# 概述

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

工作流是将一组任务组织起来以完成某个经营过程：定义了任务的触发顺序和触发条件，每个任务可以由一个或多个软件系统完成，也可以由一个或一组人完成，还可以由一个或多个人与软件系统协作完。

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

以请假为例，现在大多数公司的请假流程是这样的

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

采用工作流技术的公司的请假流程是这样的：员工使用账户登录系统——点击请假——上级登录系统点击允许，就这样，一个请假流程就结束了。

## 技术优点

从上面的例子，很容易看出：工作流系统，实现了工作流程的自动化，提高了企业运营效率、改善企业资源利用、提高企业运作的灵活性和适应性、提高量化考核业务处理的效率、减少浪费（时间就是金钱）。

而手工处理工作流程，一方面无法对整个流程状况进行有效跟踪、了解，另一方面难免会出现人为的失误和时间上的延时导致效率低下，特别是无法进行量化统计，不利于查询、报表及绩效评估。

## 主流工作流框架

**流行的工作流框架有**：Activity5.13、JBPM4.4、OSWorkFlow、WorkFlow。工作流框架底层需要有数据库提供支持，activiti5.13版本，有23张表。JBPM4.4框架底层有数据库支持，18张表。JBPM底层使用hibernate操作数据库。Activiti框架底层使用的mybatis操作数据库。

## Activiti简介

Activiti5是由Alfresco软件在2010年5月17日发布的业务流程管理（BPM）框架，它是覆盖了业务流程管理、工作流、服务协作等领域的一个开源的、灵活的、易扩展的可执行流程语言框架。Activiti基于Apache许可的开源BPM平台，创始人Tom Baeyens是JBoss jBPM的项目架构师，它特色是提供了eclipse插件，开发人员可以通过插件直接绘画出业务流程图。

Activiti支持的数据库有：h2、mysql、oracle、mssql、db2等主流数据库。

### 学习要点

1. 1个插件

在Eclipse中安装Activity插件，让你可以在Eclipse中绘制Activity工作流图。

1. 1个引擎

ProcessEngine对象，Activity工作流引擎。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。

所有的操作都是从获取引擎开始的，所以一般会把引擎作为全局变量。

ProcessEngine processEngine =ProcessEngines.getDefaultProcessEngine();

1. 1个配置文件

activiti.cfg.xml。Activiti核心配置文件，配置流程引擎创建工具的基本参数和数据库连接池参数

1. 5种数据库表，一共23张表

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。

activiti5.13版本，有23张表，不同的表存放不同方面的数据，有流程定义表、任务结点表、流程变量表、任务历史表等等。



1. 关键对象

* Deployment：流程部署对象，部署一个流程时创建。
* ProcessDefinitions：流程定义，部署成功后自动创建。
* ProcessInstances：流程实例，启动流程时创建。
* Task：任务，在Activiti中的Task仅指有角色参与的任务，即定义中的UserTask。
* Execution：执行计划，流程实例和流程执行中的所有节点都是Execution，如UserTask、ServiceTask等。

1. 服务接口

不同的Service类对应不同的功能。

* ProcessEngine：流程引擎的抽象，通过它我们可以获得我们需要的一切服务。
* RepositoryService：Activiti中每一个不同版本的业务流程的定义都需要使用一些定义文件，部署文件和支持数据(例如BPMN2.0 XML文件，表单定义文件，流程定义图像文件等)，这些文件都存储在Activiti内建的Repository中。RepositoryService提供了对 repository的存取服务。
* RuntimeService：在Activiti中，每当一个流程定义被启动一次之后，都会生成一个相应的流程对象实例。RuntimeService提供了启动流程、查询流程实例、设置获取流程实例变量等功能。此外它还提供了对流程部署，流程定义和流程实例的存取服务。
* TaskService: 在Activiti中业务流程定义中的每一个执行节点被称为一个Task，对流程中的数据存取，状态变更等操作均需要在Task中完成。TaskService提供了对用户Task 和Form相关的操作。它提供了运行时任务查询、领取、完成、删除以及变量设置等功能。
* IdentityService: Activiti中内置了用户以及组管理的功能，必须使用这些用户和组的信息才能获取到相应的Task。IdentityService提供了对Activiti 系统中的用户和组的管理功能。
* ManagementService: ManagementService提供了对Activiti流程引擎的管理和维护功能，这些功能不在工作流驱动的应用程序中使用，主要用于Activiti系统的日常维护。
* HistoryService: HistoryService用于获取正在运行或已经完成的流程实例的信息，与RuntimeService中获取的流程信息不同，历史信息包含已经持久化存储的永久信息，并已经被针对查询优化。
* FormService：表单Service，用于读取流程、任务相关的表单数据。

1. 7项基本操作

* 设计流程图（各种组件，如连线、用户任务、网关）
* 流程定义增删改查
* 流程变量增删改查
* 启动流程定义
* 任务增删改查
* 完成任务
* 历史信息查询

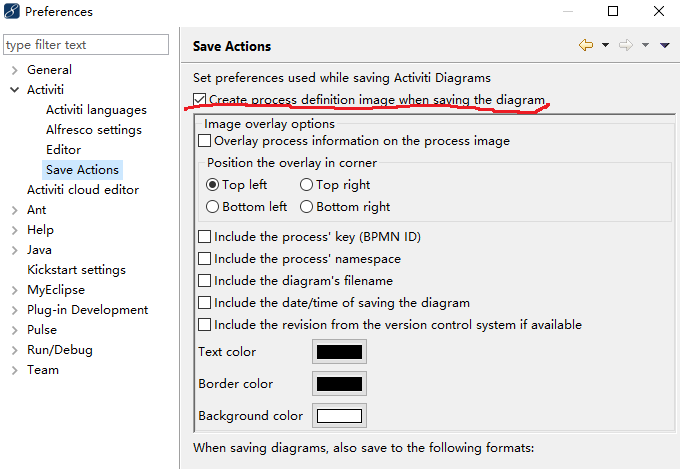
# Activiti与MyEclipse10.7整合

link方法，具体步骤如下：

1. 从<https://github.com/Activiti/Activiti-Designer/releases>中下载最新版本的压缩包；
2. 在MyEclipse安装目录下的dropins文件夹中建立activiti文件夹，并将解压后的features和plugins放到该目录下；
3. 在dropins文件夹中新建activiti.link文件，并在文件中添加一句：

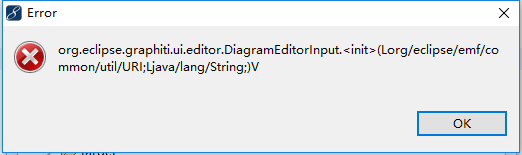
path=E:\\Program Files\\MyEclipse\\MyEclipse 10\\dropins\\activiti（例子）

1. 重新启动MyEclipse，即可在preferences中看到activiti插件。
2. 在MyEclipse的preferences -> Activiti -> Save Actions中勾选如下选项，该选项会在保存流程定义文件时生成相应的流程图片。

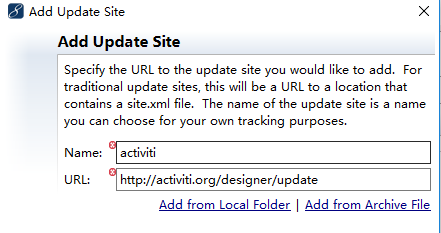


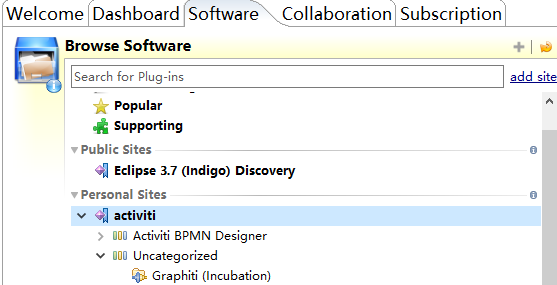
**注意**：

现象：画流程图时会提示以下错误。需另外安装graphiti插件。



解决办法：在线安装graphiti插件，name：activiti，url：<http://www.activiti.org/designer/update>，仅安装graphiti插件（activiti插件在线安装不成功）。





# 下载Activiti并部署在线设计流程

1. https://github.com/Activiti/Activiti/releases/下载所需版本的Activiti，如activiti-5.22.0.zip；
2. 解压activiti-5.22.0.zip，并获取activiti-5.22.0\wars下的activiti-explorer.war；
3. 将activiti-explorer.war解压，并在tomcat7（必须要tomcat7）的webapp下创建activiti-explorer文件夹，将解压后的文件放到该文件夹下；
4. 更改数据库，将需要的数据库驱动放到activiti-explorer\WEB-INF\lib中，然后修改activiti-explorer\WEB-INF\classes目录下的db.properties，指向相应的数据库；
5. 随后启动tomcat，接着在浏览器中输入地址localhost:8080/activiti-explorer；



1. 账号

|  |  |  |
| --- | --- | --- |
| **账号** | **密码** | **角色** |
| kermit | kermit | admin |
| gonzo | gonzo | manager |
| fozzie | fozzie | user |

# 利用MyEclipse画流程图

## Start event

启动事件元素就是启动流程实例的，也就是发起一个流程的，是流程的起点。它可以配置的很简单，也可以很复杂。

### StartEvent

StartEvent：空启动事件。它的属性介绍如下：

1. General

Id: id 标识编号

Name: 名称

asynchronous:是否异步（默认否）表示是否能被异步执行。

exclusive: 是否排他（默认是）表示同一个流程实例在同一时刻只能由一个人执行该步骤, 跟asynchronous互斥。

1. Main config

initiator:定义一个变量来记录谁发起了该流程实例。

Form Key：与其关联的一个外部表单的名字，如果采用了外部表单的话。

1. Documentation

Documentation：文档说明

1. Form

Form properties：动态表单，在这里可以定义一个动态表单，下面是具体的属性介绍。

Id: 字段的主键，唯一标识

Name: 字段的名称

Type: 字段的类型 ，如 string 、long、enum（下拉选择）、date、boolean

Expression：表达式。可以通过计算表达式设置字段的值

Variable：将字段的值以Variable指定的变量名保存。

Default：字段的默认值

Date pattern：属性type为date时，需要设置此属性定义日期格式，例如：yyyy-MM-dd

Readable：是否可以读

Writable：是否可以写

Required：是否必填

Form values：当type为enum时有效。指定多个下拉选项值让其选择。

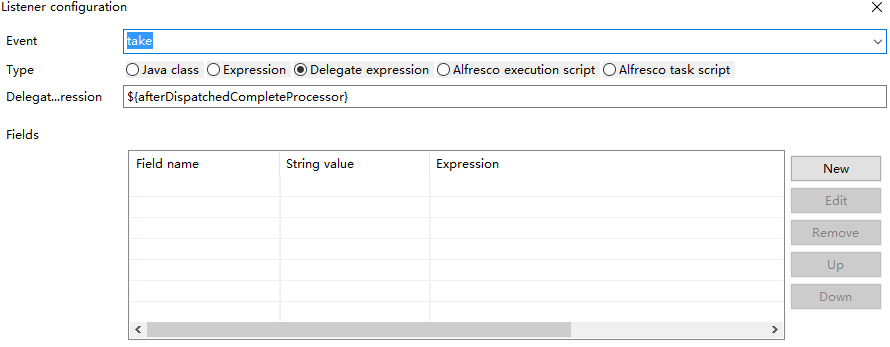
1. Listeners

执行监听器：允许在执行流程过程中执行java代码（实现了监听器的接口）或者表达式，下面是具体属性介绍：

event: 监听事件。分为start、end、take三类。分别表示监听启动、结束、经过三种事件。

Type：执行类型。

* Javaclass：通过指定项目中的类，实现监听执行。（该类需实现org.activiti.engine.delegate.ExecutionListener 接口）
* Expression：通过表达式来实现监听执行。Delegate expression: 引用一个JavaDelegate来调用java bean。



public class AfterDispatchedCompleteProcessor implements Serializable, ExecutionListener{

privatestatic final long serialVersionUID = 8513750196548027535L;

@Override

publicvoid notify(DelegateExecution execution) throws Exception {

System.out.println("流程监听器");

}

}

* Alfresco execution script: 外部定义的执行脚本
* Alfresco task script：外部定义的任务脚本

Fields: 值域。  可以作为参数值传给上面指定的执行对象。

Field Name: 值域名称

String Value：值域值

Expression：值域表达式

### TimerStartEvent

定时器事件是根据指定的时间触发的事件。可以用于开始事件(start event)， 中间事件(intermediate event)和边界事件(boundary event)。定时器事件必须含有下面一种属性的配置。

1. timeDate:指定ISO 8601格式的日期定时器激活。

<timerEventDefinition>

<timeDate>2016-08-23T18:13:00</timeDate>

</timerEventDefinition>

1. timeDuration：定义定时器经过多少时间后激活。
2. timeCycle：定义定时器重复间隔，在某些场景使用，比如周期性的启动流程，任务超时发送提醒。timeCycle的设置目前有两种方式：ISO 8601和Cron表达式(quartz任务调度框架提供的解决方案)，activiti默认是使用ISO 8601。

## Task

1. 设计TASK的表主要是：ACT\_RU\_TASK，ACT\_HI\_TASKINST；
2. 任务分为：UserTask（个人任务）、MailTask（邮件任务）等；
3. 候选人/候选组（candidate）：可以执行任务的一类人或者多个组，候选人/候选组中都可以去签收任务，一旦某人签收，就成为受理人，其他人就不能再签收受理此任务；usertask流程图中，candidate标示候选；候选人涉及的表ACT\_RU\_IDENTITYLINK；
4. 受理人（assignee）：有两种情况，一种是候选人/组中有人签收任务后成为受理人，另外一种是流程图中直接指定受理人，但是可以指定一个动态受理人；受理人涉及的表ACT\_RU\_TASK；
5. 持有人（owner）：持有人设置主要是存入历史表中，用于历史任务的查询，涉及的表ACT\_HI\_TASKINST。

### UserTask

UserTask个人任务属性：

1. Main config

Assignee：直接指定一个办理人；

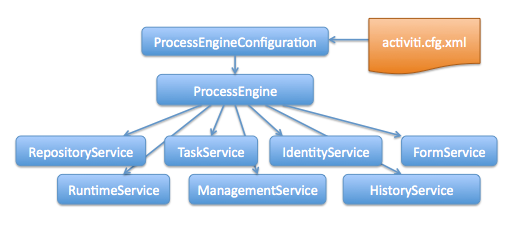
Candidate users：候选人

Candidate group：候选组

候选人/候选组中都可以去签收任务，一旦某人签收，就成为受理人，其他人就不能再签收受理此任务。



# 流程编码



## 编码创建23张表

### 直接编码创建

/\*\*

\* 使用代码创建工作流需要的23张表

\*\*/

@Test

**public** **void** createTable(){

// 获取流程引擎配置(创建一个单例子的流程引擎)

ProcessEngineConfiguration processEngineConfiguration = ProcessEngineConfiguration

.*createStandaloneProcessEngineConfiguration*();

// 设置数据库信息

processEngineConfiguration.setJdbcDriver("com.mysql.jdbc.Driver");

processEngineConfiguration.setJdbcUrl("jdbc:mysql://localhost:3306/test?useUnicode=true&characterEndocing=utf8");

processEngineConfiguration.setJdbcUsername("root");

processEngineConfiguration.setJdbcPassword("jcbkzhzx110");

/\*\*

\* processEngineConfiguration 中的配置

\* DB\_SCHEMA\_UPDATE\_FALSE = "false"; // 不能自动创建表（能够使用脚本创建）

\* DB\_SCHEMA\_UPDATE\_CREATE\_DROP = "create-drop"; // 先删除再更新

\* DB\_SCHEMA\_UPDATE\_TRUE = "true"; // 如果表不存在我就自动创建表

\*/

// 设置数据库操作的设置

processEngineConfiguration.setDatabaseSchemaUpdate(ProcessEngineConfiguration.*DB\_SCHEMA\_UPDATE\_TRUE*);

// 获取工作流的核心对象

@SuppressWarnings("unused")

ProcessEngine processEngine = processEngineConfiguration.buildProcessEngine();

}

### 通过配置文件创建

/\*\* 使用配置文件创建流程引擎 \*/

@Test

**public** **void** createByConfig(){

@SuppressWarnings("unused")

ProcessEngine processEngine = ProcessEngineConfiguration

.*createProcessEngineConfigurationFromResource*("config/activiti.cfg.xml")

.buildProcessEngine();

}

配置文件：

<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"*>

<!-- 创建单例子的流程引擎的配置文件 -->

<bean id=*"processEngineConfiguration"* class=*"org.activiti.engine.impl.cfg.StandaloneProcessEngineConfiguration"*>

<!-- 链接数据库配置

jdbcUrl

jdbcDriver

jdbcUsername

jdbcPassword

jdbcMaxActiveConnections

jdbcMaxIdleConnections

jdbcMaxCheckoutTime

jdbcMaxWaitTime

-->

<property name=*"jdbcUrl"* value=*"jdbc:mysql://localhost:3306/test?useUnicode=true&amp;characterEncoding=utf-8"* />

<property name=*"jdbcDriver"* value=*"com.mysql.jdbc.Driver"* />

<property name=*"jdbcUsername"* value=*"root"* />

<property name=*"jdbcPassword"* value=*"jcbkzhzx110"* />

<!-- 创建表配置自动创建 -->

<property name=*"databaseSchemaUpdate"* value=*"true"*></property>

</bean>

</beans>

详看：https://www.activiti.org/userguide/index.html#configuration

## 部署流程

### 使用文件部署

/\*\*

\* 使用文件部署流程

\*/

**public** **void** deploymentProcessDefinition\_classpath(){

**ProcessEngine ProcessEngine = createByConfig();**//创建工作流引擎

RepositoryService repositoryService = ProcessEngine.getRepositoryService(); // 与流程定义和部署对象相关的Service

Deployment deployment = repositoryService.createDeployment() // 创建一个部署对象

.name("员工请假流程") // 设置对应流程的名称

.addClasspathResource("leave.bpmn") // 从Classpath的资源中加载，一次只能加载一个文件(windows与linux下面要区分)

.addClasspathResource("leave.png") // 从Classpath的资源中加载，图片

.deploy(); // 完成部署

System.*out*.println("部署Id：" + deployment.getId()); // 部署Id：20001

System.*out*.println("部署名称：" + deployment.getName()); // 部署名称：流程定义

}

### 使用zip文件部署

/\*\*

\* 使用zip文件部署流程

\*/

**public** **void** deploymentProcessDefinition\_zip(){

InputStream in = **this**.getClass().getClassLoader().getResourceAsStream("leave.zip");

**ProcessEngine ProcessEngine = createByConfig()**;//创建工作流引擎

RepositoryService repositoryService = ProcessEngine.getRepositoryService(); // 与流程定义和部署对象相关的Service

Deployment deployment = repositoryService.createDeployment() // 创建一个部署对象

.name("zip流程定义") // 添加部署的名称

.addZipInputStream(**new** ZipInputStream(in)) // 制定zip格式文件完成部署

.deploy(); // 完成部署

System.*out*.println("部署Id："+deployment.getId()); // 部署Id:22501

System.*out*.println("部署名称："+deployment.getName()); // 部署名称:zip流程定义

}

### 使用输入流部署

/\*\*部署流程定义zip文件\*/

**public** **void** deploymentProcessDefinition\_inputStream(){

InputStream inputStreambpmn = **this**.getClass().getClassLoader().getResourceAsStream("leave.bpmn");

InputStream inputStreampng = **this**.getClass().getClassLoader().getResourceAsStream("leave.png");

ProcessEngine ProcessEngine = createByConfig();//创建工作流引擎

RepositoryService repositoryService = ProcessEngine.getRepositoryService(); // 与流程定义和部署对象相关的Service

Deployment deployment = repositoryService.createDeployment() // 创建一个部署对象

.name("流程定义输入流") // 添加部署的名称

.addInputStream("leave.bpmn", inputStreambpmn)// 使用资源文件名称（要求与资源文件的名称要一致），和输入流完成部署

.addInputStream("leave.png", inputStreampng)// 使用资源文件名称（要求与资源文件的名称要一致），和输入流完成部署

.deploy(); // 完成部署

System.*out*.println("部署Id："+deployment.getId()); // 部署Id:55001

System.*out*.println("部署名称："+deployment.getName()); // 部署名称:流程定义输入流

}

## 查询流程及流程图

### 查询流程

/\*\*

\* 查询流程定义

\*/

@Test

**public** **void** getProcessDefinition(){

**ProcessEngine ProcessEngine = createByConfig();**//创建工作流引擎

RepositoryService repositoryService = ProcessEngine.getRepositoryService(); // 与流程定义和部署对象相关的Service

/\*

\* 有两个对象，一个是ProcessDefinition（流程定义），一个是Deployment（流程部署）

\*/

ProcessDefinitionQuery processDefinitionQuery = repositoryService.createProcessDefinitionQuery()

.orderByProcessDefinitionKey().asc()

.orderByProcessDefinitionVersion().desc();

List<ProcessDefinition> processDefinitionList = processDefinitionQuery.list();

**for** (ProcessDefinition processDefinition : processDefinitionList) {

String deploymentId = processDefinition.getDeploymentId();

Deployment deployment = repositoryService.createDeploymentQuery().deploymentId(deploymentId).singleResult();

System.*out*.println("流程定义编号：" + processDefinition.getId()

+ " 部署编号：" + processDefinition.getDeploymentId()

+ " 名称：" + processDefinition.getName()

+ " 流程key：" + processDefinition.getKey()

+ " 版本号：" + processDefinition.getVersion()

+ " 源文件：" + processDefinition.getResourceName()

+ " 图片：" + processDefinition.getDiagramResourceName()

+ " 部署时间：" + deployment.getDeploymentTime());

}

}

### 查看流程图

/\*\*

\* 获取流程定义资源

\* **@param** processDefinitionId 流程id

\* **@param** resourceType 资源类型

\*/

@Test

**public** **void** getProcessDefinitionResource(String processDefinitionId, String resourceType) {

ProcessEngine ProcessEngine = createByConfig();//创建工作流引擎

RepositoryService repositoryService = ProcessEngine.getRepositoryService(); // 与流程定义和部署对象相关的Service

ProcessDefinition processDefinition = repositoryService.createProcessDefinitionQuery().processDefinitionId(processDefinitionId).singleResult();

String resourceName = "";

**if** (resourceType.equals("image")) {

resourceName = processDefinition.getDiagramResourceName();

} **else** **if** (resourceType.equals("xml")) {

resourceName = processDefinition.getResourceName();

}

//获取资源文件流

InputStream resourceAsStream = repositoryService.getResourceAsStream(processDefinition.getDeploymentId(), resourceName);

}

## 删除流程

/\*\*

\* 使用部署ID删除流程定义

\*/

**public** **static** **void** deleteProcessDefinition(String deploymentId) {

ProcessEngine processEngine = *createByConfig*();//创建工作流引擎

/\*

\* 不带级联的删除

\* 只能删除没有启动的流程，如果流程启动，就会抛出异常

\*/

// processEngine.getRepositoryService()//与流程定义和部署对象相关的Service

// .deleteDeployment(deploymentId);

/\*

\* 能级联的删除

\* 能删除启动的流程，会删除和当前规则相关的所有信息，正在执行的信息，也包括历史信息

\*/

processEngine.getRepositoryService()//与流程定义和部署对象相关的Service

.deleteDeployment(deploymentId, **true**);

}

## 启动一任务流程

## 跟踪一任务流程

## 获取待办任务

## 签收一任务流程

## 审批一任务流程

# MyEclipse整合Activiti

## spring文件配置

<!-- Activiti配置 -->

<bean id=*"processEngineConfiguration"* class=*"org.activiti.spring.SpringProcessEngineConfiguration"*>

<!-- 数据源 -->

<property name=*"dataSource"* ref=*"dataSource"*/>

<!-- 数据库类型（一般不需要） h2, mysql, oracle, postgres, mssql, db2 -->

<property name=*"databaseType"* value=*"mysql"*/>

<!-- 统一事物处理 -->

<property name=*"transactionManager"* ref=*"transactionManager"*/>

<!-- 生成表策略(建议初始化后关闭)

false（默认）：检查已有数据库表但不生成或更新表，如果版本不一致或表不存在，则会抛出异常

true：检查已有数据库表，如果版本不一致或表不存在，则会更新或生成表

create-drop：引擎启动时生成表，引擎关闭时删除表 -->

<property name=*"databaseSchemaUpdate"* value=*"true"*/>

<!-- 是否开启job引擎，该引擎会定时的查询act\_ru\_job表，如某一条记录的时间要求达到了，就会执行这一个JOB

和Job Executor相对应，activiti从5.17.0版本开始提供性能更好，数据库更友好的Async Executor

来执行异步任务，所以activiti推荐使用Async Executor

工作流引擎初始化时，job executor默认是被激活的 -->

<property name=*"jobExecutorActivate"* value=*"true"*/>

<!-- activiti默认激活使用job executor，要使用async executor替代job executor需在配置文件中定义一下两个属性

<property name="asyncExecutorEnabled" value="true" />

<property name="asyncExecutorActivate" value="true" /> -->

<!-- 历史归档的级别

none：忽略所有的历史归档。尽管对于运行期流程执行来说这是性能最高的，但是没有历史信息保留

activity：归档所有流程实例和活动实例。不归档细节。

audit：缺省级别。它归档所有流程实例，活动实例和提交的表单属性。以至于通过表单的所有用户交互都是可跟踪并可以被审计。

full：历史归档的最高级别。所以是最慢的。这个水平保存audit级别的所有信息加上像流程变量的所有其它可能的细节。 -->

<property name=*"history"* value=*"audit"*/>

<!-- 流程定义缓存配置

所有被解析的流程定义都会可以被缓存起来，因为流程定义是不会改变的，没有必要多次请求数据库。

默认的activiti是没有限制缓存数量的 -->

<property name=*"processDefinitionCacheLimit"* value=*"100"*/>

<!-- 每个涉及到用户的系统都有一套用户权限管理平台或者模块，用来维护用户以及在系统内的功能、数据权限，

Activiti工作流引擎配套设计了包括User、Group的Identify模块，需要实现数据同步

方法一：用视图覆盖同名的ACT\_ID\_系列表（act\_id\_group、act\_id\_info、act\_id\_membership、act\_id\_user

1.删除已创建的act\_id\_group、act\_id\_info、act\_id\_membership、act\_id\_user表

2.创建视图 创建的视图要保证数据类型一致

3.修改引擎默认配置 设置属性dbIdentityUsed为false即可 -->

<property name=*"dbIdentityUsed"* value=*"false"*/>

<!-- 生成流程图的字体 -->

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

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

<!-- 是否开启工作的数据日志 -->

<property name=*"enableDatabaseEventLogging"* value=*"false"* />

</bean>

<!-- 加载activiti引擎 -->

<bean id=*"processEngine"* class=*"org.activiti.spring.ProcessEngineFactoryBean"*>

<property name=*"processEngineConfiguration"* ref=*"processEngineConfiguration"*/>

</bean>

<!-- activiti的各种服务接口 -->

<bean id=*"repositoryService"* factory-bean=*"processEngine"* factory-method=*"getRepositoryService"*/>

<bean id=*"runtimeService"* factory-bean=*"processEngine"* factory-method=*"getRuntimeService"*/>

<bean id=*"taskService"* factory-bean=*"processEngine"* factory-method=*"getTaskService"*/>

<bean id=*"historyService"* factory-bean=*"processEngine"* factory-method=*"getHistoryService"*/>

<bean id=*"managementService"* factory-bean=*"processEngine"* factory-method=*"getManagementService"*/>

<bean id=*"identityService"* factory-bean=*"processEngine"* factory-method=*"getIdentityService"*/>

## maven配置