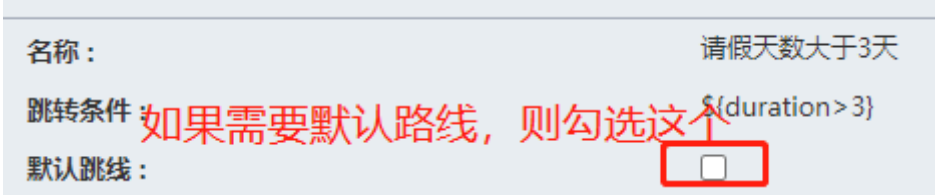
1. 点击连线设置跳转条件 `${duratin<=3}`，判断请假天数大于3天则流向总经理审批



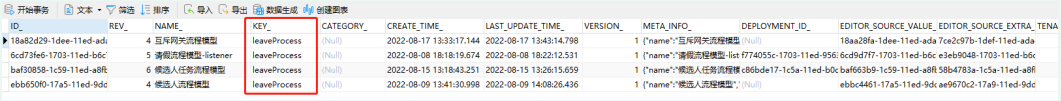
2. 点击连线设置跳转条件 `${duratin>3}` ，判断请假天数小于等于3天则流向行政审批



3. 如果需要默认执行路径，则点击连线，然后**勾选**默认跳线



问题：查看表act_re_model，发现KEY_都是“leaveProcess”



ID_	REV_	NAME_	KEY_	CATEGORY_	CREATE_TIME_	LAST_UPDATE_TIME_	VERSION_	META_INFO_	DEPLOYMENT_ID_	EDITOR_SOURCE_VALUE_	EDITOR_SOURCE_EXTRA_	TENA
18a82d29-1dee-11ed-ad	4	互斥网关流程模型	leaveProcess	(Null)	2022-08-17 13:33:17.144	2022-08-17 13:43:14.798	1	{"name":"互斥网关流程模型	(Null)	18aa28fa-1dee-11ed-ada	7ce2c97b-1def-11ed-ada	
6cd73fe6-1703-11ed-b6c	5	请假流程模型-listener	leaveProcess	(Null)	2022-08-08 18:18:19.674	2022-08-08 18:22:12.531	1	{"name":"请假流程模型-list	f774055c-1703-11ed-956	6cd9d7f7-1703-11ed-b6c	e3eb9048-1703-11ed-b6c	
baf30858-1c59-11ed-a8f	6	候选人任务流程模型	leaveProcess	(Null)	2022-08-15 13:18:43.251	2022-08-15 13:26:15.659	1	{"name":"候选人任务流程模	c86bde17-1c5a-11ed-b0c	baf663b9-1c59-11ed-a8f	58b4783a-1c5a-11ed-a8f	
ebb650f0-17a5-11ed-9dc	4	候选人流程模型	leaveProcess	(Null)	2022-08-09 13:41:30.998	2022-08-09 14:08:26.436	1	{"name":"候选人流程模型","(Null)		ebbc4461-17a5-11ed-9dc	ae9670c2-17a9-11ed-9dc	

解决方法：

在接口ModelSaveRestResource里方法saveModel里 `model.setName(values.getFirst("name"));` 后面添加以下代码：

```
// 获取唯一标识key
String key =
objectMapper.readTree(values.getFirst("json_xml")).get("properties").get("pr
ocess_id").textValue();
model.setKey(key);
```

12.1.4 代码测试

1. 部署流程定义：**http://localhost:8080/workflow/model/deploy**，注意方法的模型id值
2. 启动流程

   注意启动流程实例的时候要设置请假天数(duration)变量，有了这个变量流程图的网关才会决定走哪条路

```
@Autowired
private RuntimeService runtimeService;

/** 启动流程实例 */
@PostMapping("/startProcess")
public String startProcess() {
    // 流程变量
    Map<String, Object> variables = new HashMap<>();
    // 天数
    variables.put("duration", 2);
    // 启动流程实例
    runtimeService.startProcessInstanceByKey("testExclusiveGateway",
variables);
    return "启动流程成功";
}
```

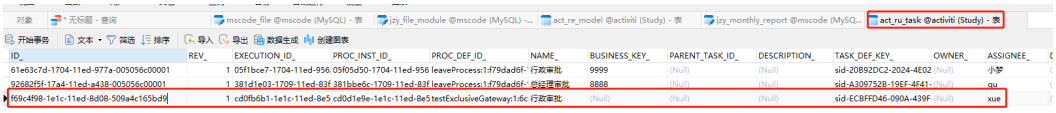
3. 查询任务

```
@Autowired
private TaskService taskService;

/** 查询指定办理人的待办任务 */
@RequestMapping("/findWaitTask")
public List<Map<String, Object>> findWaitTask() {
    // 办理人
    String assignee = "meng";
    // 指定个人任务查询
    List<Task> taskList = taskService.createTaskQuery()
        .taskAssignee(assignee)
        .processDefinitionKey("testExclusiveGateway")
        .list();
    return taskList.stream().map(task -> {
        Map<String, Object> map = new HashMap<>();
        map.put("任务id", task.getId());
        map.put("任务名称", task.getName());
        map.put("流程状态", (task.isSuspended() ? "已挂起" : "已激活"));
        map.put("任务的办理人", task.getAssignee());
        return map;
    }).collect(Collectors.toList());
}
```

4. 完成任务

```
@Autowired
private TaskService taskService;

/** 完成任务 */
@RequestMapping("/completeTask")
public String completeTask() {
    taskService.complete("cd166d74-1e1c-11ed-8e50-509a4c165bd9");
    return "完成任务";
}
```

上面设置的是2天，直接领导完成后，就到了行政审批的办理人xue了，如下图：



ID_	REV_	EXECUTION_ID_	PROC_INST_ID_	PROC_DEF_ID_	NAME_	BUSINESS_KEY_	PARENT_TASK_ID_	DESCRIPTION_	TASK_DEF_KEY_	OWNER_	ASSIGNEE_
61e63c7d-1704-11ed-977a-005056c00001	1	05f1bce7-1704-11ed-956	05f05d50-1704-11ed-956	leaveProcess:1:f79dad6f-	行政审批	9999	(Null)	(Null)	sid-20B92DC2-2024-4E02	(Null)	小梦
92682f5f-17a4-11ed-a438-005056c00001	1	381d1e03-1709-11ed-83f	381bbe6c-1709-11ed-83f	leaveProcess:1:f79dad6f-	总经理审批	8888	(Null)	(Null)	sid-A3097528-19EF-4F41-	(Null)	qu
f69c4f98-1e1c-11ed-8d08-509a4c165bd9	1	cd0fb6b1-1e1c-11ed-8e5	cd0d1e9e-1e1c-11ed-8e5	testExclusiveGateway:1:6c	行政审批	(Null)	(Null)	(Null)	sid-ECBFFD46-090A-439F	(Null)	xue

后面处理流程就跟正常流程一样了

12.2 并行网关ParallelGateway

12.2.1 概念

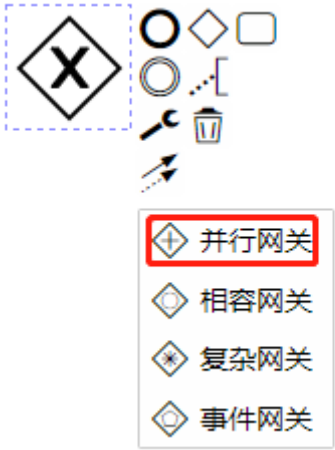
并行网关用于对流程中的并发性建模，在流程模型中引入并发性的最简单的网关是并行网关，将流程分成多条分支，然后多条分支汇聚到一起

- 分支(fork)：并行所有外出顺序流，为每个顺序流都创建一个并发分支
- 汇聚(join)：所有到达并行网关，在此等待的进入分支，只有当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

> - 分支与汇聚成对出现。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
> - 并行网关在业务应用中常用于会签任务，会签任务即多个参与者共同办理的任务

12.2.2 流程定义模型

bpmn并行网关符号：



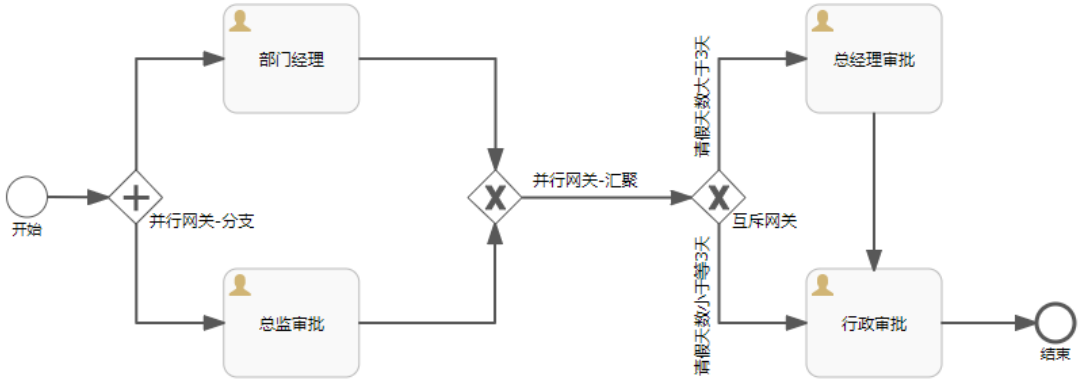
说明：**http://localhost:8080/workflow/model/create**

流程核心：开启流程实例->分成两个分支(部门经理审核，总监审批)->两个分支任务都完成了->汇聚分支

属性：流程唯一标识key：testParallelGateway，流程名：并行网关流程模型

流程定义里用户任务节点的办理人分别是：

- 部门经理审批meng
- 总监审批xue
- 总经理gu
- 行政审批 小梦



12.2.3 代码测试

1. **部署流程定义**：**http://localhost:8080/workflow/model/deploy**，注意方法的模型id值，然后执行方法
2. **启动流程**

```
@Autowired
private RuntimeService runtimeService;

/** 启动流程实例 */
@PostMapping("/startProcess")
public String startProcess() {
    // 流程变量
    Map<String, Object> variables = new HashMap<>();
    // 天数
    variables.put("duration", 2);
    // 启动流程实例
    runtimeService.startProcessInstanceByKey("testParallelGateway",
variables);

    return "启动流程成功";
}
```

启动流程后，到数据库查询数据：

- 查询act_ru_execution正在执行实例3条：

ID_	REV_	PROC_INST_ID_	BUSINESS_KEY_	PARENT_ID_	PROC_DEF_ID_
05f05d50-1704-11ed-9563-005056c00001	1	05f05d50-1704-11ed-956	9999	(Null)	leaveProcess:1:f79dad6f-1703-11
05f1bce7-1704-11ed-9563-005056c00001	4	05f05d50-1704-11ed-956	(Null)	05f05d50-1704-11ed-9563-005056c00001	leaveProcess:1:f79dad6f-1703-11
381bbe6c-1709-11ed-83f3-005056c00001	1	381bbe6c-1709-11ed-83f	8888	(Null)	leaveProcess:1:f79dad6f-1703-11
381d1e03-1709-11ed-83f3-005056c00001	3	381bbe6c-1709-11ed-83f	(Null)	381bbe6c-1709-11ed-83f3-005056c00001	leaveProcess:1:f79dad6f-1703-11
9a419c3b-1eb8-11ed-9f0f-509a4c165bd9	1	9a419c3b-1eb8-11ed-9f0	(Null)	(Null)	testParallelGateway:5:95c5ff2a-1el
9a425f8e-1eb8-11ed-9f0f-509a4c165bd9	1	9a419c3b-1eb8-11ed-9f0	(Null)	9a419c3b-1eb8-11ed-9f0f-509a4c165bd9	testParallelGateway:5:95c5ff2a-1el
9a45b9f1-1eb8-11ed-9f0f-509a4c165bd9	1	9a419c3b-1eb8-11ed-9f0	(Null)	9a419c3b-1eb8-11ed-9f0f-509a4c165bd9	testParallelGateway:5:95c5ff2a-1el

  可以看到流程实例ID都是一样的，说明它们是一个流程的任务实例

  只不过下面两条实例有一个PARENT_ID就是第一个实例的id

  可以理解为，第1条实例就是启动的流程实例，下面两条是正在运行的任务实例

- 查询act_ru_task正在执行任务有2条

ID_	REV_	EXECUTION_ID_	PROC_INST_ID_	PROC_DEF_ID_	NAME_
61e63c7d-1704-11ed-977a-005056c00001	1	05f1bce7-1704-11ed-956	05f05d50-1704-11ed-956	leaveProcess:1:f79dad6f-1703-11ed-9563-005056c00001	行政审批
92682f5f-17a4-11ed-a438-005056c00001	1	381d1e03-1709-11ed-83f	381bbe6c-1709-11ed-83f	leaveProcess:1:f79dad6f-1703-11ed-9563-005056c00001	总经理审批
9a4803e3-1eb8-11ed-9f0f-509a4c165bd9	1	9a425f8e-1eb8-11ed-9f0f	9a419c3b-1eb8-11ed-9f0	testParallelGateway:5:95c5ff2a-1eb8-11ed-9f0f-509a4c165bd9	部门经理
9a482af5-1eb8-11ed-9f0f-509a4c165bd9	1	9a45b9f1-1eb8-11ed-9f0f	9a419c3b-1eb8-11ed-9f0	testParallelGateway:5:95c5ff2a-1eb8-11ed-9f0f-509a4c165bd9	总监审批

部门经理、总监审批是当前并行执行的任务

3. **完成并行任务**

   完成并行任务不分前后，由任务的负责人去执行即可

   比如：当部门经理审批完成任务后，查询act_ru_task运行中任务表，发现在act_ru_task表的部门经理任务数据已被删除

ID_	REV_	EXECUTION_ID_	PROC_INST_ID_	PROC_DEF_ID_	NAME_
61e63c7d-1704-11ed-977a-005056c00001	1	05f1bce7-1704-11ed-956	05f05d50-1704-11ed-956	leaveProcess:1:f79dad6f-1703-11ed-9563-005056c00001	行政审批
92682f5f-17a4-11ed-a438-005056c00001	1	381d1e03-1709-11ed-83f	381bbe6c-1709-11ed-83f	leaveProcess:1:f79dad6f-1703-11ed-9563-005056c00001	总经理审批
9a482af5-1eb8-11ed-9f0f-509a4c165bd9	1	9a45b9f1-1eb8-11ed-9f0f	9a419c3b-1eb8-11ed-9f0	testParallelGateway:5:95c5ff2a-1eb8-11ed-9f0f-509a4c165bd9	总监审批

12.3 相容网关InclusiveGateway

12.3.1 概念

相容网关可以看做是互斥网关和并行网关的结合体。和互斥网关一样，你可以在连线上定义条件表达式，定义的表达式会被包含网关解析。但是相容网关主要区别是也可以不指定表达式，同时选择多条分支，这和并行网关是一样的

简而言之：

- 互斥网关：解析条件，流向条件为true的节点，或者报错(没有满足条件的)
- 并行网关：不解析条件，不报错，分支节点都执行
- 相容网关：有条件就解析条件，条件成立就流向，不成立就不流向，全都不成立报错；没有条件直接流向

12.3.2 流程定义模型

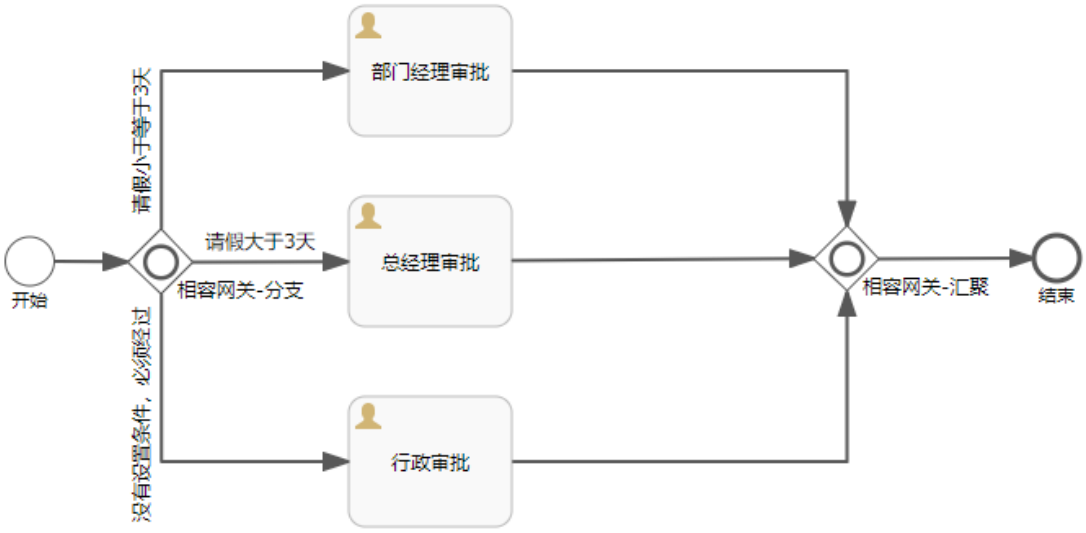
说明：**http://localhost:8080/workflow/model/create**

流程核心：开启请假流程实例->相容网关分支->如果小于等于3天的部门经理审批，大于3天的有总经理审批，请假必须经过行政审批->汇聚分支->流程结束

属性：流程唯一标识key：`testInclusiveGateway`，流程名：`相容网关流程模型`

流程定义里用户任务节点的办理人分别是：

- 部门经理审批 meng
- 总经理 xue
- 行政审批 gu



12.3.4 代码测试

1. 部署流程定义：**http://localhost:8080/workflow/model/deploy**，注意方法的模型id值，然后执行方法
2. 启动流程实例

```
@Autowired
private RuntimeService runtimeService;

@RequestMapping("/startProcess")
public String startProcess() {
    // 流程变量
    Map<String, Object> variables = new HashMap<>();
    // 请假天数
    variables.put("duration", 2);
    // 启动流程实例
    runtimeService.startProcessInstanceByKey("testInclusiveGateway",
variables);

    return "启动流程实例成功";
}
```

启动流程后，到数据库查询数据：

- 查询act_ru_execution正在执行实例3条：

ID_	REV_	PROC_INST_ID_	BUSINESS_KE	PARENT_ID_	PROC_DEF_ID_
05f05d50-1704-11ed-9563-005056c00001	1	05f05d50-1704-11ed-956	9999	(Null)	leaveProcess:1:f79dad6f-1703-11ed-9563-005056c00001
05f1bce7-1704-11ed-9563-005056c00001	4	05f05d50-1704-11ed-956	(Null)	05f05d50-1704-11ed-9563-005056c00001	leaveProcess:1:f79dad6f-1703-11ed-9563-005056c00001
381bbe6c-1709-11ed-83f3-005056c00001	1	381bbe6c-1709-11ed-83f	8888	(Null)	leaveProcess:1:f79dad6f-1703-11ed-9563-005056c00001
381d1e03-1709-11ed-83f3-005056c00001	3	381bbe6c-1709-11ed-83f	(Null)	381bbe6c-1709-11ed-83f3-005056c00001	leaveProcess:1:f79dad6f-1703-11ed-9563-005056c00001
9a419c3b-1eb8-11ed-9f0f-509a4c165bd9	2	9a419c3b-1eb8-11ed-9f0	(Null)	(Null)	testParallelGateway:5:95c5ff2a-1eb8-11ed-9f0f-509a4c165bd9
9a425f8e-1eb8-11ed-9f0f-509a4c165bd9	2	9a419c3b-1eb8-11ed-9f0	(Null)	9a419c3b-1eb8-11ed-9f0f-509a4c165bd9	testParallelGateway:5:95c5ff2a-1eb8-11ed-9f0f-509a4c165bd9
9a45b9f1-1eb8-11ed-9f0f-509a4c165bd9	1	9a419c3b-1eb8-11ed-9f0	(Null)	9a419c3b-1eb8-11ed-9f0f-509a4c165bd9	testParallelGateway:5:95c5ff2a-1eb8-11ed-9f0f-509a4c165bd9
cbb2fa2a-1ec8-11ed-a28f-509a4c165bd9	1	cbb2fa2a-1ec8-11ed-a28	(Null)	(Null)	testInclusiveGateway:1:ca0941c9-1ec8-11ed-a28f-509a4c165bd9
cbb3bd7d-1ec8-11ed-a28f-509a4c165bd9	1	cbb2fa2a-1ec8-11ed-a28	(Null)	cbb2fa2a-1ec8-11ed-a28f-509a4c165bd9	testInclusiveGateway:1:ca0941c9-1ec8-11ed-a28f-509a4c165bd9
cbbac260-1ec8-11ed-a28f-509a4c165bd9	1	cbb2fa2a-1ec8-11ed-a28	(Null)	cbb2fa2a-1ec8-11ed-a28f-509a4c165bd9	testInclusiveGateway:1:ca0941c9-1ec8-11ed-a28f-509a4c165bd9

  可以看到流程实例ID都是一样的，说明它们是一个流程的任务实例

  只不过下面两条实例有一个PARENT_ID就是第一个实例的id

  可以理解为，第1条实例就是启动的流程实例，下面两条是正在运行的分支任务实例

- 查询act_ru_task正在执行任务有2条

ID_	REV_	EXECUTION_ID_	PROC_INST_ID_	PROC_DEF_ID_	NAME_
61e63c7d-1704-11ed-977a-005056c00001	1	05f1bce7-1704-11ed-956	05f05d50-1704-11ed-956	leaveProcess:1:f79dad6f-1703-11ed-9563-005056c00001	行政审批
92682f5f-17a4-11ed-a438-005056c00001	1	381d1e03-1709-11ed-83f	381bbe6c-1709-11ed-83f	leaveProcess:1:f79dad6f-1703-11ed-9563-005056c00001	总经理审批
9a482af5-1eb8-11ed-9f0f-509a4c165bd9	1	9a45b9f1-1eb8-11ed-9f0f	9a419c3b-1eb8-11ed-9f0	testParallelGateway:5:95c5ff2a-1eb8-11ed-9f0f-509a4c165bd9	总监审批
cbbb3792-1ec8-11ed-a28f-509a4c165bd9	1	cbb3bd7d-1ec8-11ed-a2	cbb2fa2a-1ec8-11ed-a28	testInclusiveGateway:1:ca0941c9-1ec8-11ed-a28f-509a4c165bd	部门经理审批
cbbb5ea4-1ec8-11ed-a28f-509a4c165bd9	1	cbbac260-1ec8-11ed-a28	cbb2fa2a-1ec8-11ed-a28	testInclusiveGateway:1:ca0941c9-1ec8-11ed-a28f-509a4c165bd	行政审批

只有部门经理审批、行政审批。没有总经理审批

3. 完成相容任务

```
@RequestMapping("/completeTask")
public String completeTask() {
    taskService.complete("cbbb3792-1ec8-11ed-a28f-509a4c165bd9");

    return "完成审批任务";
}
```

完成相容任务不分前后，由任务的负责人去执行即可

比如：当部门经理审批完成任务后，查询act_ru_task运行中任务表，发现act_ru_task表的部门经理任务数据已被删除

ID_	REV_	EXECUTION_ID_	PROC_INST_ID_	PROC_DEF_ID_	NAME_
61e63c7d-1704-11ed-977a-005056c00001	1	05f1bce7-1704-11ed-956	05f05d50-1704-11ed-956	leaveProcess:1:f79dad6f-1703-11ed-9563-005056c00001	行政审批
92682f5f-17a4-11ed-a438-005056c00001	1	381d1e03-1709-11ed-83f	381bbe6c-1709-11ed-83f	leaveProcess:1:f79dad6f-1703-11ed-9563-005056c00001	总经理审批
9a482af5-1eb8-11ed-9f0f-509a4c165bd9	1	9a45b9f1-1eb8-11ed-9f0f	9a419c3b-1eb8-11ed-9f0	testParallelGateway:5:95c5ff2a-1eb8-11ed-9f0f-509a4c165bd9	总监审批
cbbb5ea4-1ec8-11ed-a28f-509a4c165bd9	1	cbbac260-1ec8-11ed-a28	cbb2fa2a-1ec8-11ed-a28	testInclusiveGateway:1:ca0941c9-1ec8-11ed-a28f-509a4c165bd	行政审批
f69c4f98-1e1c-11ed-8d08-509a4c165bd9	1	cd0fb6b1-1e1c-11ed-8e5	cd0d1e9e-1e1c-11ed-8e5	testExclusiveGateway:1:6c8a8c2e-1e1c-11ed-a8ec-509a4c165b	行政审批

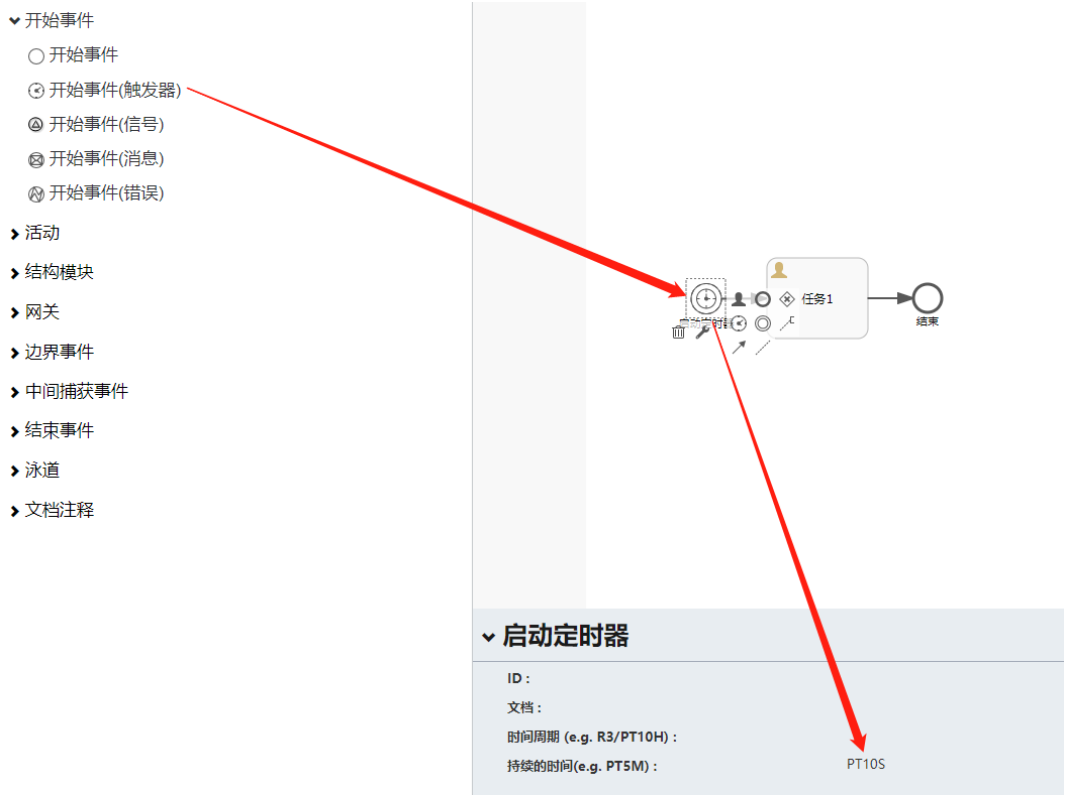
只有行政审批任务了

十三、定时器时间和邮件任务

涉及核心表：ACT_RU_TIMER_JOB

13.1 开始定时器事件

- 可以设置时间，定时开始启动流程实例
- 部署流程后，不用startProcessInstanceByXxxx启动流程实例，而是到达设定时间后自动启动
- 到达设定时间后，查询下是否有对应的流程实例



- **触发时间(ISO-8601格式标准)：**

表示何时触发；标签元素 `<timeDate>`，时间格式是ISO 8601的固定格式，比如：

```
<timerEventDefinition>
    <timeDate>2022-08-18T19:08:18</timeDate>
</timerEventDefinition>
```

`2022-08-18T19:08:18`，T是日期和时间分割标记

- **持续时间(例如 PT5M)：**

  表示定时器经过多少时间后触发；标签元素 `<timeDuration>`，时间格式为ISO 8601的PT格式，比如：

  ```
  <timerEventDefinition>
      <timeDuration>2022-08-18T19:08:18</timeDuration>
  </timerEventDefinition>
  ```

  要设置 `一年两个月三天四小时五分六秒`，可以写成 `P1Y2M3DT4H5M6S`

  P是开始标记，T是日期和时间分割标记。没有日期只有时间T是不能省去，只有日期没有时间T直接省去

  比如：1小时候执行应该写成 `PT1H`，10秒后执行 `PT10S`，3天后执行 `P3D`。

- **时间周期(e.g. R3/PT10H)**

  表示重复触发的间隔时间；标签元素 `<timeCycle>`，时间格式为ISO 8601的PT格式或Rn格式或变量，比如：

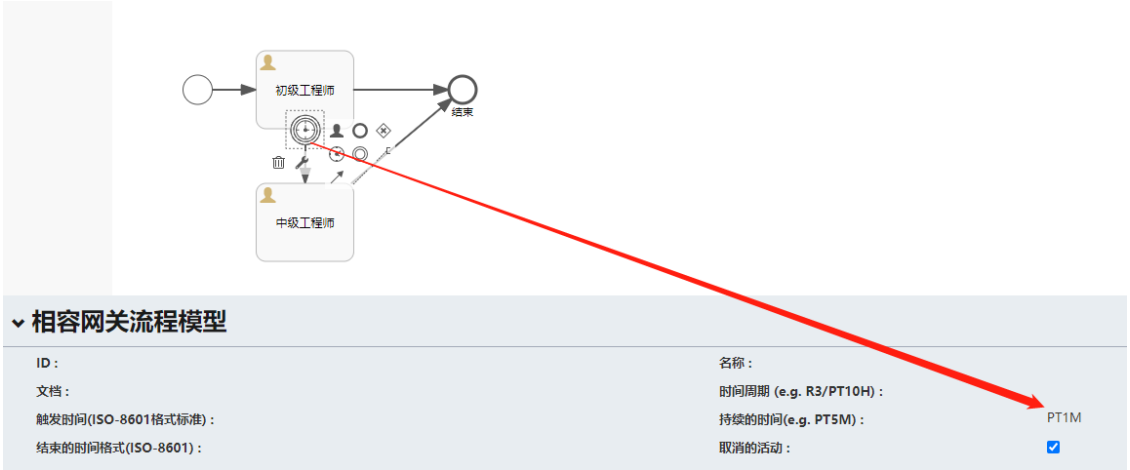
  ```
  <timerEventDefinition>
      <timeCycle>R3/PT10H</timeCycle>
  </timerEventDefinition>
  ```

  - R表示永远重复；R1重复一次；R231重复231次
  - R3/PT10H表示重复3次每次间隔10小时

13.2 边界定时器事件

用于向某节点上添加边界定时事件

需求：模拟修一部手机，先交给初级工程师修理，定时1分钟后没有修好就给中级工程师修理



- **触发时间(ISO-8601格式标准)：**

  表示何时触发；标签元素 `<timeDate>`，时间格式是ISO 8601的固定格式，比如：

  ```
  <timerEventDefinition>
      <timeDate>2022-08-18T19:08:18</timeDate>
  </timerEventDefinition>
  ```

  `2022-08-18T19:08:18`，T是日期和时间分割标记

- **持续时间(例如 PT5M)：**

  表示定时器经过多少时间后触发；标签元素 `<timeDuration>`，时间格式为ISO 8601的PT格式，比如：

  ```
  <timerEventDefinition>
      <timeDuration>2022-08-18T19:08:18</timeDuration>
  </timerEventDefinition>
  ```

  要设置 一年两个月三天四小时五分六秒，可以写成 `P1Y2M3DT4H5M6S`

  P是开始标记，T是日期和时间分割标记。没有日期只有时间T是不能省去，只有日期没有时间T直接省去

  比如：1小时候执行应该写成 `PT1H`，10秒后执行 `PT10S`，3天后执行 `P3D`。

- **时间周期(e.g. R3/PT10H)**

  表示重复触发的间隔时间；标签元素 `<timeCycle>`，时间格式为ISO 8601的PT格式或Rn格式或变量，比如：

  ```
  <timerEventDefinition>
      <timeCycle>R3/PT10H/${EndDate}</timeCycle>
  </timerEventDefinition>
  ```

  - R表示永远重复；R1重复一次；R231重复231次
  - R3/PT10H表示重复3次每次间隔10小时
  - `${EndDate}` 为变量，表示重复触发结束时间，时间格式是ISO 8601的固定格式，但是这个时间还支持cron表达式的时间

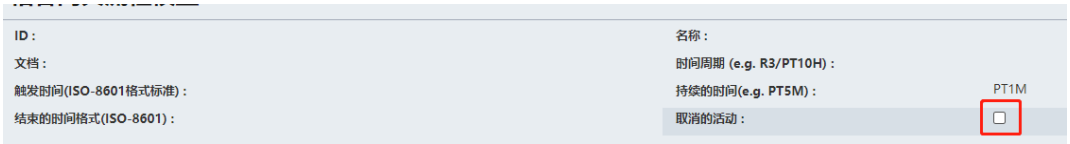
- **结束的时间格式(ISO-8601)**

  表示某个时间让重复触发的失效；会在标签元素 `<timeCycle>` 上添加包含 `activiti:endDate` 属性(该属性为可选非必须的)，格式为：

```
<timerEventDefinition>
    <timeCycle activiti:endDate="2022-08-
18T18:42:11+00:00">R3/PT4H</timeCycle>
</timerEventDefinition>
```

- **取消的活动(cancel activity)**

  设置取消的活动参数，如下：



  - 勾选取消的活动 设置持续的时间10秒，当开始流程后，如果初级工程师在10秒内没有处理完，则结束当前任务；创建分支实例走向下一节点，不在初级工程师处停留。（只能查询到下一节点任务）
  - 不勾选取消的活动，设置持续的10秒，当开始流程后，如果初级工程师在10秒内没有处理完，则不会结束当前任务，保留此任务节点，然后创建分支实例走向下一节点。（能查询到初级工程师和下一节点两个任务）
- **测试代码**