activiti第一天课堂笔记

# 课程安排：

4 天activiti课。

第一天：activiti入门，流程部署，流程定义

第二天：流程实例、任务

第三天：流程监控查询、流程变量、流程连线

第四天：网关、组任务

教学思路：

上午理论（activiti的api）

下午案例（将activiti在采购系统中进行应用）

第一天：activiti入门，流程部署，流程定义

什么是工作流？（掌握）

学习没有加入工作流的采购系统(一定掌握)：

框架：Springmvc+mybatis+mysql

总结：没有加入工作流，系统中问题有哪些？

引出工作流的概念，加入工作流后解决上边总结的问题。

什么是activiti？

Activiti概念，

Activiti具体使用步骤（重点，今天有一个整体activiti流程的认识）

部署activiti的环境。

Activiti的入门程序：

包括：流程部署、启动流程实例、查询任务、办理任务、流程完成

开始学习activiti：

Activiti的服务架构（掌握）

Activiti流程部署，相当于流程定义！（重点）

案例：

在采购系统中实现流程部署和流程定义。

将采购系统和activiti进行整合：spring+activity

开发功能：实现流程部署和流程定义

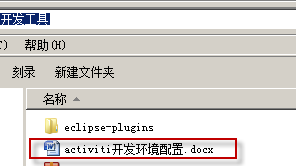
# 采购系统学习

架构：springmvc3.1.4+mybatis3.2.3+mysql5.1

## 系统配置：

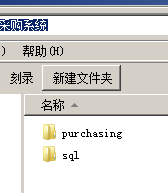
Eclipse：eclipse-jee-juno

参考文件进行配置



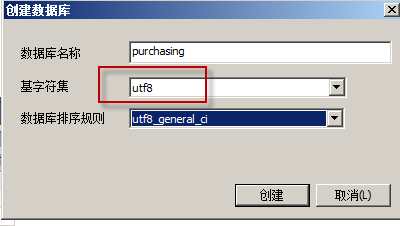
Web容器：tomcat7

采购系统目录：

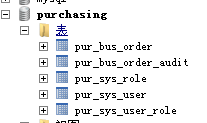


在mysql5.1中导入sql目录 purchasing.sql

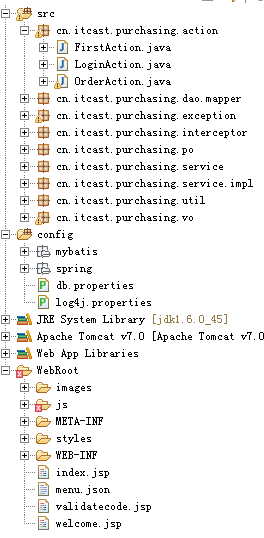
创建数据库：



导入sql目录 purchasing.sql

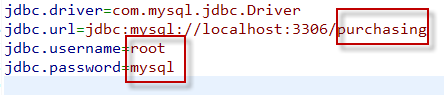


在eclipse-juno导入purchasing目录的工程

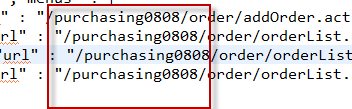


## 配置文件：

* 在config/db.properties配置数据库连接参数：



* 修改WebRoot/menu.json



## 需求

需求：实现员工创建采购单，需要经过三级审核（部门经理审核、总经理审核、财务审核）

完整的业务流程：



为了方便学习简化流程：



分析系统角色：

员工、部门经理、总经理、财务管理人员

## 业务的概念

什么是业务？

用户的需求就是业务。

什么是业务功能？

通过计算机技术实现用户需求，开发功能实现用户需求。

什么是业务系统？

实现用户需求，所开发的软件系统，叫业务系统。

什么是业务系统核心（灵魂）？

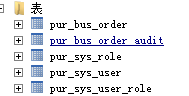
技术不是业务系统核心，业务流程是业务系统核心灵魂。

什么是业务流程？

很多的功能组成一个流程，完成用户需求为目标，叫业务流程。

## 数据表结构

5张表：



结合需求/业务功能学习数据库表。

创建采购单：

创建一个采购单信息表: pur\_bus\_order

此表记录采购单基本信息

Name:采购单名称

Price：采购金额

Content：采购内容

Createtime:创建时间

Status：采购状态： 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

此状态字段用于标记采购流程走到哪一步。

部门经理审核采购单：

创建采购单审核信息表：pur\_bus\_order\_audit

记录了部门经理、总经理及财务的审核信息

order\_id：采购单id（外键）

user\_id：审核人

audit\_info：审核意见

audit\_type：审核类型 审核类型1:部门经理审核,2:总经理审核,3:财务审核

status：审核状态1：通过，0：不通过

createtime：审核时间

用户及角色表：

用户表：pur\_sys\_user

角色表：pur\_sys\_role

用户角色关系表：pur\_sys\_user\_role

## 功能学习

创建采购单

需求：员工创建一个新的采购单

操作流程：

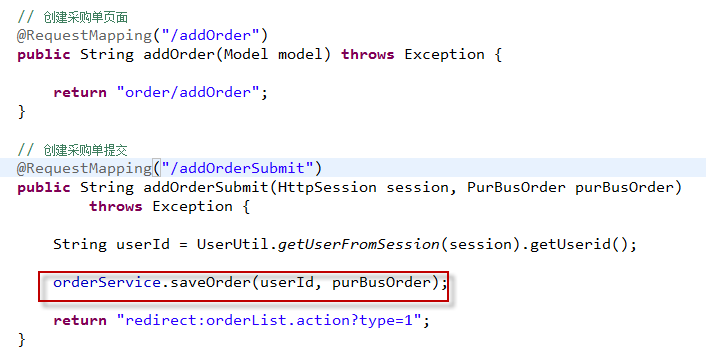
1. 填写采购单的基本信息



向采购单信息表插入一条记录

代码实现：

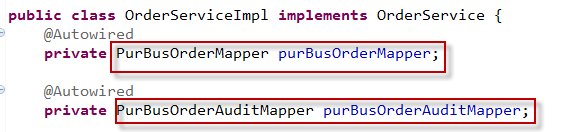
Action：参考OrderAction.java

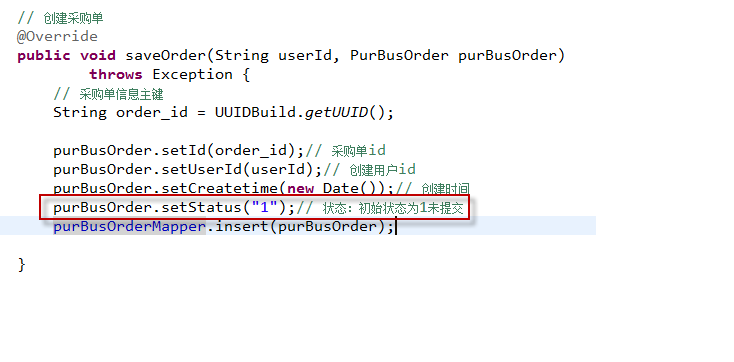


Service：



注入mapper：





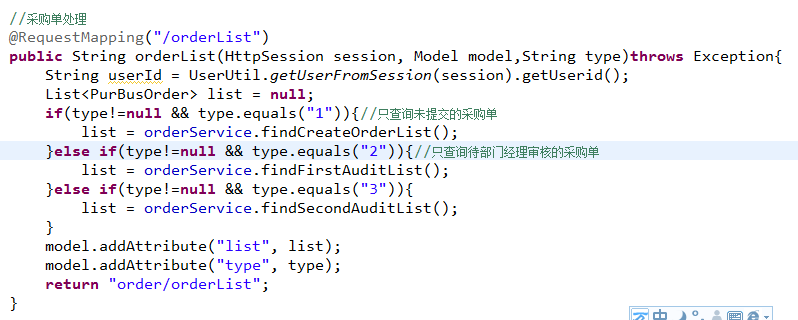
Service在保存采购单，初始状态为1：（未提交）

1. 维护采购单修改

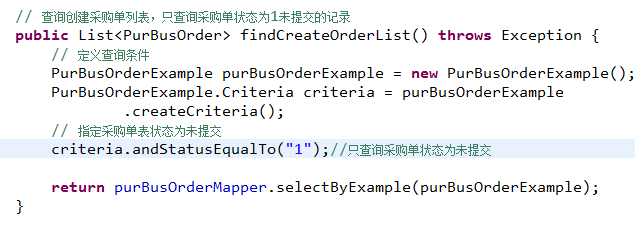


代码实现;

Action:



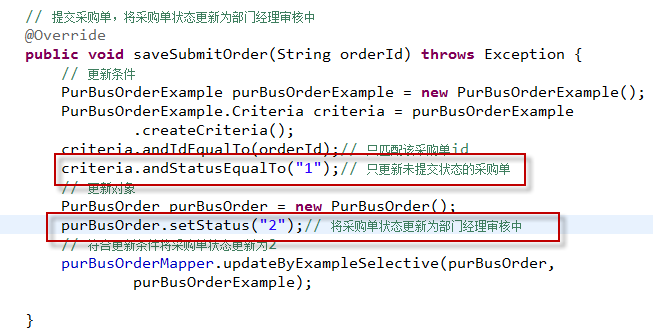
Service：



1. 提交采购单

采购单信息不再修改，提交采购单

Service：



### 部门经理审核采购单

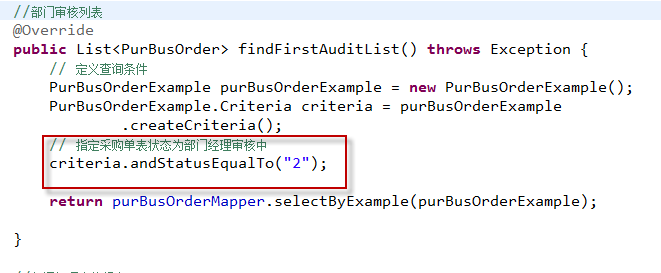
操作流程：

使用lisi登录系统，点击“部门经理审核采购单”

1. 查询待审核的采购单

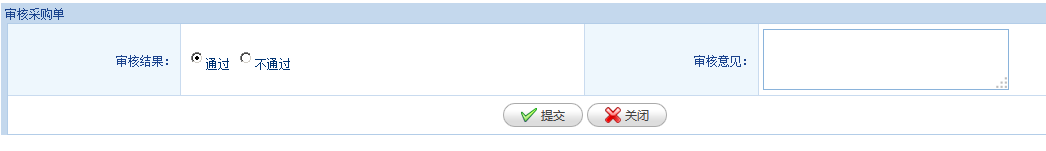


Service：



只查询采购单状态为2（部门经理审核中）

2、部门经理进行审核



审核提交：

@Override

**public** **void** saveFirstAuditSubmit(String userId,String orderId,

PurBusOrderAudit purBusOrderAudit) **throws** Exception {

//保存审核信息

purBusOrderAudit.setId(UUIDBuild.*getUUID*());

purBusOrderAudit.setUserId(userId);

purBusOrderAuditMapper.insert(purBusOrderAudit);

//下边更新采购单表中的状态

//更新条件

PurBusOrderExample purBusOrderExample = **new** PurBusOrderExample();

PurBusOrderExample.Criteria criteria = purBusOrderExample

.createCriteria();

criteria.andIdEqualTo(orderId);// 只匹配该采购单id

criteria.andStatusEqualTo("2");// 只更新部门经理审核中的采购单

//更新对象

PurBusOrder purBusOrder = **new** PurBusOrder();

//获取审核状态，根据审核状态确定采购单状态的值

String status = purBusOrderAudit.getStatus();

**if**(status!=**null** && status.equals("1")){//如果审核通过

purBusOrder.setStatus("4");// 将采购单状态更新总经理审核中

}**else**{//如果审核不通过

purBusOrder.setStatus("3");// 部门经理审核不通过

}

//更新采购单状态

purBusOrderMapper.updateByExampleSelective(purBusOrder,

purBusOrderExample);

}

总经理审核

查询采购单状态为4（总经理审核中）

填写审核信息

提交审核

Service;

如果总经理审核通过，将采购单状态更新6，如果不通过更新为5。

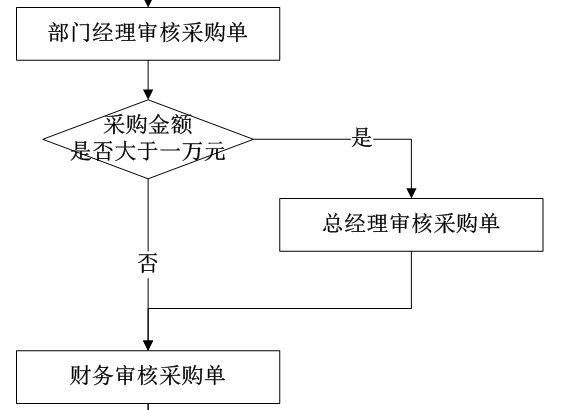
小结：

对采购流程的管理是通过采购状态字段，在该字段中设置采购流程每一个步骤的结点：

状态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

## 需求变更

需求变更1：



修改代码;

修改部门审核提交的service：

判断采购金额，如果大于等于1万元，如果审核通过将采购单状态更新为4总经理审核中

如果采购金额小于1万元，如果审核通过将采购单状态更新为6财务审核中

需求变更2



修改代码：

修改部门经理审核采购单提交的service：

如果部门经理审核通过将采购单状态修改为6财务审核中

修改财务审核采购单提交的service：

如果财务审核通过，将采购单状态修改为4总经理审核中

修改总经理审核采购单

如果审核通过则流程结束，添加一个状态值8总经理审核通过

总经理审核通过将采购单状态更新为8

## 没有加入工作流采购系统的问题

1. 在数据库中通过一个字段控制/管理采购流程，控制采购流程执行顺序，代码中需要对状态值进行**硬编码不利于系统维护。**

使用工作流解析上边的问题，让工作流管理业务流程，业务系统就可以集中精力开发业务功能。

**注意：不使用工作流也能实现业务流程，需要业务系统自己管理业务流程，对于复杂的业务流程建议使用工作流管理业务流程，对于简单的业务流程不建议使用工作流，因为使用工作流会增加开发的工作量，会增加系统开发成本。**

# 什么是工作流？

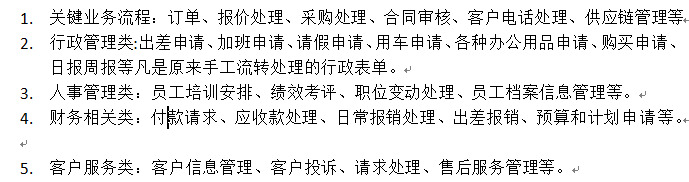
工作流(Workflow)，就是通过计算机对业务流程自动化执行管理。实现多个参与者共同完成一个业务流程，使用业务流程自动化去执行。

什么工作流系统？

具有工作流功能的系统叫做工作流系统，如果具有工作流功能，如果流程变化只需要修改流程，而不需要修改业务功能。

了解工作流系统应用行业及具体应用场景：

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



# 什么是activiti？

Activiti是一个工作流引擎，引擎需要嵌入到业务系统中才可以发挥作用。通过activiti实现工作流。

将业务流程抽取成一个流程定义，流程定义按照bpmn2.0标准进行定义，activiti就会按照讲预先定义好的流程进行流程的执行，从而实现流程的自动化管理，减少系统开发由业务流程变更增加的成本。

Activiti是前身就是jbpm4（也是一个工作流引擎），Activiti中的api及service接口和jbpm4很相似。

## 术语

* BPM：

BPM（Business Process Management），即业务流程管理，是一种以规范化的构造端到端的卓越[业务](http://baike.baidu.com/view/64906.htm" \t "_blank)流程为中心，以持续的提高组织业务[绩效](http://baike.baidu.com/view/122994.htm" \t "_blank)为目的的系统化**方法**，常见商业管理教育如[EMBA](http://baike.baidu.com/view/29918.htm" \t "_blank)、MBA等均将BPM包含在内。

最早在企业提出进行业务流程管理。企业中对业务流程进行管理优化，减少企业成本。

* Bpm软件

实现了业务流程管理的软件，相当于上边说的工作流系统。

* Bpmn2.0(掌握)

Bpmn2.0和上边的bpm是不同的概念！！！

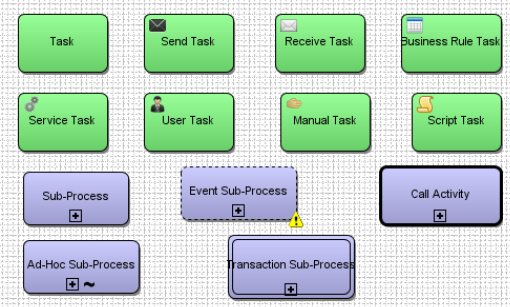
BPMN（Business Process Model And Notation）- 业务流程模型和符号，是由BPMI（Business Process Management Initiative）开发的一套标准的业务流程建模符号，使用BPMN提供的符号可以创建业务流程。

Activiti是基于bpmn2.0标准进行流程定义，activiti在进行流程定义时，使用这些模型和符号绘制流程图。

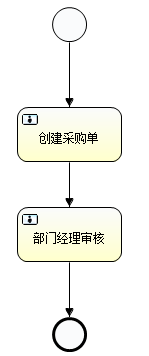
Event 用一个圆圈表示，它是流程中运行过程中发生的事情。



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如何使用？

Activiti是一个工作流引擎，通常情况下activiti需要嵌入到业务系统中运行。

## 第一步：部署activiti

Activiti运行需要在工程中加入jar包：

Activiti本身的jar包

Activiti和spring的整合jar

依赖jar包

创建数据库（存储流程运行的数据）。

## 第二步：使用activiti流程建模工具(activity-designer)定义业务流程(.bpmn文件)

Activiti官方提供了一个和eclipse集成插件（**activity-designer**），用来画流程图，使用bpmn2.0标准(定义模型和符号)，画流程图过程就是流程定义的过程。

在线下画流程图。

## 第三步：向activiti部署业务流程定义（.bpmn文件）

将.bpmn文件（流程定义文件），让activiti识别，让activiti按照流程定义文件的内容去管理业务流程。

将.bpmn文件部署到activiti的数据库中，流程定义完成了。

## 第四步：启动一个流程实例

流程定义完成后，如果有参者来发起一个流程，activiti可以按照.bpmn文件内容去管理流程。

比如：张三创建一个采购单表示发起了一个采购流程。李四创建一个采购单，李四也发起一个采购流程。

流程定义和流程实例关系：静态和动态，类似java类和java对象。

## 第五步：查询待办任务

因为现在系统的业务流程已经交给activiti管理，通过activiti就可以查询当前流程执行到哪了，**当前用户需要办理什么任务了**，这些activiti帮我们管理了，而不像上边需要我们在sql语句中的where条件中指定当前查询的状态值是多少。

## 第六步：办理任务

当前流程运行到哪了这是由activiti自动管理的，当前参与者就可以完成/办理属于他的**当前任务**。

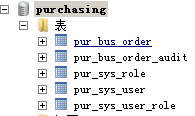
## 第七步：流程结束

当任务办理完成没有下一个任务/结点了，这个流程实例就完成了。

# 关于环境

第一个环境：没有加入工作流采购系统

作用：主要学习业务流程和springmvc+mybatis



第二个环境：activiti测试环境

作用：用于测试activiti的api，提供各种service接口。

需要创建一个数据库：

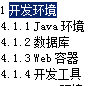
仅仅有activiti的数据表

第三个环境：activiti应用环境，加入工作流的采购系统

需要创建一个数据库：

包括activiti的数据表和业务表（采购系统的表）

# activiti测试环境

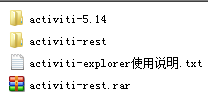


Activiti自身环境包括：activiti的jar包和数据库

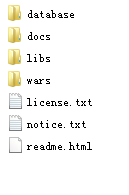
## activiti的jar包

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

下载activiti5.14版本，



解压：



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。

参考

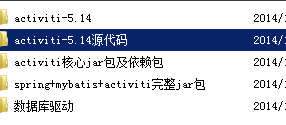


中文的说明文档（activiti各各api和service）

1. **Libs**

Activiti所需要的jar 包

源码：



核心包：

activiti-engine-5.14.jar

和spring整合包：

activiti-spring-5.14.jar

1. **Wars**

官方自带的示例工程。

## Activiti Designer流程设计器

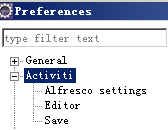
用于进行流程定义，按照bpmn2.0标准画流程图。

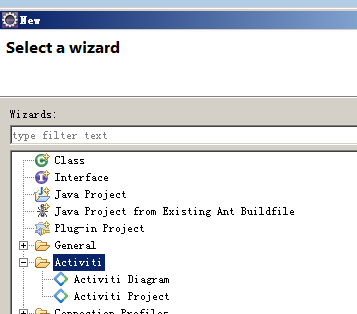
在eclipse-juno版本中安装activiti designer设计器：

参数文档开发工具目录下的“activiti开发环境配置.docx”中“eclipse插件安装”，其中包括了Activiti插件。

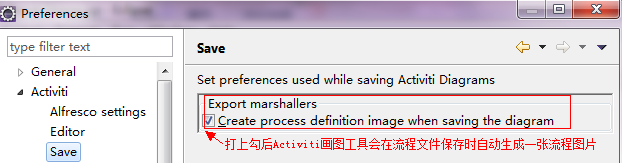
流程图画好后，生成一个.bpmn的文件（内容遵循bpmn2.0标准），通过插件可以同时生成一个png图片（流程图）。

Activiti的设计器插件安装后：

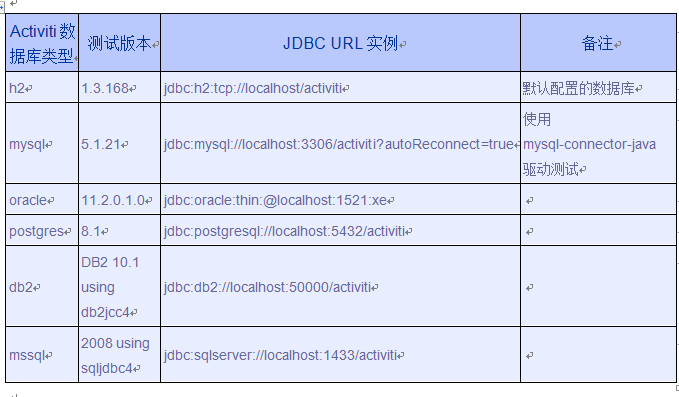




打开菜单Windows->Preferences->Activiti->Save下流程图片的生成方式:



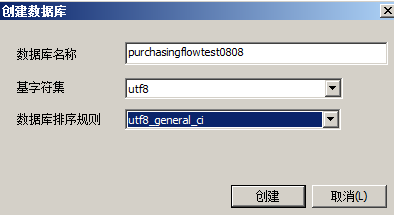
## Activiti的数据库



本教程使用mysql数据库

使用mysql5.1版本。

创建一个用于activiti测试的数据库



创建数据库表

方式1：从下载activiti5.14目录/database中执行sql脚本

方法2：通过java程序

## 通过java程序创建数据库

创建数据库思路：

Activiti运行需要核心门面接口ProcessEngine，在创建processEngine时自动检查数据库环境，可以通过数据库操作策略操作数据库。

第一步：创建一个java 工程

这是第二个环境，测试activiti。

第二步：加入jar



上边是activiti单独运行的jar包

下边是activiti和spring整合的jar，比上边目录多了spring的jar包：tx、springmvc。。。

使用activiti和spring整合的jar

第三步：log4j.properties

第四步：activiti.cfg.xml

Activiti的全局配置文件，是spring的配置文件。

如果和spring没有整合时，配置*数据源*和*processEngineConfiguration*

*数据源：使用dbcp第三方连接池*

*processEngineConfiguration：用于创建processEngine*

Activiti通过spring容器在创建*processEngineConfiguration。*



数据库操作策略：

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。**

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。

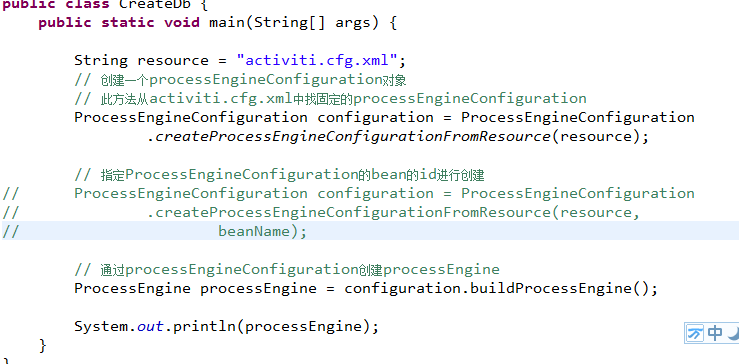
drop-create：先删除表再创建表。

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。

第五步：编写java程序

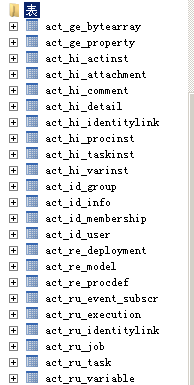
创建一个processEngineConfiguration对象

通过processEngineConfiguration创建procesEngine对象，同时自动创建数据库表。

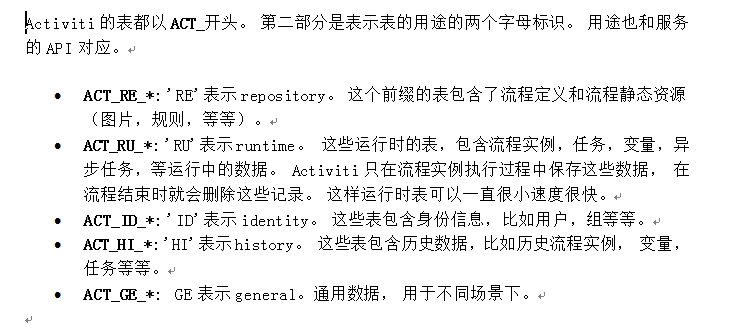


数据库创建完成，

共23张表：

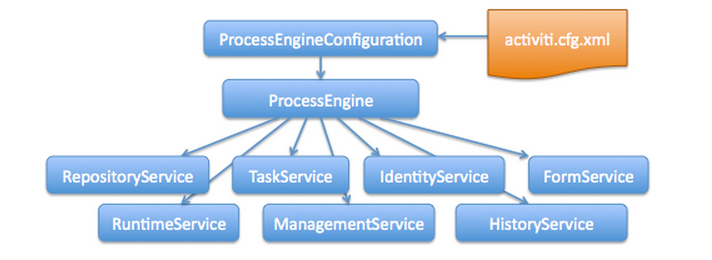


表的命名规则



# Activiti的服务架构

## 架构图



根据activiti.cfg.xml创建processEngineConfiguration

通过processEngineConfiguration创建ProcessEngine，

通过ProcessEngine调用getXXXXService文件得到各各Service，

## Activiti.cfg.xml

Activiti单独运行：

activiti.cfg.xml是activiti的核心配置文件，名称不固定。

在此文件中配置processEngineConfiguration，通过processEngineConfiguration创建processEngine

Activiti单独运行通过StandaloneProcessEngineConfiguration创建processEngine

<!-- processEngineConfiguration： 用于创建processEngine -->

<bean id=*"processEngineConfiguration"*

class=*"org.activiti.engine.impl.cfg.StandaloneProcessEngineConfiguration"*>

<!-- 数据源 -->

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

<!-- activiti数据库表处理策略 -->

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

</bean>

Activiti与spring整合

创建applicationContext-activiti.xml，在此文件配置activiti组件 名称不固定。

使用**SpringProcessEngineConfiguration让activiti与spring整合。**

配置：*processEngineConfiguration*

<!-- 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>

在此文件中配置processEngine

<!-- 流程引擎 -->

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

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

</bean>

配置service;

<!-- 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"* />

## ProcessEngine

相当于一个门面接口，通过ProcessEngineConfiguration创建processEngine，通过ProcessEngine创建各各service接口。

门面是一个设计模式，定义一个简单bean，定义属性（门面包括组件java对象），定义get/set方法，调用门面get方法获取组件。

* 创建方式1;

ProcessEngine processEngine = ProcessEngineConfiguration

.createProcessEngineConfigurationFromResource(resource).buildProcessEngine();

* 创建方式2：



## service

|  |  |
| --- | --- |
| RepositoryService | activiti的资源管理类，部署流程定义、查询流程定义、删除流程定义、查看流程定义文件（.bpmn，.png） |
| RuntimeService | activiti的流程运行管理类，启动一个流程实例，查询当前运行流程 |
| TaskService | activiti的任务管理类,查询当前待办任务，办理当前的任务 |
| HistoryService | activiti的历史管理类，查询历史流程信息、查询历史任务信息 |
| IdentityService | activiti的用户身份管理类，用户授权（用户信息、组信息、用户与组关系信息） |
| FormService | activiti的表单管理类，在流程图中配置表单数据 |
| ManagerService | activiti的引擎管理类，监控activiti运行状态，通常用超级管理员使用 |

注：红色标注为常用service。

通过调用ProcessEngine.getXXXService()方法：

RepositoryService repositoryService = processEngine.getRepositoryService ()

RuntimeService runtimeService = processEngine.getRuntimeService();

…..

# activiti入门体验

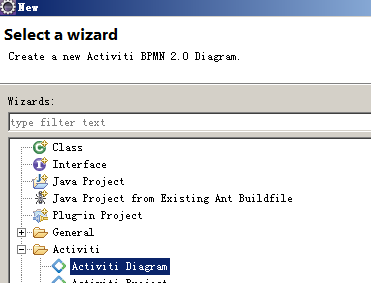
## 需求：

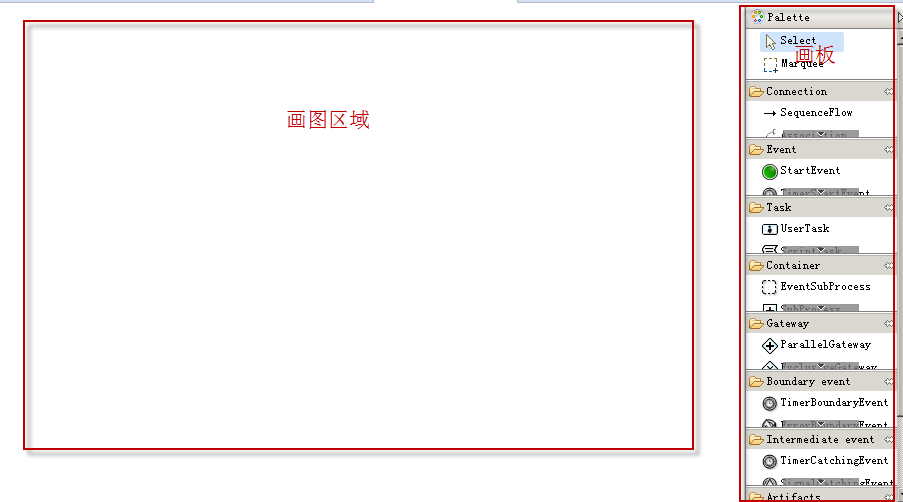
员工创建采购单，经过部门经理、总经理、财务三级审核。业务流程图如下：



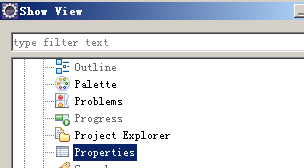
## 流程定义：

使用流程设计器，定义流程，生成.bpmn文件、.png文件。

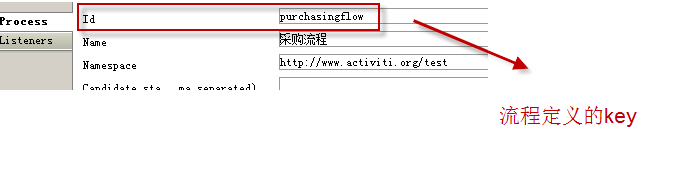




打开properties视图，

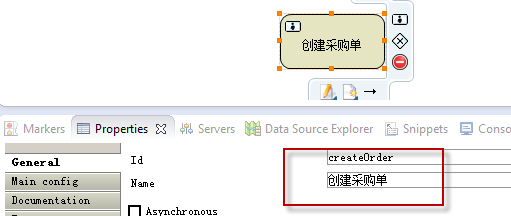


* 流程定义的key (点击空白处，出现这个界面设置)

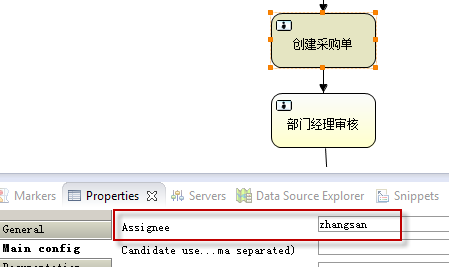


流程定义key用于标识符一个业务流程，针对相同的业务流程key是相同的，比如两个采购流程，由于前后修改采购流程不一样，key定义为一样，因为它就是一个采购流程。

* 创建采购单结点：



指定任务的办理人：



在assignee中设置任务办理人。对每个userTask结点都设置assignee

自动生成流程定义文件：.bpmn,.png

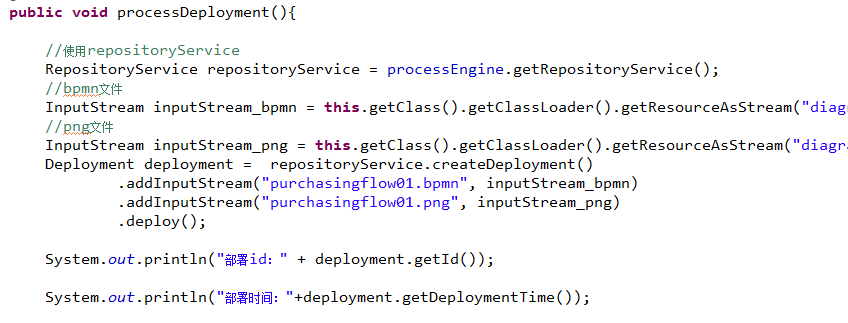


## 部署流程定义

流程修改后，一定要重新部署流程！！！！！

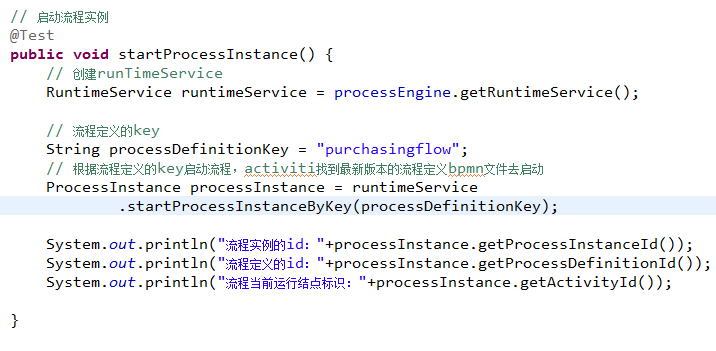
将流程定义文件部署到activiti的数据库中。

使用RepositoryService进行流程定义部署。



## 启动流程实例

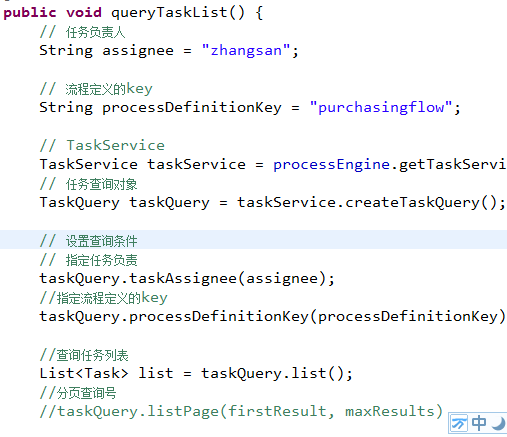
由参与者发起一个流程，使用runTimeService



## 查询待办任务

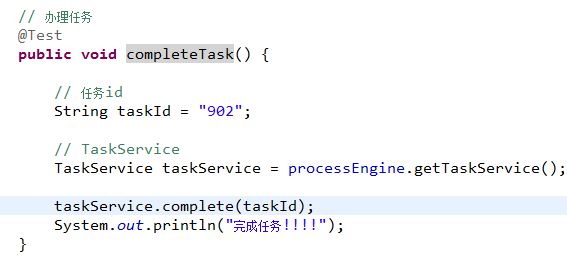
用户查询当前自己需要办理的任务。

使用TaskService查询当前待办任务。



## 办理任务

使用TaskService完成任务



# 流程定义

## 操作流程：

1. 用户在线下定义bpmn

生成bpmn文件，png不是必须要生成的，在web页面上提供一个功能查询流程定义图这个功能。activiti本身也能生成png，但是生成图片会中文乱码，图中结点坐标位置错乱，所以要提前通过eclipse插件生成png文件。

需要两个文件：.bpmn、.png

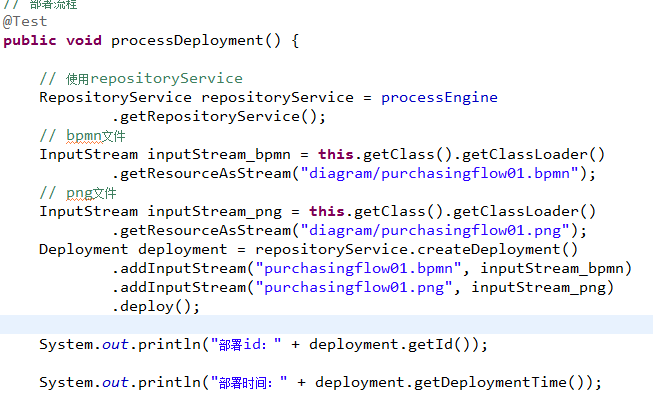
1. 通过repositoryService对流程定义进行部署

## 部署方法

单个文件部署

将.bpmn、.png两个文件部署到activiti

* 代码：



* 数据库操作

向流程部署表写一条记录

SELECT \* FROM act\_re\_deployment #流程部署表

向流程定义表写记录，如果一次部署多个流程需要写多条记录。

SELECT \* FROM act\_re\_procdef #流程定义表

Id\_ 流程定义id：流程定义key：版本号：流水号

Version\_ 流程定义版本：流程定义key相同时，版本号会自动加1

Key\_ 流程定义key：用于标识一个业务流程，不管版本是否改变，相同的业务流程使用相同的key

deployment\_id\_ 流程部署id：

RESOURCE\_NAME\_： bpmn文件名称

DGRM\_RESOURCE\_NAME\_： png文件名称

将bpmn和png写到资源表：

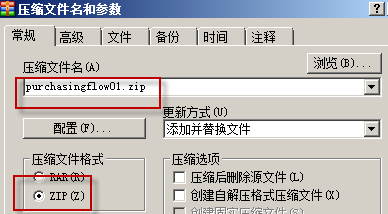
SELECT \* FROM act\_ge\_bytearray #资源表

注意：流程部署表和流程定义表是一对多关系。建议一次部署只部署一个流程，方便管理。

建议使用单个文件部署部署。

压缩包部署

将.bpmn、.png两个文件压缩成zip文件，通过repositoryService进行部署



通过代码部署zip文件;

@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());

}

## 流程定义查询

通过此功能查询本系统通过actviti管理的流程有哪些。

使用repositoryService查询流程定义，可以根据流程定义的key查询某个业务流程在activiti中的流程定义。

// 流程定义的查询

@Test

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

//流程定义的key

String processDefinitionKey = "purchasingflow";

// 使用repositoryService

RepositoryService repositoryService = processEngine

.getRepositoryService();

//流程定义查询对象

ProcessDefinitionQuery processDefinitionQuery = repositoryService

.createProcessDefinitionQuery();

//设置查询条件

processDefinitionQuery.processDefinitionKey(processDefinitionKey);

//得出查询列表

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

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

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

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

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

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

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

System.*out*.println("bpmn文件名："+processDefinition.getResourceName());

System.*out*.println("png文件名："+processDefinition.getDiagramResourceName());

}

}

## 流程定义的资源文件查询

需求：查看activiti中流程定义的资源文件（.bpmn和.png），程序员需要查看bpmn（xml格式），终端用户查询图片。

使用repositoryService。

// 流程定义资源文件查看

@Test

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

// 使用repositoryService

RepositoryService repositoryService = processEngine

.getRepositoryService();

// 流程定义id

String processDefinitionId = "purchasingflow:3:1104";

// 查询一个流程定义

ProcessDefinition processDefinition = repositoryService

.createProcessDefinitionQuery()

.processDefinitionId(processDefinitionId).singleResult();

String bpmn\_name = processDefinition.getResourceName();

String png\_name = processDefinition.getDiagramResourceName();

// 部署id，来源于流程部署表

String deploymentId = processDefinition.getDeploymentId();

// 资源文件名称

String resourceName\_bpmn = bpmn\_name;

String resourceName\_png = png\_name;

//bmpn的输入流

InputStream inputStream\_bpmn = repositoryService.getResourceAsStream(deploymentId, resourceName\_bpmn);

//png的输入流

InputStream inputStream\_png = repositoryService.getResourceAsStream(deploymentId, resourceName\_png);

//将输入流通过文件输出流写到磁盘

FileOutputStream fileOutputStream\_bpmn = **new** FileOutputStream(**new** File("D:/purchasingflow.bpmn"));

FileOutputStream fileOutputStream\_png = **new** FileOutputStream(**new** File("D:/purchasingflow.png"));

**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);

}

//释放资源

inputStream\_bpmn.close();

inputStream\_png.close();

fileOutputStream\_bpmn.close();

fileOutputStream\_png.close();

}

## 流程定义删除

使用repositoryService删除



流程一但启动是不删除的，给超级管理开放级联的功能。可以暂停/激活流程的执行。

# Activiti应用（部署流程定义）

## 搭建activit应用环境（采购系统）

在原有purchasing系统基础上，加入activiti。

第一步：创建一个javaweb工程 purchasingflow0808

将原有purchasing内容拷贝到新工程。

第二步：加入jar包

使用下边的jar替换原始工程的jar包



第三步：创建activiti和spring整合配置文件

applicationContext-activiti.xml文件：

配置

*processEngineConfiguration：*

将业务系统中事务管理器注入到processEngineConfiguration，使用统一事务管理器管理activiti和业务系统事务

<!-- 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 Spring对业务系统的service和activiti的service使用一个事务管理器 -->

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

<!-- 数据库策略 -->

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

<!-- activiti的定时任务关闭 -->

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

</bean>

配置processEngine、XXXXService

<!-- 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 -->

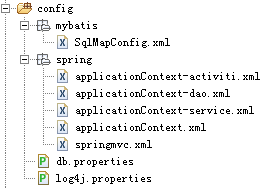
<bean id=*"managementService"* factory-bean=*"processEngine"*

factory-method=*"getManagementService"* />

在web.xml中加载此配置文件，设置/WEB-INF/classes/spring/applicationContext-\*.xml

所有配置文件：

红色标注是需要添加的文件。



SqlMapConfig.xml：mybatis的配置文件(原工程)

Springmvc.xml：springmvc框架的配置文件(原工程)

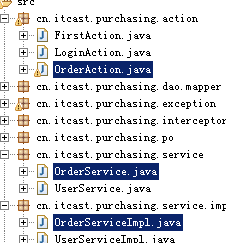
applicationContext.xml：spring容器配置文件，这里配置了数据源和事务管理(原工程)

applicationContext-dao.xml：配置了mybatis的自动扫描器(原工程)

applicationContext-service.xml： 配置了服务接口(原工程)

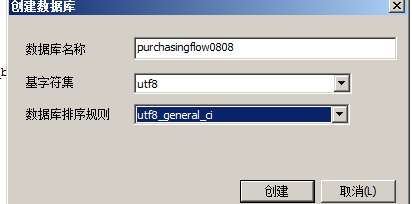
applicationContext-activiti.xml：是spring与actiivti整合的配置文件（参考上面章节的spring-activiti.cfg.xml）

第四步：删除旧文件

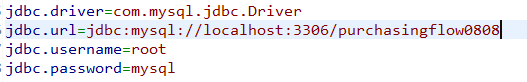


创建新的数据库

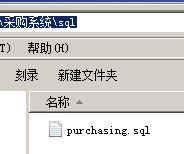
数据库用于activiti和业务系统整合的正式环境。

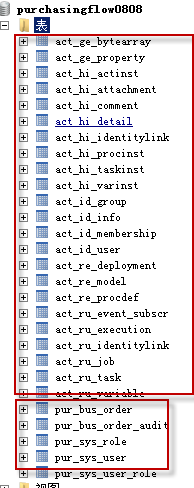


修改db.properties文件：



导入原采购系统的表：



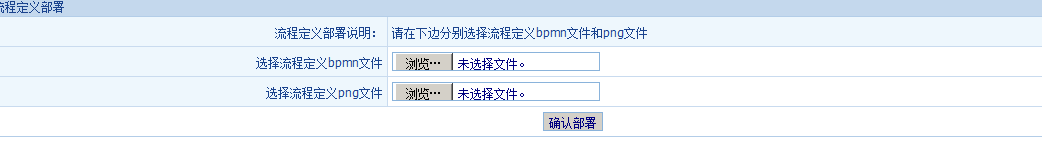


## 功能开发

在采购系统中增加流程定义功能，功能如下：

流程定义部署

效果：



思路：

通过springmvc上传流程定义的资源

调用activiti的service执行流程定义部署

本功能不是业务功能，为了activiti整合而开发。

springmvc上传文件

配置上传文件的参数解析。

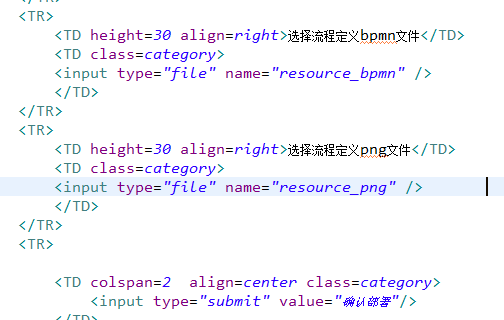
在springmvc.xml配置：



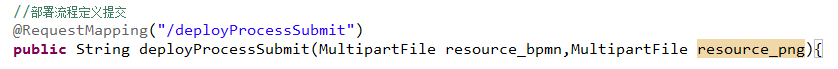
代码：

上传页面：





Action方法：



// 部署流程定义提交

@RequestMapping("/deployProcessSubmit")

**public** String deployProcessSubmit(MultipartFile resource\_bpmn,

MultipartFile resource\_png) **throws** IOException {

// 通过activiti的repositoryService进行流程定义部署

// bpmn文件

InputStream inputStream\_bpmn = resource\_bpmn.getInputStream();

String bpmn\_name = resource\_bpmn.getOriginalFilename();

// png文件

InputStream inputStream\_png = resource\_png.getInputStream();

String png\_name = resource\_png.getOriginalFilename();

Deployment deployment = repositoryService.createDeployment()

.addInputStream(bpmn\_name, inputStream\_bpmn)

.addInputStream(png\_name, inputStream\_png)

.deploy();

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

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

//转到流程定义查询页面

**return** "redirect:queryProcessDefinition.action";

}

修改menu.json

{

"menus" : [{"icon" : "icon-sys","menuid" : "1","menuname" : "流程管理","url" : "","menus" : [

{"icon" : "icon-log","menuid" : "1\_1","menuname" : "部署流程","url" : "/purchasingflow0808/flow/deployProcess.action"

}]

}

]

}

流程定义查询

需求：

查询流程定义信息：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 流程部署id | 流程定义id | 流程定义名称 | 流程定义key | 流程定义版本 |

点击“查看bpmn”、“查看流程图片”链接，查询流程定义资源文件内容。

点击“删除流程定义”，删除已部署的流程定义。

开发

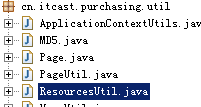
* 查询流程定义

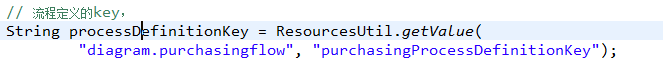
使用activiti的repositoryService

流程定义的key事先配置在配置文件中，在程序中从配置文件得到。

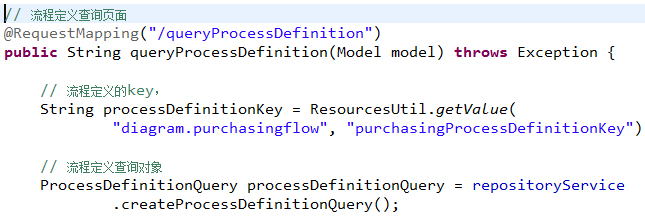


通过工具类读取配置文件：





参考action方法;



页面：



* 根据流程定义的id删除流程定义

使用activiti的repositoryService

// 根据流程定义的id删除流程定义

@RequestMapping("/deleteDeployment")

**public** String deleteDeployment(String deploymentId) {

// 通过级联删除将流程定义及相关的所有内容全部删除，不限制流程是否启动

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

**return** "flow/queryProcessDefinition";

}

* 根据流程定义的id查看资源文件

使用activiti的repositoryService

分析：

页面将processDefinitionId流程定义id传到action

还需要传一个资源文件类型（bpmn,png）

通过repositoryService得到资源文件的输入流，将输入流的数据通过response得到输出流输出到客户端。

// 查询资源文件

@RequestMapping("/queryResource")

**public** **void** queryResource(HttpServletResponse response,String processDefinitionId, String resourceType) **throws** IOException {

// 查询一个流程定义

ProcessDefinition processDefinition = repositoryService

.createProcessDefinitionQuery()

.processDefinitionId(processDefinitionId).singleResult();

// 部署id，来源于流程部署表

String deploymentId = processDefinition.getDeploymentId();

// 资源名称

String resourceName = **null**;

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

resourceName = processDefinition.getResourceName();

} **else** {

resourceName = processDefinition.getDiagramResourceName();

}

// 资源输入流

InputStream inputStream = inputStream = repositoryService

.getResourceAsStream(deploymentId, resourceName);

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

**int** len = -1;

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

response.getOutputStream().write(b, 0, len);

}

}