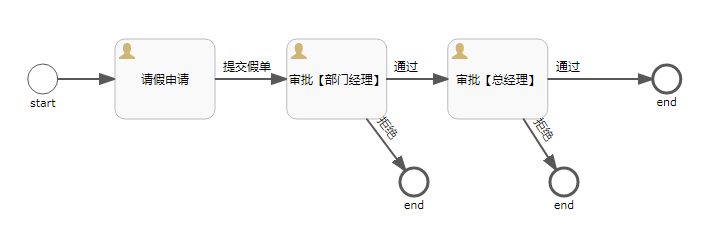
Activiti工作流

# 工作流概念：

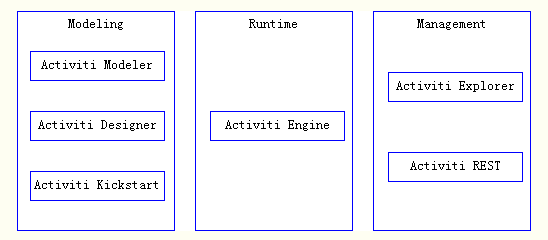
工作流（Workfolw），就是“业务过程的部分与整体在计算机应用环境下的自动化”，它主要解决的是“使在多个参与者之间按照某种预定义的规则传递文档、信息或任务的过程自动进行，从而实现某个预期的业务目标，或者促使此目标的实现”。”



# 工作流介绍

Activiti 的创始人 Tom Baeyens 是JBPM（Java Business Process Management）的创始人，由于与合作伙伴在JBPM的未来架构上产生意见分歧，其在2010年离开了Jboss并加入Alfresco公司，他的离开使得JBPM5完全放弃了JBPM4的架构，基于Drools Flow (规则引擎)重新开发，在2010年5月发布了第一个Activiti版本（5.0 alphal）,由此看来，Activiti更像是JBPM 4的延续，也许是为了让其看起来更像JBPM4的延续，Activiti团队直接将Activiti的第一个版本定义为5.0。

## 2.1、Activiti框架



Activiti Engine：作为最核心的模块，提供针对BPMN 2.0规范的解析、执行、创建、管理（任务、流程实例）、查询历史记录并根据结果生成报表。

Activiti Modeler：是模型设计器，其并非由Activiti公司所开发，而是由业界认可的Signavio公司赠送的（Signavio原本是收费的产品，现在被免费授权给Activiti用户使用）。适用于业务人员把需求转换为规范流程定义。

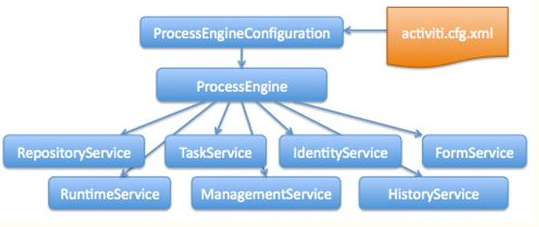
Activiti Designer：功能和Activiti Modeler类似，同样提供了基于BPMN 2.0规范的可视化设计功能，但是目前还没有完全支持BPMN规范的定义。适用于开发人员，可以把业务需求人员用Signavio设计的流程定义（XML格式）导入到Designer中，从而让开发人员将其进一步加工成为可以运行的流程定义。

Activiti Kickstart：基于表格的流程设计工具。用Activiti引擎可用的构件（constructs）的子集提供给快速创建“即席（adhoc）”的业务流程。

Activiti Explorer：可以用来管理仓库、用户、组，启动流程、任务办理等。此组件使用REST风格API（目的在于让开发人员快速入门），提供一个基础的设计模型。如果业务简单，也可以直接使用无需开发。还可以作为后台管理员的流程、任务管理系统使用。

Activiti REST：提供Restful风格的服务，允许客户端以JSON的方式与引擎的REST API交互，通用的协议具有跨平台、跨语言的特性。

### 2.2、Activiti服务架构

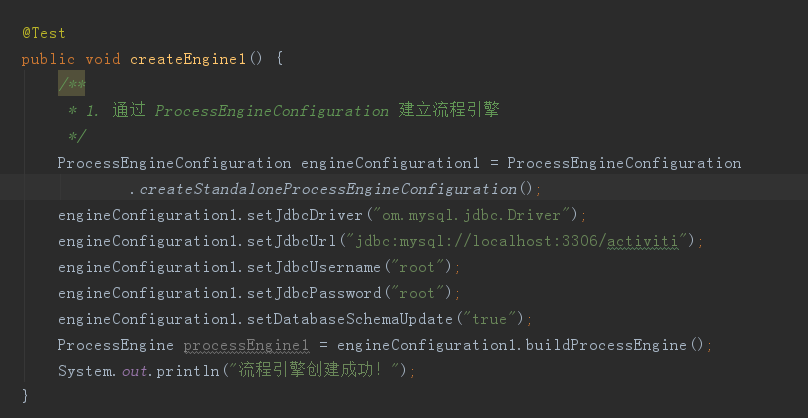


# 工作流引擎

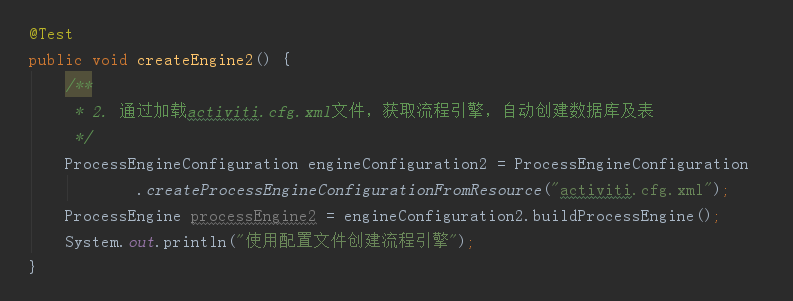
ProcessEngine对象，这是Activiti工作的核心，负责生成流程操作的各种操作对象，来管理各种实例及数据、监控和管理流程的运行。

## 3.1、工作流程引擎的创建方式

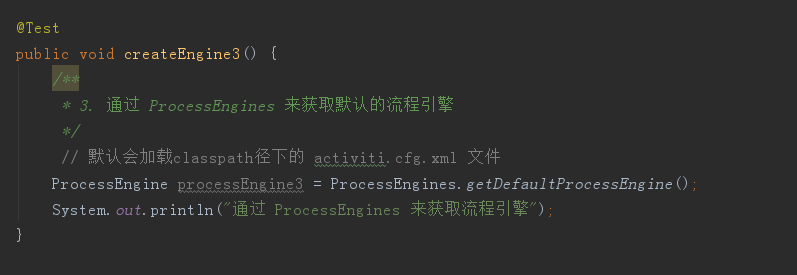
（1）通过 ProcessEngineConfiguration 的createStandaloneProcessEngineConfiguration()方法建立流程引擎



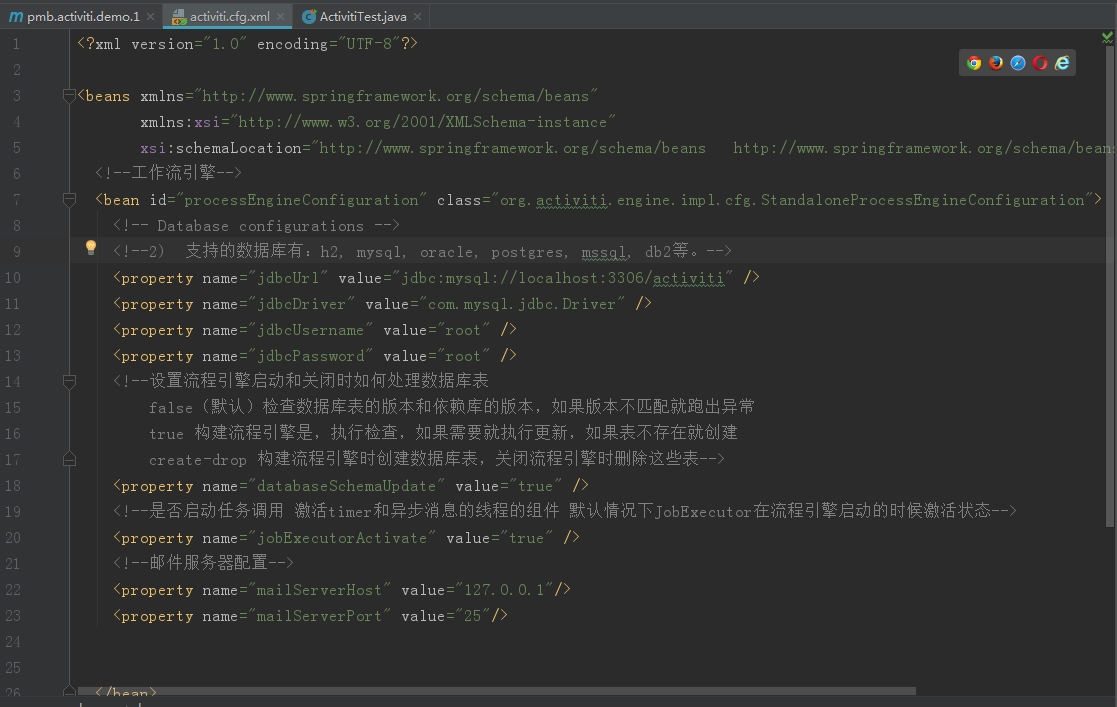
（2）通过 ProcessEngineConfiguration 的createProcessEngineConfigurationFromResource()方法加载\*.cfg.xml 文件建立流程引擎。



（3）通过 ProcessEngines 的 getDefaultProcessEngine()方法获取默认的流程引擎。



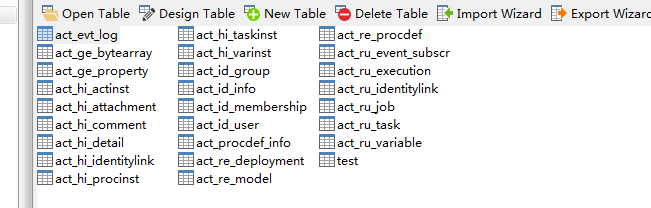
## 3.2activiti.cfg.xml简介



这只是一些主要的配置，还有更过的想要了解的话可以自己去[www.baidu.com](http://www.baidu.com)去查询学习。

# 数据库表

在构建流程引擎时，将属性databaseSchemaUpdate属性设置为true，当你启动流程引擎时，数据库没有表的，则自动帮你生成activiti所需的表格，不同版本的activiti，表的数量不同。但最少也有23张表格，随着版本的升级，表的数量会增加，我采用的是5.19.0.3版本，共有25张表，百度会有23张表的具体解释



## 4.1教你看懂数据库表名称

Activiti的后台是有数据库的支持，所有的表都以ACT\_开头。 第二部分是表示表的用途的两个字母标识。 用途也和服务的API对应。

act\_re\_\*: 'RE'表示repository。 这个前缀的表包含了流程定义和流程静态资源 （图片，规则，等等）。

act\_ru\_\*: 'RU'表示runtime。 这些运行时的表，包含流程实例，任务，变量，异步任务，等运行中的数据。 Activiti只在流程实例执行过程中保存这些数据， 在流程结束时就会删除这些记录。 这样运行时表可以一直很小速度很快。

act\_id\_\*: 'ID'表示identity。 这些表包含身份信息，比如用户，组等等。

act\_hi\_\*: 'HI'表示history。 这些表包含历史数据，比如历史流程实例， 变量，任务等等。

act\_ge\_\*: 通用数据， 用于不同场景下，如存放资源文件。

## 4.2表结构操作：

### 4.2.1：资源库流程规则表

1. act\_re\_deployment 部署信息表
2. act\_re\_model 流程设计模型部署表
3. act\_re\_procdef 流程定义数据表

### 4.2.2：运行时数据库表

1. act\_ru\_execution 运行时流程执行实例表
2. act\_ru\_identitylink 运行时流程人员表，主要存储任务节点与参与者的相关信息
3. act\_ru\_task 运行时任务节点表
4. act\_ru\_variable 运行时流程变量数据表
5. act\_ru\_event\_subscr 事件描述表

### 4.2.3：历史数据库表

1. act\_hi\_actinst 历史节点表
2. act\_hi\_attachment 历史附件表
3. act\_hi\_comment 历史意见表
4. act\_hi\_identitylink 历史流程人员表
5. act\_hi\_detail 历史详情表，提供历史变量的查询
6. act\_hi\_procinst 历史流程实例表
7. act\_hi\_taskinst 历史任务实例表
8. act\_hi\_varinst 历史变量表

### 4.2.4：组织机构表

1. act\_id\_group 用户组信息表
2. act\_id\_info 用户扩展信息表
3. act\_id\_membership 用户与用户组对应信息表
4. act\_id\_user 用户信息表

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

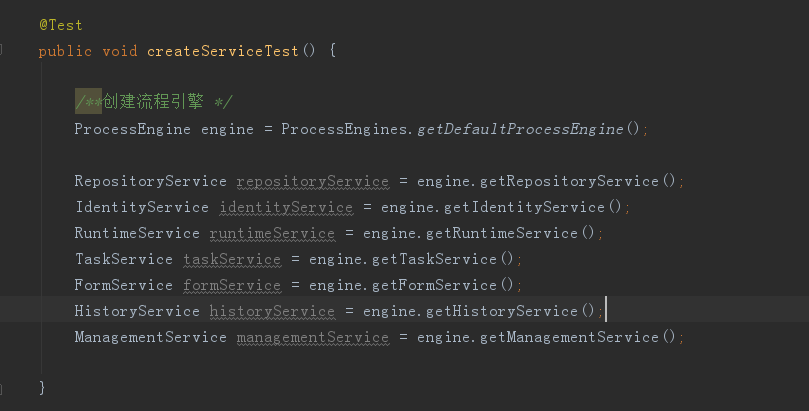
### 4.2.5：通用数据表

1. act\_ge\_bytearray 二进制数据表
2. act\_ge\_property 属性数据表存储整个流程引擎级别的数据,初始化表结构时，会默认插入三条记录，

# 七大服务组件

从以上的内容中我们知道了流程引擎的创建，Activiti所必要的数据库表，接下来我们就要开始讨论通过什么样的手段来操作这些数据库表来搭建我们都的工作流。

## 5.1、七大服务组件的创建及说明

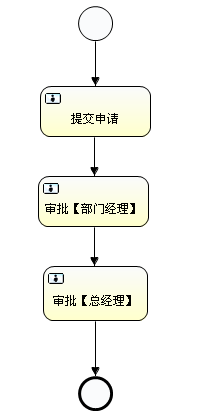


|  |  |
| --- | --- |
| repositoryService | 流程仓库Service，用于管理流程仓库，例如部署、删除、读取流程资源 |
| identityService | 身份Service，用于管理和查询用户、组之间的关系 |
| runtimeService | 运行时Service，可以也拿过来处理所有正在运行状态的流程实例、任务等 |
| taskService | 任务Service，用于管理和查询任务，例如签收、办理、指派等 |
| formService | 表单Service，用于读取流程、任务相关的表单数据 |
| historyService | 历史Service，用于查询所有历史数据，例如流程实例、任务、活动、变量、附件 |
| managementService | 引擎管理Service，和具体业务无关，主要可以查询引擎配置、数据库、作业等 |

## 5.2 服务组件的使用 HelloWorld程序

服务组件的应用就是对Activiti数据库表做一些CURD的操作，类似于Hibernate框架接下来我们通过一个简单的HelloWorld来演示服务组件的使用。

### 5.2.1、流程图



### 5.2.2、部署流程定义



流程资源有多种方式，如图所示，当流程部署定义好之后，数据库中有四张表会有数据

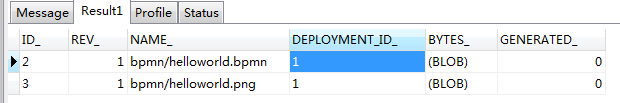
（1）#act\_re\_deployment 部署信息表 select \* FROM act\_re\_deployment



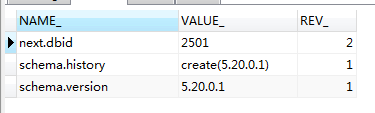
（2）#act\_re\_procdef 流程定义数据表 select \* FROM act\_re\_procdef



（3）act\_ge\_bytearray 二进制数据表 select \* FROM act\_ge\_bytearray

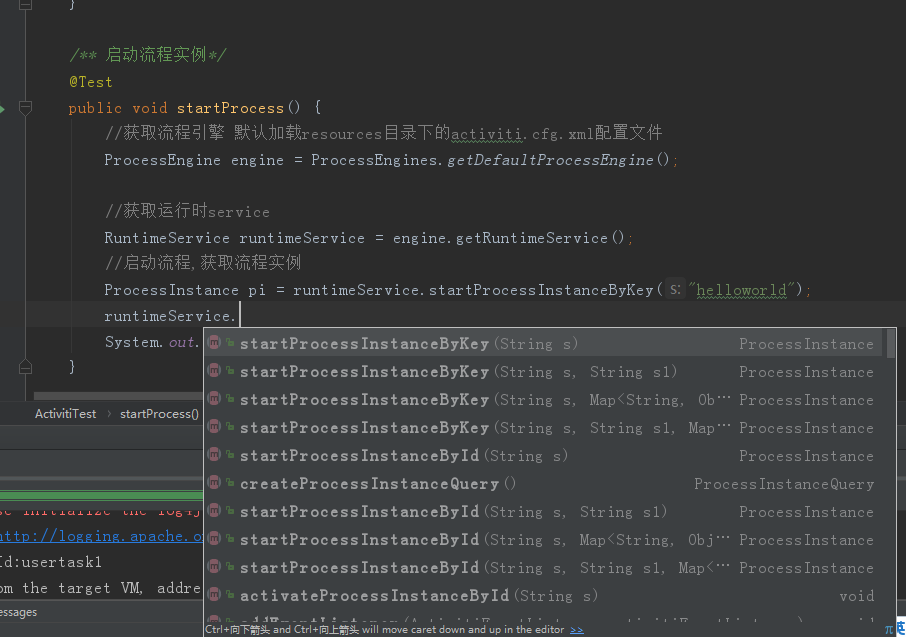


（4）#act\_ge\_property 属性表 select \* FROM act\_ge\_property



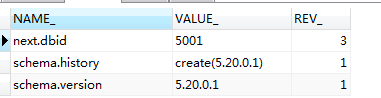
这是主键生成策略，下个id就是2501

### 5.2.3 启动流程实例



启动流程的方式有很多，有id启动，key启动，还可以带参数（参数的使用后面写请假案例的时候会说明），这里采用的是key的启动方式，这种方式的好处就是，每一次启动都是最新的流程定义，启动该流程之后数据库中表的变化

（1）#act\_ge\_property 属性表 select \* FROM act\_ge\_property



指明下一个id为5001

（2）act\_hi\_actinst 历史节点表

（3）act\_hi\_identitylink 历史流程人员表

（4）act\_hi\_procinst 历史流程实例表

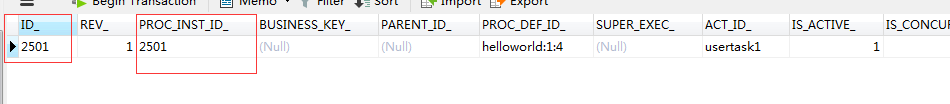
（5）act\_hi\_taskinst 历史任务实例表

（6）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

（7）act\_ru\_task 运行时任务节点表

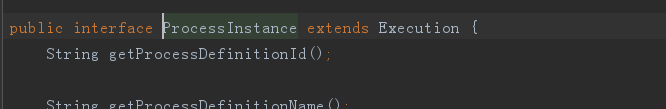
当然还有一些表当启动流程有参数时，也会有数据插入，具体自己分析，接下来讲一个有点意思的表的数据

（8）act\_ru\_execution 运行时流程执行实例表



ExecutionId = 2501,说明了属性表主键生成策略的存在，也说明了，流程启动时，第一条插入的数据就是这张表，ID 和 流程实例id PROC\_INST\_ID一样，这里存在一个问题。

**Execution** Activiti用这个对象去描述流程执行的每一个节点。在没有并发的情况下，Execution就是同ProcessInstance。流程按照流程定义的规则执行一次的过程，就可以表示执行对象Execution。看源码：



从源代码中可以看出ProcessInstance就是Execution。但在现实意义上有所区别：

在单线流程中，如上图的贷款流程，ProcessInstance与Execution是一致的。

多线程时：如：wire money(汇钱)和archive(存档)是并发执行的。 这个时候，总线路代表ProcessInstance，而分线路中每个活动代表Execution。

总结：

\* 一个流程中，执行对象可以存在多个，但是流程实例只能有一个。

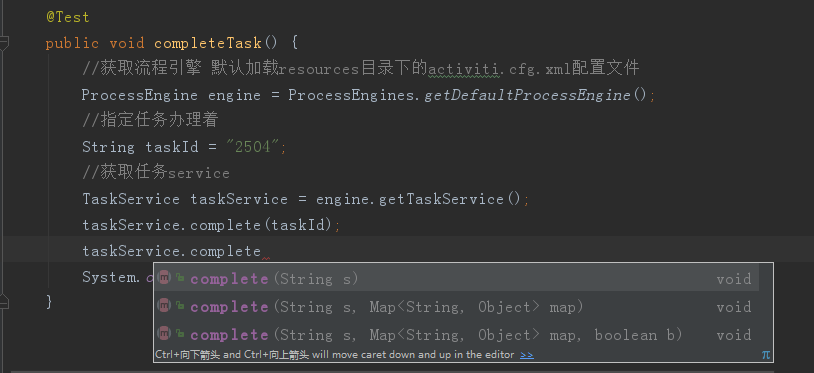
\* 当流程按照规则只执行一次的时候，那么流程实例就是执行对象。

### 5.2.4 查看正在执行中的个人任务



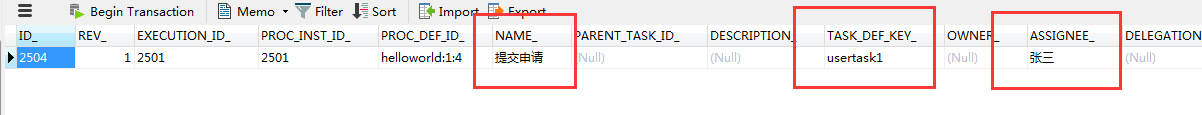
看到链式查询没有，是不是很熟悉，是不是跟Hibernate很相似，如果你能想到这一点，说明还有其他条件的查询方法，需要你自己动手去……点点点就弹出来，因为方法太多，我就不一一列出了，有兴趣可以去研究研究

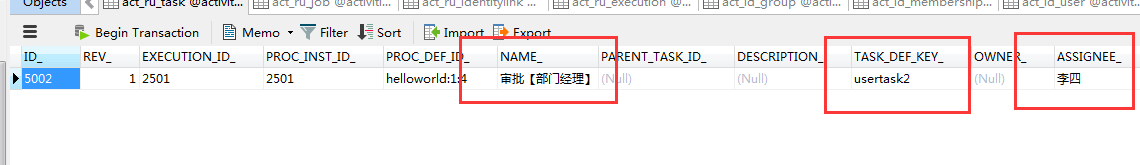
### 5.2.5 完成任务



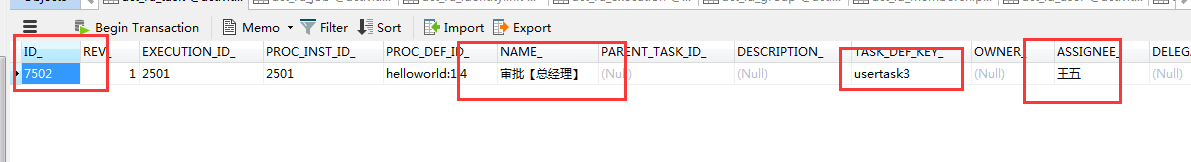
完成任务中可以带有参数，参数可以指定下一个任务由谁办理，或者满足那些条件，该流程怎么走，这跟流程图（bpmn的参数设计有关，待会将）

当任务完成后，开始往下一个任务执行，看数据库的变化：



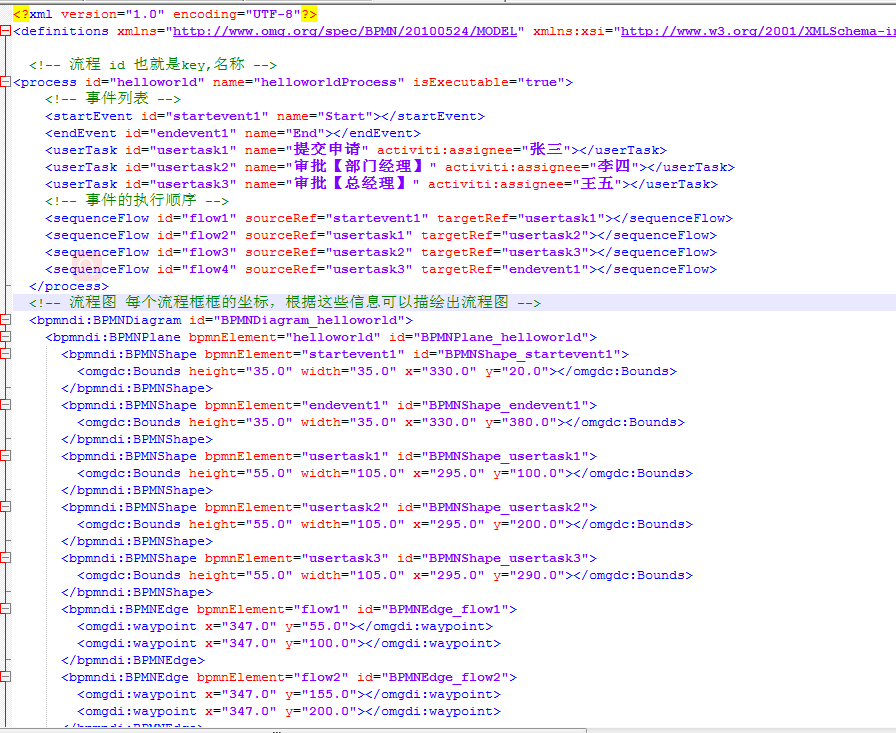


再来一次完成任务，，



由此可见，Activiti工作流的意义就很明显了。开始→完成任务→下一个任务→完成任务→下一个任务→完成任务………………→流程结束。

# bpmn 文件



# 问题抛出

看完上面的文档，应该能有简单的应用activiti完成一个简单的demo了，但是你会不会有疑问：

1. 流程图都是事先准备好的，而且是本地的，能不能自己画好流程图直接部署到数据库中，不用本地添加？
2. 任务办理人都是画流程图时写死的，能不能动态指定任务办理人？
3. 当流程有分支，如何去判断该走那一条分支？
4. 如何实现流程管理业务，让流程与业务建立联系？