**JBPM4.4工作流程整合说明文档**

编写人：csx

# 目录

目录

[1. 目录 2](#_Toc303680350)

[2. JBPM工作流引擎介绍 4](#_Toc303680351)

[3. JBPM开发环境准备 4](#_Toc303680352)

[1.1 JBPM4环境要求 6](#_Toc303680355)

[1.2 安装开发环境 6](#_Toc303680356)

[1.3 设计第一个流程 8](#_Toc303680357)

[３. JBPM整合步聚 11](#_Toc303680358)

[3.1 执行创建的脚本 12](#_Toc303680359)

[3.2 加入依赖包 12](#_Toc303680360)

[3.3 整合Jbpm至应用的事务管理 13](#_Toc303680361)

[3.3.1 引入Spring的配置管理(app-jbpm.xml) 14](#_Toc303680362)

[3.3.2 引入jbpm.cfg.xml配置文件 14](#_Toc303680363)

[3.3.3 配置jbpm.cfg.xml 15](#_Toc303680364)

[3.3.4 配置jbpm.tx.spring.cfg.xml 16](#_Toc303680365)

[４. JBPM工作流扩展 18](#_Toc303680366)

[4.1 JBPM常用术语 18](#_Toc303680367)

[4.2 资源库和运行时表结构 19](#_Toc303680368)

[4.3 历史数据库表结构 20](#_Toc303680369)

[4.4 身份认证表结构 20](#_Toc303680370)

[4.5 扩展表E-R图 21](#_Toc303680371)

[4.6 扩展表说明 21](#_Toc303680372)

[4.7 流程发布 24](#_Toc303680373)

[4.8 流程任务处理人员设置 27](#_Toc303680374)

[4.9 在线流程设计器 29](#_Toc303680375)

[4.10 ESTBPM工作流扩展表单 30](#_Toc303680376)

[4.10.1 在线表单 30](#_Toc303680377)

[4.10.2 EXT表单 31](#_Toc303680378)

[4.10.3 URL模板表单 34](#_Toc303680379)

[５. 实现过程部分说明 35](#_Toc303680380)

[5.1 流程启动调用序列图 36](#_Toc303680381)

[5.2 流程执行任务实现序列图 37](#_Toc303680382)

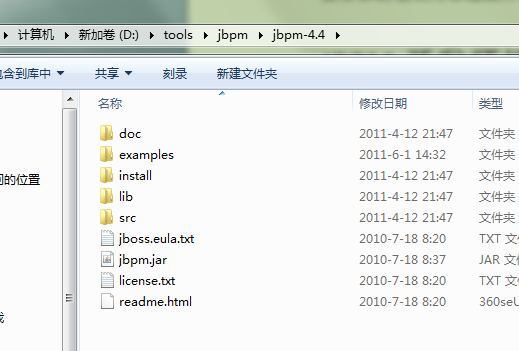
[5.3 部分关键类实现说明 38](#_Toc303680383)

# JBPM工作流引擎介绍

JBPM是JBoss开源的一个比较知名的项目，是一个灵活的、可扩展的工作流引擎，它拥有直观的流程定义语言，可用任务、异步等待状态、定时器、自动化动作等来表示业务流程图，并且把这些业务操作绑定在一起，使之完成流程中的各项操作。由于采用Java语言开发，并且采用了开源的Hibernate框架，所以可以做到跨平台部署，跨数据库。并获得广大软件开发商的认可，并且越来越多的业务系统也采用该框架作为首先的工作流引擎。

# JBPM开发环境准备

首先从JBPM代码托管地址上（http://sourceforge.net/projects/jbpm/）下载４.4ＧＡ版本，下载完后，解压到本地目录（installDir），如下图所示：



学习资料可以查看其下的doc目录下的devguide与userguide.

以下为该主要目录的说明：

* + - doc: 用户指导，API文档，开发手册
    - examples: 用户指导中的流程例子，非常不错的学习例子
    - install/jdbc为jbpm的数据库连接字符串
    - install/src/cfg 为jbpm的hibernate的各种详细配置
    - install/src/db 为jbpm的数据库脚本
    - install/src/gpd 为jbpm的eclipse控件
    - install/src/signavio 为jbpm的在线流程设计器
    - lib: 第三方依赖包及一些Jbpm的发布包
    - src: Jbpm源代码
    - jbpm.jar:　Jbpm的主要二进制包



## 1.1 JBPM4环境要求

JDK 1.5 以上版本，如需要执行Ant脚本，需要Ant 1.7以上的版本。

## 1.2 安装开发环境

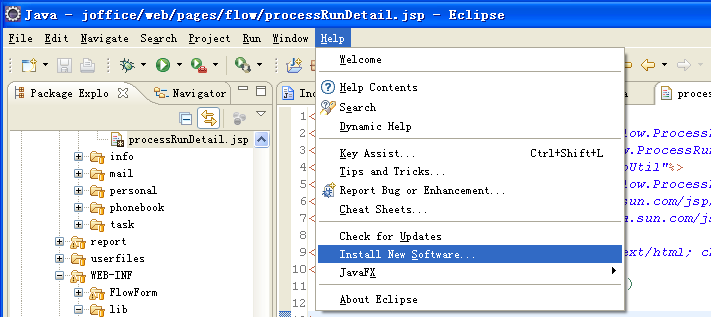
1. 安装eclipse

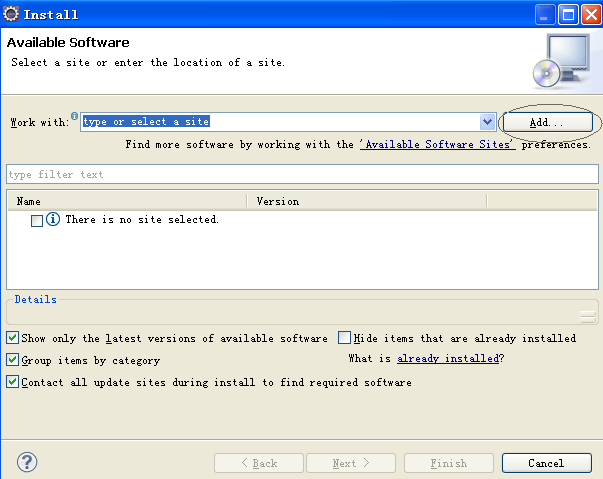
运行配置基于eclipse 3.5开发环境，开发工具可在以下地址下载：

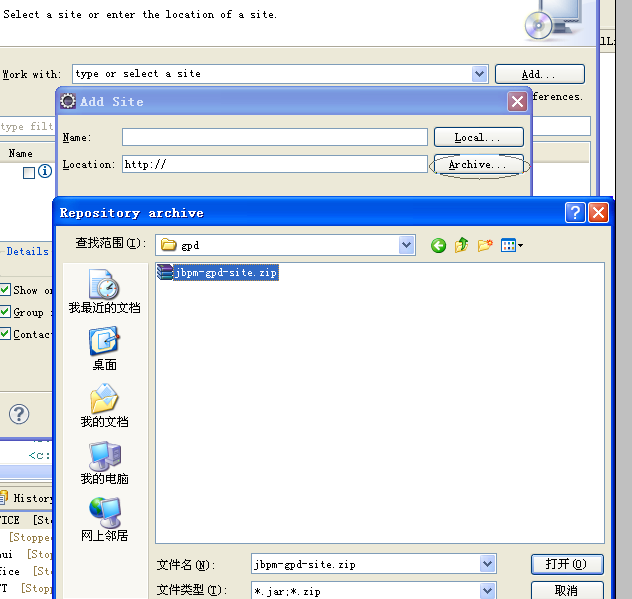
<http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/galileo/SR1/eclipse-jee-galileo-SR1-win32.zip>

1. 安装Jbpm设计器插件

* 添加update site 至Eclipse中
* Help🡪Install New Software
* 点击“Add”
* 在弹出对话框，选择“Archive…”
* 导航到Jbpm的解压目录选择install/src/gpd/jbpm-gpd-site.zip，然后点击“打开”。
* 在Add Site 对话框选择“ＯＫ”，这将进入安装
* 选择jPDL 4 GPD的Update Site，点击“Finish”
* 选择Approval the license
* 安装完成后，重启eclipse



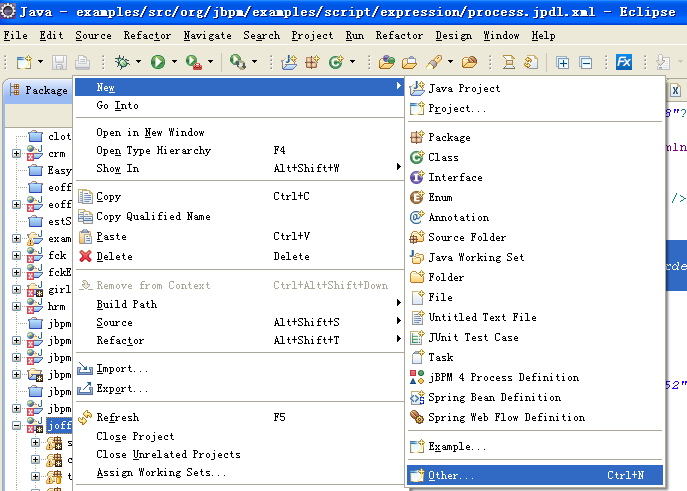


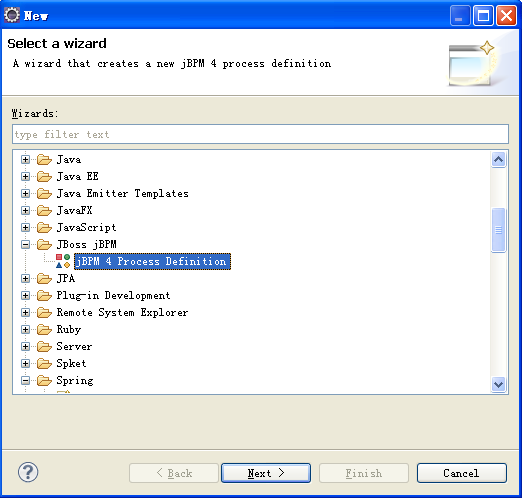


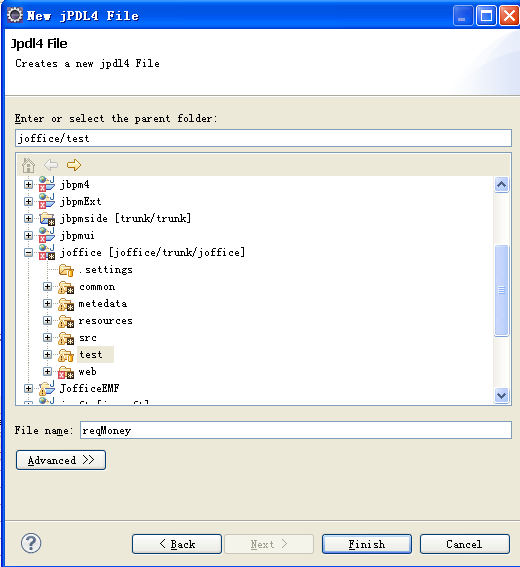
## 1.3 设计第一个流程

安装好Jbpm设计器的插件后，就可以开始自己的流程设计之旅。

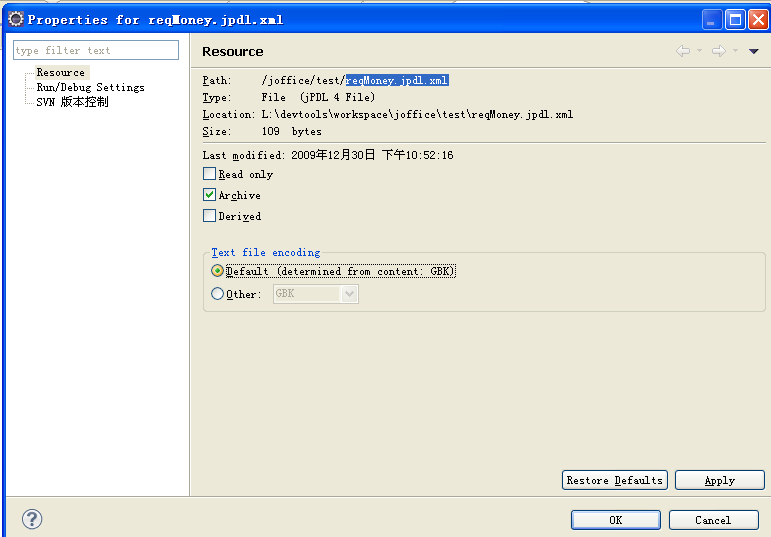
新建流程



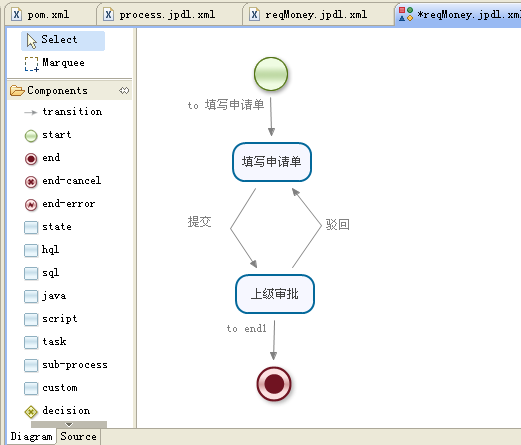




点“Finish”完成后，需要注意自己的开发环境的编码，即reqMoney.jpdl.xml需要设置为GBK以使其支持中文标签。



进入设计器设计流程，如下所示：



由于使用了中文，因而需要进入source模式下，把XML的encoding设置为”GBK”， 否则保存时，会出现乱码。

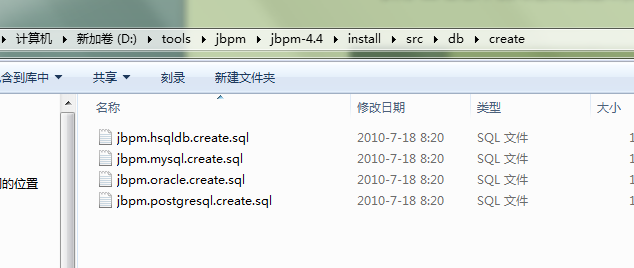
为了避免以上的设计问题，可以把eclipse的文件编码统一设置为UTF-8,进入Eclipse的安装目录，找到eclipse.ini文件，打开并加上以下字符串即可，如下所示：

-Dfile.encoding=UTF-8

# JBPM整合步聚

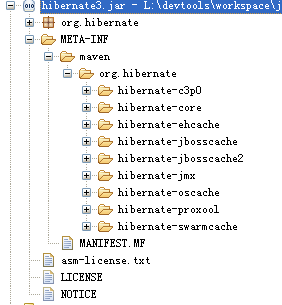
## 执行创建的脚本

执行db目录下的对应的数据库类型的创建脚本，如下图所示：



## 加入依赖包

1. Jbpm.jar
2. 由于Jbpm使用Hibernate作为ORM底层，因而需要把Hibernate的依赖包也加入来。Hibernate依赖的库可以从hibernate.jar包看到：



1. 加入流程有脚本引擎支持，即在Jbpm4的流程定义中引入类似如下的脚本方式，如条件表达式：

<script name=*"invoke script"*

expr=*"Send packet to* ***#{order.address}****"*

var=*"text"*　g=*"96,16,104,52"*>

这需要引入juel-engine.jar,juel-impl.jar,juel-api.jar。

注意：引入juel.jar包后，在tomcat6的环境下，运行时，会与其本身的el-api.jar有冲突，解决办法是：把tomcat6下的lib的el-api.jar替换为juel.jar，同时不要在应用程序下加入juel.jar。

## 整合Jbpm至应用的事务管理

Jbpm底层使用了自己Hibernate的SessionFactory，而我们的程序中也有自己Hibernate的SessionFactory，若在执行我们的程序时，需要调用JBPM的相关流程业务，就会相当在两个Connection完成对数据库的操作，这对于事务管理上带来了困难。因此，我们把Jbpm中的SessionFactory指向我们的SessionFactory来。

### 引入Spring的配置管理(app-jbpm.xml)

同时，由于Jbpm提供对于Spring的较好支持，所以我们加入app-jbpm.xml，其配置如下：

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<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-2.5.xsd"*

default-lazy-init=*"true"*>

<bean id=*"springHelper"* class=*"org.jbpm.pvm.internal.processengine.SpringHelper"* />

<bean id=*"processEngine"* factory-bean=*"springHelper"* factory-method=*"createProcessEngine"* />

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

<bean id=*"executionService"* factory-bean=*"processEngine"* factory-method=*"getExecutionService"* />

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

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

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

</beans>

### 引入jbpm.cfg.xml配置文件

以上jbpmConfiguration Bean 作为Jbpm与Spring的整合的总入口，其配置来自jbpm.cfg.xml文件，我们把该文件定义为：

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<jbpm-configuration>

<import resource=*"jbpm.default.cfg.xml"* />

<import resource=*"jbpm.tx.spring.cfg.xml"* />

<import resource=*"jbpm.jpdl.cfg.xml"* />

<import resource=*"jbpm.bpmn.cfg.xml"* />

<import resource=*"jbpm.identity.cfg.xml"* />

<import resource=*"jbpm.businesscalendar.cfg.xml"* />

<!--

<import resource="jbpm.console.cfg.xml" />

<import resource="jbpm.jobexecutor.cfg.xml" />

-->

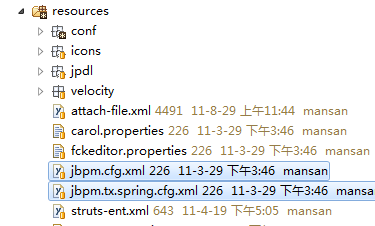
<process-engine-context>

<string name=*"spring.cfg"* value=*"conf/app-jbpm.xml"* />

</process-engine-context>

</jbpm-configuration>

以上其引用的jbpm的xml文件均可以在jbpm.jar包中找到，也可以在Jbpm的解压目录下的src找到。把对应的文件拷至开发目录下的src或其他源代码的根目录。如：



### 配置jbpm.cfg.xml

去掉jbpm.default.cfg.xml中的自带的Hibernate配置，使其直接用我们Spring中的Hibernate配置，如下所示：

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<jbpm-configuration>

<import resource=*"jbpm.default.cfg.xml"* />

<import resource=*"jbpm.tx.spring.cfg.xml"* />

<import resource=*"jbpm.jpdl.cfg.xml"* />

<import resource=*"jbpm.bpmn.cfg.xml"* />

<import resource=*"jbpm.identity.cfg.xml"* />

<import resource=*"jbpm.businesscalendar.cfg.xml"* />

<!--

<import resource="jbpm.console.cfg.xml" />

<import resource="jbpm.jobexecutor.cfg.xml" />

-->

<process-engine-context>

<string name=*"spring.cfg"* value=*"conf/app-jbpm.xml"* />

</process-engine-context>

</jbpm-configuration>

*说明：在这里也可以看到，其是指定使用juel的脚本引擎库，所以需要把juel的依赖库加至系统中来。*

### 配置jbpm.tx.spring.cfg.xml

配置jbpm的事务，以保证spring的系统的事务与jbpm的事务的统一管理。

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<jbpm-configuration spring=*"enabled"*>

<process-engine-context>

<command-service name=*"newTxRequiredCommandService"*>

<retry-interceptor />

<environment-interceptor policy=*"requiresNew"* />

<spring-transaction-interceptor transaction-manager=*"txManager"* policy=*"requiresNew"* />

</command-service>

<!-- Default command service has a Spring transaction interceptor-->

<command-service name=*"txRequiredCommandService"*>

<retry-interceptor />

<environment-interceptor />

<spring-transaction-interceptor transaction-manager=*"txManager"* />

</command-service>

</process-engine-context>

<transaction-context>

<transaction type=*"spring"* />

<hibernate-session current=*"true"* />

</transaction-context>

</jbpm-configuration>

从以上可以看出，我们保证系统的其他模块的hibernate调用与jbpm的hibernate调用均由同一SessionFactory产生（原理是利用线程绑定方式），因而达到实现事务的统一管理的目的。

【说明】

完成以上配置后，现在我们可以基于Spring来进行JBPM的流程管理，在应用程序中，我们是通过app-jbpm.xml中的Bean提供的接口来完成的。以下这些Bean的说明：

ProcessEngine：流程引擎服务总入口，可以获得其他Service

RepositoryService：流程定义库服务类

ExecutionService：流程执行服务类

TaskService：流程任务服务类

HistoryService：历史任务服务类

IdentityService：用户与角色服务类

# JBPM工作流扩展

## JBPM常用术语

1. 工作流引擎组件：核心组合负责生成流程运行的各实例及数据，以及管理流程的运行及结束。
2. 流程定义：预先定义好的业务逻辑流程。
3. 流程实例：业务流程实际运行过程
4. 流程库（Repository）存放所有的流程定义，也可以认为是流程发布的存放库。
5. 任务：逻辑上定义给需要流程的参与人员处理的工作
6. 用户组：代表一组用户，可认为某一类型或某些角色的所有用户。
7. 用户：参与流程任务的特定用户
8. 候选用户：参与流程任务的候选用户，即某些任务在某些情况下可以转给候选用户来处理。
9. 待办任务：指流程参与者目前需要处理的任务
10. 任务表单：用户在处理某任务时，需要填写内容，称之为表单。

【说明】由于JBPM只是处理流程，所以在流程的处理过程中，其任务表单需要由开发人员来定制开发，而表单的数据也没有进行较好的存储，这为查询及管理流程任务带来很大不便，因而需要进行扩展。在扩展之前，我们需要对jbpm的表结构的有所了解。

## 资源库和运行时表结构

JBPM4\_DEPLOYMENT 工作流发布表，存放流程发布定义的id,状态（如挂起，激活）等。

JBPM4\_DEPLOYPROP 流程定义的属性中的key，Id,版本等。

JBPM4\_LOB 存储流程定义相关的部署信息 ，如流程的定义中的XML文件也存储在该表中。

JBPM4\_EXECUTION 存放流程执行中的运行状态(任务也是Execution中的一种)

JBPM4\_TASK存放需要人来完成的Activities(活动)，需要人来参与完成的Activity 被称为Task

JBPM4\_PARTICIPATION参与者表，存放参与者信息，参与者的种类有Candidate、Client、Owner、Replaced Assignee和Viewer。而具体的参与者既可以是单一用户，也可以是用户组

JBPM4\_SWIMLANE泳道表。SwimLane是一种Runtime Process Role。通过SwimLane，多个Task可以一次分配到同一Actor身上

JBPM4\_JOB 存放的是Timer 的定义

JBPM4\_VARIABLE 存的是流程实例或任务实例中的变量，一般用于存储需要在流运算的相关变量或业务相关的数据。

## 历史数据库表结构

JBPM4\_HIST\_PROCINST 与JBPM4\_HIST\_ACTINST 分别存放Process Instance和Activity Instance的历史记录

JBPM4\_HIST\_DETAIL 保存 Variable的变更记录

JBPM4\_HIST\_VAR 保存历史的变量

JBPM4\_HIST\_TASK Task的历史信息

## 身份认证表结构

JBPM4\_ID\_GROUP ，

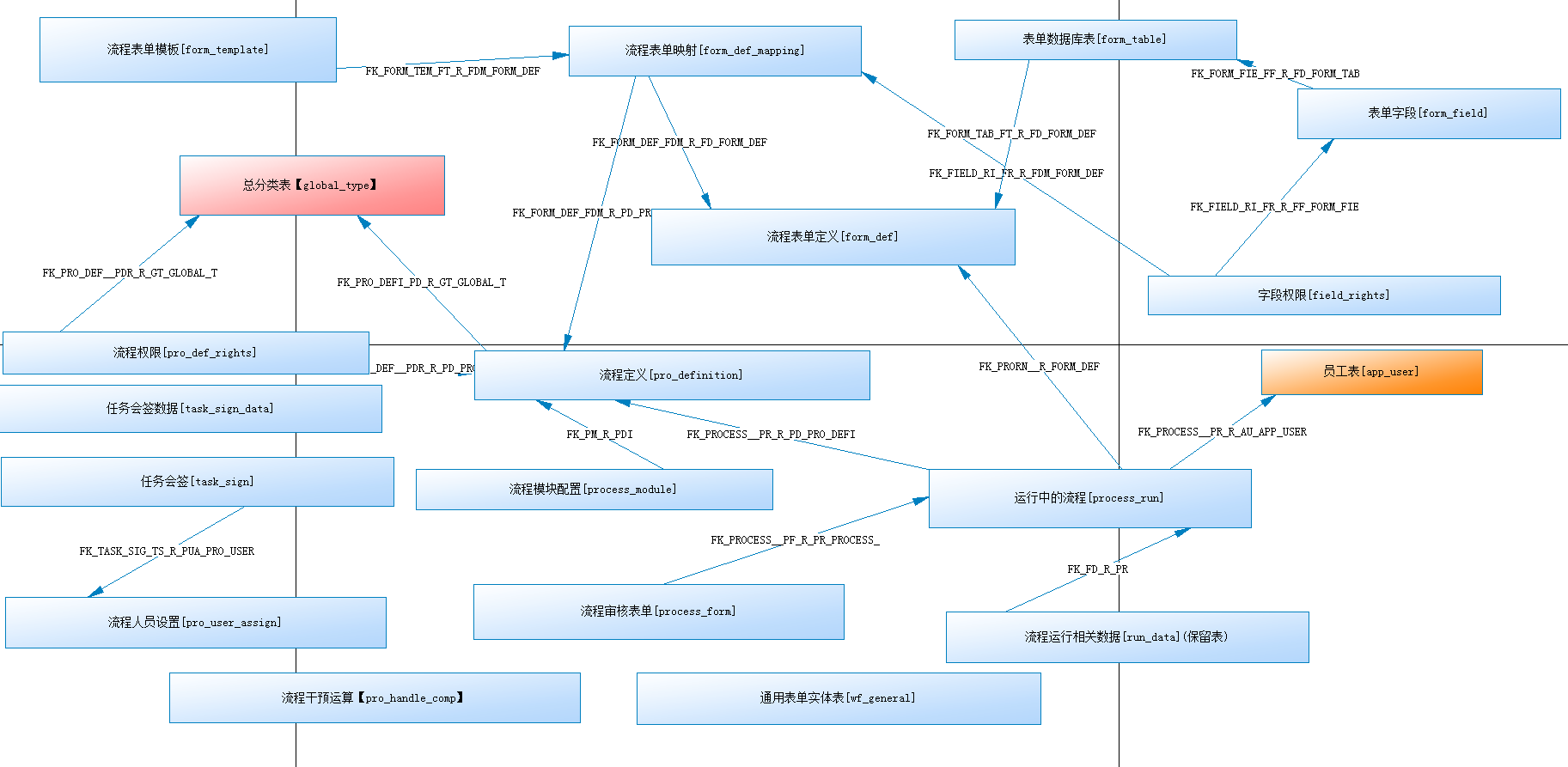
JBPM\_ID\_MEMBERSHIP ，

JBPM4\_ID\_USER

绝大部分系统都有自己的一套用户认证机制，所以一般我们都自己扩展用户的表以实现我们自己的用户认证（主要是为了加载我的待办事项，查找经办人对应的处理任务），

尽管JBPM4也提供了扩展的用户的接口（UserSession），但可惜JBPM4对任务的查找提供的API比较弱，因此也需要我们自己进行扩展。首先我们把用户存在JBPM4\_TASK中的assignee\_，JBPM4\_PARTICIPATION里的USERID\_,若为用户组，我们则存于JBPM4\_PARTICIPATION里的GROUPID\_。

## 扩展表E-R图



## 扩展表说明

* form\_template

表单模板内容的配置，存放是表单的配置，表单模板设计器设计生成的内容，包括URL模板其对应的路径（允许每个任务节点对应不同的模板）,

EXT模板目前是放置在应用程序的目录下，如:WEB-INF/FlowForms/流程版本号/节点名.vm

* form\_def

在线制作流程表单的定义，主要存储表单制作的内容及标题的定义等信息。

* form\_table

在线制作表单生成的物理表表名定义

* form\_field

在线制作表单生成的物理表对应的字段定义

* field\_rights

在线制作表单的字段权限定义，如在某个节点上，A字段仅能读。

* pro\_def\_rights

流程定义权限，包括定义的分类权限，具体流程的权限，权限限制于是具体人、部门还是角色

* form\_def\_mapping

表单与流程定义的映射表，每个流程的任务节点一般都会映射为对应的业务表单，业务表单可来自在线设计的表单也可以为表单模板。

* pro\_definition

流程定义，放置最新的流程定义，包括流程的最新在线设计xml及jbpm的定义，版本等信息

* app\_user

流程参与用户表，也是系统的用户表

* task\_sign

任务会签的设置表，如一票否决，少数服从多数的设置。

* task\_sign\_data

任务会签的数据保存，记录会签的用户投票的情况

* process\_run

流程运行实例，跟jbpm流程实例对应起来，

* run\_data

流程运行表单（在线表单）数据，与jbpm4\_variable数据一样，

* process\_form

流程审批表单的审批历史及数据

* process\_module

流程模块，用于为业务模块加上流程启动的功能（尚未使用，为扩展表）

* wf\_general

在线流程表单的通用数据

* pro\_user\_assign

流程的任务节点的人员设置。

流程任务的人员授予

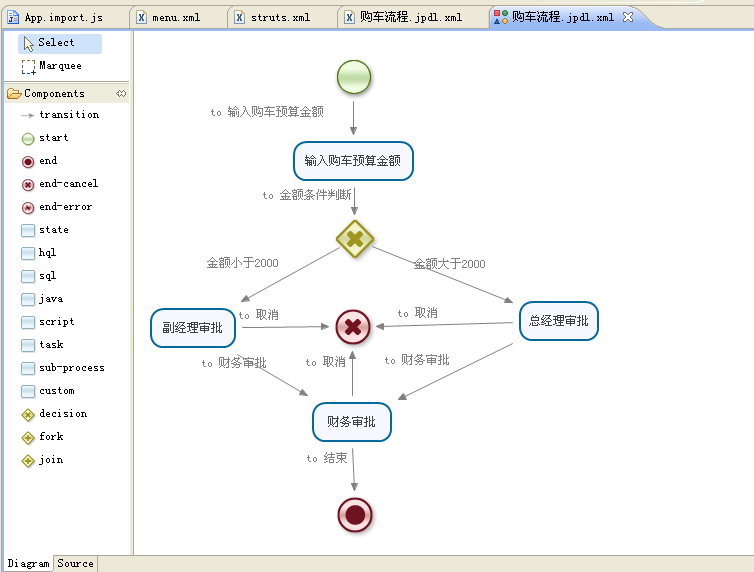
流程任务最终都需要指定人员来执行，任务的执行人员可以在流程的定义文件（ＸＭＬ）中来指定，这种方式我们也称之为前绑定角色或人员，但是那样会导致流程在运行起来修改指定的执行人员并不方便，要加强这种灵活，则需要进行使用后绑定的方式，则流程发布后，可以再通过程序或界面去设置其每个任务执行的人员或角色，或在任务执行的过程中，指定其对应的执行人员。后设置的每个任务指定的执行人员或角色的信息则存储在pro\_user\_assign表中。

## 流程发布

方法一：

**定义流程**

使用Jbpm Eclipse plugin定义流程如下所示：



其定义的XML文件如下所示：

<?xml version=*"1.0"* encoding=*"GBK"*?>

<process key=*"buyCarFlow"* name=*"购车流程"* xmlns=*"http://jbpm.org/4.0/jpdl"*>

<start g=*"196,22,48,48"* name=*"开始"*>

<transition g=*"-143,-11"* name=*"to 输入购车预算金额"* to=*"输入购车预算金额"*/>

</start>

<task g=*"153,104,133,52"* name=*"输入购车预算金额"*>

<transition g=*"-92,-11"* name=*"to 金额条件判断"* to=*"金额条件判断"*/>

</task>

<decision g=*"197,184,48,48"* name=*"金额条件判断"*>

<transition g=*"-83,-17"* name=*"金额小于2000"* to=*"副经理审批"*>

<condition expr=*"#{reqAmount&lt;2000}"*/>

</transition>

<transition g=*"-28,-16"* name=*"金额大于2000"* to=*"总经理审批"*>

<condition expr=*"#{reqAmount&gt;2000}"*/>

</transition>

</decision>

<task g=*"10,271,98,52"* name=*"副经理审批"*>

<transition g=*"-47,-17"* name=*"to 取消"* to=*"取消"*/>

<transition g=*"-71,-17"* name=*"to 财务审批"* to=*"财务审批"*/>

</task>

<task g=*"379,264,92,52"* name=*"总经理审批"*>

<transition g=*"-47,-17"* name=*"to 取消"* to=*"取消"*/>

<transition g=*"-71,-17"* name=*"to 财务审批"* to=*"财务审批"*/>

</task>

<task g=*"172,365,92,52"* name=*"财务审批"*>

<transition g=*"-47,-17"* name=*"to 取消"* to=*"取消"*/>

<transition g=*"-47,-17"* name=*"to 结束"* to=*"结束"*/>

</task>

<end-cancel g=*"195,272,48,48"* name=*"取消"*/>

<end g=*"197,460,48,48"* name=*"结束"*/>

</process>

说明：

1．注意以下这部分的定义

<transition g=*"-83,-17"* name=*"金额小于2000"* to=*"副经理审批"*>

<condition expr=*"#{reqAmount&lt;2000}"*/>

</transition>

<transition g=*"-28,-16"* name=*"金额大于2000"* to=*"总经理审批"*>

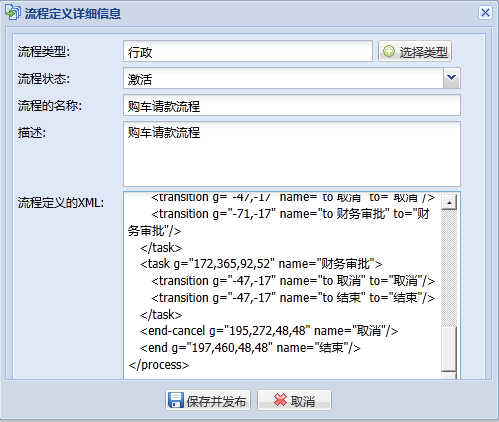
<condition expr=*"#{reqAmount&gt;2000}"*/>

</transition>

其使用了表达式来进行条件的跳转, *reqAmount代表为流程表单中提交的数据字段名称，在后面我们会在表单中使用该字段名称。*

**发布流程**

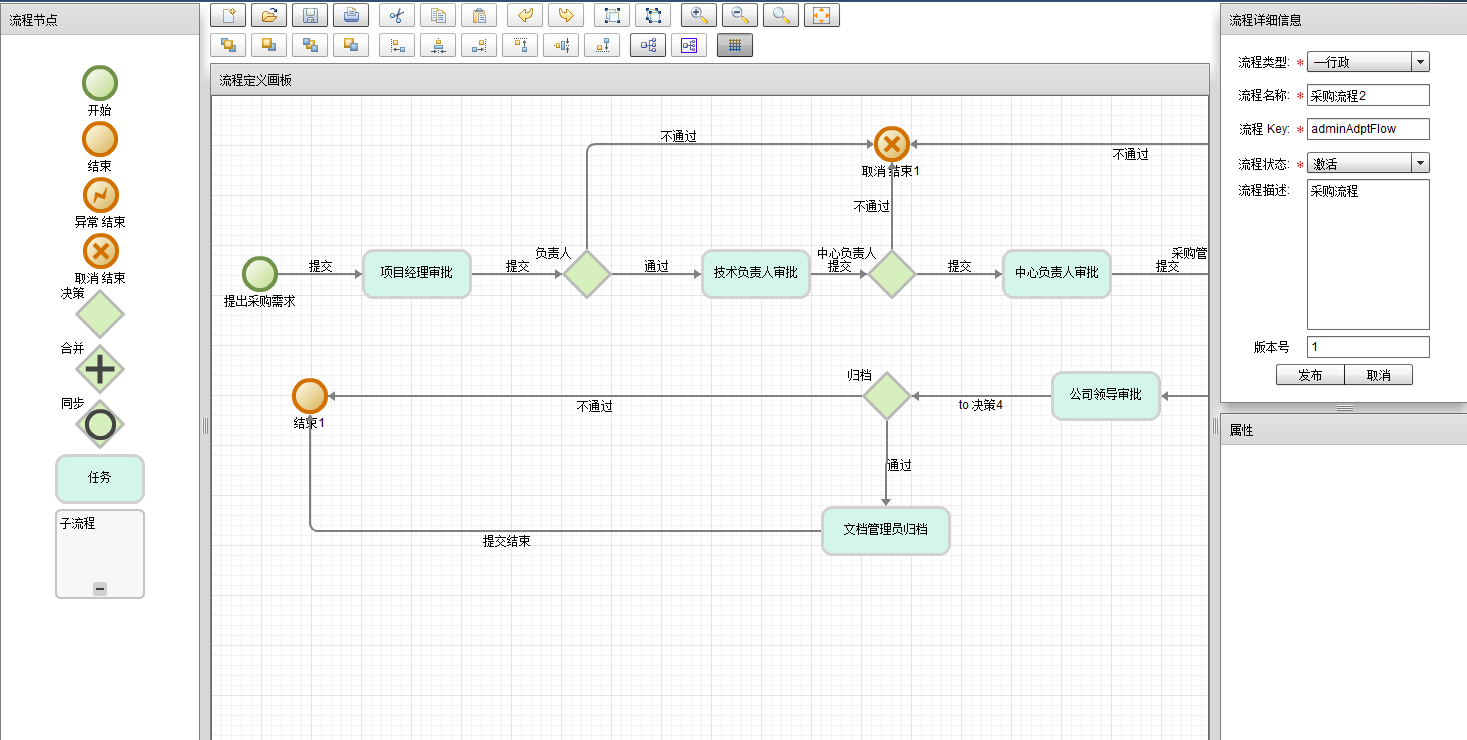
点击流程管理菜单中的流程定义管理，点击左流程左分类其中一项，再点“发布流程”按钮，如下所示：



把以上XML文件拷至该对话框。并且填写好流程名称为“购车请款”，注意，流程名称写好后，最好不要更改，否则后面的流程表单的路径也需要更改。再点击“保存”按钮，则成功发布了一个流程定义。

**方法二：**

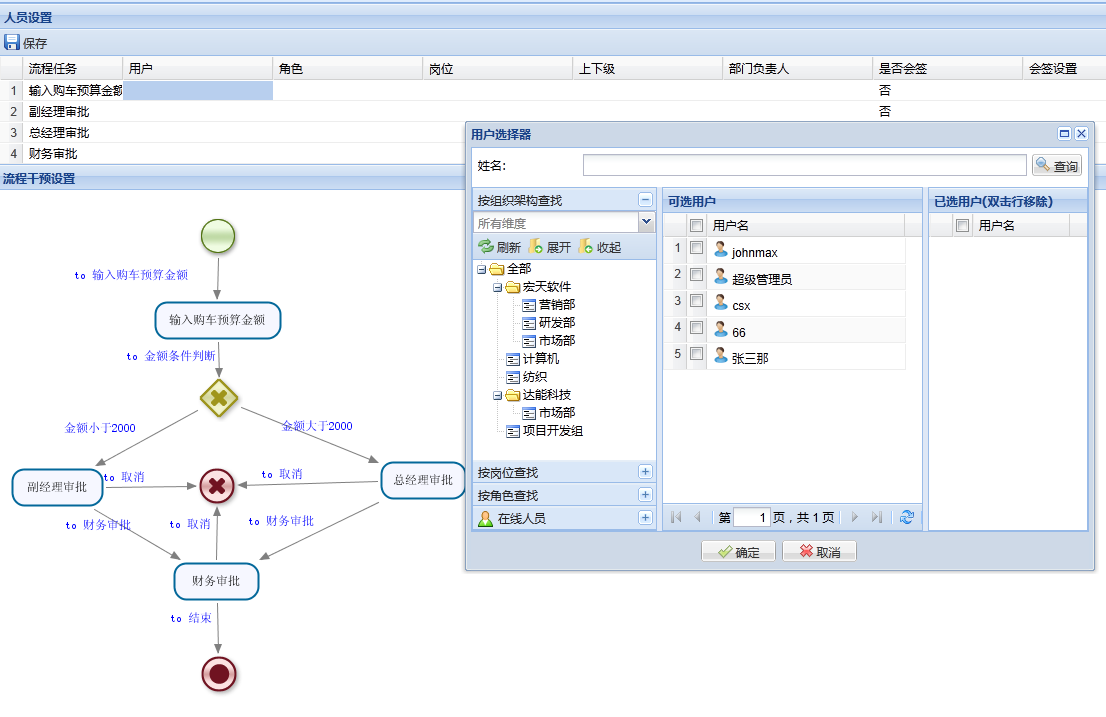
在线定义并发布



直接点击发布则可。

## 流程任务处理人员设置

在流程管理列表中，点击“购车请款”流程行中的“”按钮，进入流程任务人员的设置界面，如下图所示：



说明：

系统中的任务执行指派如下所示：



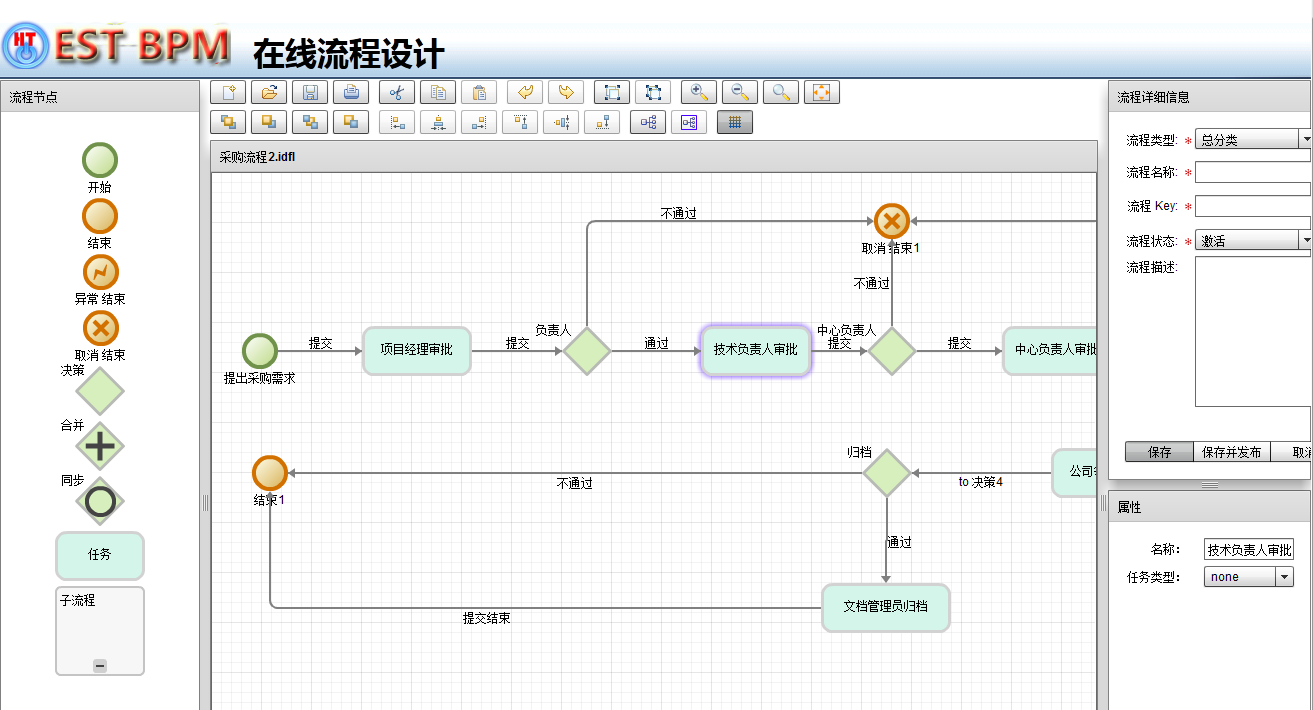
关于以上图的说明：

1. 流程执行过程中，以在流程表单指定的人员为优先
2. 若在执行过程中，没有指定对应的任务的执行人，则需要按后台的流程任务的人员配置进行设置。
3. 后台设置的人员是合并规则，即同时允许一个节点即设置固定的人员、岗位人员、上下级人员等。

## 在线流程设计器

在ESTBPM中，通过结合IBM ILOG的Elixir Enterprise 3.0提供的可视化组件，以实现在线的流程编辑设计功能。该设计器生成对应的idfl文件，ESTBPM通过自己的扩展引擎解析该文件，并且把它转化成JBPM的流程定义文件，以实现在线的流程设计，同时保留idfl文件的内容，以方便后续的流程编辑更改的实现。

而通过扩展，把jbpm的常用节点放在设计器的左边，可根据需要不断增加流程的节点，但在中国，流程的节点绝大部分是需要人去参与的，所以只是把任务节点作为重要的节点展示出来，若需要其他自动化的节点，即需要在设计器上进行扩展即可，其最终实现的效果如下：



## ESTBPM工作流扩展表单

### 在线表单

在线表单提供后台的配置管理功能，通过FCK表单编辑器，提供编辑html表单的功能，并且生成流程的表单，其工作原理如下所示：



1. 通过后台的表单设计器，生成表单的界面及对应的后台的数据库表结构，并且存于Form\_table及form\_field表中。
2. 通过Form\_table及Form\_field表生成Hibernate的实体的映射文件。
3. 通过重写Hibernate的SessionFactory，提供实时动态加载映射文件，生成该映射文件对应的物理表，以方便存储其流程中的数据。
4. 在流程执行的任务中，如ProcessNextForm.js中，即通过ProcessActivityAction来加载流程的业务表单及填充其数据。
5. 在流程历史表单中，通过ProcessRunDetail.jsp加载流程业务数据。

### EXT表单

若在流程的设置中，指定了某个任务节点其表单为EXT表单，则前台加载表单时，会加载其对应的业务表单模板，并且把业务表单加载至其流程的执行表单中。其原理如下所示：

通过ProcessNextForm.js或ProcessRunStart.js



流程任务的表单划分为几个模块：

1. 流程任务功能按钮，如驳回，执行下一步等。
2. 流程的执行下一步的路径
3. 流程执行一步的人员列表
4. 流程表单的展示区（包括在线表单及EXT模板）
5. 显示流程表单的审批历史

在第四点上，EXT模板则是通过js函数，动态创建一个Panel ,并且动态加入至外围的FormPanel,验证则通过外围的FormPanel来进行验证，则对于内置的表单也提供自定义的验证接口调用，一般在内部表单中，通过validate方法的定义，定义其验证规则，该方法只要返回true则代表通过验证，若返回false则表示内置的表单数据不合法。

其表单模板加载原理图如下所示：



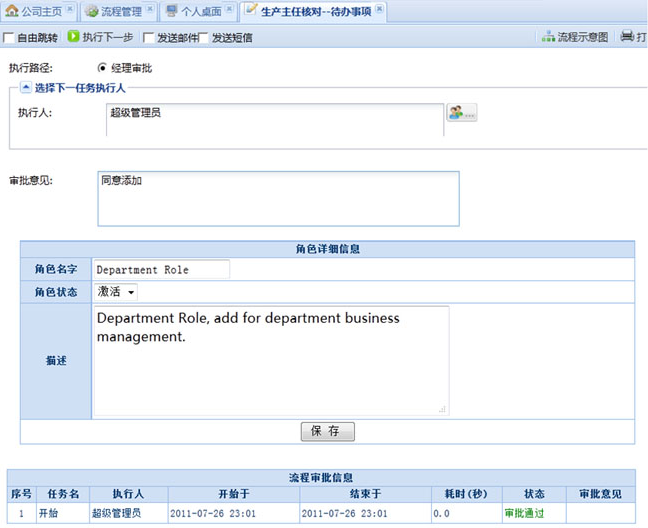
1. ProcessRunStart.js或ProcessNextForm加载其对应的业务表单模板，通过发起请求至ProcessActivityAction的get方法，返回其业务表单内容代码（EXT模板，为一个EXT的容器类，如Panel），在ProcessActivityAction中处理模板时，会把相应的Model加载至模板，如该流程所有的变量，当前任务的节点名称（ActivityName）,因此，在任务表中，可以通过${varName}的方式获取到变量的值
2. ProcessRunStart或ProcessNextForm取得业务表单后，把业务表单的内容动态执行，如this.formExtPanel=eval('new ('+valExt+')('+vmParams+');');
3. this.formExtPanel被动态加入至外围表单

其流程图如下所示：



### URL模板表单

系统也支持直接通过URL引入对应的任务表单，该种方式适合于不需要过多对现在系统的表单的修改。



中单的那部分表单是来自另一个URL的展示

# 实现过程部分说明

### 流程启动调用序列图



1. ProcessRunStart.js通过defId，在saveAndStart方法，把流程表单中提交至ProcessActivityAction的save方法，进行流程启动。
2. ProcessActivityAction通过调用doStartFlow方法进行流程的启动工作
3. 在doStartFlow方法中，首先会调用用户自定义的前置方法，该方法的定义一般置于用户的自定义表单中，以变量名为preHandler，值为xxxService.methodName的定义方式。用户则需要在他的xxService类中，定义该方法，该法的参数类型为FlowRunInfo类型，返回的参数值为Integer，1代表成功执行，0代表执行失败，－1则代表不需要执行。
4. 调用FlowService类的doSaveData进行业务表单的数据保存，这种处理是针对在线表单的数据保存操作。
5. 保存数据后，调用JbpmService进行JBPM流程数据的启动，并且返回流程实例对象ProcessRun
6. 进行流程的短信、邮件的通知调用notice
7. 再次执行用户的后置任务方法(由afterHandler参数指定)，该方法的定义与preHandler 的方式类似

### ****流程执行任务实现序列图****



该执行过程与启动流程类似，只不过是通过taskId进行任务的执行跳转。

### 部分关键类实现说明

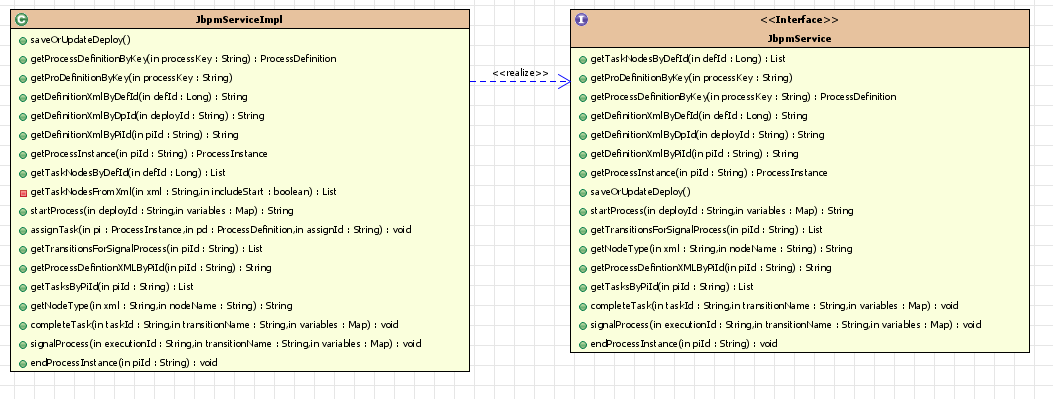
* 为了实现的方便，系统把工作流的相关操作放置在JbpmServiceImpl.java类中，该类提供了流程的相关处理方法，如发布流程：

**jbpmService.saveOrUpdateDeploy(proDefinition);**

删除流程：

**jbpmService.doUnDeployProDefinition(new Long(id));**

该类是整个工作流引擎的最核心的类，其下层是Jbpm的底层操作接口，对外还同时保存了流程的相关信息放置Jbpm流程外的表中进行存储，以使得查询更方便。



* 流程的定义发布由ProDefinitionAction中提供了实现，添加流程或删除流程定义等，其通过JbpmService接口来完成的。
* 流程的人员分配由ProUserAssignAction来完成，如保存流程任务的人员分配设置时，由该类的save方法完成
* 流程的启动及执行过程由ProcessActivityAction类提供实现，其实现细节请参与其源代码
* 流程的任务操作，如锁定任务，任务代办，释放任务由TaskAction提供实现
* 流程运行过程中的详细信息由ProcessRunDetailAction提供了实现