工作流-Activiti入门（api）

# Activiti概述

## 工作流是什么？

工作流（Workflow），指“业务过程的部分或整体在计算机应用环境下的自动化”。是对工作流程及其各操作步骤之间业务规则的抽象、概括描述。

通俗的说，流程（过程）就是多个人在一起合作完成某件事情的步骤，把步骤编程计算机能理解的形式，就是工作流。

工作流是研究一个群体如何在计算机的帮助下实现协同工作的，**其主要解决的问题是：为了实现某个业务目标，利用计算机在多个参与者之间按某种预定规则自动传递文档、信息或者任务。**

## 为什么要用工作流？

一个问题：我们需要工作流么？

### 原始的业务处理有弊端

原始的手工处理和软件处理流程的对比：

在公司内部的诸多表单里，包括行政、人事、IT等方面的，大部分都是采用纸张填写，手工传递或者Word / Excel文档填写，邮件发送的方式。

手工处理工作流程，

1. 一级一级审批签字难免会出现人为的失误和时间上的延时，导致工作效率低下。
2. 无法对整个流程状况进行有效跟踪、了解。
3. 无法进行量化统计，不利于查询、报表及绩效评估。

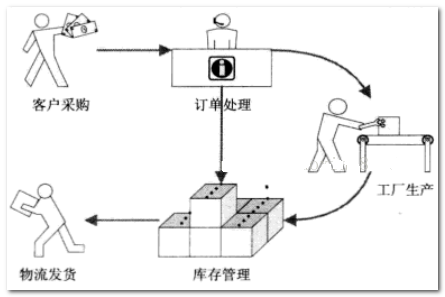
**而采用工作流软件，使用者只需在电脑上填写有关表单，会按照定义好的流程自动往下跑，下一级审批者将会收到相关资料，**并可以根据需要修改、跟踪、管理、查询、统计、打印等，大大提高了效率，实现了知识管理，提升了公司的核心竞争力。

简单的说：就是将传统的流程电子化了。

### 开发架构层面的问题

【业务场景：订货流程】

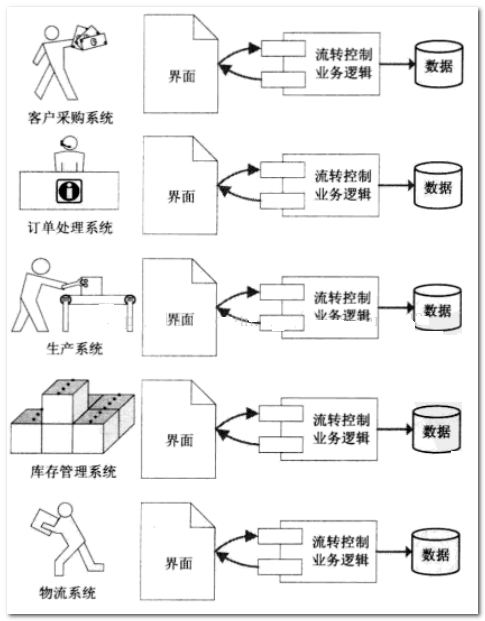
1. 客户提交采购订单
2. 业务员执行订单处理
3. 如果缺货，转工厂生产
4. 仓库出货
5. 物流发货



【不使用工作流技术：】

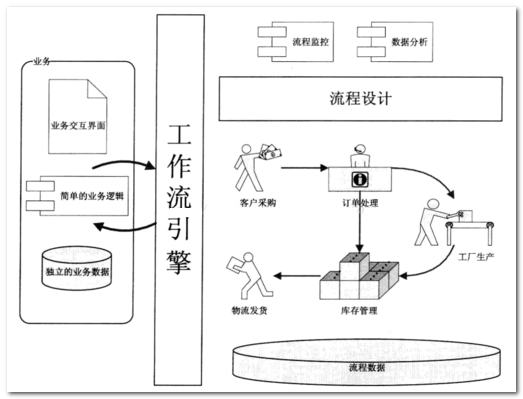
从头开始开发这个订购流程的业务逻辑，我们需要：

* 每个活动点都需要开发交互页面和后台处理程序
* 每个活动的流转都需要硬性判断下一步活动节点及其操作人
* 每次操作都需要维护业务数据和流程的相关数据



这还不包括监视、控制、分析流程的部分。

【使用工作流技术：】



这样看起来就简单多了。右侧的工作流管理系统，接管了所有订货业务在流程方面的定义和执行。这样做的好处很多：

* 使用专门的流程数据系统，维护所有涉及流程流转的数据。
* 提供“流程设计”工具，帮助用户定义订货流程的模型，而且一般都提供了可视化的界面。
* 所有的流程都依靠流程引擎来处理，避免了需求更改与硬编码之间矛盾的产生。
* 工作流引擎还提供了众多的API，可以很方便的将工作流的管理和业务操作完美结合。

### 业务需求中的问题

业务流程的不断变化变化：

乙方技术人员：你们的业务总变，你们都说不清，让我们怎么开发。

甲方业务人员：我们要是业务从来不变，每个人都能说清，要你们开发系统做什么？

双方的矛盾是不可调和的,谁都认为是对方的错,是对方在浪费时间

上述业务流程在发生变化时，就要不断的修改代码才能达到目的

而使用工作流技术，可以做到不用或者很少修改代码就能达到目标。

在流程经常变化的情况下，这种优势更为明显

### 工作流技术的好处小结

* 降低开发风险

通过使用诸如活动、流转、状态、行为这样的术语，使得业务分析师和开发人员使用同一种语言交谈成为可能。优秀的流程设计建模工具，甚至能使开发人员不必将用户需求转化成详细设计文档。

* 流程实现的集中统一

应对业务流程经常变化的情况，使用工作流技术的最大好处是使业务流程的实现代码，不再散落在各式各样的业务系统中。

* 加速开发

开发者不用再关注流程的参与者、活动节点的衔接、流转控制……因为这些工作很多被工作流框架接管了。因而开发者开发起来更快、代码出错更少、系统更加容易维护。

* 提升对迭代开发的支持

如果系统中业务流程部分被硬编码，就不容易更改，需求分析师就会花费很大的精力在开发前的业务分析中，并且希望一次成功。但可悲的是，在任何软件项目开发中，这都很少能实现。工作流管理系统使得业务流程很容易部署和重新编排，业务流程相关的应用开发可以以一种“迭代／渐进”的方式推进，也就是说工作流技术在某种程度上支持“需求分析不必一次完全成功”。

p.s: 软件都是人与人之间合作和斗争的工具，软件本身不是关键，人与人之间的矛盾和关系才是关键。

使用工作流技术后的产品，流程要能够很容易被业务人员画出来，流程还要很容易被系统自动控制。

## 工作流的技术特点

工作流只管理流程，是不能完成业务操作的。换句话说，工作流是相对独立的、通用的，与业务管理无直接关系。

比如请假流程，工作流管理的是请假的流程（谁申请，谁审批），请假的业务（请假的原因，请假的时间，，）

工作流虽然与业务无直接关系，但是，在开发的时候，需要考虑和业务整合。

## 应用的行业

工作流最常见的应用是在办公自动化的系统上。

办公自动化（Office Automation,简称OA）是将现代化办公和计算机网络功能结合起来的一种新型的办公方式。办公自动化没有统一的定义，凡是在传统的办公室中采用各种新技术、新机器、新设备从事办公业务，都属于办公自动化的领域。在行政机关中，大都把办公自动化叫做电子政务，企事业单位就都叫OA，即办公自动化。通过实现办公自动化，或者说实现数字化办公，可以优化现有的管理组织结构，调整管理体制，在提高效率的基础上，增加协同办公能力，强化决策的一致性，最后实现提高决策效能的目的。

OA-----》无纸化办公，无址化办公。

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

【阅读】

|  |
| --- |
| 工作流业务在互联网行业也存在这个业务。为什么互联网行业一般不用这个呢？  因为互联网行业的流程比较简单，没有传统行业的复杂，并且变化小。  比如：购物流程：下单》付款》卖家发货》买家收货》确认》好评。  所以一般这个流程，都是自己开发。  而传统的行业的流程比较复杂，比如政府的，种类多，流程复杂。  以前：  开个公司：工商局提交申请》核实名字》税务局》。。。。。。。》工商局》办结  该流程不只是着一种，比如，工商局内部子流程：副主任审批，主任审批，副局长审批，局长审批（流程判断—流程分支），联合审批（N个部门都要审批通过，才能走下一个流程）。  所以，我们使用工作流引擎机制，配合写的程序，可以通过画图的方式就可以基本满足我们的流程需要。 |

## 常见的工作流框架



jbpm:2006年2.0版本，被jboss收购。变的很强大。。jbpm6。

## Activiti是什么？

Activiti是一套业务流程管理(BPM)框架，它覆盖了业务流管理、工作流、服务协助等领域，它是一个开源的（Apache许可2.0）、灵活的、易扩展的可执行流程语言框架。

Activiti是Alfresco软件在2010年5月17日宣布启动的一个开源项目，其创始人和首席架构师由业务流程管理BPM的专家 Tom Baeyens担任。Tom Baeyens是JBoss jBPM的项目架构师。

提示：Jbpm4版本之后，该作者才做的Activiti，所以，Activiti很像jbpm4，理念又有点像hibernate。但是，jbpm5与4版本的差距比较大。

它是一项新的基于Apache许可的开源BPM平台，从基础开始构建，提供支持新的BPMN 2.0标准。（直观的认识就是XML）

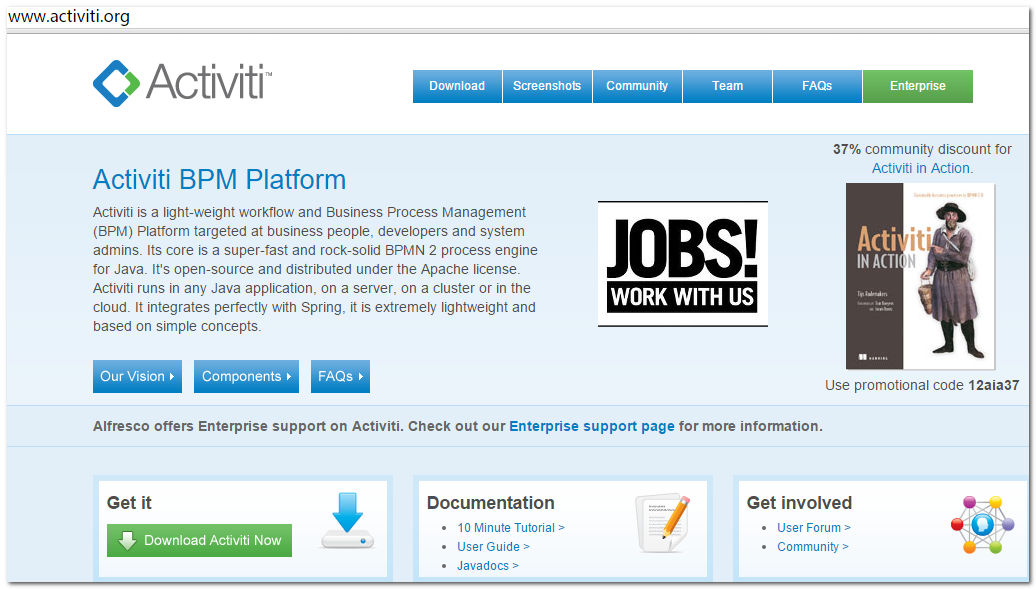
Activiti提供了流程设计器，开发人员可以直接通过流程设计器直接画出业务流程图。

Activiti的持久层使用的Mybatis。（api将dao封装了，无需会mybatis）

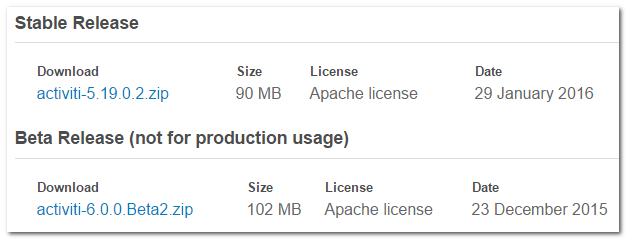
# Activiti环境搭建

## 开发包下载

官网：http://www.activiti.org/

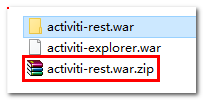


下载开发包



开发包的结构：

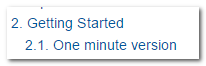


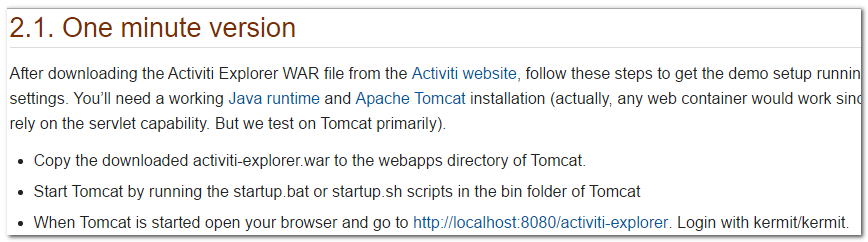


是空的工程，提供了工作流的基本环境（jar，配置文件的示例）--不用maven可以用这个。

## 一分钟版本（官方Demo演示）

官方API文档：

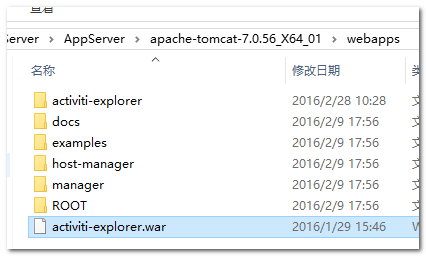






官方提供的可用的示例，完整演示工作流的使用。（包含设计流程，发布流程，启动流程，运行流程。）

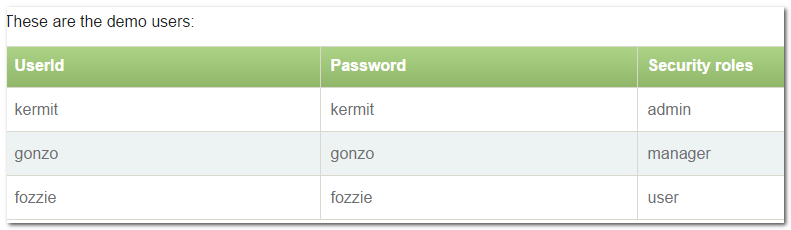
部署：



h2数据库

启动和访问。

内置：

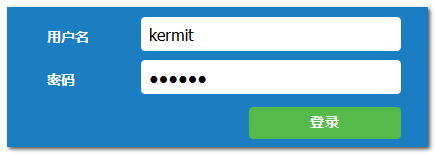


访问系统：

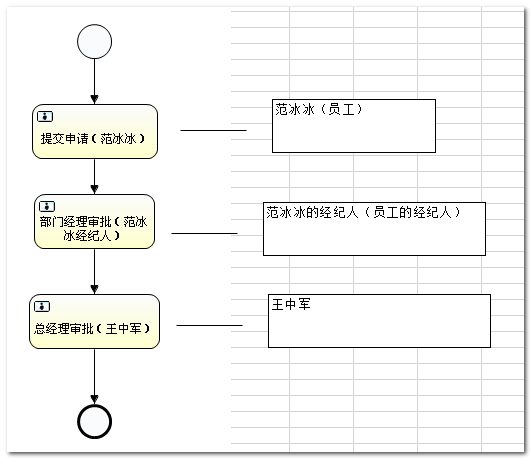
http://localhost:8080/activiti-explorer/

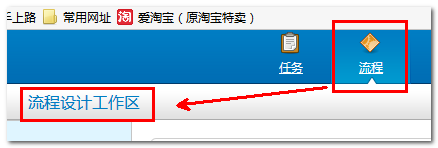
开发步骤：

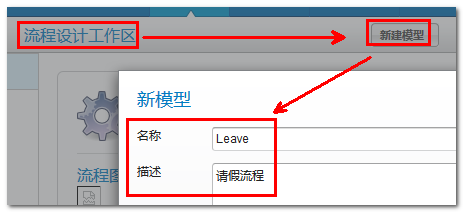
第一步：用管理员进去定制流程。



设计流程（类似以下流程）：

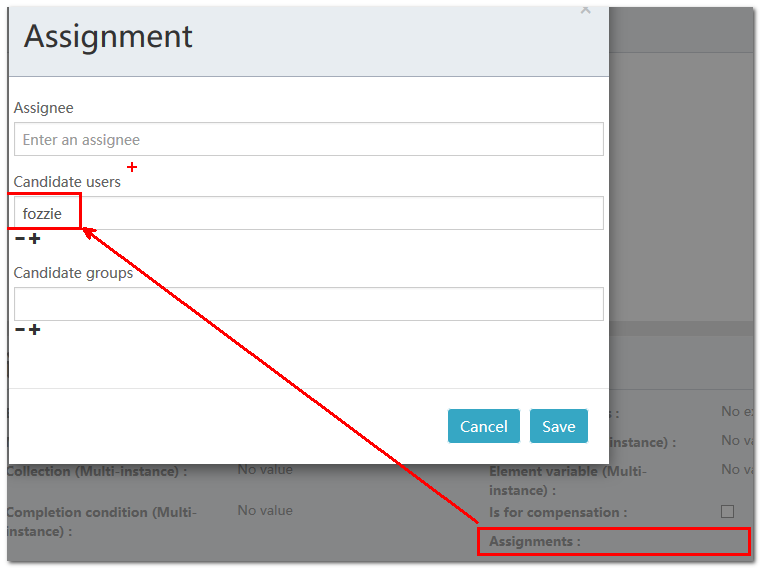


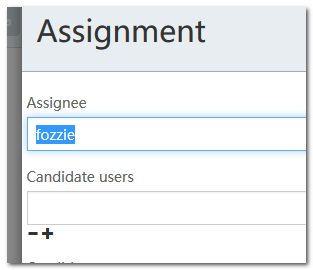


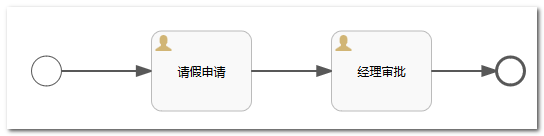


自动打开在线编辑器：（js写的）

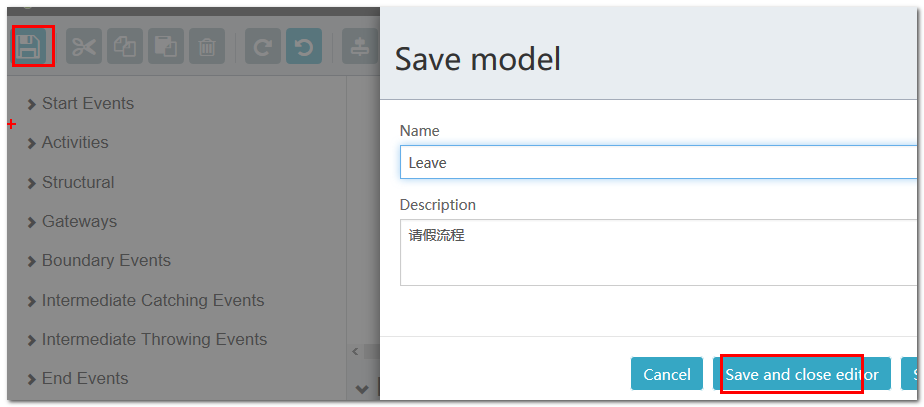
设置办理人：



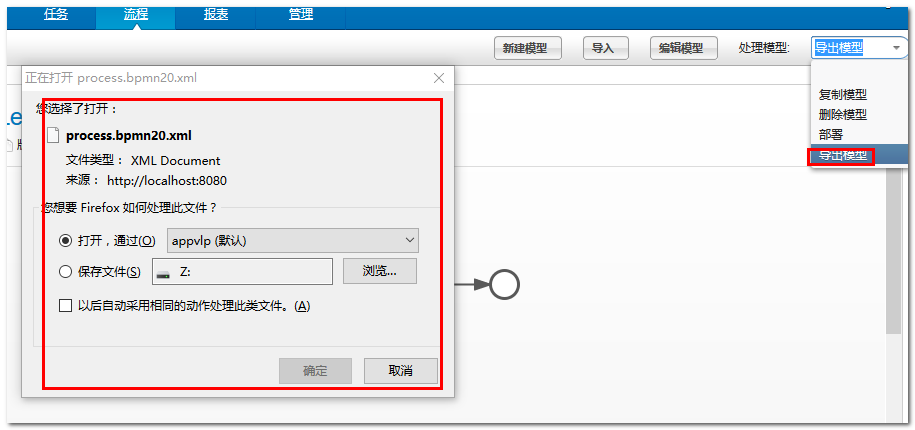




保存流程定义：

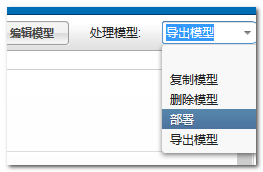


可以将流程导出备用，或导入其他的工作流系统：



第二步：部署和启动流程。

直接部署：



启动流程：



第三步：使用流程

换个人来登录



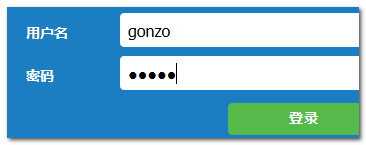
可以查看代办的任务：

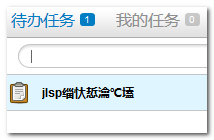


办理当前的任务：



换个经理来登录：







梳理：

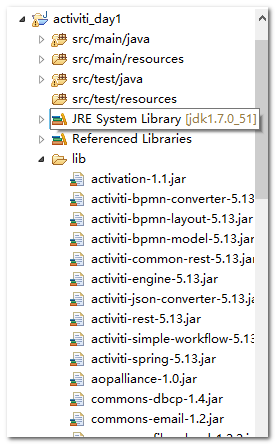
使用工作流系统需要三步：

1. 画流程图（流程文档xml定义）---管理员
2. 部署到工作流的系统中---管理员
3. 启动流程，开始执行任务，直到任务结束。--用户

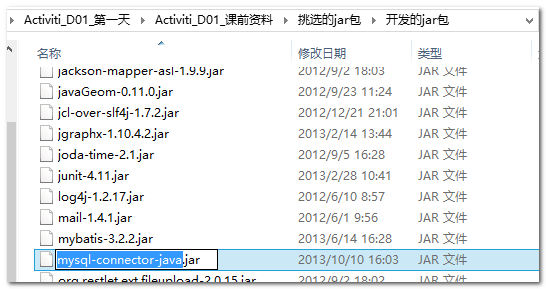
## 导入Jar

【传统的项目】--了解

在activiti-5.13->wars目录下是一些示例项目，解压activiti-rest项目，导入activiti-rest目录中WEB-INF\lib下所有包。添加到classpath中。



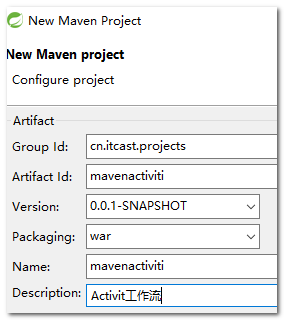
由于我们使用的是Mysql数据库，Mysql数据库的链接驱动Activiti官方包中并没有提供，需要我们自己导入。手动导入mysql-connector-java.jar，添加到classpath下。

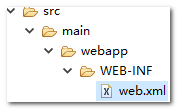


将所有jar添加到classpath中。

【Maven的项目】

新建maven的web工程：





Maven坐标（工作流引擎activiti-engine，日志实现slf4j-log4j12，数据库驱动oracle，测试junit）：

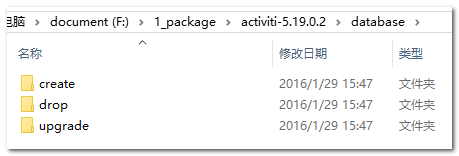
|  |
| --- |
| <properties>  <activiti.version>5.19.0.2</activiti.version>  <slf4j.version>1.7.5</slf4j.version>  <oracle.version>10.2.0.4.0</oracle.version>  <c3p0.version>0.9.1.2</c3p0.version>  <junit.version>4.11</junit.version>  </properties>  <dependencies>  <dependency>  <groupId>org.activiti</groupId>  <artifactId>activiti-engine</artifactId>  <version>${activiti.version}</version>  </dependency>  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-log4j12</artifactId>  <version>${slf4j.version}</version>  </dependency>  <dependency>  <groupId>com.oracle</groupId>  <artifactId>ojdbc14</artifactId>  <version>${oracle.version}</version>  <scope>runtime</scope>  </dependency>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>${junit.version}</version>  <scope>test</scope>  </dependency>  </dependencies> |

配置文件：

将activiti-rest.war\WEB-INF\classes中的log4j.properties复制进工程的resources文件夹中

## 初始化Activiti数据库

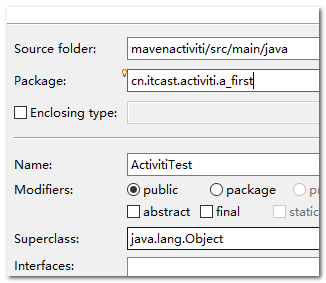
初始化数据库可以使用开发包中的一堆脚本，自己执行。



但不推荐。

Jar中已经内置了脚本，你只需要获取工作流引擎对象就可以了。自己会执行脚步呢，生成数据库。

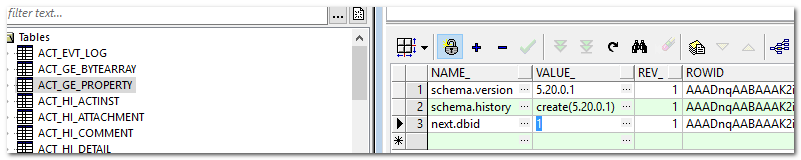
新建一个测试类：





|  |
| --- |
| @Test  //创建表：通过创建工厂（流程引擎：用来生成其他的服务api来具体操作）  **public** **void** testCreateTable(){  //1.得到引擎的配置对象:单服务器使用  ProcessEngineConfiguration processEngineConfiguration=  ProcessEngineConfiguration.*createStandaloneProcessEngineConfiguration*();    //2.在引擎配置对象上设置相应的数据库连接参数  processEngineConfiguration.setJdbcUrl("jdbc:mysql://localhost:3306/activiti\_api\_sh15?createDatabaseIfNotExist=true&userUnicode=true&characterEncoding=utf-8");  //驱动  processEngineConfiguration.setJdbcDriver("com.mysql.jdbc.Driver");  //用户名  processEngineConfiguration.setJdbcUsername("root");  //密码  processEngineConfiguration.setJdbcPassword("");    //设置自动建表  //三种参数值：  //false:只是检查表结构，但不创建也不更新。适合已经有表的时候用  //create-drop：启动的时候检查和创建表，引擎关闭的是的删除  //true:检查表结构，如果表存在，则创建，如果表结构不一致，则更新。---常用  processEngineConfiguration.setDatabaseSchemaUpdate(ProcessEngineConfiguration.***DB\_SCHEMA\_UPDATE\_TRUE***);  //3.创建引擎（工厂）  ProcessEngine processEngine = processEngineConfiguration.buildProcessEngine();  System.***out***.println("引擎创建成功！数据库创建成功！引擎对象："+processEngine);    } |

自动建立25张数据表。



版本号：工作流引擎系统，可以升级。主要作用就是为了版本升级用的。

Next.dbid:生成主键。（一定规则。。。）

## Activiti的核心

Activity核心：表+java的api -》一套工作流规则。

### Activiti的核心—数据库表

Activiti控制流程的执行是通过25张表（23+2）进行的，而这25张表又通过Java的几个服务(Service)API进行操作的。

Activit数据库表是工作流的后台支持，所有的表都以ACT\_开头，表的命名中的第二部分的两个字母是标识表的用途，而用途也与服务的API相对应。

|  |
| --- |
| ACT\_RE\_\*: 'RE'表示repository。 这个前缀的表包含了流程定义和流程静态资源 （图片，规则，等等）。  ACT\_RU\_\*: 'RU'表示runtime。 这些运行时的表，包含流程实例，任务，变量，异步任务，等运行中的数据。 Activiti只在流程实例执行过程中保存这些数据， 在流程结束时就会删除这些记录。 这样运行时表可以一直很小速度很快。  ACT\_ID\_\*: 'ID'表示identity。 这些表包含身份信息，比如用户，组等等。  ACT\_HI\_\*: 'HI'表示history。 这些表包含历史数据，比如历史流程实例， 变量，任务等等。  ACT\_GE\_\*: ‘GE’表示general。通用数据， 用于不同场景下，如存放资源文件。  ACT\_EVT\_LOG：事件日志 |

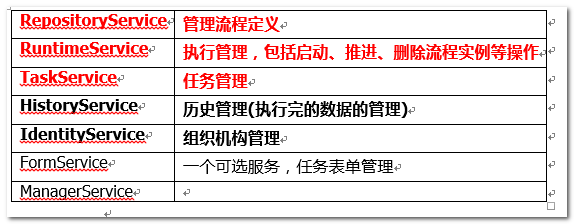
因此，从用途上划分主要有如下几类：

* 资源库流程规则表
* 运行时数据库表
* 历史数据库表
* 用户组织机构表
* 通用数据表
* 事件日志表

### Activiti的核心--API

ProcessEngine：流程引擎接口，是最核心的API类，其他的API类都由他而来。它是其他API产生的基础！

ProcessEngine对象，这是Activiti工作的核心。负责生成流程运行时的各种实例及数据、监控和管理流程的运行。



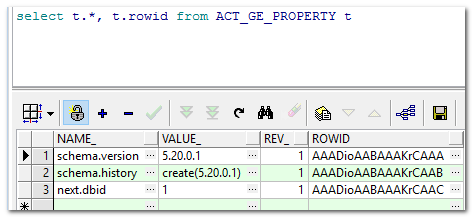


* RepositoryService 流程定义管理 操作 ACT\_RE\_\* 数据表
* RuntimeService 流程实例管理 操作 ACT\_RU\_EXECUTION 数据表
* TaskService 任务管理 操作 ACT\_RU\_TASK 数据表
* IdentitiService 认证管理 操作 ACT\_ID\_\* 数据表
* HistoryService 历史记录管理 操作 ACT\_HI\_\* 数据表
* ManagementService 定时任务管理，创建job，ACT\_RU\_JOB 数据表
* FormService 表单管理，生成动态任务表单页面，某个字段存的数据

### 学习方法

要掌握Activiti，必须要学会这些核心API，以及数据表的数据存储的情况。！

刚建立好数据库后，只有一张表有数据：



工作流引擎：针对我们activiti，就是一个核心接口类，---》可以构建相关的api来建立、执行、运行工作流的流程。

流程模型：流程图—必须提前画好---工作流才能按照这章图来跑。----规则！

## Activiti和Spring的集成(未整合)

我们只是利用spring的创建bean的能力，代码写法变成配置了。

导入maven坐标（spring相关5个+C3P0）

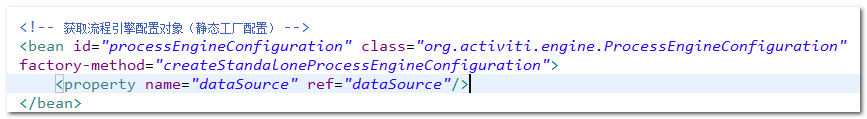
|  |
| --- |
| <properties>  <spring.version>3.2.12.RELEASE</spring.version>  <activiti.version>5.19.0.2</activiti.version>  <slf4j.version>1.7.5</slf4j.version>  <oracle.version>10.2.0.4.0</oracle.version>  <c3p0.version>0.9.1.2</c3p0.version>  <junit.version>4.11</junit.version>  </properties>  <dependencies>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-aspects</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-orm</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-web</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-test</artifactId>  <version>${spring.version}</version>  </dependency>  <dependency>  <groupId>org.activiti</groupId>  <artifactId>activiti-engine</artifactId>  <version>${activiti.version}</version>  </dependency>  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-log4j12</artifactId>  <version>${slf4j.version}</version>  </dependency>  <dependency>  <groupId>com.oracle</groupId>  <artifactId>ojdbc14</artifactId>  <version>${oracle.version}</version>  <scope>runtime</scope>  </dependency>  <dependency>  <groupId>c3p0</groupId>  <artifactId>c3p0</artifactId>  <version>${c3p0.version}</version>  </dependency>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>${junit.version}</version>  <scope>test</scope>  </dependency>  </dependencies> |

配置web.xml，引入Spring

|  |
| --- |
| <!-- spring配置文件位置 -->  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:applicationContext.xml</param-value>  </context-param>  <!-- spring核心监听器 -->  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener> |

ApplicationContext.xml:

|  |
| --- |
| <!-- 引入外部属性文件 -->  <context:property-placeholder location=*"classpath:db.properties"* />  <!-- 连接池 -->  <bean id=*"dataSource"* class=*"com.mchange.v2.c3p0.ComboPooledDataSource"*>  <property name=*"driverClass"* value=*"${jdbc.driver}"* />  <property name=*"jdbcUrl"* value=*"${jdbc.url}"* />  <property name=*"user"* value=*"${jdbc.user}"* />  <property name=*"password"* value=*"${jdbc.password}"* />  </bean>  <!-- 流程引擎配置对象 -->  <bean id=*"processEngineConfiguration"* class=*"org.activiti.engine.impl.cfg.StandaloneProcessEngineConfiguration"*>  <!-- 注入数据源 -->  <property name=*"dataSource"* ref=*"dataSource"*/>  <!-- 自动建表 -->  <property name=*"databaseSchemaUpdate"* value=*"true"*/>  </bean>  <!-- 流程引擎对象 -->  <bean id=*"processEngine"* factory-bean=*"processEngineConfiguration"* factory-method=*"buildProcessEngine"*/> |



Db.properties

|  |
| --- |
| jdbc.driver=oracle.jdbc.driver.OracleDriver  jdbc.url=jdbc:oracle:thin:@localhost:1521:xe  jdbc.user=mybos  jdbc.password=mybos |

## Activiti流程设计器

### BPMN—工作流模型

业务流程建模与标注（Business Process Modeling Notation，BPMN)　描述基本的BPMN符号，包括这些图元如何组合成一个业务流程图（Business Process Diagram）

BPMN的主要目标是提供一些被所有业务用户容易理解的符号，从创建流程轮廓的业务分析到这些流程的实现，直到最终用户的管理监控。

简单的说：BPMN对电脑和用户都能读得懂。（XML实现）

Activiti支持最新的BPMN 2.0标准。

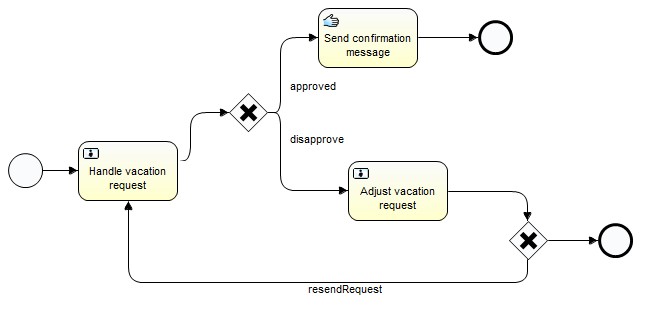
补充：流程图中的哪些符号就是由这个BPMN2。0标准规定的。

总之：工作流模型是对业务流程抽象的一个模型，是整个工作流引擎的基础。

### 流程设计器

BPMN采用xml进行描述，虽然人可以直接读写，但还是需要有一定技术能力的人才能操作，不直观。

为解决这个问题，Activiti提供了eclipse插件，可以直接通过图形化的界面进行BPMN的设计和编写。插件画出来的BPMN图效果如下：



### 设计器的安装

设计器有在线和离线两种：

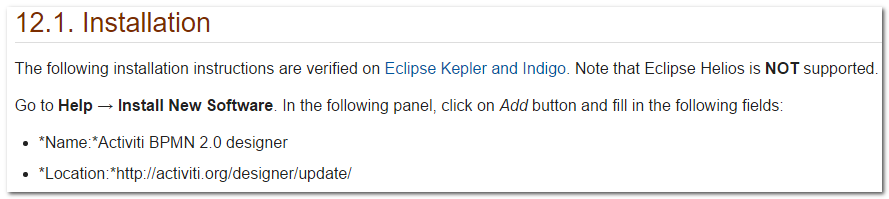
* 在线设计器（官方提供项目）--不推荐
* 离线设计器：eclipse+插件

离线设计器的安装又分在线安装和离线安装

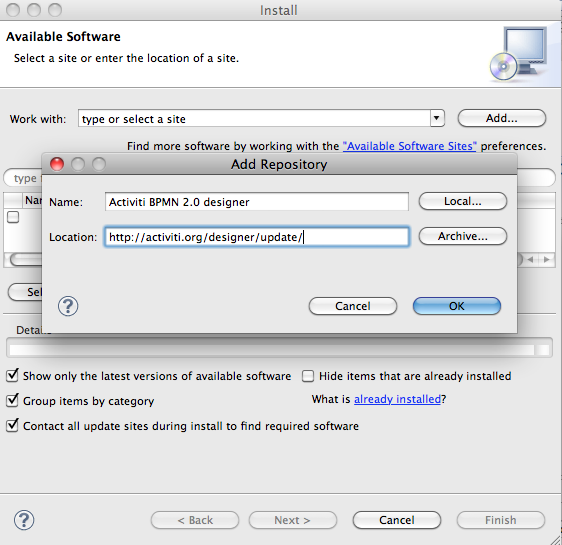
|  |
| --- |
| eclipse插件（设计器）离线下载：  http://www.activiti.org/designer/archived/  eclipse插件（设计器）在线安装：  http://www.activiti.org/designer/update/ |

两种安装方式：在线安装和手动安装.

* 在线安装：



|  |
| --- |
| \*Name:\*Activiti BPMN 2.0 designer  \*Location:\*http://activiti.org/designer/update/ |

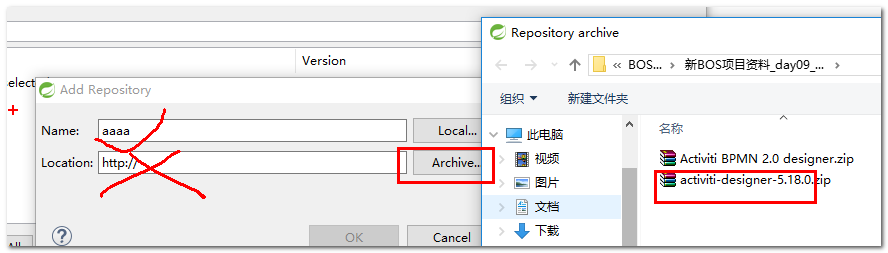


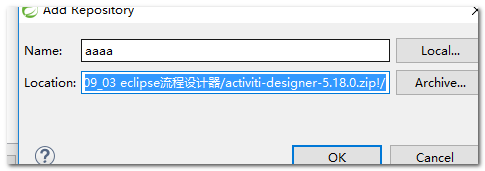
* 离线安装：

将解压到dropin目录

打开编辑器，手动安装压缩包：（过程和在线一样）





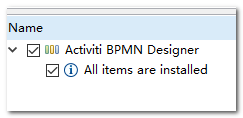


但：这种方式经常不成功，原因eclipse版本和插件必须完全匹配。

* 半离线安装

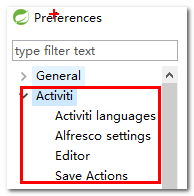
打开编辑器，手动安装压缩包：



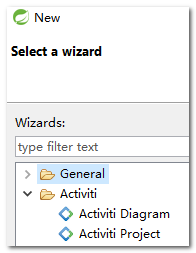


之后有些组件会到网上下载，后面如果有提示都点击确定。

注意：本版本适应。



或

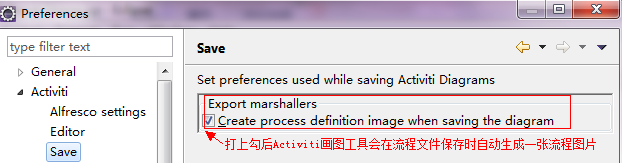


说明安装成功了

提示：我们的这个插件在新版的eclipse下安装的时候，离线安装可能不成功。

### 设计器的设置

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



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

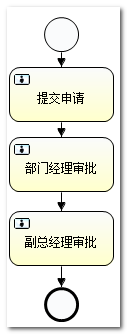
所在完成以上配置后，会由我们自己来管理流程图片。在发布流程时把流程规则文件和流程图片一起上传就行了。

补充：

---bpmn

# 业务需求

一个简单的请假流程：



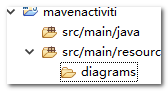
一般步骤是：

1. 定义流程（画流程图）
2. 部署流程定义
3. 启动流程实例
4. 查询办理人的个人任务
5. 完成办理人的个人任务

# 流程文档定义—画流程图

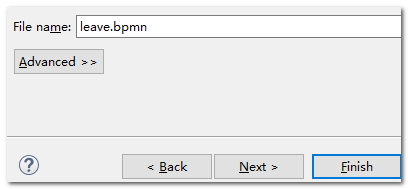
习惯上，在资源文件包中建立新包：



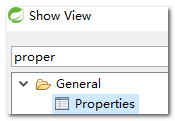


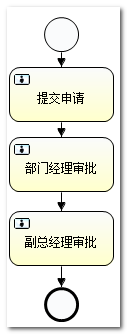
新建流程定义图





要打开，设置一些值：

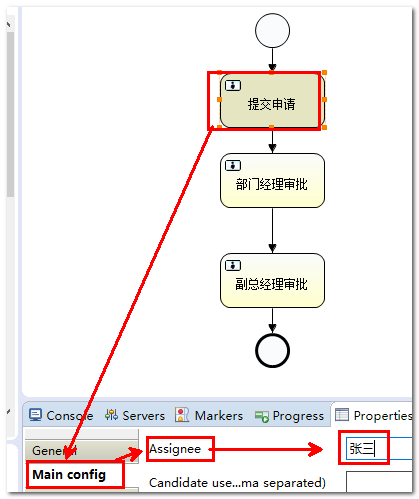




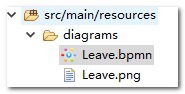
设置：流程id（非常重要-将来程序里面叫key—流程定义的唯一标识），必须英文



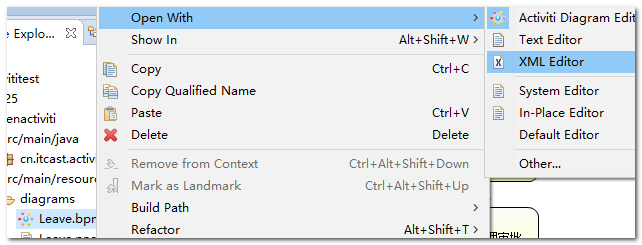
添加办理人：



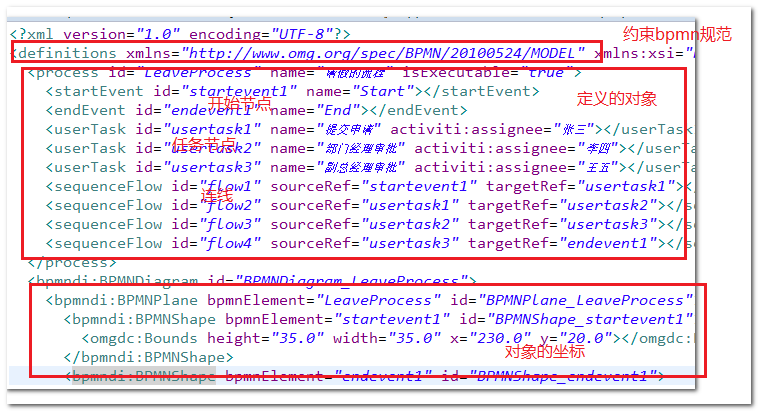
直接保存：



查看流程定义的文档：

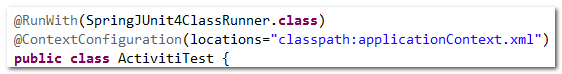


Bpmn文件解读：



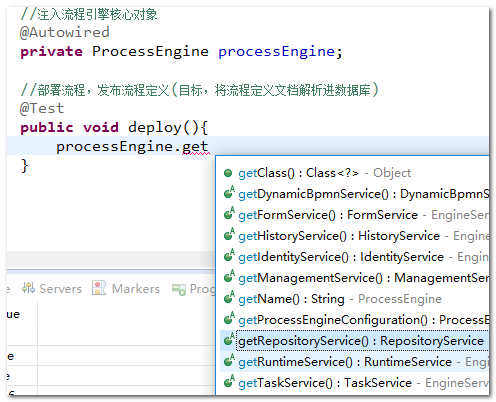
# 流程定义管理





## 流程部署+流程定义

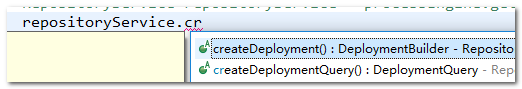
核心对象：



关于工作流的api，有个规律：要获取某对象，都是createXxx；

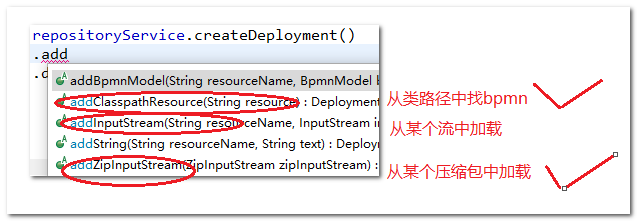
如果是CreateXxx，获取的就是xxx对象，该对象主要用来增删改对应的表。

如果是CreateXxxQuery，获取的是xxxQuery对象，该对象主要用来查询的。



工作流的api都支持链式编程。

加载资源的方法：



加载流程文档的时候，可以单独加载文档bpmn，工作流会自动生成一张图片存在数据库。

但也可以两个都加载，那么工作流就不会自动生成图片，而使用你上传的图片。

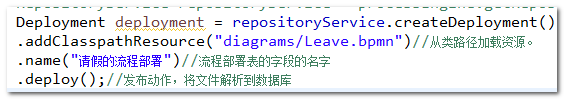
cn.tadpole.b\_processdefinition

|  |
| --- |
| @RunWith(SpringJUnit4ClassRunner.**class**)  @ContextConfiguration(locations="classpath:applicationContext.xml")  **public** **class** ActivitiTest {  //注入仓库Service  @Autowired  **private** RepositoryService repositoryService;    //发布流程:只发布流程图，自动生成图片  @Test  **public** **void** deployment1(){  Deployment deployment = repositoryService.createDeployment()//获取发布对象  .addClasspathResource("diagrams/Leave.bpmn")//添加流程文件  .name("请假流程1")//发布的名字  .deploy();//发布  System.***out***.println("发布的编号："+deployment.getId());  System.***out***.println("发布的名称："+deployment.getName());  System.***out***.println("发布的时间："+deployment.getDeploymentTime());  }  //发布流程:发布流程图和图片--第一种方法  @Test  **public** **void** deployment21(){  Deployment deployment = repositoryService.createDeployment()//获取发布对象  .addClasspathResource("diagrams/Leave.bpmn")//添加流程文件  .addClasspathResource("diagrams/Leave.png")//添加流程图片  .name("请假流程21")//发布的名字  .deploy();//发布  System.***out***.println("发布的编号："+deployment.getId());  System.***out***.println("发布的名称："+deployment.getName());  System.***out***.println("发布的时间："+deployment.getDeploymentTime());  }  //发布流程:发布流程图和图片--第二种方法  @Test  **public** **void** deployment22() **throws** FileNotFoundException{  Deployment deployment = repositoryService.createDeployment()//获取发布对象  .addZipInputStream(**new** ZipInputStream(**this**.getClass().getClassLoader().getResourceAsStream("diagrams/Leave.zip")))  .name("请假流程22")//发布的名字  .deploy();//发布  System.***out***.println("发布的编号："+deployment.getId());  System.***out***.println("发布的名称："+deployment.getName());  System.***out***.println("发布的时间："+deployment.getDeploymentTime());  }  } |

ApplicationContext.xml:

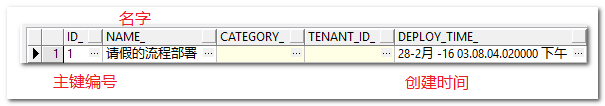
|  |
| --- |
| <!-- 流程控制的相关service -->  <!-- 仓库对象 -->  <bean id=*"repositoryService"* factory-bean=*"processEngine"* factory-method=*"getRepositoryService"*></bean> |

表分析：

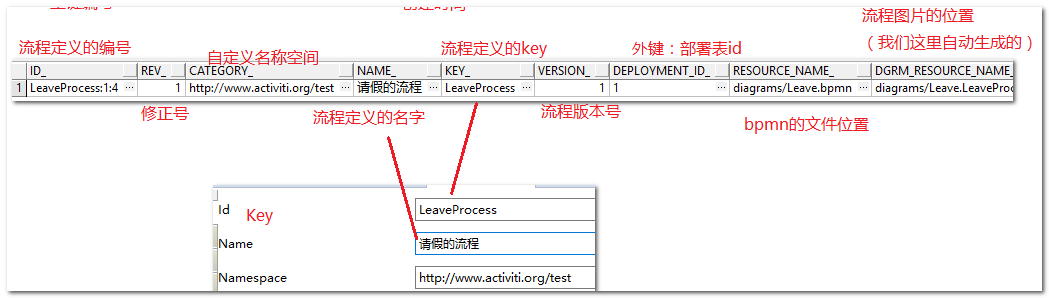


当流程发布的时候：发生了什么？表的变化。

部署对象表：



流程定义表：

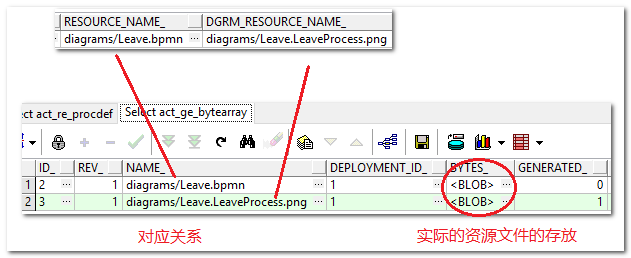


流程定义编号（流程定义信息表主键）：规律：流程定义的key:版本号:“随机”编号

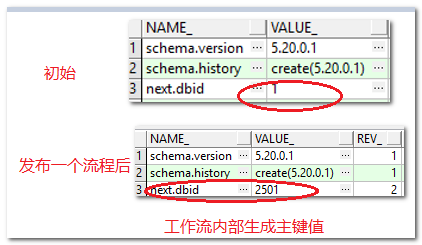
版本号：如果再次部署流程，如果key一样，会升级版本号。

资源文件的位置在哪里呢？

资源文件表：



系统属性表：



主键值生成规律：刚开始1，将1，先给第一张需要主键 的表来用。2，3

如果是下一次操作，

紧接着，下一个： 2501，如果用了，下一个是2502，，，，

上述都是同一次操作。

如果是下一次操作，

5001》5001》5002

7501》7501

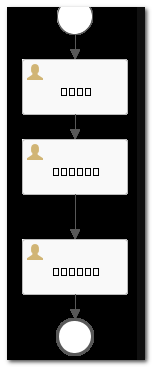
这里的值，会为下一次做好准备。。

再次部署一个流程：



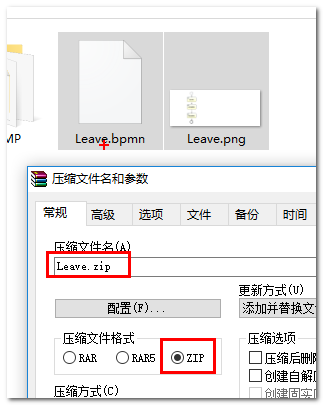
发生了什么。。。

工作流服务端自动生成的图片的乱码问题？



|  |
| --- |
| 解决方案1：  在流程引擎配置对象上面设置字体属性：  <!-- 发布流程生成图片是正常显示中文 -->  <property name="activityFontName" value="宋体"/>  <property name="labelFontName" value="宋体"/>  解决方案2：  流程图的图片在设计器中生成好，上传到服务器，不让服务器自动生成图片。 |

一次性就部署bpmn和图片，使用压缩包（服务器会自动解压）



涉及到的表：

|  |
| --- |
| select t.\*, t.rowid from ACT\_RE\_DEPLOYMENT t;--部署对象表  select t.\*, t.rowid from Act\_Re\_Procdef t;--流程定义表(procee definition)  select t.\*, t.rowid from act\_ge\_bytearray t;--资源存放表（存放图标、图片对象）  select t.\*, t.rowid from act\_ge\_property t;--系统的属性表 |

## 部署信息查询

目标：查询这章表，ACT\_RE\_DEPLOYMENT

|  |
| --- |
| @Test//查询部署表对象数据  **public** **void** queryDeployment(){  List<Deployment> list = repositoryService.createDeploymentQuery()  //条件查询  .deploymentName("请假的流程部署")  .deploymentId("1")//  // .singleResult()//查询出一条结果  .orderByDeploymenTime()//排序  // .listPage(firstResult, maxResults)//分页  // .count()//统计  .list();  **for** (Deployment deployment : list) {  System.***out***.println("部署编号："+deployment.getId());  System.***out***.println("部署名称："+deployment.getName());  System.***out***.println("------------------------");  }  } |

## 流程定义信息查询

|  |
| --- |
| @Test  //查询流程定义的表的数据  **public** **void** queryProceeDefinition(){  List<ProcessDefinition> list = repositoryService.createProcessDefinitionQuery()  // .processDefinitionKey(processDefinitionKey)//根据key来查询  .list();  **for** (ProcessDefinition processDefinition : list) {  System.***out***.println("流程定义的编号："+processDefinition.getId());  System.***out***.println("流程部署的编号："+processDefinition.getDeploymentId());  }  } |

## 流程图查看

|  |
| --- |
| //查询流程图  @Test  **public** **void** findProcessDefinitionDiagram() **throws** IOException{  //获取任意的ACT\_ge\_bytearray的资源，包括xml和图片  // InputStream in = repositoryService.getResourceAsStream("5001", "diagrams/leave.LeaveProcess.png");  InputStream in = repositoryService.getProcessDiagram("LeaveProcess:1:4");  OutputStream out = **new** FileOutputStream("z:/Leave.png");  FileUtil.*copyStream*(in, out);  } |

自动生成的流程图的乱码问题解决：

|  |
| --- |
| <!-- 流程引擎配置对象 -->  <bean id=*"processEngineConfiguration"* class=*"org.activiti.engine.impl.cfg.StandaloneProcessEngineConfiguration"*>  <!-- 注入数据源 -->  <property name=*"dataSource"* ref=*"dataSource"*/>  <!-- 自动建表 -->  <property name=*"databaseSchemaUpdate"* value=*"true"*/>  <!-- 字体 -->  <property name=*"activityFontName"* value=*"宋体"*/>  <property name=*"labelFontName"* value=*"宋体"*/>  </bean> |

## 流程定义的修改

流程定义不能修改，如果流程需要变化，只需要重新画流程图，然后重新部署，增加一个新的流程。

旧的流程，如果没有走完，依然走旧的。

如果新发起的流程，就走新的，

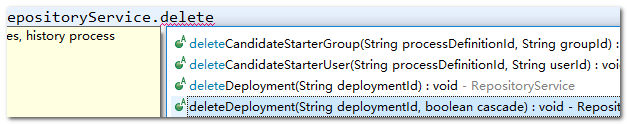
版本号+1解决

演示。。。

## 流程的删除-流程销毁

通过部署id来删除

|  |
| --- |
| //删除流程定义（通过部署id）  @Test  **public** **void** deleteProcessDefinition(){  repositoryService.deleteDeployment("1", **true**);  } |



级联和不级联。

场景：如果流程已经开启了，如果不使用级联，则会抛出异常！（原因：定义表有外键关联它），此时只能用级联。

# 流程实例管理

## 流程定义和流程实例的区别

作为java面向对象，class(User)----类型的定义 user=new user()—实例

流程定义就像class

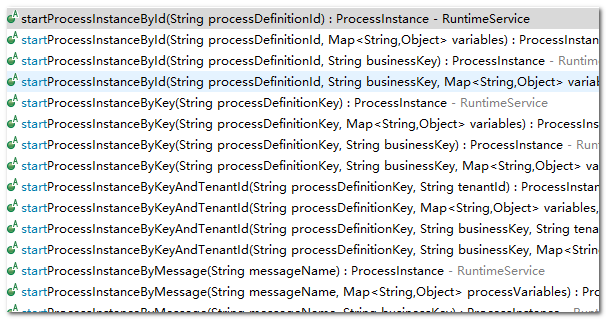
实例就像对象----new 流程定义----启动流程

问题：一个流程定义，可以启动N次么？可以！

每次启动流程，都会是一个新的流程实例！

## 流程实例的启动

启动的api有三大类：



* 根据流程定义的id来启动：可以启动指定的流程定义。
* 根据流程定义的key来启动：自动选择版本号最高（最新）的流程定义来启动
* 根据消息的名字来启动

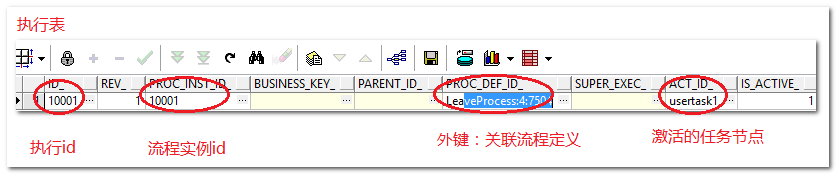
|  |
| --- |
| //注入运行时service  @Autowired  **private** RuntimeService runtimeService; |
| //启动流程实例  @Test  **public** **void** startProcessInstance(){  ProcessInstance processInstance = runtimeService.startProcessInstanceByKey("LeaveProcess");  System.***out***.println("流程实例的id："+processInstance.getId());  System.***out***.println("流程实例的名字："+processInstance.getName());  System.***out***.println("流程实例对应的流程定义："+processInstance.getProcessDefinitionId());  } |
| <!-- 运行时Service -->  <bean id=*"runtimeService"* factory-bean=*"processEngine"* factory-method=*"getRuntimeService"*/> |

分析：

流程启动发生了什么？？看表。

插入数据，哪些表呢？

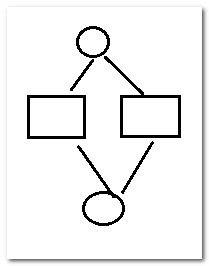
**流程执行表（包含了流程实例）**



执行id和流程实例id是否要一样呢？

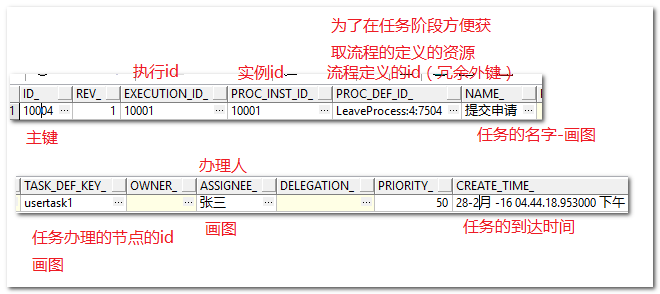
对于没有分支的流程，一个实例就是一次执行流程，两个值一样。

但如果有分支的流程，一个实例就有多次执行 ，两个值就可以不一样了。（一个流程定义只能new出来一个实例）



一个实例，但有两个执行。

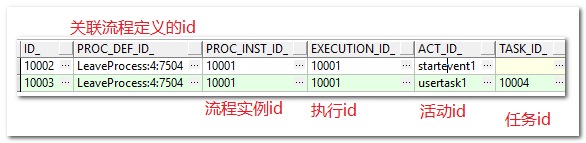
**任务表：**



办理人的表：

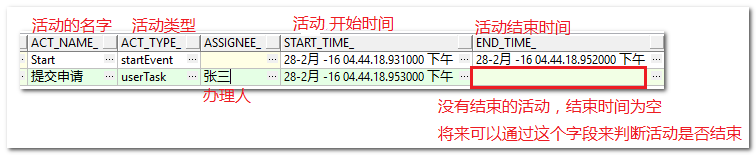


历史的活动表：



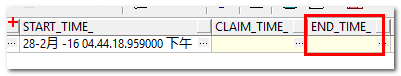
什么是活动？任何的节点的都是活动，开始+结束+任务+网关+。。。。。

什么是任务？只有是带小人的方块才是任务，一般是需要人来办理。



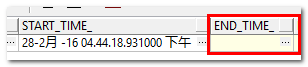
**任务历史表**

该表的字段和活动历史表基本一样，只是数据少，（因为它只记录任务）



如果任务没有结束，endtime就为空。

流程实例的历史表：



想法：如何判断流程是否结束？

* 看历史流程实例表endtime是否为空，如果为空，还没结束。
* 还可以判断正在执行的实例表有没有实例id对应的数据

**历史的人员表：**



涉及的表：

|  |
| --- |
| select t.\*, t.rowid from act\_ru\_execution t; *--执行表-shi例-与流程定义是1对多*  select t.\*, t.rowid from act\_ru\_task t;*--任务表*  select t.\*, t.rowid from act\_ru\_identitylink t;*--管理办理人*  select t.\*, t.rowid from act\_hi\_procinst t;*--流程实例的历史表*  select t.\*, t.rowid from act\_hi\_actinst t;*--活动历史表-action instant-*  select t.\*, t.rowid from act\_hi\_taskinst t;*--任务的历史表*  select t.\*, t.rowid from act\_hi\_identitylink t;*--历史的人员表* |

## 流程实例的查询

|  |
| --- |
| //查询流程实例  @Test  **public** **void** queryProcessInstance(){  List<ProcessInstance> list = runtimeService.createProcessInstanceQuery()  .list();  **for** (ProcessInstance processInstance : list) {  System.***out***.println("流程实例的id："+processInstance.getId());  System.***out***.println("流程实例的名字："+processInstance.getName());  System.***out***.println("流程实例对应的流程定义："+processInstance.getProcessDefinitionId());  System.***out***.println("--------------------------");  }  } |

## 流程当前节点的坐标

如何让用户看到当前流程运行位置？

在节点上加一个红框。

怎么加红框？

先获取坐标和大小，然后用div来写。

单个活动节点：

|  |
| --- |
| //查询活动节点坐标  @Test  **public** **void** queryActiveNodeZuobiao(){  //活动节点  String act\_id ="usertask1";  GraphicInfo graphicInfo = repositoryService.getBpmnModel("LeaveProcess:4:7504")  .getGraphicInfo(act\_id);//某活动节点的坐标  System.***out***.println("高："+graphicInfo.getHeight());  System.***out***.println("宽："+graphicInfo.getWidth());  System.***out***.println("x："+graphicInfo.getX());  System.***out***.println("y："+graphicInfo.getY());  } |

多个活动节点：

|  |
| --- |
| List<String> activeActivityIds = runtimeService.getActiveActivityIds("10001");  **for** (String activeActivityId : activeActivityIds) {  GraphicInfo graphicInfo1 = repositoryService.getBpmnModel("LeaveProcess:4:7504")  .getGraphicInfo(activeActivityId);//某活动节点的坐标  System.***out***.println("高："+graphicInfo1.getHeight());  System.***out***.println("宽："+graphicInfo1.getWidth());  System.***out***.println("x："+graphicInfo1.getX());  System.***out***.println("y："+graphicInfo1.getY());  } |

## 流程实例的删除（强行终止）

场景：用户A请假了，流程已经在审批了。但此时由于公司需要，不能请假了，销毁请假实例。

|  |
| --- |
| //流程实例的删除  @Test  **public** **void** deleteProcessInstance(){  runtimeService.deleteProcessInstance("LeaveProcess:4:7504", "不想请假了");  } |

提示：

* 删除流程定义：可以级联删除流程定义以及流程实例相关表数据。（寸草不生）
* 删除流程实例：只删除流程实例相关（运行时）表的数据，对流程定义没有影响。

# 任务管理

## 个人任务查询（待办任务查询）

|  |
| --- |
| //注入任务service  @Autowired  **private** TaskService taskService; |
| //个人任务查询  @Test  **public** **void** queryPersionTask(){  List<Task> list = taskService.createTaskQuery()  .taskAssignee("张三")  .list();  **for** (Task task : list) {  System.***out***.println("任务编号："+task.getId());  System.***out***.println("任务名称："+task.getName());  System.***out***.println("任务所属流程成实例id："+task.getProcessInstanceId());  }  } |

## 个人任务办理

|  |
| --- |
| //个人任务办理  @Test  **public** **void** completePersonTask(){  taskService.complete("12504");  } |

业务：

会走到下一个流程。（自动，只要你流程定义好了）

表会有什么变化：

自己去看—了解工作原理。

## 任务办理完成

如何将任务办结。

流程一步一步一步往下走。。。

当end节点之前的节点办理完成后，会自动完成end节点。当前流程实例都结束了。

重点：

1. 理解为什么要用工作流？
2. 流程引擎的核心对象：ProcessEngine，出来具体的操作对象：（操作就是25表-流程引擎核心）

RepositoryService:操作部署信息相关+获取资源相关+获取流程定义相关

RuntimeService：操作流程启动、流程启动后的流程实例信息查询、执行实例信息查询

TaskService：操作流程人的任务，查询个人任务，办理任务（任务id）

3．流程图会画（bpmn model---xml）

|  |
| --- |
| *--通用属性表（版本号，主键信息）*  *--主键：每次操作+2500，每次操作中的小操作，依次增加的*  select t.\*, t.rowid from ACT\_GE\_PROPERTY t;  *--部署信息表*  select t.\*, t.rowid from ACT\_re\_deployment t;  *--流程定义表(存放流程定义的必要信息)*  select t.\*, t.rowid from ACT\_re\_procdef t;  *--通用资源表（存放二进制信息）*  select t.\*, t.rowid from ACT\_ge\_bytearray t;  *--正在执行实例的表(执行实例对象和流程实例对象)*  select t.\*, t.rowid from ACT\_ru\_execution t;  *--正在执行的任务表（存放方框带小人的节点信息）*  select t.\*, t.rowid from ACT\_ru\_task t;  *--正在执行的身份表（参与人和候选人信息）*  select t.\*, t.rowid from ACT\_ru\_identitylink t;  *--流程实例的历史表*  select t.\*, t.rowid from ACT\_hi\_procinst t;  *--历史的任务表（任务节点）*  select t.\*, t.rowid from ACT\_hi\_taskinst t;  *--历史的活动表(任何节点，包含任务节点、开始结束节点)*  select t.\*, t.rowid from ACT\_hi\_actinst t;  *--历史的身份表*  select t.\*, t.rowid from ACT\_hi\_identitylink t; |