Activiti 第一天 流程定义、流程定义部署

Activiti四天课：

第一天：流程定义、流程部署

Activiti是什么？(掌握)

Activiti解决了什么问题？(掌握)

流程定义(掌握)

流程定义部署(掌握)

第二天：流程实例、个人任务

以完整的业务流程，用activiti实现。

Activiti是如何开发！！！

第三天：流程监控、流程变量、连接（流程分支）

Activiti深入学习，将案例流程进行完善

第四天：网关、组任务

Activiti深入学习，将案例流程进行完善

统一讲课的思路：

先讲api

再讲案例

第一天：流程定义、流程部署

学习没有加入activiti的系统（采购系统）：

部署一个环境没有加入activiti的系统(springmvc3.1.4+mybatis3.2.3+mysql5.1)

总结：没有工作流的问题。

引出工作流：

工作流是什么？(掌握)

Activiti是什么？(掌握)

学习activiti：

部署环境：

1. activiti的api测试环境
2. activiti应用案例（采购系统）环境

activiti是如何工作的，工作步骤：（掌握）

activiti流程定义、流程定义部署(掌握)

流程定义、流程定义部署在采购中应用：

1. spring+activiti整合
2. 流程定义、流程定义部署功能开发

# 学习采购系统

## 需求

员工在企业中要采购办公用品等，先创建采购单、先由部门经理审核、再由总经理审核、最后由财务审核。

角色：

员工

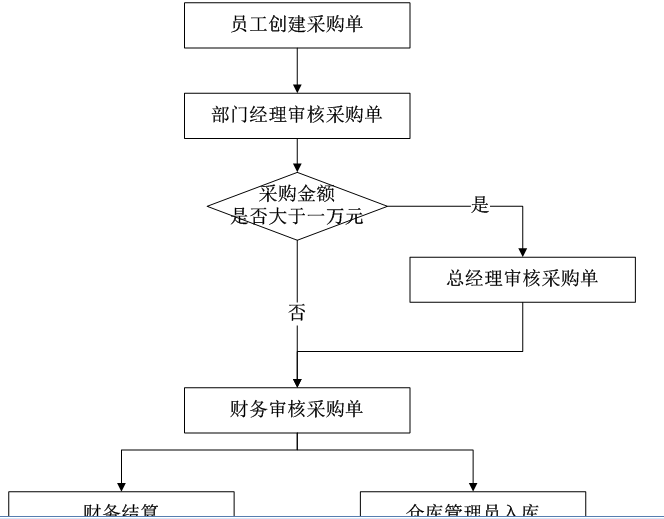
部门经理

总经理

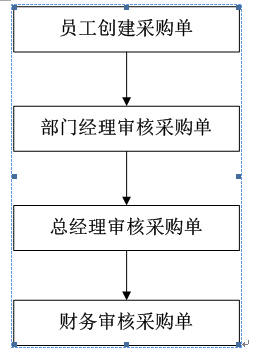
财务人员

## 业务流程

从需求中分析业务流程，业务流程由不同的用户角色共同完成任务。



将上边的流程简化：



## 业务概念

什么是业务？

用户的需求就是业务。

什么是业务功能？

使用计算机技术开发软件满足用户需求，开发软件功能就是业务功能。

什么是业务系统？

开发一个软件系统，满足用户的需求，就是业务系统。

什么是业务流程？

由多个（可以一个）用户角色，使用业务功能操作一个流程，就是业务流程，业务功能组成了业务流程。

业务系统核心（灵魂）是什么？

技术不是业务系统核心，业务流程是业务系统核心。

## 采购系统部署

开发环境：

Eclipse indigo

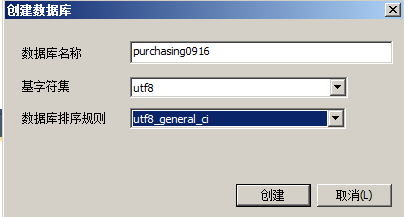
Jdk1.7.72

系统架构：spring3.1.4(包括springmvc)+mybatis3.2.3+mysql

数据库

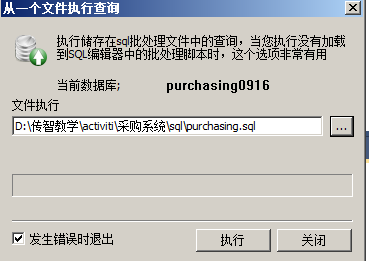
创建数据库：

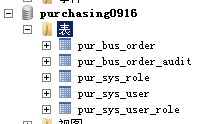
用于采购系统(没有加入工作流)学习。



导入mysql数据库（purchasing0916）：







Pur\_sys\_role：角色表

Pur\_sys\_user：用户表

Pur\_sys\_user\_role：用户角色表(关系表)

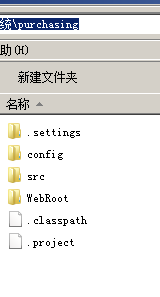
Pur\_bus\_order：采购单的订单表

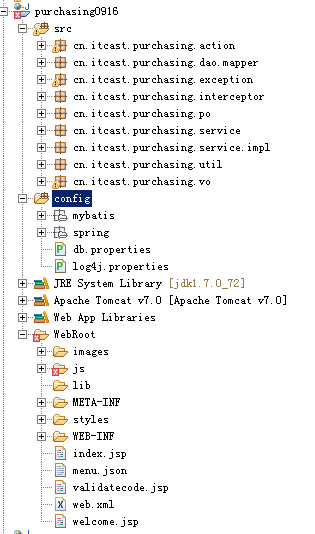
存储员工创建的采购单信息

Pur\_bus\_order\_audit：采购单审核信息表

存储采购单审核信息

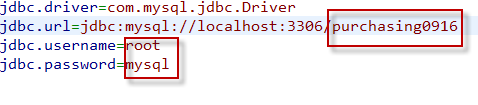
在eclipse中创建web工程

将原始工程内容拷贝到新创建的工程中；  


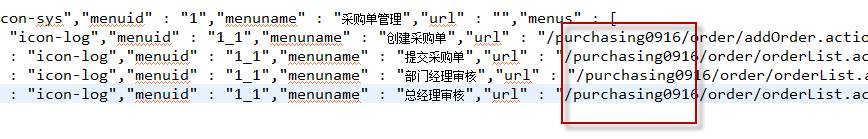


## 修改配置文件

修改 db.properties



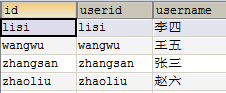
修改menu.json



## 部署系统运行

在tomcat中部署purchasing0916，启动tomcat。

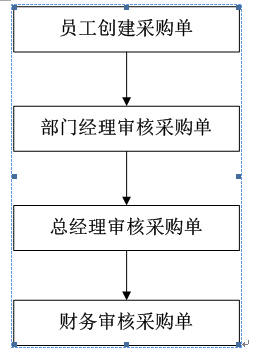
登陆账号：



密码为111111

## 采购系统功能

业务流程：



创建采购单

用工员账号登陆，点击“创建采购单”：



输入采购信息，点击提交，进入提交 采购单页面：

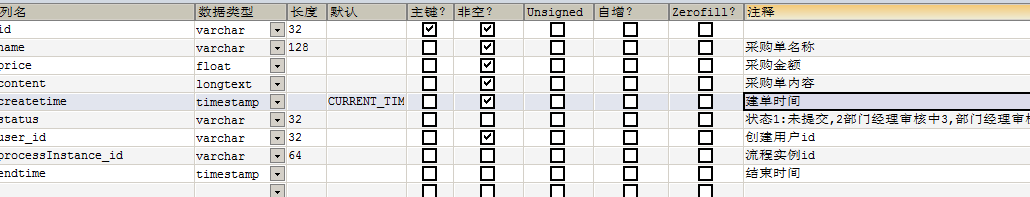
当采购单不再修改时，执行提交操作。



数据库操作

创建采购单：

向pur\_sys\_order表插入一条记录（采购单状态设置为1:未提交）



采购单状态 ：

业务代码，每一个代码表示一种状态，使用状态字段记录采购单流程运行的过程。

1:未提交,

2部门经理审核中

3,部门经理审核不通过,

4总经理审核中,

5总经理审核不通过

6财务审核中,

7财务审核不通过

提交采购单：

更新采购单状态为2部门经理审核中

部门经理审核

使用部门经理账号登陆系统，进入部门经理审核页面：



点击“部门经理审核”链接：



输入审核意见,提交。

数据库操作

查询待审核的采购单：

只查询采购单状态为2部门经理审核中的记录

审核采购单：

插入pur\_bus\_order\_audit表记录

更新采购单状态：

如果审核不通过，更新为3,部门经理审核不通过,

如果审核通过，更新为4总经理审核中

总经理审核

使用总经理账号登陆系统，进入总经理待审核列表：



点击“总经理审核”，



输入审核意见，提交。

数据库操作

查询总经理待审核的列表：

只查询采购单状态为4部门经理审核中的记录

总经理审核：

插入pur\_bus\_order\_audit表记录

更新采购单状态：

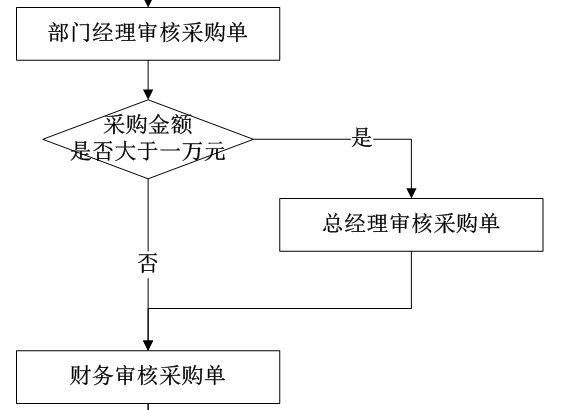
如果审核不通过，更新为5,总经理审核不通过,

如果审核通过，更新为6财务审核中

## 需求变更

变更1

针对上边的流程变更如下：



系统修改：

修改部门经理审核代码：

在部门经理审核提交 时，判断采购金额

如果采购金额大于等于10000元：

如果部门经理审核通过，将采购状态更新为4总经理审核中

如果采购金额小于10000元：

如果部门经理审核通过，将采购状态更新为6财务审核中

变更2

将上边的流程中总经理审核和财务审核位置颠倒，变为：



修改部门经理审核代码、财务审核代码、总经理审核代码。

比如财务审核采购单修改：

审核通过更新pur\_bus\_order表中的status状态为4（总经理审核中）

## 采购系统问题

采购单状态意义：业务系统设置该状态为了管理流程运行过程(创建采购单完成，由部门经理审核、再由总经理审核…)。

上边需求变更（变更了流程）需要修改大量的代码(主要是修改更新采购单状态代码)，因为将采购单状态硬编码在java代码中。系统可扩展性比较差。

工作流就能解决上边的问题，工作流可以将上边的业务流程运行进行统一管理(如果 仅是流程变更不需要业务系统修改代码)，不用业务系统单独处理业务流程执行过程。

好处：

业务系统不用单独处理业务流程过程，就可以集中精力去处理业务功能（用户需求）。

注意：

上边不用工作流也能实现业务流程管理，建议针对简单的业务流程可以由业务系统来管理业务流程，为什么不用工作流来管理，因为使用工作流增加成本，增加开发工作量。针对复杂的业务流程建议使用工作流管理业务流程，如果流程节点较多，一旦修改流程需要修改代码量比较大。

# 什么工作流？

## 什么工作流？

工作流(Workflow)，就是通过计算机对业务流程自动化执行管理。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自动进行传递文档、信息或任务的过程，从而实现某个预期的业务目标，或者促使此目标的实现”。

工作流就是 通过计算机技术对业务流程进行自动化管理。实现多个参与者按照预定的流程去自动执行业务流程。

## 工作流系统

具有工作流功能的系统就是工作流系统。工作流系统能够对业务流程进行自动化管理。

适用行业：

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。

工作流系统应用

1. 关键业务流程：订单、报价处理、采购处理、合同审核、客户电话处理、供应链管理等
2. 行政管理类:出差申请、加班申请、请假申请、用车申请、各种办公用品申请、购买申请、日报周报等凡是原来手工流转处理的行政表单。
3. 人事管理类：员工培训安排、绩效考评、职位变动处理、员工档案信息管理等。
4. 财务相关类：付款请求、应收款处理、日常报销处理、出差报销、预算和计划申请等。
5. 客户服务类：客户信息管理、客户投诉、请求处理、售后服务管理等。
6. 特殊服务类：ISO系列对应流程、质量管理对应流程、产品数据信息管理、贸易公司报关处理、物流公司货物跟踪处理等各种通过表单逐步手工流转完成的任务均可应用工作流软件自动规范地实施。

# 什么是Activiti

Activiti是一个工作流引擎（框架，组件，jar包），通常activiti引擎需要嵌入到业务系统中去运行。

Alfresco软件在2010年5月17日宣布Activiti业务流程管理（BPM）开源项目的正式启动，其首席架构师由业务流程管理BPM的专家 Tom Baeyens担任，Tom Baeyens就是原来jbpm的架构师，而jbpm是一个非常有名的工作流引擎，当然activiti也是一个工作流引擎。

Activiti前身 是jbpm4，Activiti架构和jbpm4基本上是一样的。

Activiti是一个工作流引擎， activiti可以将业务系统中复杂的业务流程抽取出来，使用专门的建模语言（BPMN2.0）进行定义，业务系统按照预先定义的流程进行执行，实现了业务系统的业务流程由activiti进行管理，减少业务系统由于流程变更进行系统升级改造的工作量，从而提高系统的健壮性，同时也减少了系统开发维护成本。

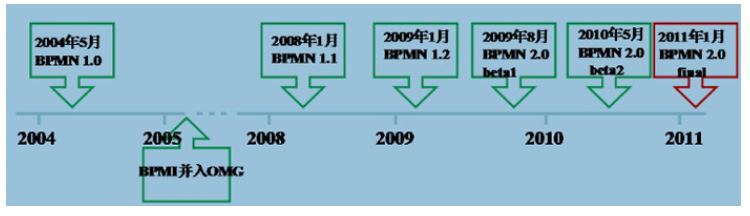
## BPM

Bpm（Business Process Management），业务流程管理，是一种理念，以持续的提高组织业务[绩效](http://baike.baidu.com/view/122994.htm)为目的系统化方法，常见商业管理教育如[EMBA](http://baike.baidu.com/view/29918.htm)、MBA等均将BPM包含在内。

## bpmn

bpmn(Business Process Model and Notation)，业务流程模型和符号。和bpm是两个不同的概念。Activiti使用bmpn标准进行流程建模。是由BPMI（Business Process Management Initiative）开发的一套标准的业务流程建模符号，使用BPMN提供的符号可以创建业务流程。

具体发展历史如下:

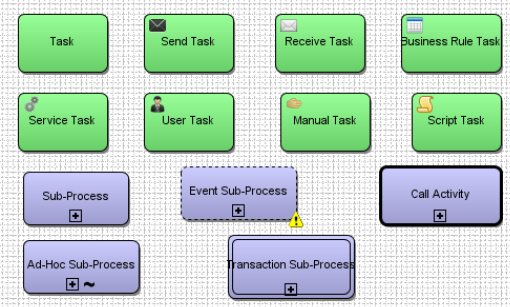


BPMN 是目前被各 BPM 厂商广泛接受的 BPM 标准。Activiti 就是使用 BPMN 2.0 进行流程建模、流程执行管理，它包括很多的建模符号，比如：

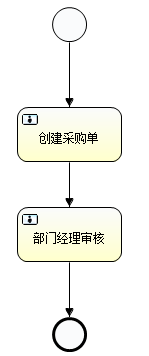
Event 用一个圆圈表示，它是流程中运行过程中发生的事情。



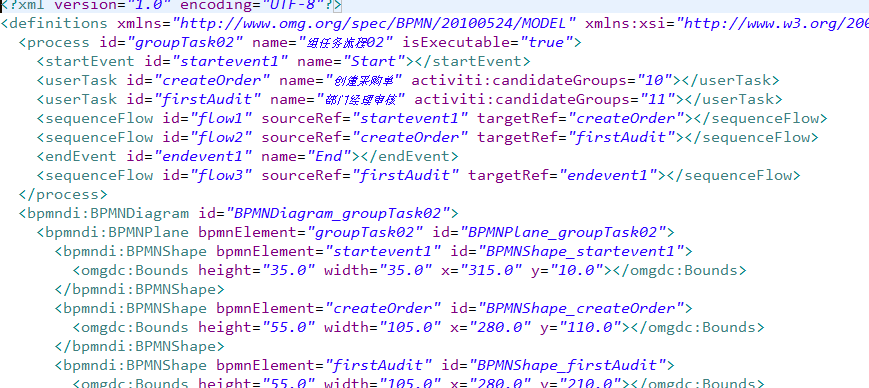
活动用圆角矩形表示，一个流程由一个活动或多个活动组成



一个bpmn图形的例子：



Activiti使用Bpmn2.0标准进行流程定义(流程建模)，bpmn采用xml描述 。



如何使用bpmn进行流程建模：

通常情况下，使用建模工具进行流程图绘制，生成一个图形，通过工具查看bpmn的xml描述 。

如果熟练，手动编写pbmn的xml文档。(对坐标和结点的大小通过工具完成)

# activiti的使用过程

## 第一步：部署activiti环境

Jar包、数据库（activiti支持很多类型的数据库）

## 第二步：流程定义

使用bpmn2.0标准进行流程定义，使用activiti提供流程建模工具(activity-Desinger)

业务分析人员（也可以包括系统设计人员）在线下使用activity-Desinger绘制流程图。

比如：定义一个采购流程(bpmn的xml描述),静态

## 第三步：向activiti中部署流程定义

让activiti去管理某个业务流程，需要使用activiti的api将线下定义的流程部署到activiti数据库中。

## 第四步：启动一个流程实例

流程实例：如果要让用户按照定义的流程（xml）去执行（由activiti管理），启动一个流程实例。

比如：张三创建采购单启动一个流程实例，李四创建采购单，也启动一个流程实例。

简单理解 为：java类和java对象。

## 第五步：查询待办任务

当前用户查询自己名下当前需要办理的任务，由activiti自动化管理了。

当前用户登陆系统，只要通过activiti的api得到当前用户的待办任务（不限采购流程，activiti所有管理流程，该用户的待办任务都能查询到），不需要由业务流程进行处理了。

## 第六步：办理任务

办理任务要填写的信息由业务系统来处理，任务办理后流程执行到哪一步由activiti自动管理。

任务完成后，**流程**的执行由activiti自动管理。

## 第七步：流程结束

当流程运行到最后一个结点，流程结束。



# 环境准备

## 三个环境

1. 未加入工作流的采购系统

Springmvc3.1.4+mybatis3.2.3+mysql5.1

1. Activiti的api测试环境

Java工程

1. 加入工作流的采购系统

Web工程

## Activiti测试环境

包括activiti的jar和activiti数据库

下载activiti

Activiti下载地址：<http://activiti.org/download.html>



1. **Database：**

activiti运行需要有数据库的支持，支持的数据库有：h2, mysql, oracle, postgres, mssql, db2等，该目录存放activiti的建表脚本。

1. **Docs**

Activiti的帮助文档。

1. **Libs**

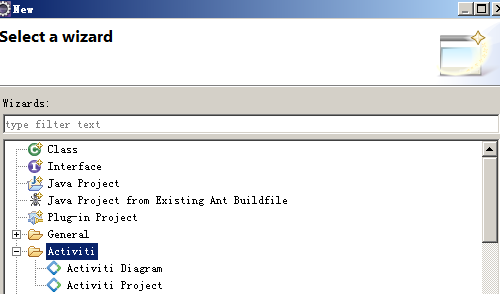
Activiti所需要的jar 包

1. **Wars**

官方自带的示例工程。

安装流程设计器(activiti-Designer)

采用离线安装方法，将eclipse的activiti-Designer插件安装在eclipse中。



官方提交 的Designer插件支持两个版本：

Eclipse –juno

Eclipse – indigo（采用该版本）

Activiti支持的数据库



创建数据库的方式：

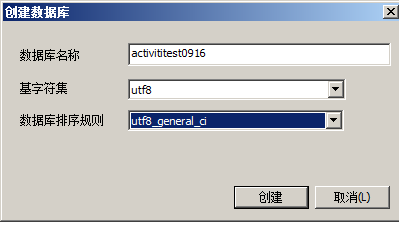
方式1、使用activiti目录 database下的数据库创建脚本创建数据库。

方式2、使用java程序来创建数据库（自动创建数据库），采用java程序创建数据库

Java程序可以创建数据库、可以删除数据库、可以升级数据库，这个过程是java程序 自动执行的。

Java程序创建数据库

在mysql中创建数据库



第一步：创建java工程activiti测试环境

创建java工程，当作activiti测试环境。



第二步：

向该工程加入jar包



上边：activiti单独 运行的jar

Activiti在运行时依赖spring

下边：activiti和spring整合后的jar包（使用这里边的jar包）

包括：activiti和spring的所有jar、数据库驱动、dbcp连接池..

第三步：配置activiti.cfg.xml

该配置文件是spring的配置文件,

Activiti通过该文件创建processEngineConfiguration，通过processEngineConfiguration创建processEngine

在classpath下创建activiti.cgf.xml文件：

包括：数据源、processEngineConfiguration



配置数据库执行策略

关于*processEngineConfiguration*中的**databaseSchemaUpdate参数，**通过此参数设计activiti数据表的处理策略，参数如下：

false（默认）：检查数据库表的版本和依赖库的版本， 如果版本不匹配就抛出异常。

true: 构建流程引擎时，执行检查，如果需要就执行更新。 如果表不存在，就创建。（常用）

create-drop: 构建流程引擎时创建数据库表， 关闭流程引擎时删除这些表。

drop-create：先删除表再创建表。（常用，使用完成后改为true）

create: 构建流程引擎时创建数据库表， 关闭流程引擎时不删除这些表。

第四步：编写日志 文件

log4j.properties

# Global logging configuration

log4j.rootLogger=DEBUG, stdout

# Console output...

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n

第五步：编写程序

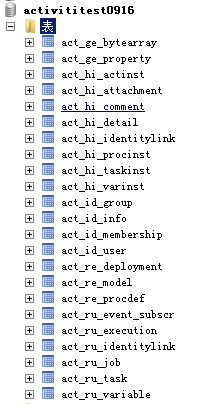
Java程序创建数据库思路：

通过java程序创建activiti核心引擎对象（ProceccEngine），创建引擎对象时候，自动检测数据库环境，根据设置activiti数据库操作策略，操作数据库。



第六步：数据库完成

总共23张表



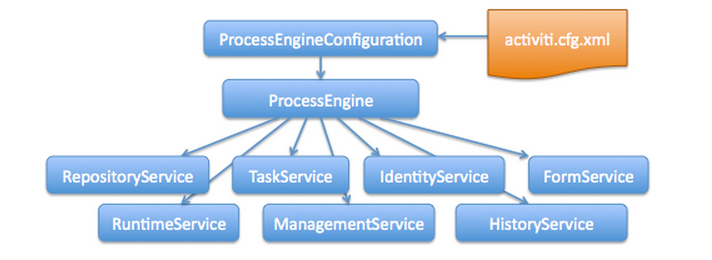
表的命名规则：

Activiti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\_\***: GE表示general。通用数据， 用于不同场景下。

# Activiti服务架构

## 架构图



Activiti核心配置文件：activiti.cfg.xml

ProcessEngineConfiguration：通过activiti.cfg.xml创建。

ProcessEngine:通过ProcessEngineConfiguration创建

Service：通过ProcessEngine获取。 (核心重点)

## activiti.cfg.xml

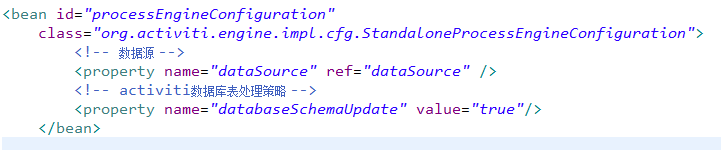
是spring配置文件。

配置数据源、ProcessEngineConfiguration，通过activiti.cfg.xml创建ProcessEngineConfiguration.

## ProcessEngineConfiguration

StandaloneProcessEngineConfiguration

用于activiti单独 运行。



**SpringProcessEngineConfiguration**

Spring和activiti整合后使用该类。

和spring整合后，直接从spring容器中获取**ProcessEngine及各各service。**

<!-- 工作流引擎配置bean -->

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

<!-- 数据源 -->

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

<!-- 使用spring事务管理器 -->

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

<!-- 数据库策略 -->

<property name=*"databaseSchemaUpdate"* value=*"drop-create"* />

<!-- activiti的定时任务关闭 -->

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

</bean>

<!-- 流程引擎 -->

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

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

</bean>

<!-- 资源服务service -->

<bean id=*"repositoryService"* factory-bean=*"processEngine"*

factory-method=*"getRepositoryService"* />

<!-- 流程运行service -->

<bean id=*"runtimeService"* factory-bean=*"processEngine"*

factory-method=*"getRuntimeService"* />

<!-- 任务管理service -->

<bean id=*"taskService"* factory-bean=*"processEngine"*

factory-method=*"getTaskService"* />

<!-- 历史管理service -->

<bean id=*"historyService"* factory-bean=*"processEngine"*

factory-method=*"getHistoryService"* />

<!-- 用户管理service -->

<bean id=*"identityService"* factory-bean=*"processEngine"*

factory-method=*"getIdentityService"* />

<!-- 引擎管理service -->

<bean id=*"managementService"* factory-bean=*"processEngine"*

factory-method=*"getManagementService"* />

<!-- 数据源 -->

<bean id=*"dataSource"* class=*"org.apache.commons.dbcp.BasicDataSource"*>

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

<property name=*"url"* value=*"jdbc:mysql://localhost:3306/activiti"* />

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

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

<property name=*"maxActive"* value=*"3"* />

<property name=*"maxIdle"* value=*"1"* />

</bean>

<!-- 事务管理器 -->

<bean id=*"transactionManager"*

class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>

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

</bean>

<!-- 通知 -->

<tx:advice id=*"txAdvice"* transaction-manager=*"transactionManager"*>

<tx:attributes>

<!-- 传播行为 -->

<tx:method name=*"save\*"* propagation=*"REQUIRED"* />

<tx:method name=*"insert\*"* propagation=*"REQUIRED"* />

<tx:method name=*"delete\*"* propagation=*"REQUIRED"* />

<tx:method name=*"update\*"* propagation=*"REQUIRED"* />

<tx:method name=*"find\*"* propagation=*"SUPPORTS"* read-only=*"true"* />

<tx:method name=*"get\*"* propagation=*"SUPPORTS"* read-only=*"true"* />

</tx:attributes>

</tx:advice>

<!-- 切面，根据具体项目修改切点配置 -->

<aop:config proxy-target-class=*"true"*>

<aop:advisor advice-ref=*"txAdvice"*

pointcut=*"execution(\* cn.itcast.purchasing.service.impl.\*.\*(..))"* />

</aop:config>

### **创建processEngineConfiguration**

ProcessEngineConfiguration configuration = ProcessEngineConfiguration

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

上边的代码要求activiti.cfg.xml中必须有一个processEngineConfiguration的bean

也可以使用下边的方法，更改bean 的名字：

ProcessEngineConfiguration.createProcessEngineConfigurationFromResource(String resource, String beanName);

## ProcessEngine

是一个门面接口，通过该门面得到各各service。

编程方式：

// 通过ProcessEngineConfiguration创建引擎 对象ProcessEngine

ProcessEngine processEngine = configuration.buildProcessEngine();

简单方式：

不通过ProcessEngineConfiguration创建，processEngine，使用下边的代码创建成功。

//使用classpath下的activiti.cfg.xml（名称固定，配置中有id为的processEngineConfiguration配置）中的配置创建processEngine

ProcessEngine processEngine = ProcessEngines.*getDefaultProcessEngine*();

System.*out*.println(processEngine);

## service

包括的service

|  |  |
| --- | --- |
| RepositoryService | activiti的资源管理类 ，用于流程部署，获取流程部署的资源。。 |
| RuntimeService | activiti的流程运行管理类，用于启动一个流程实例，操作当前运行的流程信息 |
| TaskService | activiti的任务管理类，操作个人任务，组任务 |
| HistoryService | activiti的历史管理类，用于操作历史 流程信息 |
| IdentityService | activiti的用户身份管理类，设置用户、组(角色)、用户和组关系 。。 |
| FormService | activiti的表单管理类，实现activiti中配置表单数据。 |
| ManagerService | activiti的引擎管理类，用于activiti运行的监控，给超级管理员使用。 |

注：红色标注为常用service。

创建方式

通过ProcessEngine创建Service。

调用 ProcessEngine.getXXXXX方法获取service。

注意：如果和spring整合后，service直接可以从spring容器中获取。

# 课程内容小结

1、学习没有加入工作流的采购系统

总结问题：

在采购单表 pur\_bus\_order中设置status字段(采购单状态)，为了记录流程执行的过程。

1:未提交,2部门经理审核中3,部门经理审核不通过,4总经理审核中,5总经理审核不通过 6财务审核中,7财务审核不通过

业务系统自己实现流程的管理。

如果使用工作流来管理业务系统业务流程执行过程，业务系统就不用去管理流程的执行过程，业务系统就可以集中精力作业务(用户需求)开发。

2、工作流是什么？

是通过计算机自动化管理业务系统业务流程。实现多个参与者按照预定义的流程自动化去执行。

工作流系统，具体工作流功能的系统就是工作流系统。

3、activiti是什么？

Activiti是实现工作流的一个引擎，通过activiti开发工作流系统。

1. activiti执行过程（重点）

第一步：部署activiti

包括：activiti的jar包、activiti数据库

第二步：流程定义

Activiti遵循bpmn2.0标准进行流程建模。

线下由业务分析人员通过可视化的图形工具进行流程建模。生成xml的文件（是静态文件，bpmn2.0标准描述 流程使用xml描述 ）

第三步：将流程定义部署到activiti

让activiti去管理部署的业务流程定义。

第四步：启动一个流程实例

启动一个流程实例表示一个新的流程执行开始。

第五步：查询待办任务

查询当前 用户当前待办任务。

第六步：办理任务

选择要办理的任务，完成任务 。

第七步：流程执行完成

流程执行到最后一个结点，流程执行完成。

5、服务架构

1>activiti.cfg.xml

是一个spring配置文件，是activiti的核心配置文件

2》ProcessEngineConfiguration

ProcessEngineConfiguration用于创建 ProcessEngine

Activiti单独运行：StandaloneProcessEngineConfiguration

Activiti和spring整合运行：**SpringProcessEngineConfiguration**

3>ProcessEngine

ProcessEngine 是一个门面接口，引擎核心对象，通过ProcessEngine就可以获取各各service

4》Service

RepostitoryService：用于资源管理，用于：流程定义部署、查询流程定义资源

TaskService：任务管理，用于个人任务、组任务。

RuntimeService：流程当前运行管理，用于：启动一个流程实例是、查询当前运行的流程。。

historyService：流程历史管理。

identityService：用户身份管理，用于：创建用户、组、用户和组的关系 …

# activiti入门体验

实现：

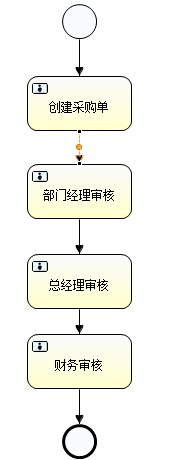
从流程定义、流程定义部署、启动一个流程实例、查询待办任务、办理任务、流程结束 。

修改流程，测试如果不改代码是否能执行通过。

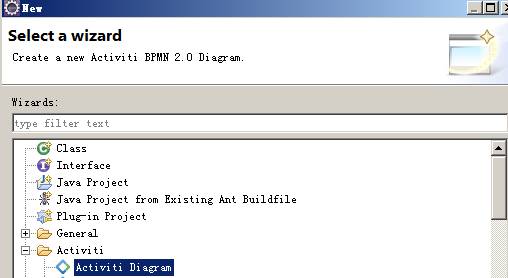
对activiti有一个整体的体验。

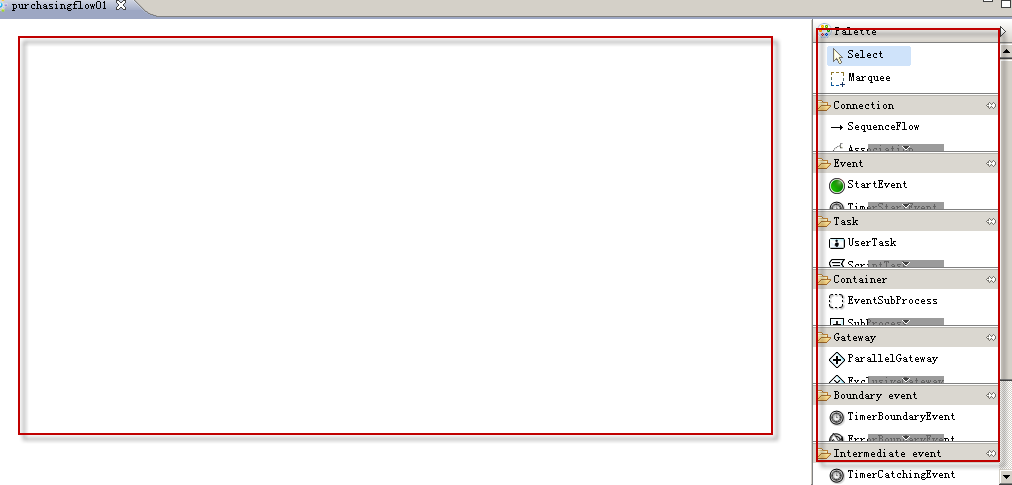
## 流程定义

使用activiti流程设计 器(eclipse的插件)绘制一个业务流程。

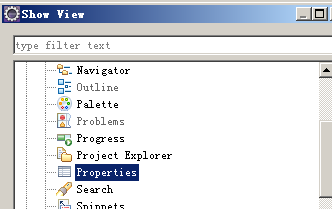


新建一个activiti流程：

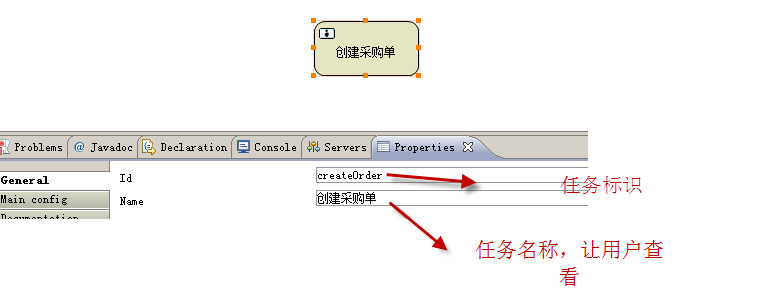




打开properties视图：



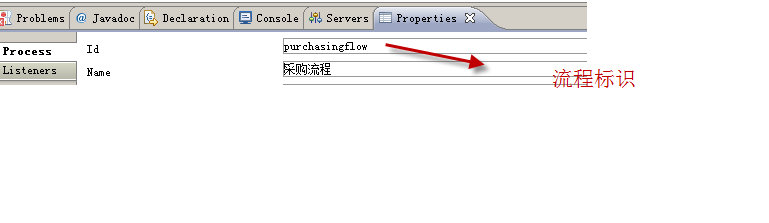
添加一个新的任务，设置任务id（标识），任务名称（用户去查看）





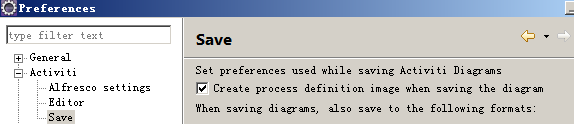
连线

设置流程标识：



流程标识 ：processDefinitionKey（流程定义key），不管采购流程版本有几个，标识只一个，这个标识就表示该流程是一个采购流程。

设置保存bpmn文件时，生成一个图片：





## 流程定义部署

通过activiti的api操作。

使用RepositoryService进行流程定义部署。

## 流程整体执行过程

// 创建processEngine

**private** ProcessEngine processEngine = ProcessEngines

.*getDefaultProcessEngine*();

// 部署流程定义

@Test

**public** **void** deployProcess() {

// 使用RespositoryService

RepositoryService repositoryService = processEngine

.getRepositoryService();

// 部署bpmn文件和png文件

// bpmn文件名

String resourceName\_bpmn = "purchasingflow01.bpmn";

InputStream inputStream\_bpmn = **this**.getClass().getClassLoader()

.getResourceAsStream("diagram/purchasingflow01.bpmn");

String resourceName\_png = "purchasingflow01.png";

InputStream inputStream\_png = **this**.getClass().getClassLoader()

.getResourceAsStream("diagram/purchasingflow01.png");

// 部署对象

Deployment deployment = repositoryService.createDeployment()

.addInputStream(resourceName\_bpmn, inputStream\_bpmn)// 部署bpmn文件

.addInputStream(resourceName\_png, inputStream\_png)// 部署png文件

.deploy();

// 部署对象id

System.*out*.println("部署id：" + deployment.getId());

System.*out*.println("部署时间：" + deployment.getDeploymentTime());

}

// 启动一个流程实例

@Test

**public** **void** startProcessInstance() {

// RuntimeService

RuntimeService runtimeService = processEngine.getRuntimeService();

// 流程定义key（流程定义的标识 ）

String processDefinitionKey = "purchasingflow";

// 启动流程实例

// 根据流程定义的key启动流程，根据key找最高版本的流程定义

ProcessInstance processInstance = runtimeService

.startProcessInstanceByKey(processDefinitionKey);

System.*out*.println("流程实例id：" + processInstance.getId());

System.*out*

.println("流程定义id：" + processInstance.getProcessDefinitionId());

}

// 待办任务

// 查询当前 用户的待办任务

@Test

**public** **void** findPersonalTaskList() {

// 查询任务使用TaskService

TaskService takService = processEngine.getTaskService();

// 流程定义key（流程定义的标识 ）

String processDefinitionKey = "purchasingflow";// 采购流程 标识

// 任务 负责人

String assignee = "zhaoliu";

// 创建查询对象

TaskQuery taskQuery = takService.createTaskQuery();

// 设置查询条件

taskQuery.taskAssignee(assignee);

// 指定 流程定义key，只查询某个流程的任务

taskQuery.processDefinitionKey(processDefinitionKey);

// 获取查询列表

List<Task> list = taskQuery.list();

**for** (Task task : list) {

System.*out*.println("任务 id：" + task.getId());

System.*out*.println("任务负责人：" + task.getAssignee());

System.*out*.println("任务名称：" + task.getName());

}

}

// 办理任务

@Test

**public** **void** completeTask() {

// 查询任务使用TaskService

TaskService takService = processEngine.getTaskService();

//任务id

String taskId="702";

takService.complete(taskId);

System.*out*.println("完成任务："+taskId);

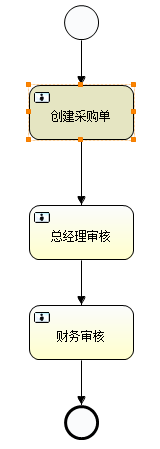
}

## 修改流程

添加任务负责人Assignee信息。

注意：修改完成流程，需要重新部署。

修改流程：



目标：不用修改测试 代码，就可以执行流程。

## 总结

通过测试发现，修改流程后，不需要修改业务代码，因为activiti将流程自动化管理了。

如果业务功能修改了，就需要修改业务系统的代码。

# 流程定义

## 线下进行流程定义

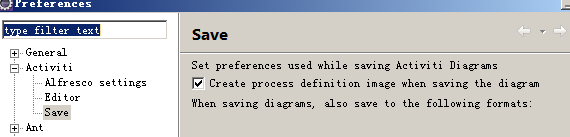
方法：

1、可以使用activiti-Designer设计器进行可视化的操作

使用activiti-Designer设计器进行画图通过是开发人员操作。

通过工具绘制流程图，生成.bpmn的文件。

设置生成 .bpmn文件的同时生成png文件。





上边文件中，.bpmn文件是必须要有的。

Png文件在流程定义时不是必须的，因为 activiti自己也会根据.bpmn文件生成图片，但是activiti生成的图片中有中文乱码。

2、使用activiti提供的b/s方式画图

通过浏览器进行图形绘制。

提供给用户通过 浏览器画图，用户执行流程定义部署。

部署：官方提供的activiti-explorer.war工程，

参考 ：

## 流程定义部署

方法1单个文件部署

// 部署流程定义

@Test

**public** **void** deployProcess() {

// 使用RespositoryService

RepositoryService repositoryService = processEngine

.getRepositoryService();

// 部署bpmn文件和png文件

// bpmn文件名

String resourceName\_bpmn = "purchasingflow01.bpmn";

InputStream inputStream\_bpmn = **this**.getClass().getClassLoader()

.getResourceAsStream("diagram/purchasingflow01.bpmn");

String resourceName\_png = "purchasingflow01.png";

InputStream inputStream\_png = **this**.getClass().getClassLoader()

.getResourceAsStream("diagram/purchasingflow01.png");

// 部署对象

Deployment deployment = repositoryService.createDeployment()

.addInputStream(resourceName\_bpmn, inputStream\_bpmn)// 部署bpmn文件

.addInputStream(resourceName\_png, inputStream\_png)// 部署png文件

.deploy();

// 部署对象id

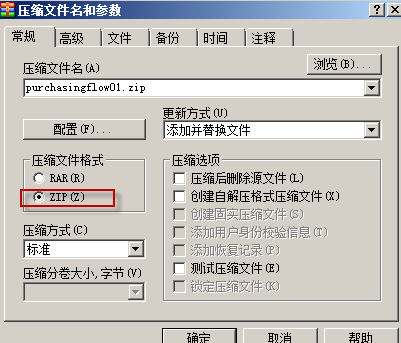
System.*out*.println("部署id：" + deployment.getId());

System.*out*.println("部署时间：" + deployment.getDeploymentTime());

}

方法1压缩包部署

第一步：将要部署的文件打成zip包



第二步：通过程序部署zip到activiti中

@Test

**public** **void** deployProcessByZip() {

// 定义zip输入流

InputStream inputStream = **this**

.getClass()

.getClassLoader()

.getResourceAsStream(

"cn/itcast/activiti/first/diagram/purchasing/purchasingflow01.zip");

ZipInputStream zipInputStream = **new** ZipInputStream(inputStream);

// 获取repositoryService

RepositoryService repositoryService = processEngine

.getRepositoryService();

// 流程部署

Deployment deployment = repositoryService.createDeployment()//

.addZipInputStream(zipInputStream)//

.deploy();

System.*out*.println("流程部署id：" + deployment.getId());

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

}

流程定义部署数据库操作

执行流程定义部署后，向下边表中写数据：

SELECT \* FROM act\_re\_deployment #流程定义部署表

记录每一次流程部署的信息，一次部署只写一条记录。

字段：

Id\_：部署id

SELECT \* FROM act\_re\_procdef #流程定义信息表

记录每一次部署的流程定义信息，一次部署可以部署多个流程定义。

字段：

id\_：流程定义id

结构：流程定义key：流程定义版本：流水号

Name\_：流程定义名称，用户要查询

Key\_：流程定义的标识，一个流程定义一个key，不管版本号，一个采购流程就是一个key

Version\_：流程定义的版本号，相同的key，每次部署版本号自动增加1

Deployment\_Id\_：对应的流程部署表的id

RESOURCE\_NAME\_：流程定义资源中bpmn文件的名称

DGRM\_RESOURCE\_NAME\_：流程定义资源中png文件的名称

act\_re\_deployment和act\_re\_procdef是一对多关系 。

SELECT \* FROM act\_ge\_bytearray #资源表(通用表)

记录了流程定义的资源信息（bpmn和png）



## 流程定义查询

流程定义部署后，通过流程定义查询功能，查询系统中activiti所管理的流程有哪些。

使用RepositoryService查询流程定义。

//流程定义查询

@Test

**public** **void** queryProcessDefinition(){

//得到repositoryService

RepositoryService repositoryService = processEngine.getRepositoryService();

//流程定义的key

String processDefinitionKey = "purchasingflow";//采购流程标识

//创建查询对象

ProcessDefinitionQuery processDefinitionQuery = repositoryService.createProcessDefinitionQuery();

//设置查询条件

//只查询采购流程的定义

processDefinitionQuery.processDefinitionKey(processDefinitionKey);

//查询列表

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

//分页查询

//processDefinitionQuery.listPage(firstResult, maxResults)

//根据流程定义的id查询一条记录

//processDefinitionQuery.processDefinitionId(definitionId).singleResult();

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

System.*out*.println("================================");

System.*out*.println("流程定义id："+processDefinition.getId());

System.*out*.println("流程定义部署id："+processDefinition.getDeploymentId());

System.*out*.println("流程定义的key："+processDefinition.getKey());

System.*out*.println("流程定义的名称："+processDefinition.getName());

System.*out*.println("bpmn资源名称："+processDefinition.getResourceName());

System.*out*.println("png资源名称："+processDefinition.getDiagramResourceName());

}

}

## 流程定义删除

使用RepositoryService删除流程定义。

1. 根据流程部署id删除这一次部署的所有流程定义

repositoryService.deleteDeployment(deploymentId);

约束：如果该流程定义没有启动流程实例可以删除，如果该流程定义已经启动流程实例，不允许删除，如果删除就是抛出异常。



建议：一次部署只部署一个流程，根据流程部署id删除一个流程定义。

2、级联删除

不管该流程定义是否启动流程实例（是否使用），通过级联删除将该流程定义及相关的信息全部删除。

//级联删除

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

一般情况下不使用级联删除，一般情况对流程定义执行暂停操作。

特殊情况下需要删除流程定义及相关的信息，就要使用级联删除，删除的功能给超级管理员使用。

代码如下：



## 流程定义修改

Activiti不提供流程定义修改接口。

通过先删除再部署实现流程定义的修改。

**注意：一般情况下通过暂停流程定义，可以暂停流程定义下边所有运行的流程实例。**

## 流程定义资源查询

需求：

流程定义部署功能是由开发人员登陆业务系统，通过网页执行部署。

对于部署的bpmn文件，开发人员需要通过此功能查询。

对于一般用户，需要通过此功能，查询png文件。

目标：实现将activiti数据库中的bpmn文件和png文件保存到本地。

将来这个功能通过网页实现，实现在线查看bpmn文件和png文件。

代码如下：

// 流程定义资源文件查询

@Test

**public** **void** queryProcessDefinitionResource() **throws** IOException {

// 得到repositoryService

RepositoryService repositoryService = processEngine

.getRepositoryService();

// 根据流程定义id查询部署id

String processDefinitionId = "purchasingflow:1:4";

ProcessDefinition processDefinition = repositoryService

.createProcessDefinitionQuery()

.processDefinitionId(processDefinitionId).singleResult();

//部署id

String deploymentId = processDefinition.getDeploymentId();

// pbmn文件

//bpmn资源文件名称

String resourceName\_bpmn = processDefinition.getResourceName();

//bpmn资源文件输入流

InputStream inputStream\_bpmn = repositoryService.getResourceAsStream(

deploymentId, resourceName\_bpmn);

// png文件

//png文件名称

String resourceName\_png = processDefinition.getDiagramResourceName();

//png资源文件输入流

InputStream inputStream\_png = repositoryService.getResourceAsStream(

deploymentId, resourceName\_png);

//构造输出流

File file\_bpmn = **new** File("d:/"+resourceName\_bpmn);

File file\_png = **new** File("d:/"+resourceName\_png);

FileOutputStream fileOutputStream\_bpmn = **new** FileOutputStream(file\_bpmn);

FileOutputStream fileOutputStream\_png = **new** FileOutputStream(file\_png);

//流复制

**byte**[] b = **new** **byte**[1024];

**int** len=-1;

**while**((len=inputStream\_bpmn.read(b, 0, 1024))!=-1){

fileOutputStream\_bpmn.write(b, 0, len);

}

**while**((len=inputStream\_png.read(b, 0, 1024))!=-1){

fileOutputStream\_png.write(b, 0, len);

}

inputStream\_bpmn.close();

inputStream\_png.close();

fileOutputStream\_bpmn.close();

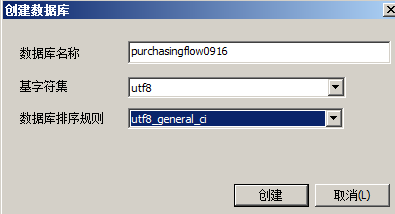
fileOutputStream\_png.close();

}

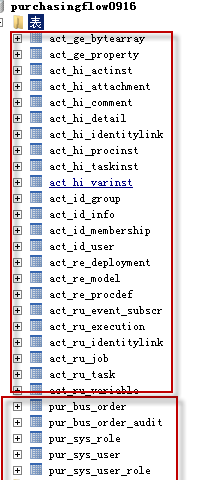
# 在采购系统加入activiti实现流程定义

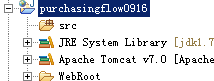
## 工程环境

创建数据库:



导入原始采购系统的sql脚本 。





在未加入activiti的采购系统中添加activiti，此环境作为第三个环境，用于activiti案例开发(activiti在业务系统中应用)

将原来采购系统中的文件拷贝到此工程中。

将activiti和spring整合：

加入activiti和spring的jar包。



编写activiti和spring整合配置文件：applicationContext-activiti.xml

<!-- 工作流引擎配置bean -->

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

<!-- 数据源 -->

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

<!-- 使用spring事务管理器 -->

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

<!-- 数据库策略 -->

<property name=*"databaseSchemaUpdate"* value=*"drop-create"* />

<!-- activiti的定时任务关闭 -->

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

</bean>

<!-- 流程引擎 -->

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

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

</bean>

<!-- 资源服务service -->

<bean id=*"repositoryService"* factory-bean=*"processEngine"*

factory-method=*"getRepositoryService"* />

<!-- 流程运行service -->

<bean id=*"runtimeService"* factory-bean=*"processEngine"*

factory-method=*"getRuntimeService"* />

<!-- 任务管理service -->

<bean id=*"taskService"* factory-bean=*"processEngine"*

factory-method=*"getTaskService"* />

<!-- 历史管理service -->

<bean id=*"historyService"* factory-bean=*"processEngine"*

factory-method=*"getHistoryService"* />

<!-- 用户管理service -->

<bean id=*"identityService"* factory-bean=*"processEngine"*

factory-method=*"getIdentityService"* />

<!-- 引擎管理service -->

<bean id=*"managementService"* factory-bean=*"processEngine"*

factory-method=*"getManagementService"* />

在web.xml中加载上边的配置文件：

<!-- 加载spring容器 -->

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/classes/spring/applicationContext.xml,/WEB-INF/classes/spring/applicationContext-\*.xml</param-value>

</context-param>

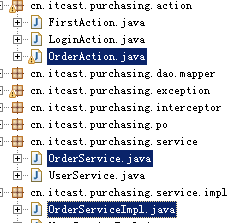
<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

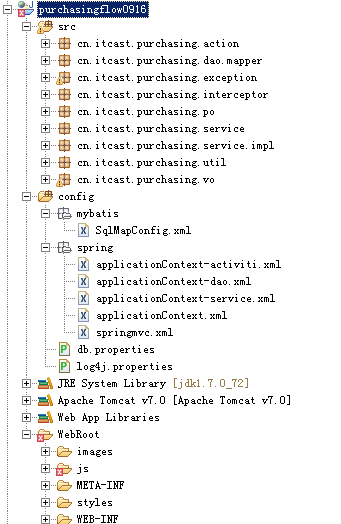
</listener>

删除多余的文件：

删除原来订单的action和service：



工程目录 结构如下：



配置文件：

Spring/applicationContext.xml---spring核心配置文件，配置数据源和事务

Spring/applicationContext-dao.xml ---配置mybatis

Spring/applicationContext-service.xml---配置service接口

Spring/applicationContext-activiti.xml—配置activiti的对象processEngine、service。。。

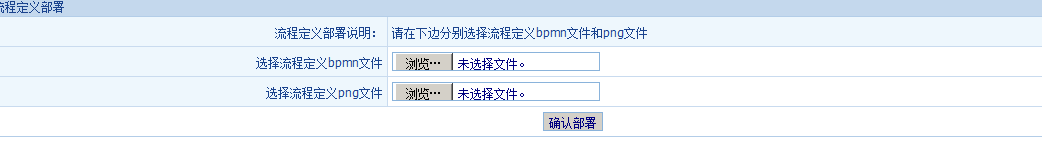
Spring/springmvc.xml---springmvc的配置文件

Mybatis/SqlMapConfig.xml---mybatis的配置文件

## 流程定义部署

需求：

用户登陆系统，上传流程定义文件（pbmn和png）



dao

不用实现，调用activiti的service执行部署。

service

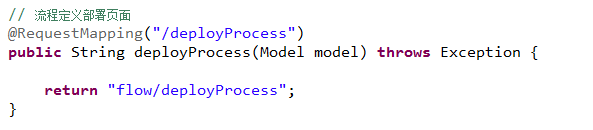
不用实现，调用activiti的service执行部署。

action

编写两个方法：

1. 进入流程定义部署页面

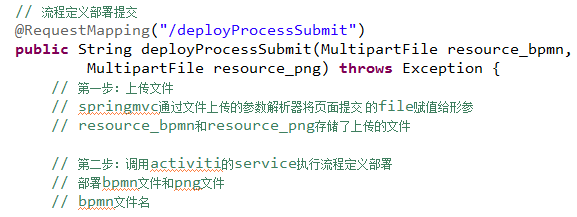
Action 返回一个jsp页面



1. 部署提交

第一步：上传文件

第二步：调用activiti的service执行流程定义部署



在springmvc.xml配置上传文件的参数解析器：

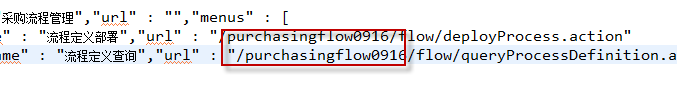


页面



在页面中file的名称和上边deployProcessSubmit方法的形参的名称一致。

修改menu.json文件,配置菜单 ：



## 流程定义查询

需求：

查询系统中采购流程的流程定义信息。



dao

不用实现，调用activiti的service执行查询

service

不用实现，调用activiti的service执行查询

action

写一个方法：

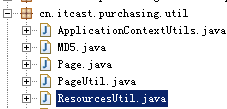
查询流程定义。

在查询时需要指定流程定义的key：processDefinitionKey，不能硬编码，需要采用配置文件配置。





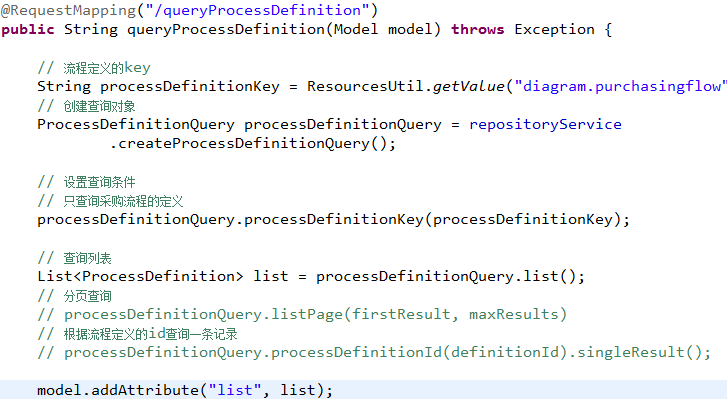
通过下边的文件读取配置文件的内容：



// 流程定义的key

String processDefinitionKey = ResourcesUtil.*getValue*("diagram.purchasingflow", "purchasingProcessDefinitionKey");// 采购流程标识

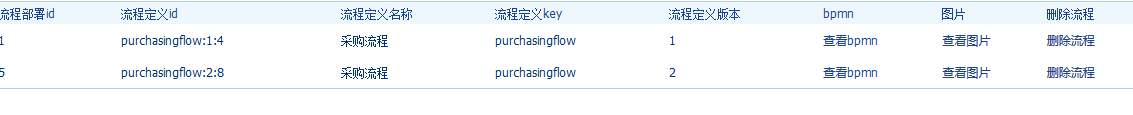
Action代码如下：



页面



测试结果：



## 流程定义删除

需求

在流程定义查询页面，点击“删除流程”，将流程定义信息从activiti中删除。

### dao

不用实现，调用activiti的service执行删除

service

不用实现，调用activiti的service执行删除

action

步骤：

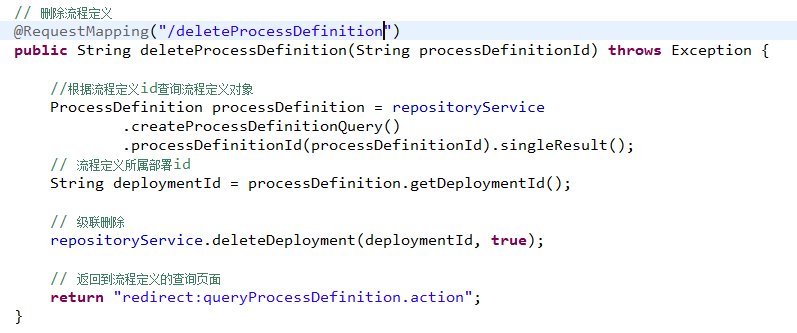
在流程定义查询页面上，

点击：执行删除操作。

流程定义删除方法：

接收页面传来的流程定义id，

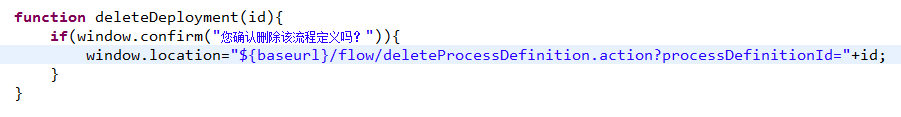
代码如下：



页面

修改流程定义查询页面，添加删除链接：





## 流程定义资源文件查看

需求

在流程定义查询页面，查看bpmn文件内容和png文件内容：



### dao

不用实现，调用activiti的service执行查询

service

不用实现，调用activiti的service执行查询

action

写一个方法：

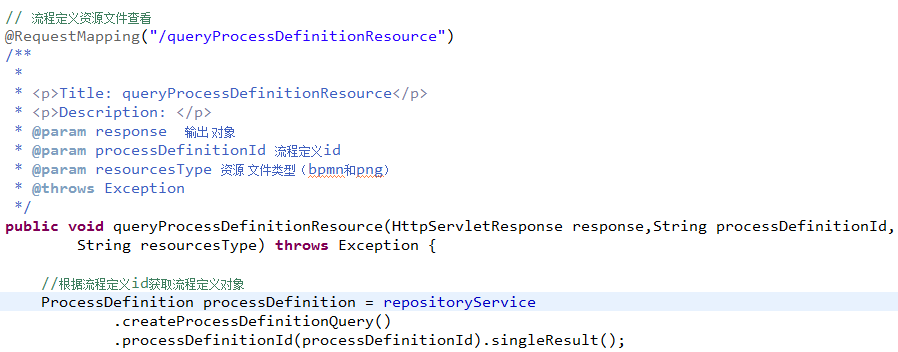
根据传入参数查询bpmn或png文件的内容

思路：通过RespositoryService得到bpmn和png的输入流，将流数据通过response输出到页面.

Action方法需要接收的参数：



代码如下：



页面

在流程定义的查询页面，添加链接：

