**BPMX3平台开发指导**

编写人：BPMX3平台开发组

# 概述

本文档目的是为了用户进行BPMX3平台的二次开发进行指导，提供框架及技术的参考说明，帮助开发用户快速掌握系统，并且进行二次开发或进行系统的功能改造。

# 系统平台运行要求：

## 软件部分

**服务端：**

|  |  |
| --- | --- |
| **软件名** | **说明** |
| **数据库** | Oracle 10g++,mysql,mssql2005 |
| **Web服务器** | **T**omcat 6++ 或WebLogic 10g+或Glassfish 2++或JBoss 4+ |
| **JDK** | JDK 1.6.0\_10 |
| **操作系统** | Windows 2003 Sp2（建议）或Windows XP以上版本，Linux（如 RedHat Enterprise Linux AS 4以上 |

**客户端：**

|  |  |
| --- | --- |
| **软件名** | **说明** |
| **浏览器** | IE 6，IE7，IE8 或Firefox 1.5+，或Google Chrome 2.0+  或Opera 10+ |
| **操作系统** | Windows XP以上版本 |

## 硬件部分

最低配置：

CPU：INTEL酷睿2 (主频2.0G以上) ,二级缓存 4M以上

内存：DRR2以上，2G以上

硬盘：6G以上，10000转/秒

# 安装及部署

本平台目前是允许不同的数据库，不同的Web服务器，不同的浏览器，可以部署在不同的操作系统上，以下的搭配是经过BPMX3开发团队测试并且可以正常使用的。

Windows 2003 Sp2

Oracle 10g

JDK 1.6\_27

Tomcat 6.0.35

## 数据库配置

安装Oracle数据库，完成后

1. 使用系统管理员登录。

Sqlplus /nolog

--在sqlplus下执行。

conn / as sysdba

1. 创建数据库表空间

表空间名称bpmx3,初始大小500M,自动扩展每次大小50M,最大为20G。

create tablespace bpmx3

logging

datafile 'D:\oradata\bpmx3.dbf'

size 500m

autoextend on

next 50m maxsize 20480m

extent management local;

1. 创建数据库用户。

创建用户bpmx3，密码bpmx3，指定默认表空间为bpmx3

create user bpmx3 identified by bpmx3

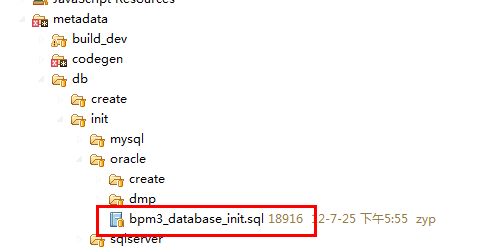
default tablespace bpmx3 ;

1. 用户授权

给bpmx3授予connect ,resource的权限。

grant connect,resource to bpmx3 ;

导入BPMX3初始化数据（使用脚本进行），执行以下脚本：



1. 执行exit命令 以用户bpmx3登录：sqlplus 用户/密码@口令。
2. 执行与本文档同一目录下的Oracle版本的初始化脚本文件bpmx3-oracle-init.sql，执行命令如下：

## 应用程序配置

1. 修改Tomcat/webapps/bpm /WEB-INF/classes/conf/app.properties,如下红色部分

jdbc.driverClassName=oracle.jdbc.OracleDriver

jdbc.url=jdbc:oracle:thin:@localhost:1521:orcl

jdbc.username=bpmx3

jdbc.password=bpmx3

#配置一个公共的对外邮件，系统默认的邮件由它来进行发送

mail.from=

mail.host=

mail.username=

mail.password=

mail.defaultEncoding=utf-8

修改成自己的邮箱配置。

1. 配置系统运行日志（可选）

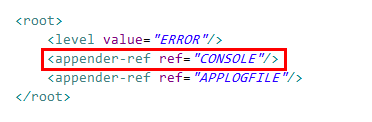
修改Tomcat/webapps/joffice/WEB-INF/classes/conf/log4j.xml

日志缺省放在程序的目录的logs下，可以不用更改，也可以直接写至某个目录，如c:/logs/bpm.log

1. 把日志写至某个文件

修改该文件中的以下标红框标记部分：

…



同时写至文件与日志，在生产环境下可以只写进文件,可以去掉红框标记部分。

1. 调整日志的级别

在调试阶段可以把以下改成DEBUG，生产环境下改成WARN。



1. 把bpm.war包拷至Tomcat/webapps目录下，启动Tomcat。
2. 运行Tomcat后，打开浏览器直接访问<http://localhost:8080/bpm/index.jsp>

初始化账号

账号： admin

密码为: 1

# 平台系统框架介绍

## 框架特点

BPMX3是基于JEE技术平台，整合大部分流行的开源成熟的技术，并且较好融合在一起，组成功能强大的开发平台，并且容易升级扩展，因此，可以保证开发系统的质量，同样可以应付不断变化的业务需求。

特点：

* 开发快速稳定
* 功能强大
* 升级容易

## 整合技术

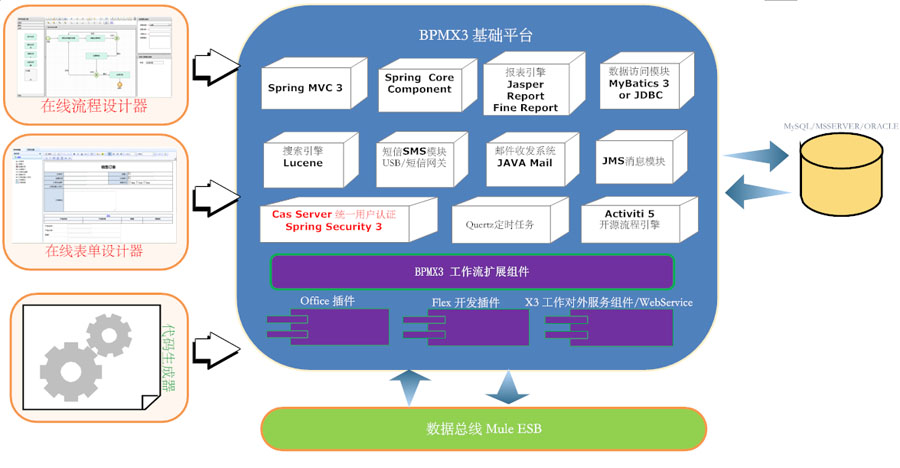
目前的版本整合的技术有：

* Spring 3.0.3
* Spring-mvc
* Spring Security 3.0
* Spring AOP
* Mybatis3.4
* UEDITOR 1.2.6
* CXF 2.3
* ACTIVITI5.10
* FREEMAKER 2.3.4
* JAVAMail
* ACTIVE MQ5.5
* JQUERY 1.7
* JSTL 1.2
* Quartz2.0
* Other: Log4j

由于使用的是轻量级的开源框架，因此比较容易进行升级及扩展，不存在软件的“黑匣子”，为开发带来方便。

## 系统框架层次

整个系统的应用开发层次结构如下所示：



整个系统大体上可以分开三层，数据展示层，业务逻辑层，数据访问层。而从开发的层次上来说，可以分为Model,Dao,Service,Controller。

### 表示层

表示层主要位于客户端，对应本产品的运行环境来说，就是IE、FireFox等浏览器。由系统用户直接使用，表示层负责页面逻辑处理和页面显示， 包括所有用于和终端用户交互的组件。主要使用HTML、DHTML、CSS、JavaScript、AJAX等技术。在BPMX3平台里，主要使用JSTL进行数据展示，JS框架使用的JQUERY。

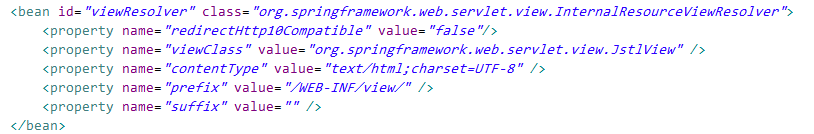
### 视图层（View）

视图（View）层主要位于服务器端，基于MVC架构模式的设计，把视图和数据与控制分离，单独作为一层。因此视图很容易修改。JSTL是本产品主要使用的视图层技术（可通过定制设置切换为其他技术，如Velocity，Freemaker，XML等）。

由于页面使用了JSTL，因此大部的列表及表单的展示页面，其数据通过控制器层通过SPRING MVC做数据传递。

在系统中视图配置如下：

这个配置在 conf/app-action.xml 中。



上图表示视图的文件放在/WEB-INF/view/目录下面。

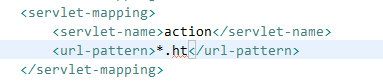
访问扩展名配置：

我们系统中访问页面都是使用.ht 作为扩展名进行访问这个是在web.xml文件中做的配置。

配置如下：



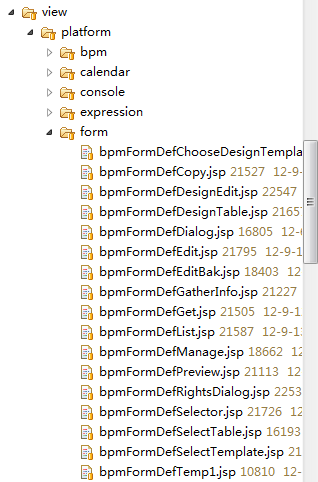
对应的路径映射配置：



在我们的系统中一般是一个表对应三个视图，其中有固定的命名规则。

分别是列表视图，明细视图，编辑视图。

系统文件的视图命名规则：



在URL地址栏中输入的地址如下：

/bpm/platform/system/sysDataSource/list.ht

系统中对应的视图页面如下：

视图目录/platform/system/sysDataSourceList.jsp 。

URL规则:

子系统/子模块/对应的表/对应的方法 +”.ht”

视图规则：

子系统/子模块/对应的表 +控制器方法名（首字母大写） +“.jsp”。

*除此之外，视图层也使用AJAX的方式展现数据。*

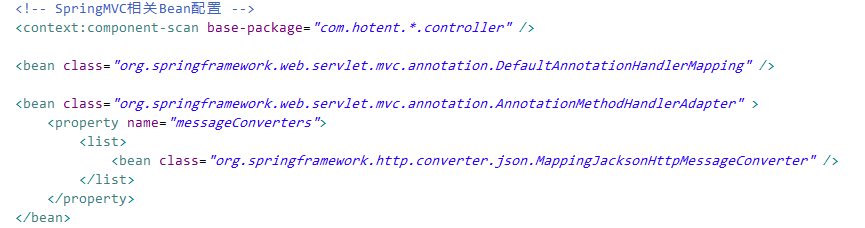
### 控制层（Controller）

控制器层主要负责接收访问请求，处理用户输入，调用业务层逻辑处理，返回业务数据和相应的视图，把结果交给视图层进行处理。

在本系统中我们使用spring-mvc做控制器，控制器是一个普通的类，使用@Controller注解即可以实现控制器。

在系统中配置如下：

系统配置文件 conf/app-action.xml 。



[通过这个配置扫描com.hotent.\*.controller.\*所有包含@Controller](mailto:通过这个配置扫描com.hotent.*.controller.*所有包含@Controller)注解的类加入到spring容器中。

其与视图层及业务层之间的关系如下所示：

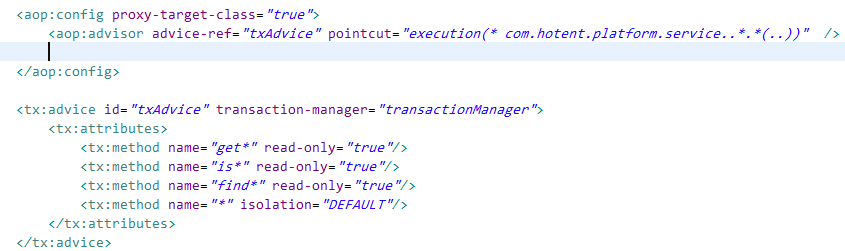


### 业务层（Business/Service）

业务层主要负责业务的具体处理（例如下订单），只与具体业务的处理流程、逻辑有关，而与Web框架、数据持续化框架等无关。基于这种设计，可以很自然的实现对SOA的支持，每个业务处理都是一个BS的方法，可以方便重用，也方便扩展，也方便通过Web Service等技术与其他系统集成。该层是系统里面非常重要的一层，事务控制也是应用在本层。通过Spring AOP对访问类的拦截控制，可以比较容易实现事务的管理。

其配置如下所示：

配置文件为app-resources.xml



意思是对所有的

com.hotent.\*.service包下的所有的方法进行拦截，除了get,is,find开头的方法名外，其他所有的方法都会在执行后进行事务的提交，出错事进行事务的回滚的操作。

管理示意图：



业务层代码没有实现任何接口，业务层代码使用了@service注解。

在app-resources.xml配置

<context:component-scan base-package="com.hotent.\*.service" />

### 数据访问层

主要使用DAO的设计模式。DAO里面不应该有业务处理逻辑，只应该有数据获取存储的处理。在本系统中，只要使用mybatis作为ORM框架，当然可以转为其他的ORM框架，系统扩展也比较容易。由于使用mybatis，数据库差异是需要解决的一个问题，我们使用方言实现不同数据库的分页。

另外：

一些不同数据库有差异的sql语句我们也有一个比较成熟的解决方案。

请查考下面的网址：

<http://bbs.jee-soft.cn:8086/showtopic-49.aspx>

另外 我们还可以使用spring的jdbctemplate实现数据库的访问和读写。

### 企业资源层（EIS）

EIS层不属于需要开发的内容，其包括是系统可以使用的资源。主要是数据库以及企业信息系统（ERP，SCM，CRM等），也可以包括外部系统的接口，例如支付网关等。BPMX3默认整合了CXF与ActiveMQ内置在里面，可以通过app-cxf.xml与app-jms.xml进行配置。

# 开发环境

## 软件准备

* JDK 1.6.0\_10 版本（Later）

http://www.oracle.com/technetwork/java/javase/downloads/index.html

* Eclipse 3.5（Later）

<http://www.eclipse.org/downloads/>

下载[Eclipse IDE for Java EE Developers](http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/heliossr2)

* Tomcat 6 （Later）/ (Weblogic 10 later)/GlassFish V2 Later

Tomcat6: <http://tomcat.apache.org/download-60.cgi>

或 Weblogic10.3

<http://www.oracle.com/technetwork/middleware/weblogic/downloads/index.html>

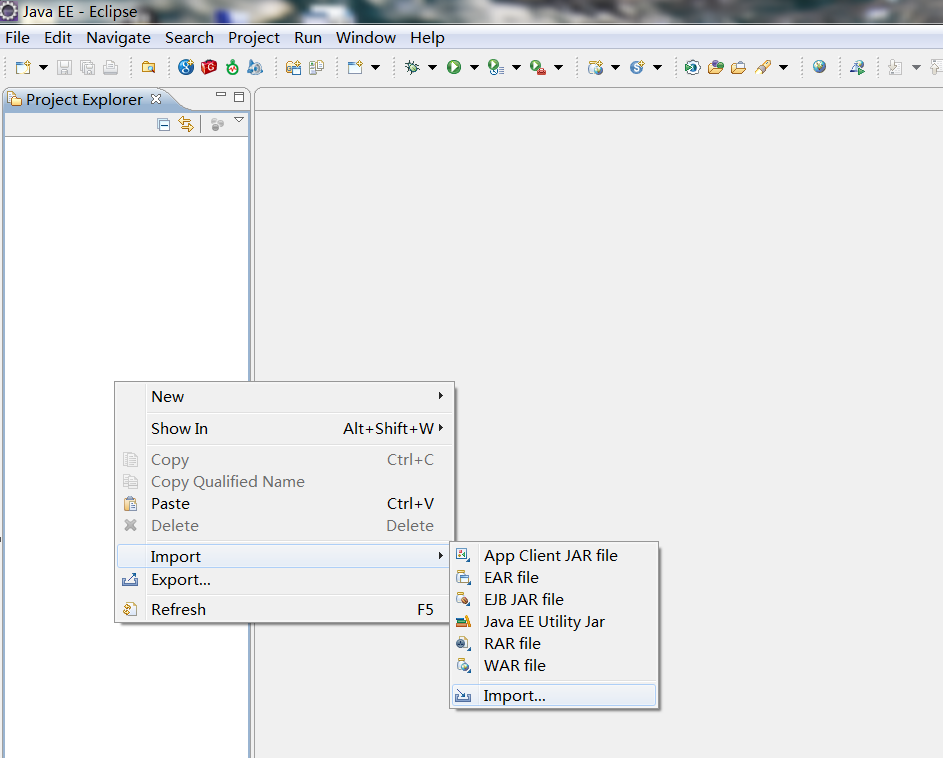
## 开发环境运行配置

运行配置基于eclipse 3.6开发环境，开发工具可在以下地址下载：

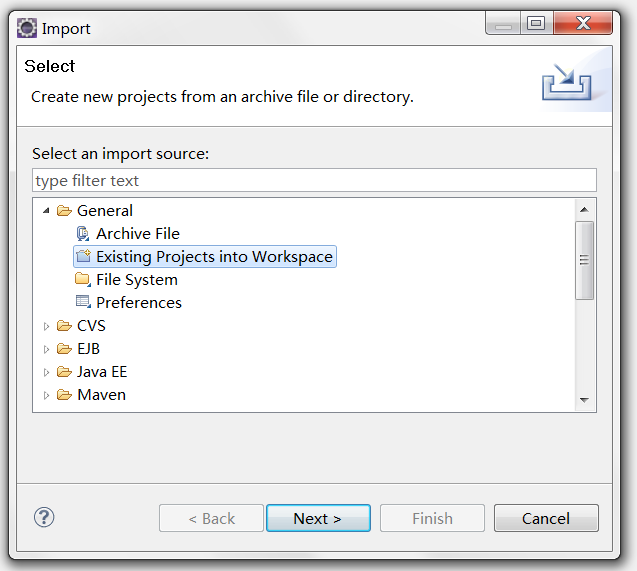
http://mirror.bjtu.edu.cn/eclipse/technology/epp/downloads/release/juno/SR1/eclipse-jee-juno-SR1-win32.zip

### 导入代码

在项目视图中，右键，选择Import🡪Import项，如下所示：

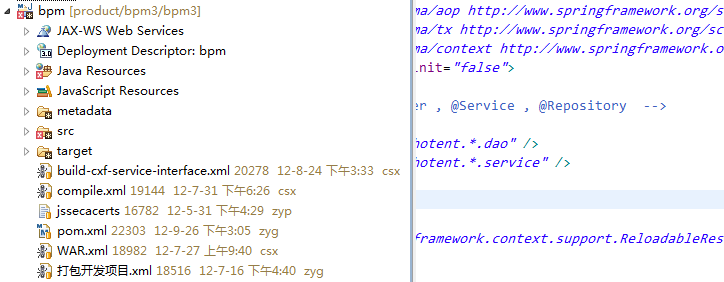


在Import窗口中选择“Existing Projects into Workspace”，选择 Next按钮



在“select root directory”选项中选择bpmx3代码目录，然后再选择中“bpmx3”，点击“Finish”按钮。

完成后，即看到以下类似的项目开发界面



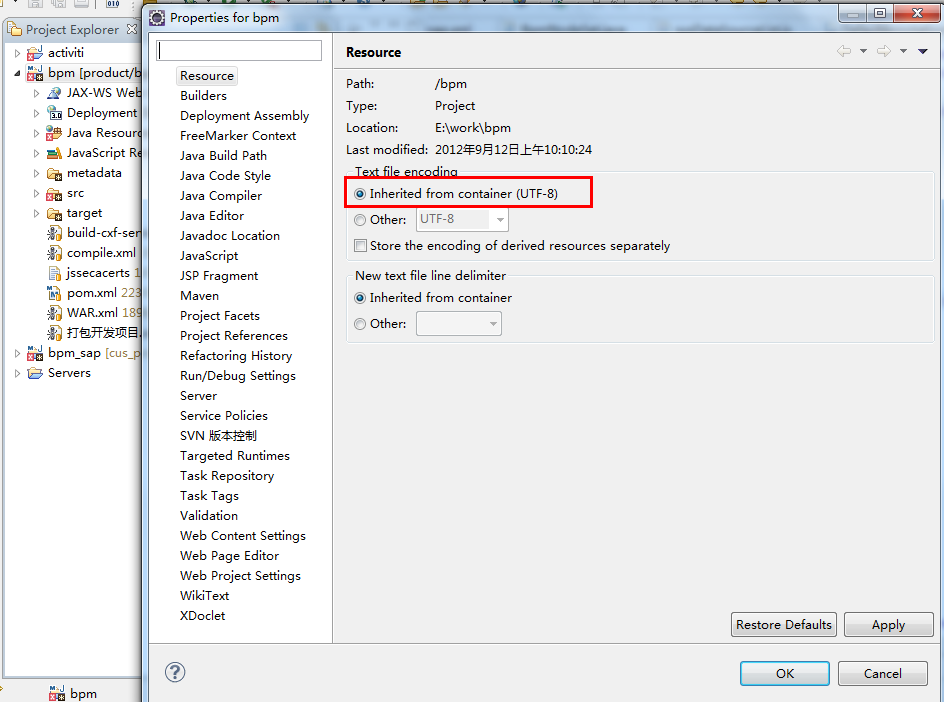
### 生成数据库

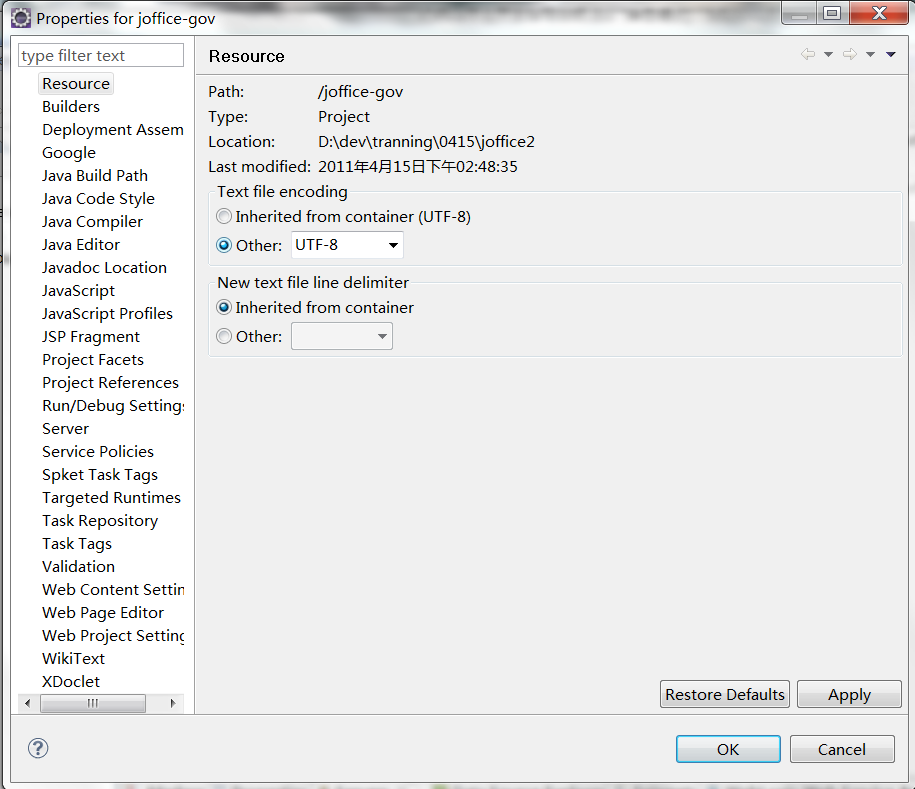
请参考【数据库配置】

### 配置开发运行环境

在安装软件的时尽量不要使用默认的目录，比如tomcat安装到programs目录下，最好选择根目录，目录中尽量不要有空格。

#### 配置字符集为UTF-8



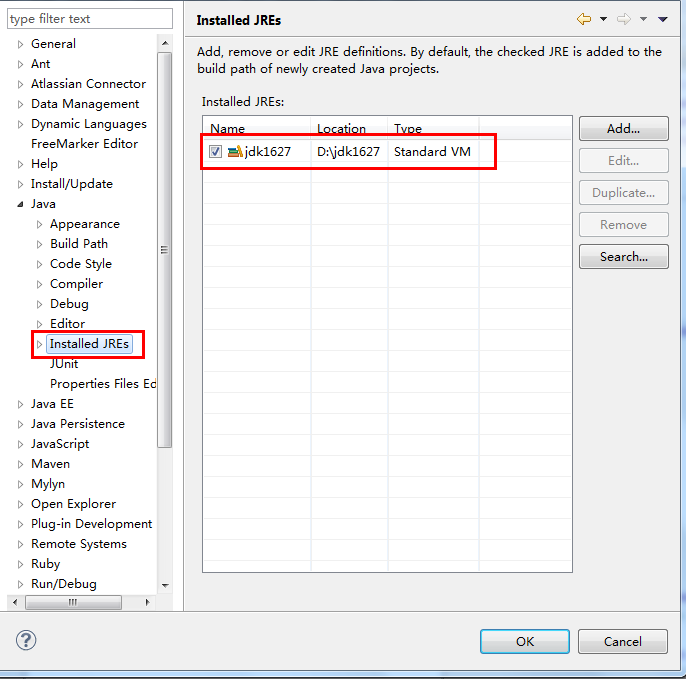


再点Apply按钮

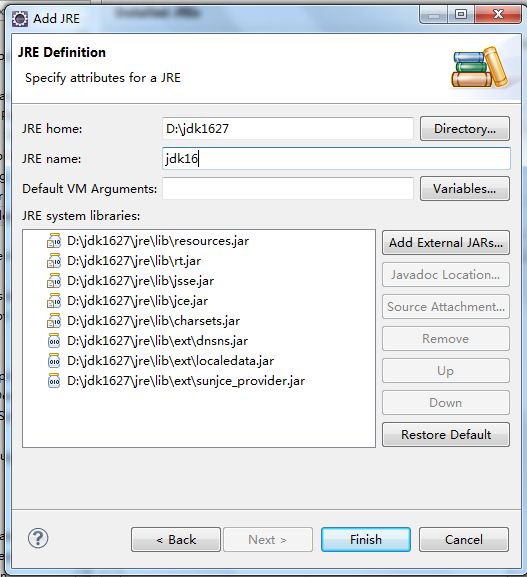
也可以通过设置workspace的编码为UTF-8

#### 配置JDK

在eclipse的Windows菜单下，选择Preferences，在弹出的窗口中，选择JAVA选项下的Installed JREs



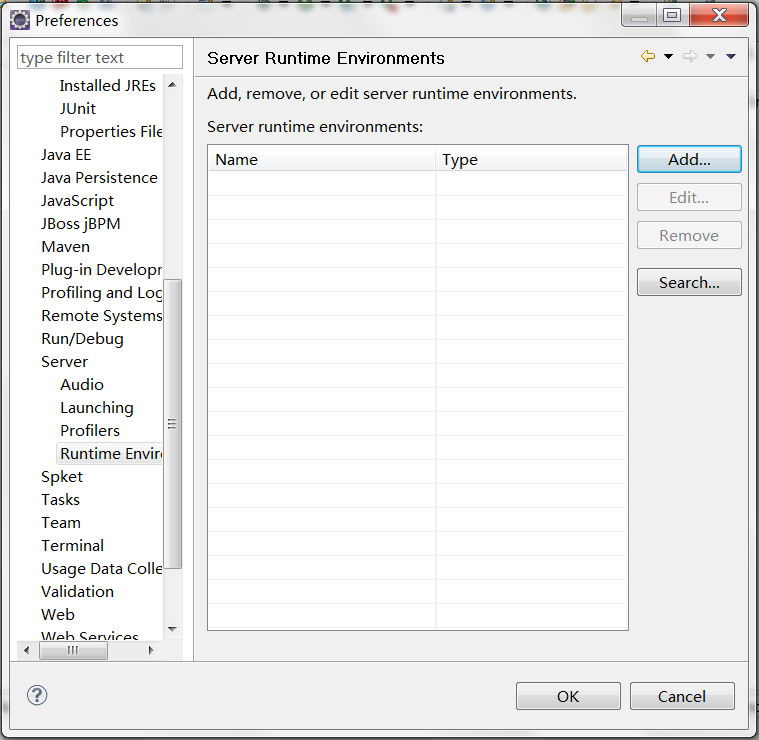
添右边的“Add…”按钮，在弹出的第一步中选择Standard Vm，第二步中选择正确的JDK目录，并且点Finish按钮。



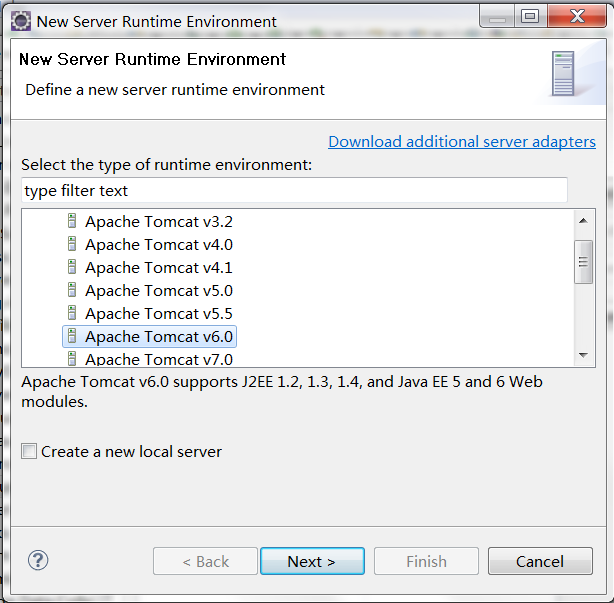
完成后，选择刚添加的JDK作为默认的运行配置。

#### 配置开发环境的Web Server

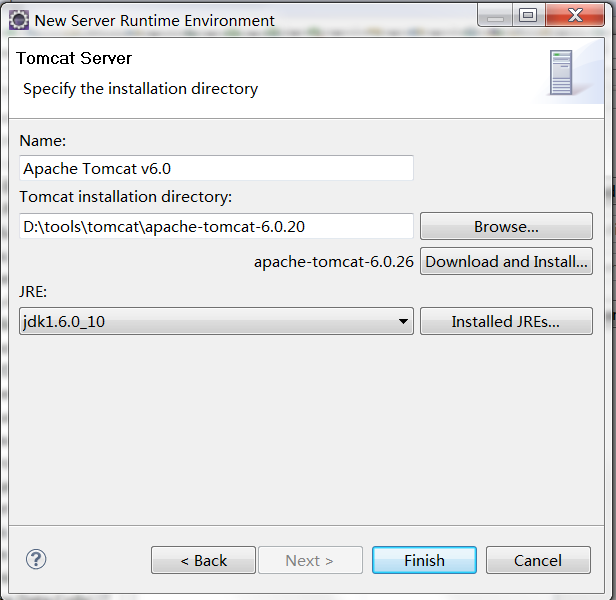
在eclipse的windows菜单下，选择Preferences，的窗口中，选择Server的Runtime Environment



在右边，点击“Add…”按钮

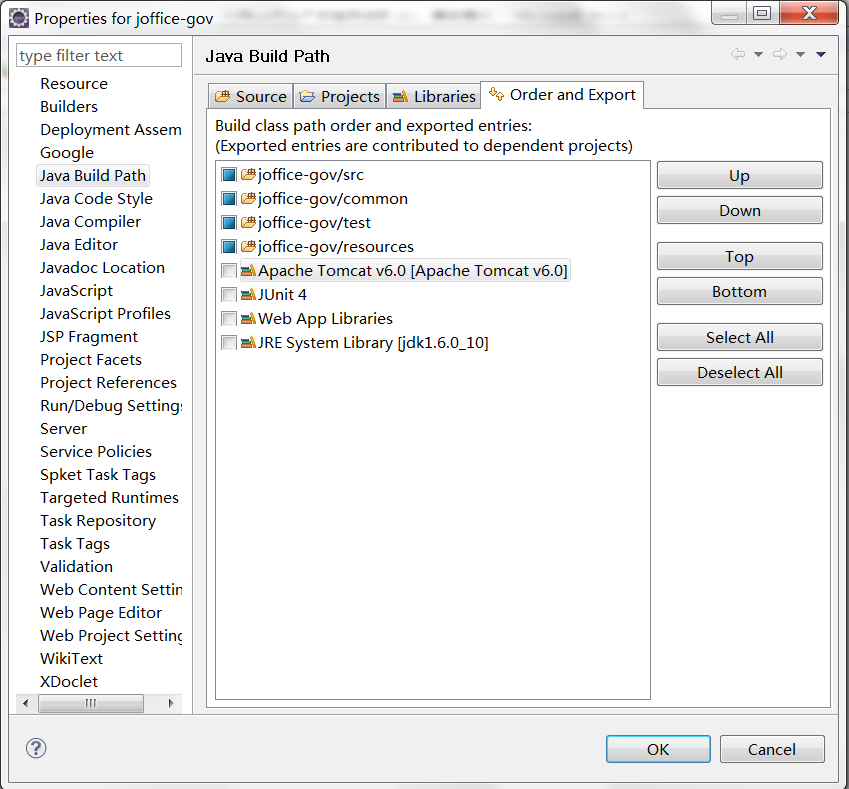
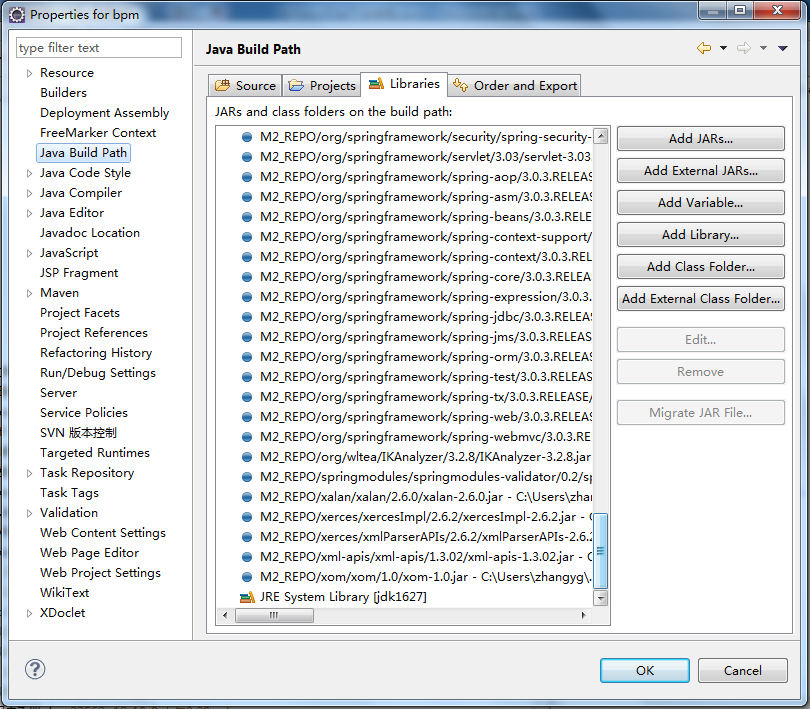


然后点击“Next”，



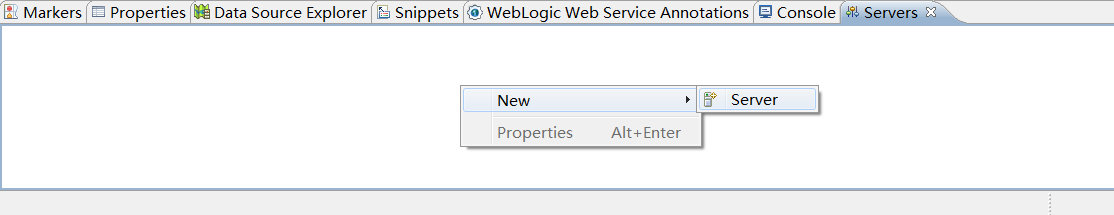
然后选择“Finish”

下一步检查整个项目的配置是否已经正确，如下所示：

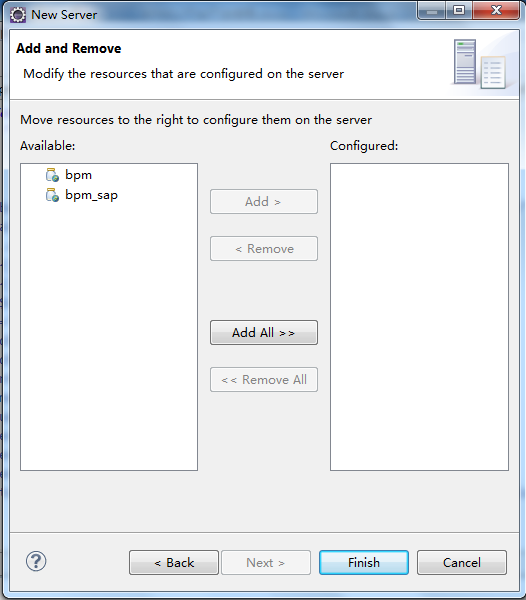


若其Jre System Libray出现“unbound”字样，表示没有配置好其运行这，删除该libray，然后添加新的则可。

#### 配置运行的WEB Server



在下面的视图中，右键，选择“New”🡪“Server”，然后选择“Apache Tomcat 6.0 ”，输入如下的信息，选择“Next”。



把左边的项目添加至右边，然后“Finish”。

现在可以启动项目了，不过eclipse这种方式在开发会有些同步的问题，即你在某些情况下修改某个文件，可能没有马上发布到Tomcat的目录去测试。这有时会为开发带来不便。所以可以把运行的目录指到开发目录则可。

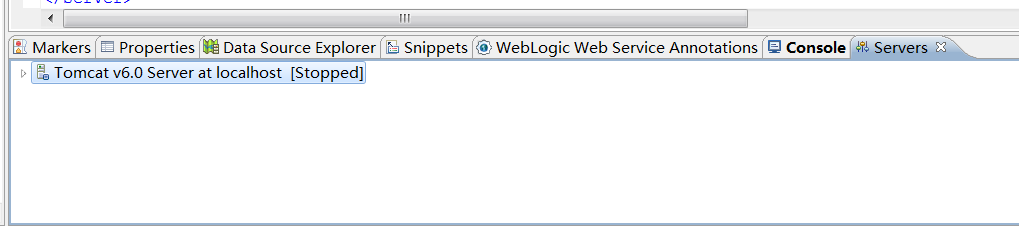
在workspace中的server目录下，修改我们刚才配置的Runtime配置下的server.xml文件，如下所示：



把<Context>标签改为以下则可,docBase则为项目的web目录。

<Context docBase="E:\work\bpm\src\main\webapp" path="/bpm"/>

然后启动项目则可



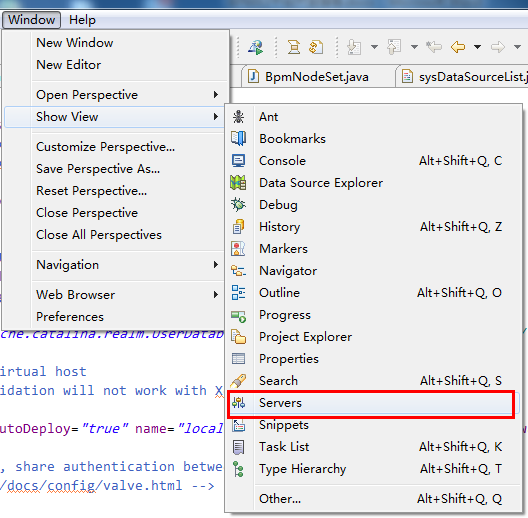
访问 http://localhost:8080/bpm/index.jsp

启动后，用admin账号，密码为1进行登录

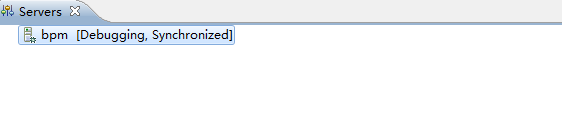
### 开发环境会经常遇到的问题

#### TOMCAT SERVER内存不足

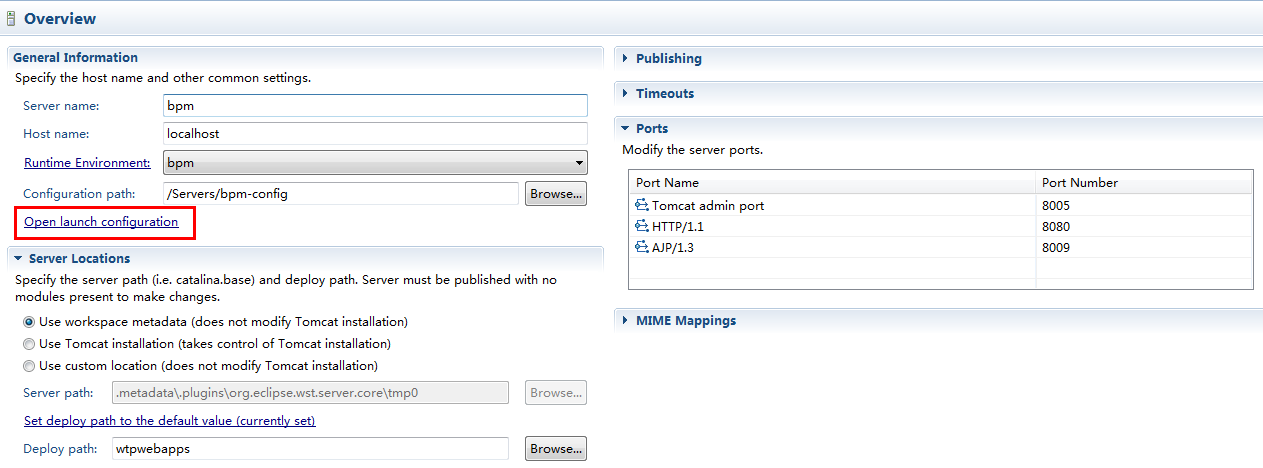
1. 打开server视图页面



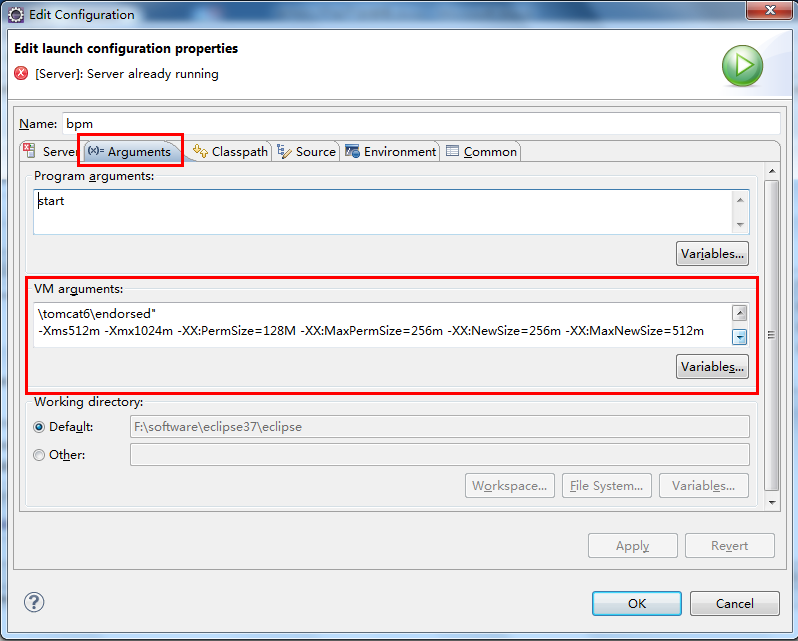
1. 双击对应的SERVER 。



1. 显示server配置



点击open launch configuaration



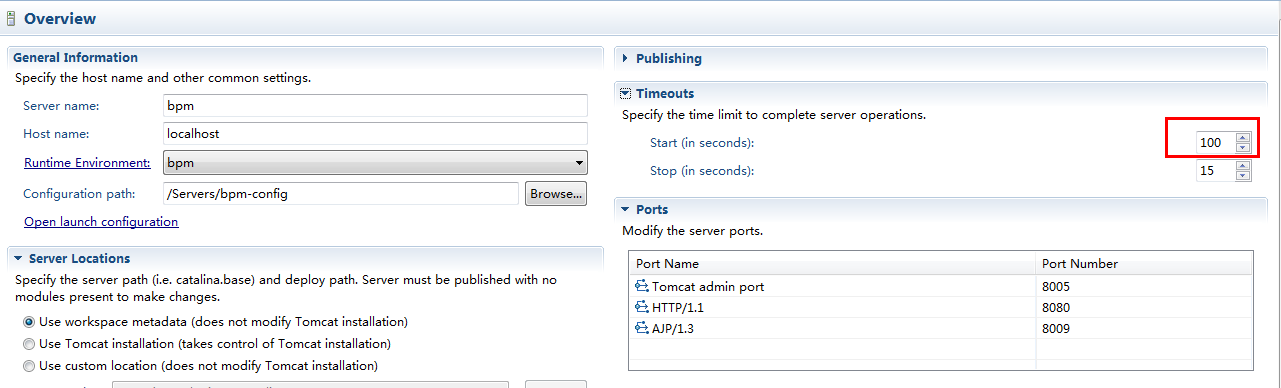
内存配置

-Xms512m -Xmx1024m -XX:PermSize=128M -XX:MaxPermSize=256m -XX:NewSize=256m -XX:MaxNewSize=512m

-Xms512m 为最小内存配置，这里一般需要配置到512m。

#### Tomcat超时

Tomcat启动的时候和机器性能有关系，有时默认的超时时间不够，在eclipse中tomcat启动超时，修改地方如下：

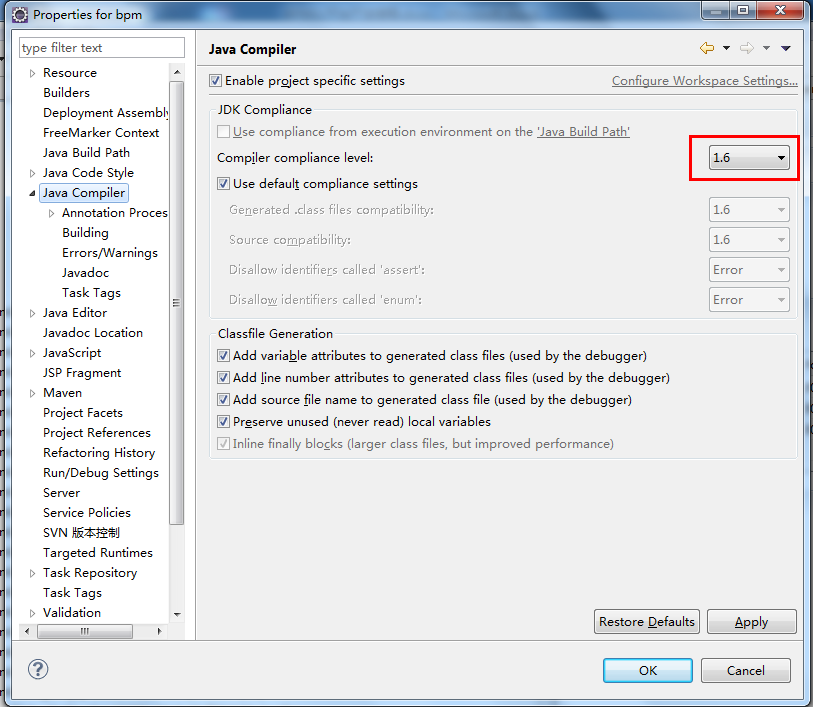


当前设置为100秒。

#### JAVA编译器问题

有时导入项目编译不对，需要检查java 编译器版本是否正确。

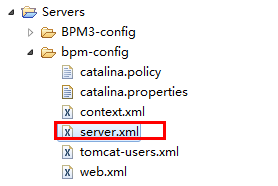
当前系统使用的是Jdk 1.6方式进行编译。



#### URL乱码问题

系统中有时后出现乱码，这是web服务器配置的问题，需要将url的编码修改为UTF-8 .

修改方法如下：



双击server.xml。

找到<Connector connectionTimeout="20000" port="8080" protocol="HTTP/1.1" redirectPort="8443"/>

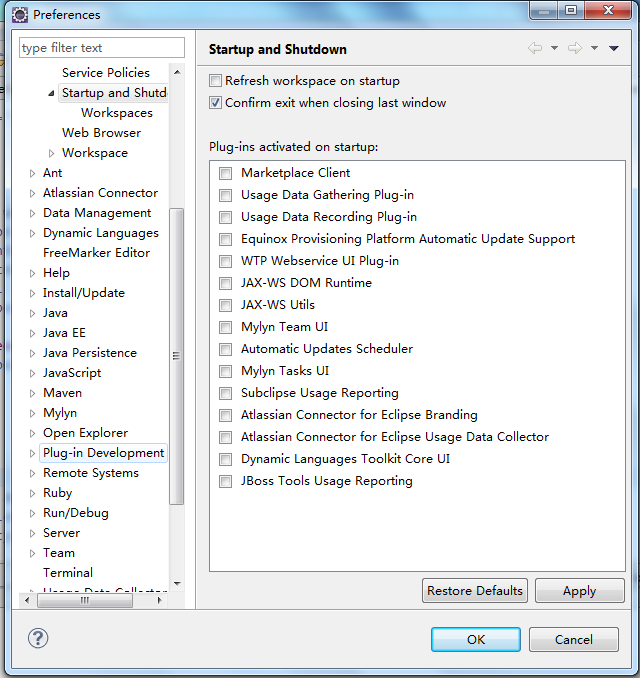
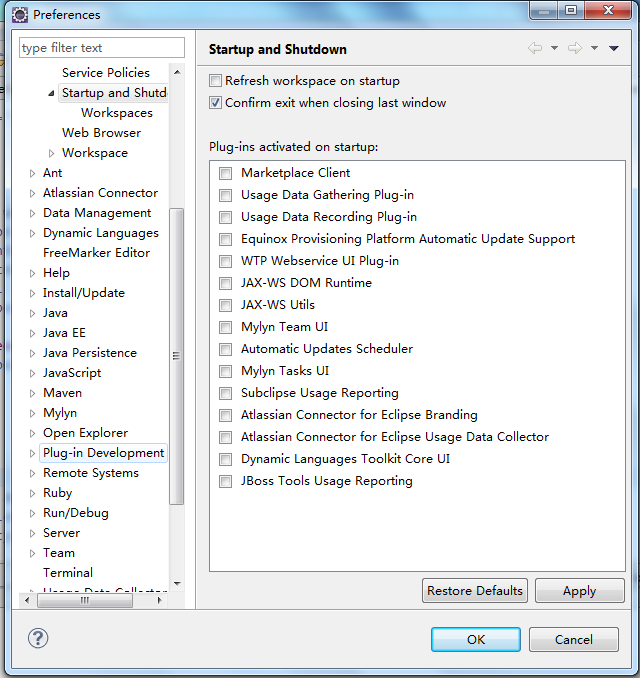
添加URIEncoding=”utf-8”

#### IDE太慢的问题

可以去掉一些验证。



去掉启动时加载的插件

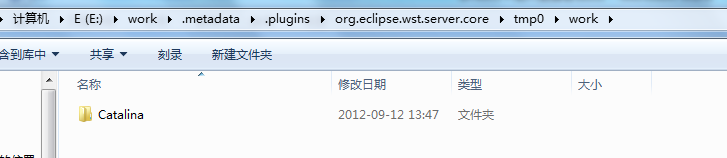
 

#### TOMCAT缓存的问题

有时候在修改代码的时候，在运行的时候不生效，排除浏览器缓存的等原因。问题可能处在tomcat的缓存问题。

删除缓存

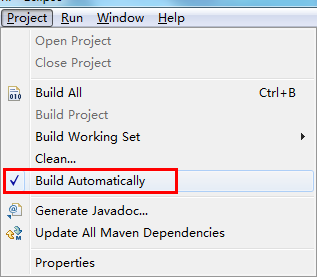




清除这个目录的缓存就可以了。

#### 编译不了问题

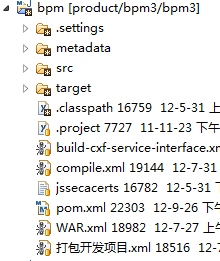
1. 检查项目有没有设置成自动编译。



1. 有时可能因为JAR文件被删除，而没有在项目应用中做删除。所以需要检查jar文件引用是否正确。

# 平台目录结构

项目导入完成后，可以在eclipse下看到整个系统源代码目录结构，如：



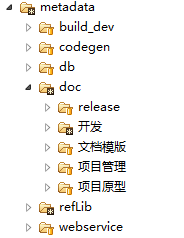
主要分成两个目录：

metadata:存放开发文档，数据库脚本，代码生成器等。

src:系统开发的源码。

本项目使用maven进行包的管理和编译。

## metadata目录



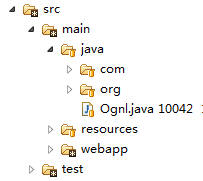
Codegen:代码生成器

Db:数据库文档

Doc:设计文档

## src 目录

源码目录



这个目录结构是maven的目录结构。

Main目录：

程序文件目录

其下有三个目录：

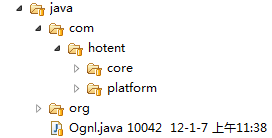
Java：java类目录

resources:配置文件目录

webapp:web应用目录

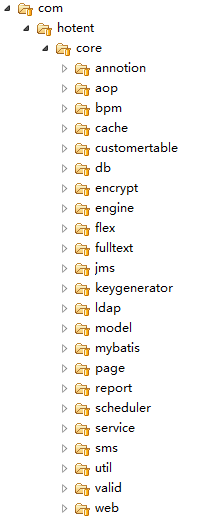
Test目录：测试用例目录

Java目录结构



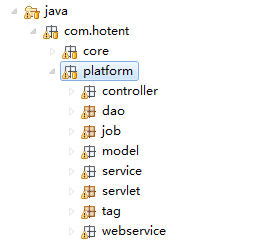
### core目录：

宏天基础类库



包括工作流，数据库基类，定时器，自定义数据库表，全文索引，JMS,web基类，短信操作，util 工具类等等。

### 平台文件目录：



开发目录分成以下几个主要的目录：

model目录：model类文件

dao目录：dao类文件

service目录：业务类文件

控制器目录：控制器类文件。

这些目录的关系为：



Dao层和Service层代码并没有象spring 2代码那样，都通过接口进行实现，spring3 使用cglib 动态代理实现事务管理，事务管理只是控制在业务层代码，所以需要控制事务的代码都写到service层代码。

另外：service和dao代码使用spring管理，并不像spring2 那样需要每个类都配置到spring 配置文件当中。

Service类使用@Service注解

Dao类使用@Repository注解

在app-resources.xml目录



Dao代码的编写：

Dao代码都继承com.hotent.core.db.BaseDao类。

继承这个类，开发人员不用每次都编写烦人的增查改删方法和分页方法。



Service 代码编写



Service代码继承自BaseService，BaseService继承自 GenericService类。

GenericService有一个抽象方法getEntityDao。这个方法在Service代码中实现。

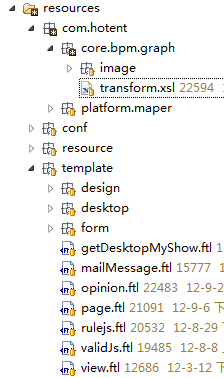
Service代码中使用其他的Dao，只需要定义私有变量。

例如：

@Resource

private SysDataSourceDao sysDataSourceDao;

## resources目录



这个目录包括平台所有的配置文件。

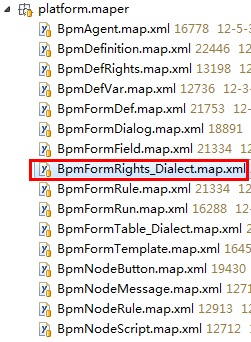
### core.bpm.graph目录

流程转换文件

transform.xsl 是将流程设计器代码转换成ACTIVITI流程文件的xsl文件。

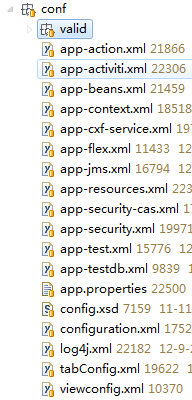
### platform.maper

com.hotent.platform.maper ： 这个目录存放了mybatis的映射文件。



文件名中使用 \_Dialect.map.xml 表示这个文件中有不同数据库的SQL语句的配置。

### conf目录

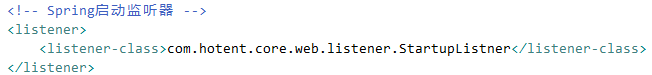


app-action.xml:

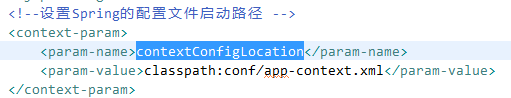
spring mvc 配置文件，在web.xml文件中配置。



Spring监听器使用的文件：



这个文件读取配置文件：



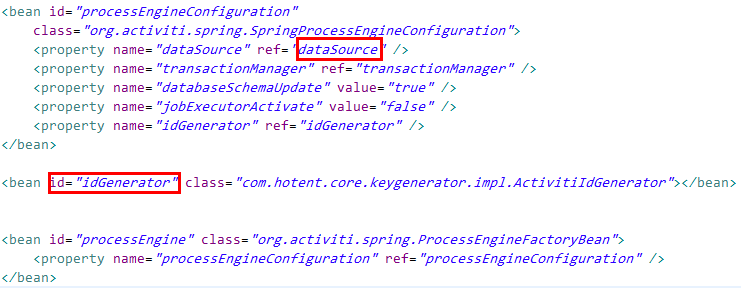
app-context.xml 文件使用模块化的方式引入配置文件：



app-resources.xml配置了：

dao，service 注解扫描，国际化资源，加载配置文件app.properties，数据库配置，事务配置，Freemaker配置，邮件配置，定时器配置，Ldap配置。

app-activiti.xml 是activiti工作流的配置。



1.配置了自定义的ID产生器。

2.配置数据源和事务管理和平台事务保持一致。

app-beans.xml 配置

这个配置了一些单独配置的spring实用类。

比如：AppUtil,会签完成类，脚本引擎类，外部调用脚本类，元数据库操作类。

在app-context.xml 引入app-security.xml，代表配置了普通spring security认证登陆；引入app-security-cas.xml代表配置了spring security单点登陆认证。

app-jms.xml

配置jms消息队列。

app-cxf-service.xml

配置cxf web服务。

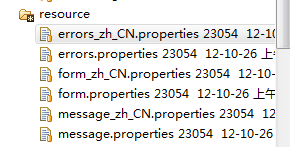
configuration.xml

配置系统访问的数据库类型。

tabConfig.xml

配置系统中功能模块用到的tab菜单。

### resource



errors\_zh\_CN.properties

用来统一设置表单页面或者一些规则判断的错误提示信息。

message\_zh\_CN.properties

用来统一设置保单提交时弹出的提示信息。

### template 模版目录

#### design子目录

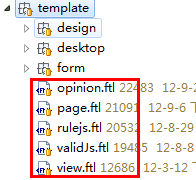
这个目录下面的文件为表单设计时用到的模版文件。

#### desktop 子目录

这个目录存放了桌面的模版文件。

#### form子目录

这个目录存放了通过表生成表单的表单模版文件。





opinion.ftl

表单意见显示模版，用于表单意见数据的显示。

page.ftl

分页导航模版，用于显示分页导航数据。

rulejs.ftl

规则模版，用于生成表单自定义规则文件(rule.js) 。

validJs.ftl

这个文件用于动态生成表单规则，用于表单数据验证。这个是用于先前验证的。之后开发的代码不再使用这个验证，新的验证使用CustomerValid.js 。

view.ftl

这个用于生成aspx视图页面模版，之前的一个项目中用到。系统中暂时没有用到。

Form/templates.xml

这个文件用来记录初始化模版的种类，同时在初始化时告诉系统一共有几个模板需要生成。

Form/ oneColumn.ftl、twoColumn.ftl、threeColumn.ftl、fourColumn.ftl

这4个文件是指利用主表生成表单时分别按照该模板生成1列、2列、3列、4列表格的表单。

Form/ inTable.ftl、inTable2.ftl、openWindow.ftl

这3个文件是利用子表生成明细表单的模板。

主表与子表生成表单的模版都可以自定义添加，只要在templates.xml添加一行模板记录（仿照已有的记录添加），然后再添加一份模板文件（比如oneColumn.ftl），

在系统中初始化模板就可以生效了。

## webapp目录

attachFiles

系统中所有通过上传上来的附件都保存在该目录。

commons

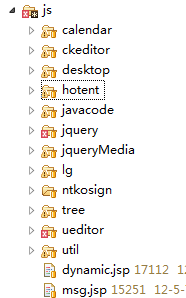
这个文件放置了一些公共的包含文件，错误处理文件等。

customForm.jsp用来引入表单用的css和js；

form.jsp用来被含有提交数据的表单页面引用；

get.jsp用来被含有列表显示的页面引用。

js 目录



Calendar 日历控件

Ckeditor 富文本框编辑控件

Desktop 桌面控件

Javacode java高亮代码

Hotent 系统使用的JS代码

Jquery

Lg ligerui控件

Tree ztree控件

Ueditor 百度编辑器

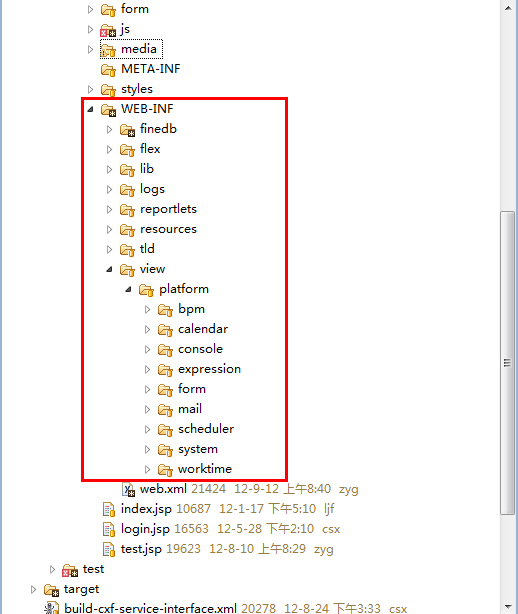
Util 实用JS工具类

Ntkosign office签章控件

Media目录

office控件，流程设计器，flex日历控件，flex文件上传控件。

Styles 样式表文件目录



WEB-INF目录

Lib 系统使用的jar目录。

View 视图目录，所有的视图jsp页面都放在这个目录中。

# 安全管理

## 概述

BPMX3 在用户安全管理这块有如下特性：

1. 支持多个子系统。

即用户可以在平台中创建多个子系统，子系统可以是部署在同一个应用中，可以是部署在同一个服务器的不同应用，也可以是不同服务器的不同应用。

如果是部署在不同服务器的应用可以使用CAS单点登录。

1. 基于角色的授权类型。

系统权限分配是基于角色的，角色分配权限资源。人和角色进行映射。

1. 授权不用编写代码。

授权是通过角色和资源做的，如果只是需要控制页面访问，不用编写任何代码。如果需要控制到页面的按钮，使用平台提供的权限标签也可以轻松实现对页面按钮权限的控制。

1. 支持CAS单点登录。

支持CAS单点登录，使用CAS作为服务器，解决多子系统部署的问题。

1. 支持AD用户同步和验证。

支持活动目录到本地用户系统的同步。AD用户验证还是统一到AD活动目录服务器进行验证。

1. 支持多维组织架构。

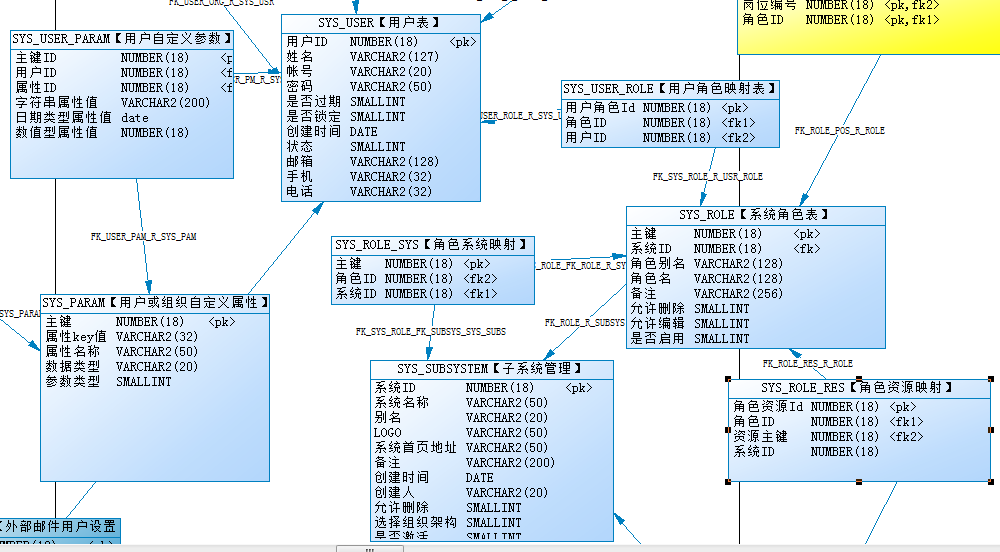
系统默认支持一个行政维度，用户可以通过自定义维度，实现自己的类似的组织架构。

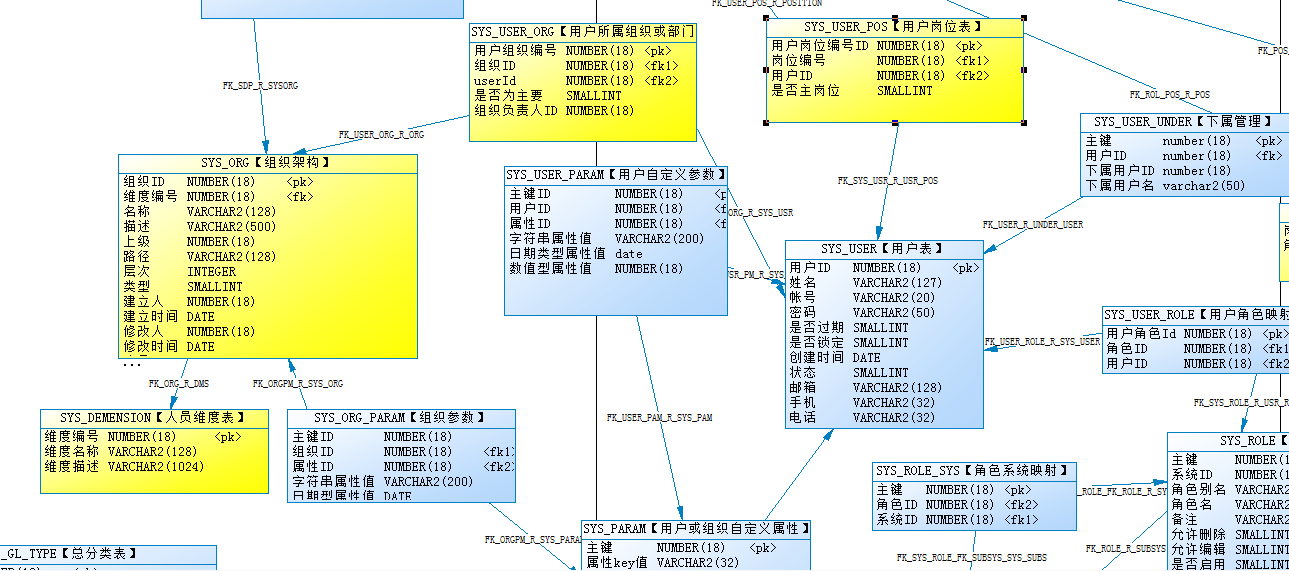
1. 支持用户岗位。

支持树形结构的用户岗位。用户岗位主要用来做为流程用户的条件。

## 系统实现

BPMX3版本，系统支持多维的系统的组织结构的管理方式，用户可以隶属于多个组织及部门下，并且可以用不同的身份进入系统。因此整个系统的用户及权限的表结构如下：





表说明：

* SYS\_USER

系统用户表，所有的系统的用户登录均记录在该表

* SYS\_SUBSYSTEM

子系统表

* SYS\_ROLE

用户角色表，所有的角色表，角色来平面，没有进行等级或继承处理，这个角色可以属于某个系统或全局。

* SYS\_ROLE\_SYS

角色和系统直接的关联。

* SYS\_RES

系统资源，每个子系统都有独立的资源数据。

资源作用：

1. 显示有当前用户在子系统中的菜单。
2. 控制URL权限。
3. 控制菜单按钮权限。

* SYS\_ROLE\_RES

角色和资源映射表。一个角色可以支持多个角色。

* SYS\_RESURL

资源和资源的URL表单映射关系。一个资源可以对应0到多个URL。

* SYS\_USER\_ROLE

用户和角色的映射，一个用户可以属于多个角色，一个角色下可以有多个用户。

* SYS\_ORG

系统把总公司、分公司、部门及其他用户分类归类为组织结构。

* SYS\_USER\_ORG

用户所属的组织,为多对多的关系。

* SYS\_USER\_UNDER

用户下属表，一个人可以多个下属。

* SYS\_POSITION

用户的岗位表

* SYS\_USER\_POS

用户和角色映射

* SYS\_DEMENSION

人员维度表。默认有一个行政维度。

* SYS\_PARAM

组织和人员自定义参数表。

* SYS\_USER\_PARAM

用户自定义参数数据。

## Spring Security的整合

Spring Security（以下简称为SS）其前身是大名鼎鼎的ACEGI，后来正式合并至Spring框架，成为Spring框架中一员，由于它支持安全策略比较多，并且集成容易，现在也越来越多的系统采用它来进行安全管理。本系统中也采用它的基于角色的管理策略，以实现系统灵活的安全管理要求。

### Spring Security核心概念

**Authentication (认证）对象**

其实就是一个可以通过Spring Security的认证的身份证明。如实现该接口的类UsernamePasswordAuthenticationToken， 表示可以通过username及password作为身份验证。

Authentication对象包含了

 Principal 标识是哪一个对象，可以认为是用户

 Credentials 信任的对象，如密码

 Authorities 权限的集合，在我们的系统中可以认为是角色的集合 (authorities要赋予给principal的)

**SecurityContextHolder**

是Spring Security的核心对象，是安全上下文的访问的入口。如取得当前的登录用户可以从该类中的相应的方法取得。该类中包含ThreadLocal私有属性用于存取SecurityContext, SecurityContext包含Authentication私有属性。如实现弹出窗口登录功能的时候，输入的用户名及密码并没有最终经过SPRING SECURITY的filter,那么如何使用得当前用户可以成功登录呢，其就是利用到这一点，通过该类拿到SecurityContext，然后设置一个认证的对象给它，SPRING SECURITY在看到该认证对象的时候，就会成功经过了身份的认证了。

其实也可以这样理解，为了处理Http请求间认证，Spring Security使用HttpSessionIntegrationFilter，HttpSessionIntegrationFilter用于在HttpSession存储Http请求间的SecurityContext。不过我们可以通过SecurityContextHolder去拿到这个SecurityContext

**AuthenticationManager**

通过Providers 验证 在当前 ContextHolder中的Authentication对象是否合法。

**AccessDecissionManager**

验证访问的资源是否授权

**Interceptors(拦截器)**

拦截器(如FilterSecurityInterceptor，JoinPoint，MethodSecurityInterceptor等)用于协调授权，认证等操作。

**GrantedAuthority(权限)**

是授予相应Principal的应用系统访问权限。

Spring Security是Spring中一个强大的安全管理框架，不过目前在我们系统中使用的仅是其中一部分的功能，则权限过滤安全检查的功能。

【说明】

*如果抛开这个框架，我们实现权限管理的时候，可能使用最多的方案还是使用Filter来进行过滤，在Filter里判断当前的用户是否为登录用户，若是登录用户，则看是否有权限访问当前的资源，若为未登录用户，则跳至登录页面。*

### 系统的安全管理方式

目前系统中已经实现了以下的方式

1. 登录时需要加上验证码
2. 所有的数据展示及访问页需要登录后才能访问
3. 用户的数据库密码存储时使用Sha-256的加密算法
4. 登录后的所有系统的访问URL均需要授权
5. 登录多少次失败后，可锁定IP，约２０分钟后才能自动解锁。（尚未加入，容易实现）

如果需要更强的安全管理策略，可以使用更为复杂的系统配置（如通过防火墙等手段）

权限设计目前是采用基于角色控制的方式，用户需要访问系统的资源，首先必须要授予一个角色，而该角色具有访问系统资源的权限的能力，也可以认为是权限的集合。因此，一个用户要访问系统的某个资源（如产品列表），则首先要授予一个能够访问产品列表资源的角色（如productAdmin）。只要任一个用户拥有了该角色，即可以访问该资源。

系统的安全涉及到两个不同的概念，认证和授权。前者是关于确认用户是否确实是他们所宣称的身份。用户进入系统的时候，首先要进行第一个操作就是进行身份认证，即Authentication。在系统中一般表现为用户用账号跟密码登录。如果都正确了，则可以登录系统。

【说明】在现实中你可以这样理解，员工在进入公司之前，需要进行身份的确认。身份确认通过后，则可以进入公司。进入公司后，并不代表可以随便进入公司的每个办公室。这时就需要每个看当前员工具有哪些角色，即授权。授权则是关于确认用户是否有允许执行一个特定的操作。如当前员工是总经理，则可以进入总经理办公室，并且可以进入普通员工的办公区域。是因为总经理已经授权可以出入这些地方。

在本系统中，权限表现为功能菜单及系统访问的URL。

因而用户、角色、权限之间的关系可以用如下的图描述：



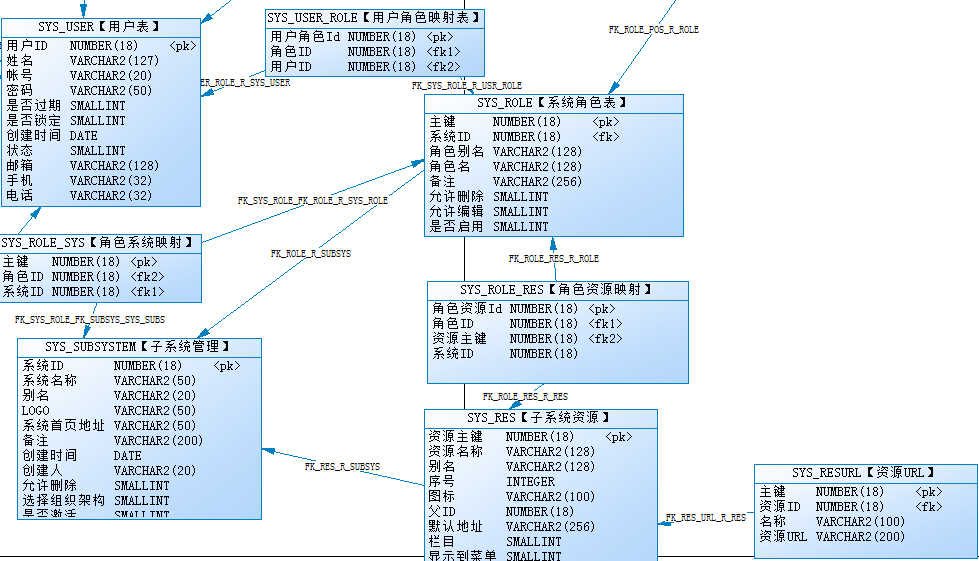
一个用户可以有多个角色，每个角色有多个功能菜单，每个功能菜单会对应多个系统访问的URL

### 整合Spring Security



#### 表设计

基于以上的思想，我们设计了以下表：



* SYS\_USER

系统用户表，所有的系统的用户登录均记录在该表

* SYS\_SUBSYSTEM

子系统表

* SYS\_ROLE

用户角色表，所有的角色表，角色来平面，没有进行等级或继承处理，这个角色可以属于某个系统或全局。

* SYS\_ROLE\_SYS

角色和系统直接的关联。

* SYS\_RES

系统资源，每个子系统都有独立的资源数据。

资源作用：

1. 显示有当前用户在子系统中的菜单。
2. 控制URL权限。
3. 控制菜单按钮权限。

* SYS\_ROLE\_RES

角色和资源映射表。一个角色可以支持多个角色。

* SYS\_RESURL

资源和资源的URL表单映射关系。一个资源可以对应0到多个URL。

* SYS\_USER\_ROLE

用户和角色的映射，一个用户可以属于多个角色，一个角色下可以有多个用户。

【说明】用户表sys\_user中的密码需要保存为密文，可以防止用户的密码泄露

#### 与Spring Security集成说明

目前我们需要扩展Spring Security的以下两部分功能

1. 身份认证

2. 授权

#### 身份认证

Spring Security是由一组的filter来进行统一的过滤，不同的filter进行相应的权限过滤功能。不过在Security跟Spring集成的过程中，其是由一个代理的类进行这些filter的统一管理。可以在web.xml中进行了查看，如下所示：



所有经过springSecurityFilterChain的url，都会转到DelegatingFilterProxy类的bean去处理。而该Bean在Spring Security 3.0中，已经内置于安全管理的缺省的配置当中，我们只需要把app-security.xml加入我们系统管理中来即可。如下：

而该类型的bean则在spring-sercurity.xml中进行了配置，如下所示：





【说明】

当用户登录时，会根据用户账号及密码进行身份验证，验证由authenticationManager来进，其会调用实现UserDetailsService接口实现类完成，在本系统，是由sysUserDao类来实现。



#### URL访问授权



而我们的用户及角色实体要成为安全框架识别的安全实体，需要相应实现不同的接口，如下所示：

系统用户类实现SysUser的UserDetails接口。

角色需要实现GranteAuthority的接口：

授权的管理是通过Filter来进行的，用户访问URL时，均需要经过Spring Security的URL进行授权。在本系统中，这个功能是通过com.hotent.core.web.filter.PermissionFilter来进行。

系统启动时，会把所有的权限以[角色—URL列表]的形式放置在一个静态的Map对象中（可参见代码com.hotent.core.web.security.HtSecurityMetadataSource中的loadRes()方法提供实现），用户访问系统的URL时，会取得URL对应的角色列表，同时获取当前用户的角色，判断用户的角色是否在URL对应的角色列表中,如果存在则表示有权限访问。

具体判断是否有权限访问的实现请参考：com.hotent.core.web.security.HtDecisionManager类的实现。

判断流程如下：

1.首先判断是否匿名访问，如果配置了匿名访问则直接通过。

2.判断用户是否已经登录，如果没有登录则返回没有权限页面。

3.如果是超级管理员直接通过。

4.页面的角色包括公开访问的角色直接通过。

5.判断是否有权限访问子系统。

6.判断页面是否有角色授权，如果当前用户有访问页面的角色则直接通过。

7.没有满足的条件，则抛出AccessDeniedException异常。



扩展说明：

在这里我们加入自己的Filter,当以上访问均没有通过认证时，其就会进入PermissionFilter，在此类中，我们根据当前用户的所拥有的角色来决定是否进入下一个Filter的处理，若没办法通过权限认证，则会抛出一个拒绝授权访问的错误，系统就会进入403页面。403页中写入响应的文件头“\_\_forbidden”，EXT展示层根据Connection取到的文件头而相应弹出相应的提示信息。

#### 按钮权限访问控制

上面说的是通过URL进行授权，本节介绍在系统如何实现页面按钮级别的权限。

按钮级别的需求很常见，比URL控制更具人性化。比如在页面上有一个删除链接，可以点击url删除某个数据，结果点击进去却被告知没有权限访问。这样的界面就不够友好，最好是不能访问，就直接灰显，让点击无效。这个时就按钮级别的权限就非常有用。

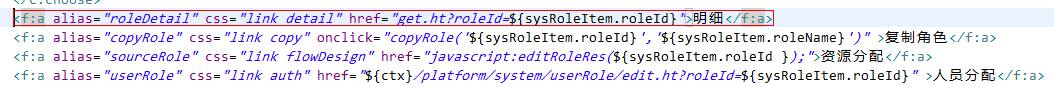
实现原理：

1. 每一个资源别名对应一个角色列表。
2. 用户登录后也有对应的角色列表。
3. 判断用户的角色是否在资源角色的列表中。
4. 使用自定义的JSTL标签实现。具体代码可以查看com.hotent.core.web.tag.AnchorTag。

在BPMX3平台中实现按钮级别权限非常简单。

下面就是步骤：

1. 定义资源和url资源是一起定义的，也可以不定义URL 。注意每个资源在子系统中都有一个独一无二的别名。
2. 在页面上需要控制权限加权限标签如下图：



和写链接标签基本一致，只是加了一个“<f:” 开头，增加了一个alias的属性，在<f:a></f:a>之间的不支持class等属性，支持css属性，这个别名就是控制连接访问权限的。

#### 菜单资源分配

将用户加入到指定的角色，对该角色分配相应的资源，那么菜单分配就可以了。这个都不再详细描述了。

# 二次开发及扩展

## 系统调试

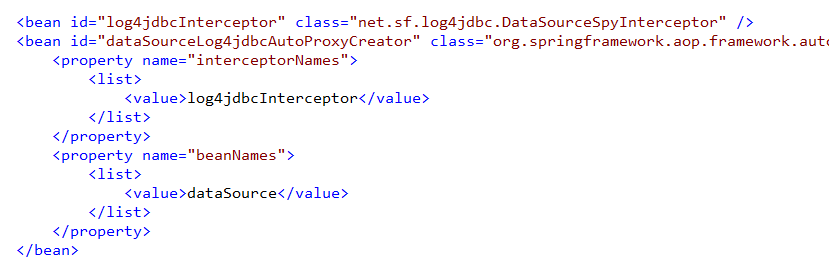
### 概要说明

我们在使用平台的时候，有时有些模块有问题，我们这个时候想知道在后台程序到底执行了那些sql语句，这样有利于排查问题，在平台中用户可以打开开关，由于启动调试后会影响到系统执行的性能。在调试完成后，需要及时关闭。

### 在平台中打开调试

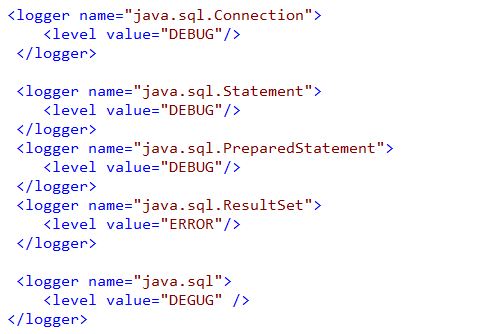
1. 找到app-resource.xml 。

找到以下代码：



把注释去掉。

1. 找到log4j.xml文件。

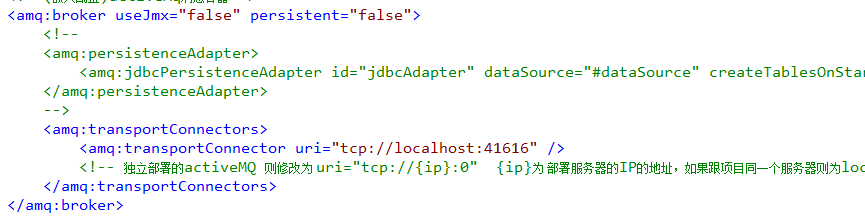


代码按照上面的配置修改。

1. 注意打开这个会带来一个副作用，会引起JMS序列化抛出事务界别报错，虽然不会影响启动，但总抛出错误不是一个好的体验。

因此我们可以暂时关闭jms序列化到数据库的配置。

找到app-jms.xml配置。



把persistent修改为false

注释掉amq:persistenceAdapter。

调试完成后修改回来。

## 工作流扩展

BPMX3 是基于ACTIVITI流程引擎扩展，在上面扩展了很多功能，使流程设计变得更加简单。

1. 是一个开放式、轻量级的开发平台
2. BPMX3多层系统架构
3. 基于模板代码生成
4. 丰富的报表展示
5. 跨数据库的数据访问支持
6. 简易的前端使用
7. 参照最新的BPMN2标准设计
8. 满足中国特色流程需求
9. 在线的可视化流程设计、可视化的流程监控
10. 完全独立的流程引擎服务，可与其他系统无缝集成
11. 基于流行的JAVA 流程引擎Activiti 5.10扩展
12. 灵活的在线流程定义
13. 灵活的嵌入式部署、独立部署
14. 灵活的在线表单定义及版本管理（支持按内表及外部表及视图生成表单）
15. 灵活的第三方表单集成
16. 开放式的流程体系
17. 灵活的组织结构支持
18. 流程的导入与导出
19. 子流程支持
20. 流程任务逐步回退处理(任务追回及多驳回 ）
21. 流程自由跳转
22. 流程跳转业务规则设置
23. 多级分发任务与任务汇总
24. 会签（并行与串行）、补签及投票处理
25. 流程版本变更管理
26. 流程催办（基于MQ短信及邮件设置）
27. 流程任务的工作日设置（用于统计流程任务执行效率）
28. 任务监控管理
29. 流程委办、转办

### 流程术语

#### 流程定义

流程定义又称流程模型，是用来描述业务过程的规定性文档，一个流程主要由一系列的活动和转移组成。流程定义需要遵从特定的语法规范。ACTIVITI支持BPMN2.0标准规范。流程定义有工作流引擎负责解释执行。

#### 流程实例

流程实例是在流程根据流程定义产生的实例，是实例化流程定义。我们说一条流程执行完毕，意思就是流程实例生命周期结束。

#### 任务执行人

具体任务所有者，这个任务有任务执行人执行。在流程数据库表中这个是ACT\_RU\_TASK 的ASSIGNEE\_字段.

#### 任务候选人

任务产生后，可以有多个人看到这个任务。这个时候需要先获取到任务再执行。数据存放在ACT\_RU\_IDENTITYLINK。

#### 会签（多实例用户任务）

一个任务需要有多个人员来完成,比如一个任务一笔经费审批，需要五个人同意，才能到财务审批，否则打回。这样的需求就可以使用会签任务来执行。对于会签可以设置会签规则。

如果不设置会签规则，那么流程将会执行完所有的分支。

#### 多实例

多实例是在流程中重复执行一个特定步骤。跟程序开发的概念相比，多实例和foreach机制类似。流程引擎根据一个给定的集合，循环执行某一个步骤，甚至一个完整的子流程。

多实例节点是一个普通的活动节点，加上一个multiInstanceLoopCharacteristics子元素。这样就会使一个节点在运行时执行多次，执行方式可以是串行或并行。

下面活动可以通过添加multiInstanceLoopCharacteristics变成多实例节点。

1. 用户任务。
2. 脚本任务。
3. Java节点。
4. Web服务节点。
5. 业务规则任务
6. 邮件任务
7. 人工任务
8. 内嵌子流程。
9. 外部子流程。

事件节点（开始，结束）和网关节点不能作为多实例节点。

产生子实例的excution，一般是主实例，会有下列的流程变量：

nrOfInstances:总共产生子实例的个数。

nrOfActiveInstances：活动实例的个数。也就是为结束的实例个数。串行多实例流程的活动实例一直都是1个。

nrOfCompletedInstances：已经完成的实例个数。

这些变量可以通过excution.getVariable (“变量名获取”);

另外还有一个子实例的本地变量。

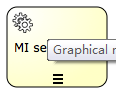
loopCounter：指明当前实例的执行次序。

在流程图中：

三条垂直的短线表示并行多实例任务。



三条水平的短线表示为串行多实例任务。



Xml定义

如果需要将一个普通的节点变成多实例节点。这个活动的xml必须有一个**multiInstanceLoopCharacteristics** 的子元素。



**isSequential**  属性表示多实例是使用串行还是并行的方式执行。

当进入多实例节点的时候，产生多少个实例，通过计算产生。

产生多个实例的方法有一下几种。

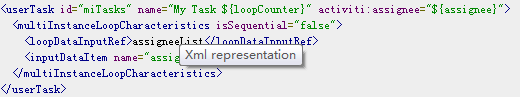
1. 直接指定产生实例的个数。



1. 使用表达式



1. 使用集合流程变量。



设定一个流程变量，这个变量为一个集合。使用loopDataInputRef指定集合变量的名称。inputDataItem 指定根据集合的元素产生流程变量的名称。

我们假定流程变量assigneeList 的集合值为 [csx,ray]。那么两个用户任务将被同时创建出来。那么每一个子实例都会产生用自己的本地变量assignee 。这个变量将会自动的赋给任务的执行人。

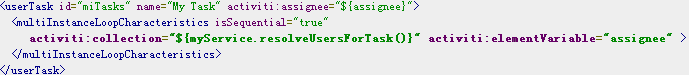
1. 使用集合表达式。

上面的方法起始有两个不好的地方。

* 1. 名称不方便记忆。
  2. BPMN2.0规范限制loopDataInputRef和inputDataItem不能使用表达式。

Activiti 为了解决上述问题，给**multiInstanceLoopCharacteristics** 元素添加collection and elementVariable属性。

例如：

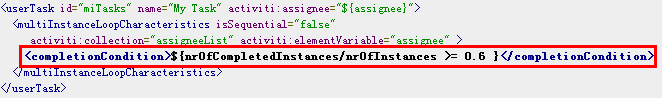


这个集合是有java代码提供一个方法提供多实例结合，另外子实例产生变量名为assignee的变量。这种方式最为灵活。

1. 多实例结束判断。

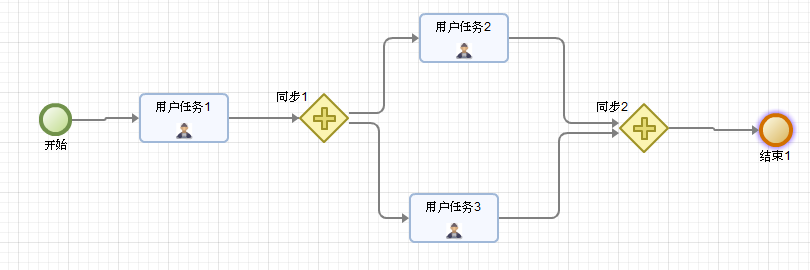
在一般情况下，多实例产生的子实例，需要全部完成，整个节点才会发生跳转。我们也可以指定一个表达式，每次在子例结束执行这个表达式。当这个表达式返回为true。流程引擎将还没执行完的子实例结束掉并且多实例节点结束。继续执行下面的流程。因此我们使用可以定义一个完成条件“completionCondition”。

示例：



这里判断的是完成的实例数和总的实例个数相除。这里是需要完成流程6成以上执行完成当前节点。不设置完成条件，那么所有的子实例就会全部执行完毕。

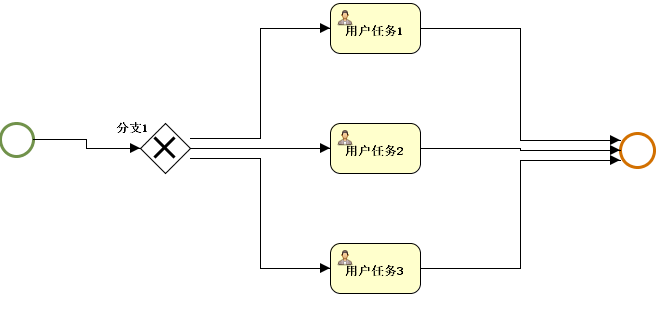
#### 同步网关



用来分组多个任务同时并行处理的情况，所有的同步任务必须都处理完全，流程才能往下走； 图标C:\Users\qing\AppData\Roaming\Tencent\Users\1004148805\QQ\WinTemp\RichOle\_DA%LM9$4DD`IOY6)YXZ{BB.jpg就是同步节点，都是一对同时出现C:\Users\qing\AppData\Roaming\Tencent\Users\1004148805\QQ\WinTemp\RichOle\`6VZEARF9@D4HKI]{5H$}$7.jpg，同步节点不需要设置任何条件；即是如果“用户任务1”完成，那么会同时产生两个分支(用户任务2和用户任务3)，并且需要这两个分支同时完成，流程才会继续往下流转。

#### 分支网关

这种网关的特点是需要判断分支的条件是否满足来决定流程的走向，如果**分支条件**满足或者**某条条件为空**，那么流程就会选择相应分支的执行，它只会选择其中的一条分支执行，如果分支条件都不满足，那么会抛出错误。



出现的情况：

1.如果三个条件都设置，并且只有一个条件成立，流程将往条件成立的分支跳；

2.如果三个条件都设置，而且有两个以上的条件成立；流程将往条件成立并且默认为第一个条件的分支跳；

3.如果第一个分支（用户任务1）不设置条件，无论其它分支设置的条件是否为真，流程都会默认第一个分支（为空条件）为真，流程将往不设条件的分支跳；

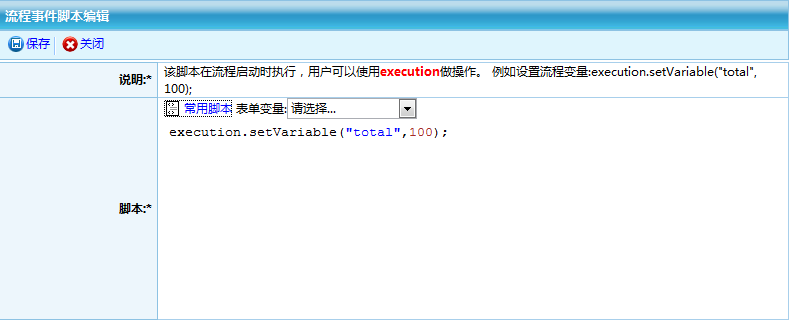
4.如果第一个分支（用户任务1）设置条件，第二个分支没设置条件，如果第一个分支成立，流程将走第一分支，否则走第二个分支，以此类推；

5.如果两个分支都不设置条件，流程将走默认的第一个分支（用户任务1）；

6.如果两个分支条件都不成立，流程将抛出异常。

测试方法如下：

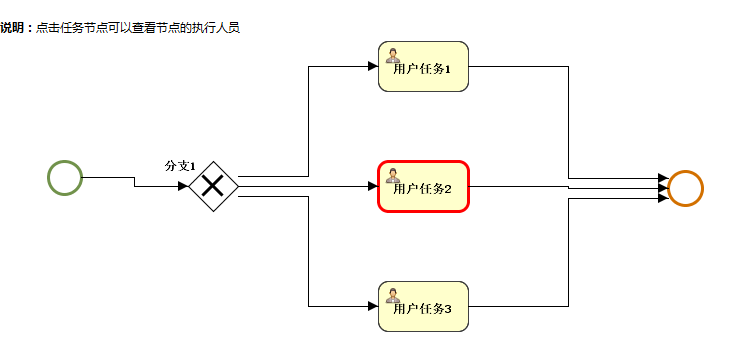
在起始节点事件写如下代码：



在网关编辑如下条件：



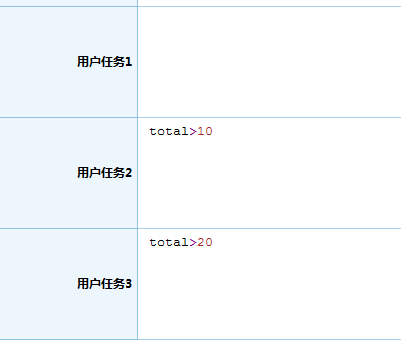
任务2和任务3都满足条件。实际上只会选择其中一条执行。



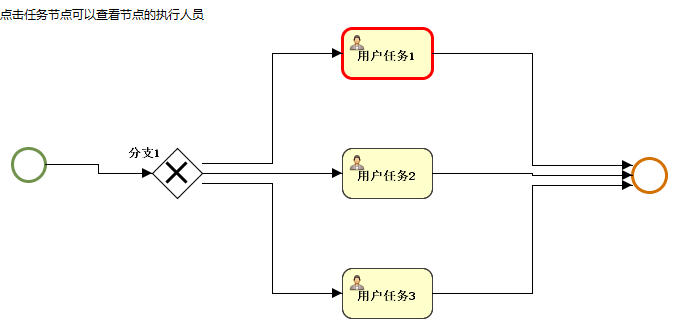
在这里流程引擎按照顺序选择了任务2 执行。

另外，去掉任务1的条件又会如何呢？

分支条件修改如下：



执行结果如下图：



流程引擎直接选择了“任务1”执行。

这个表明，流程引擎碰到分支网关，选择执行的路径规则是，只要有条件满足的分支或条件为空的分支，流程就会按照这条线路执行。

#### 条件网关

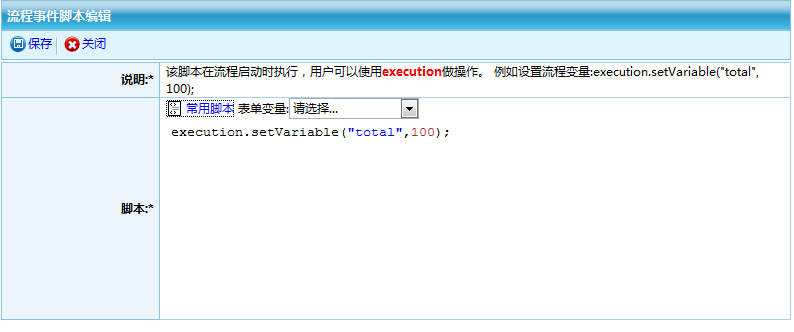
条件网关也就是条件同步可以看作是分支和并行网关的结合体。其特点。

1. 像分支网关一样可以支持在分支上定义流转条件。
2. 可以执行多条符合条件的分支，或者没有条件的分支。
3. 根据条件可以产生一个任务或者多个任务，并且所产生的任务必须全部执行完才会往下执行。
4. 条件同步节点C:\Users\qing\AppData\Roaming\Tencent\Users\1004148805\QQ\WinTemp\RichOle\3]2}BKM4Z{E1ZZHNWC}12Q2.jpg是一对同时出现。

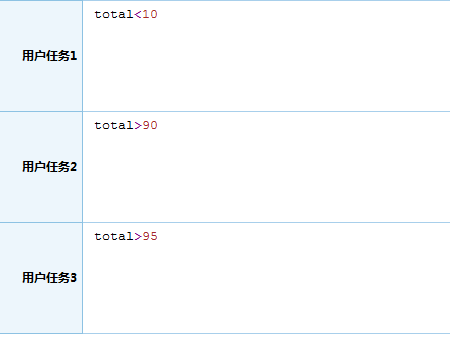
实例如下：



在开始节点设置流程变量：



网关条件如下



这个时候流程应该走“用户任务2”和”用户任务3” ，并且两个任务完成后才跳转到“用户任务4”。

出现的情况：

1.如果三个条件都设置，流程将根据设置的条件产生符合条件的任务（有几个成立就产生几个任务）；

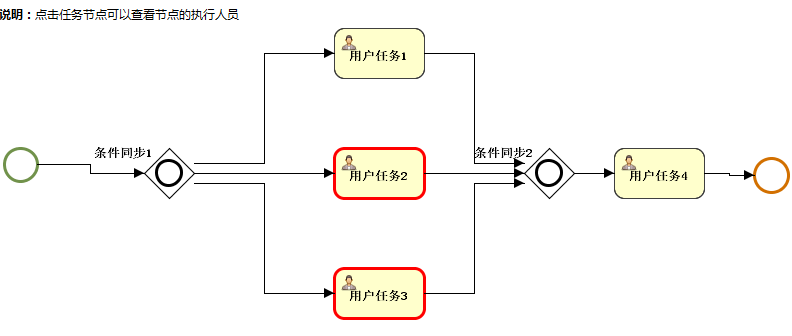
2.如果部分条件设置，部分条件没设置，流程将产生符合条件的任务和不设条件的任务；

3.如果所有条件都不设置，流程将产生所有的任务；

4.如果所有条件都设置，并且所有分支都不成立，流程将产生异常；

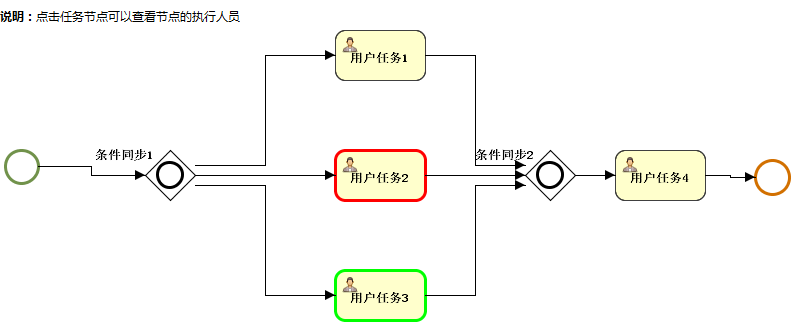
5.如果两个分支条件都不成立，流程将抛出异常。

现在启动流程：

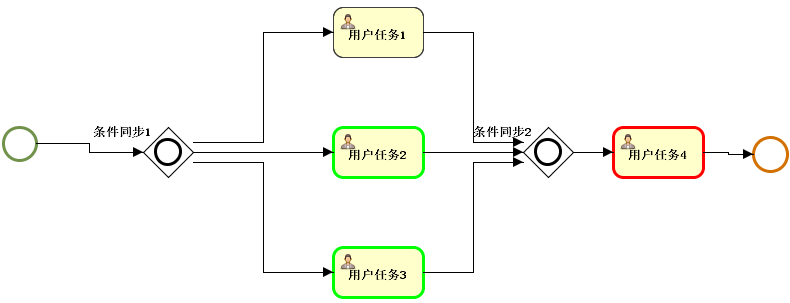


测试结果正确。

现在分别完成 2和3 。



任务2还在等待执行。



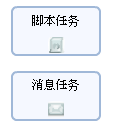
任务2完成后启动用户任务4。

#### 任务节点

任务节点是人工节点，流程执行到任务节点，会停留在这个活动节点上，

需要人工的干预才会向下执行。任务节点是我们在流程中用的最多的节点。

#### 自动节点



脚本任务和消息任务节点都是自动节点。即这样的节点不会在上面做停留。这样的节点可以用来做，消息发送，或者流程结束做一些数据处理。

#### 流程代理

流程代理是如果用户离开，可以将自己的任务代理给别人执行。执行时相当于本人执行。

#### 多实例节点

Bpmx3 支持多实例节点，多实例意思就是会在同一个节点执行多次，多实例可以分为串行和并行。

1. 串行

串行是产生一个任务执行完成后，会产生出下一个活动实例执行。串行一次只有一个活动实例。

1. 并行

并行是一次性产生多个活动实例。

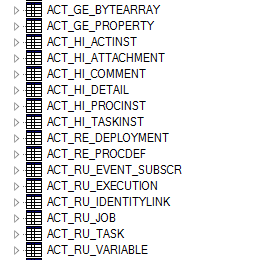
在BPMX3 系统支持多实例的有：

1. 会签任务。
2. 内嵌子流程。
3. 外部子流程。

### 数据库表介绍

数据库表分为两部分：

#### Activiti的数据库表



有兴趣的可以研究一下他的表结构：

下面简单介绍一下这些表：

Act\_GE\_BYTEARRAY

这个表存放流程图数据，流程文件数据和需要序列化的流程变量数据。

ACT\_GE\_PROPERTY

这个表的数据和我们的应用没什么关联，可以不用管。

ACT\_HI 开头的表为流程历史表。

ACT\_RE\_DEPLOYMENT

流程发布记录表。

ACT\_RE\_PROCDEF 流程定义数据表。

记录了流程定义数据，版本数据。

ACT\_RU\_ 表示流程运行时表

ACT\_RU\_EXECUTION

流程运行记录，记录流程执行情况。

ACT\_RU\_IDENTITYLINK

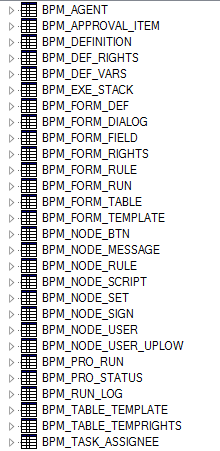
流程任务候选人表。

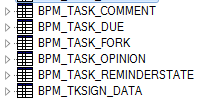
ACT\_RU\_TASK

通过任务节点产生流程任务列表。

ACT\_RU\_VARIABLE

#### 流程扩展数据表





BPMX3 扩展数据库表。

sys\_user\_agent，bpm\_agent

流程常用语

BPM\_APPROVAL\_ITEM

流程定义

BPM\_DEFINITION，记录流程版本数据，及一些扩展的数据。

流程定义授权

BPM\_DEF\_RIGHTS

流程变量

BPM\_DEF\_VARS 这个表现在没有使用到。

BPM\_EXE\_STACK

流程执行堆栈

BPM\_FORM\_DIALOG

自定义对话框。

BPM\_FORM\_FIELD

自定义表单字段。

BPM\_FORM\_RIGHTS

自定义表单权限控制。

BPM\_FORM\_RULE

表单自定义验证规则。

BPM\_FORM\_RUN

表单运行时，主要是为了实现表单的不同版本。

BPM\_FORM\_TABLE

自定义表的表定义

BPM\_FORM\_TEMPLATE

自定义表的表单模版。

BPM\_NODE\_BTN

表单自定义按钮。可以自定义启动流程按钮和流程执行下一步的按钮。

BPM\_NODE\_MESSAGE

节点消息

SYS\_MESSAGE

消息数据。

BPM\_NODE\_RULE

流程节点跳转规则。

BPM\_NODE\_SCRIPT

节点执行脚本。

BPM\_NODE\_SET

表单节点设置情况，包括流程节点的表单设置数据等等。

BPM\_NODE\_SIGN

会签节点会签规则的设置。

BPM\_NODE\_USER

流程节点人员设置。

BPM\_NODE\_USER\_UPLOW

流程节点上下级人员设置。

BPM\_PRO\_RUN

流程实例运行情况。

BPM\_PRO\_STATUS

流程节点状态

BPM\_RUN\_LOG

流程执行日志。

BPM\_TABLE\_TEMPLATE

流程表单模版，用于显示自定义表列表数据。

BPM\_TABLE\_TEMPRIGHTS

流程表单模版权限定义。

BPM\_TASK\_COMMENT

流程任务评论，暂时未用。

BPM\_TASK\_DUE

流程任务催办信息设置。

BPM\_TASK\_FORK

流程分发执行情况记录。

BPM\_TASK\_OPINION

流程任务意见。

BPM\_TASK\_REMINDERSTATE

催办提醒记录情况。

BPM\_TKSIGN\_DATA

流程会签数据。

### 脚本的编写方法

在平台中到处很多地方都使用到了脚本。这样增加了平台的扩展性，可以自己通过脚本进行扩展。但是如何编写脚本，新手可能都不大容易入手。下面介绍一下如何在平台下面编写脚本。

脚本的使用范围

1. 流程表单
2. 流程启动事件脚本
3. 流程结束事件脚本
4. 任务启动监听器
5. 任务完成监听器
6. 分支条件
7. 事件脚本
8. 流程规则脚本

脚本编写的方法

1. 脚本使用的语言为groovy，我们就使用平时使用的java代码就可以了。
2. 在脚本引擎中直接注入了 系统的所有的service和IScript 接口实现类的对象实例。所以在脚本中可以直接使用系统中的service接口和IScript对象。

在脚本中引用service，service **类名首字母小写**。然后调用相应的方法即可。

1. 如果需要使用到对象类。

可以使用import语句。

比如系统中有对象 com.hotent.platform.system.Person 。

就这样使用。

import com.hotent.platform.system.Person;

如果需要返回返回的，使用return语句进行返回。

1. 获取spring管控的类。

其实在脚本我们自动注入了service和IScript的实例对象。这样如果我们希望获取service和IScript之外的类，那么我们怎么实现呢？

例如：需要获取dao中的SysUserDao实例对象，我们可以写如下代码。

import com.hotent.core.util.AppUtil;

import com.hotent.platform.dao.system.SysUserDao;

SysUserDao dao=( SysUserDao )AppUtil.getBean(SysUserDao.class);

1. 编写代码建议

在编写代码框中尽量少写代码,也就是说编写的自定义函数时将功能都封装到java代码中,这样可以保证代码出错的可能，调试起来也相对的简单。

### 自定义脚本函数开发。

自定义脚本开发函数是基于java开发的，需要实现IScript接口，这个类被配置到spring容器中，在启动的时候注入到groovy脚本引擎当中。





我们也可以开发自己的脚本函数只要这个类继承了IScript就可以。开发完成后把这个类配置到Spring容器中进行管理。

自定义函数脚本在平台中使用相当广泛,主要使用在：

1.流程表单

2.流程启动事件脚本

3.流程结束事件脚本

4.任务启动监听器

5.任务完成监听器

6.分支条件

7.事件脚本

8.流程规则脚本

开发方式相当简单：

实例：



这样这个函数就可以在比如流程启动，结束，任务启动，结束，流程节点脚本中使用了。

调用方法很简单

类名（第一个字母小写） +“.” +方法名

例如:

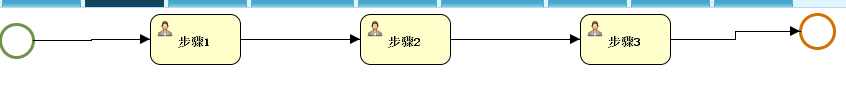
Long userId= scriptImpl.getCurrentUserId();

### 变量作用范围

#### 简单流程测试

这个测试，流程为顺序流程，没有多实例，并行执行和子流程的执行情况。

测试流程如下



在步骤一的启动事件中设置流程变量，代码如下：

task.setVariableLocal("vars","ray");  
task.setVariable("global","ray");  
task.getExecution().setVariableLocal("excutionLocal", "abc");  
task.getExecution().setVariable("excutionGlobal", "abc");

在步骤一的完成监听事件中写如下代码：

System.out.println(task.getVariableLocal("vars"));  
System.out.println(task.getVariable("global"));  
System.out.println(task.getExecution().getVariableLocal("excutionLocal"));  
System.out.println(task.getExecution().getVariable("excutionGlobal"));

输出结果：

ray

ray

abc

abc

这个说明局部变量在完成代码事件中依然可以获取到。

在步骤二的起始监听事件中

System.out.println(task.getVariableLocal("vars"));  
System.out.println(task.getVariable("global"));  
System.out.println(task.getExecution().getVariableLocal("excutionLocal"));  
System.out.println(task.getExecution().getVariable("excutionGlobal"));

分别输出：

null

ray

abc

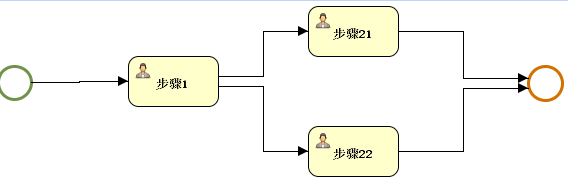
abc

这个测试说明，excution设置变量和本地变量在这个流程中是没有区别的。

Task设置流程变量 一个是全局的，一个是本地变量，本地变量在任务发生转移后变量被删除。

#### 并行测试

测试流程如下：



在步骤1起始事件监听器中设置代码如下：

task.getExecution().setVariableLocal("excutionLocal", "local");  
task.getExecution().setVariable("excutionGlobal", "global");

在步骤21：

System.out.println(task.getExecution().getVariableLocal("excutionLocal"));

System.out.println(task.getExecution().getVariable("excutionGlobal"));

步骤22：

System.out.println(task.getExecution().getVariableLocal("excutionLocal"));

System.out.println(task.getExecution().getVariable("excutionGlobal"));

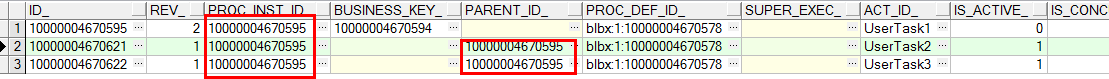
先执行步骤1：



在变量表中产生变量如上图所示。其中executionId和流程实例Id一致。

在执行下一步的时候

这个时候ACT\_RU\_EXECTION 表中的记录如下:



这个时候，流程同时产生两个子执行实例，主线的流程实例状态为非活动，新生成的excution的两条执行实例为活动状态。

这个在“步骤21”，“步骤22”节点创建任务的事件中执行上面的代码，结果分别为：

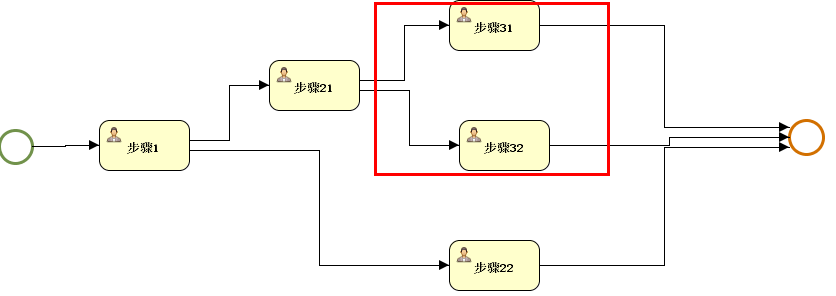
null

global

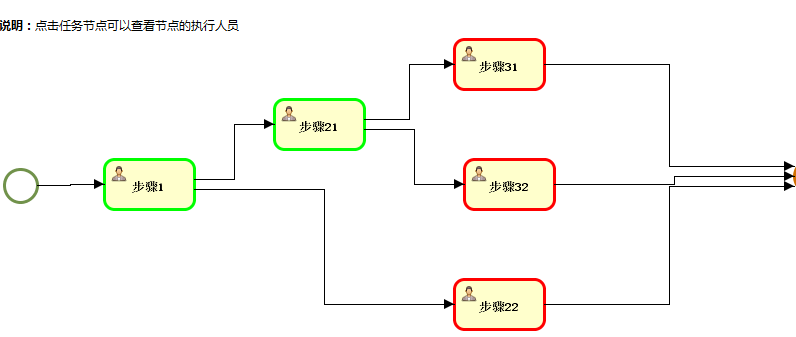
这个表示通过子的excution获取本地变量为空，获取全局变量正确。意思就是之前设置的本地变量为主实例的流程本地变量，在子实例是获取不到的。

#### 多级并行测试

流程图如上：

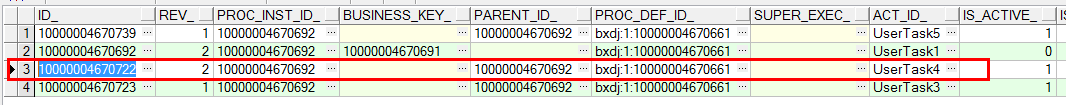


在执行步骤21的时候，流程图如下：



这里有4个excution分别为：

流程主实例，和三个子活动。

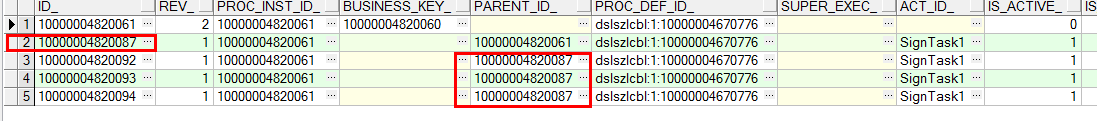


这个表名excution的local变量只和当前的excution相关。全局的变量这个流程都是可以获取到的。

#### 多实例测试



多实例节点也会产出多个子实例出来。



执行实例包括：

1. 主线程
2. 线程执行到多实例节点后，会在会签节点产生一个实例出来。
3. 这个多实例节点会根据选择人，分别产生出一条执行实例。

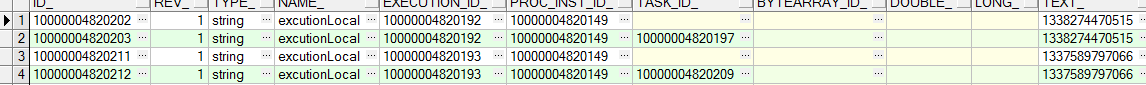
数据库可以看上图的数据，查看parent\_id\_, 流程实例始终只有一个。

这样我们可以写如下的代码，给每个excution设置一个流程变量。

String assignee=task.getAssignee();  
task.getExecution().setVariableLocal("excutionLocal", assignee);  
task.setVariableLocal("excutionLocal", assignee);

这样就正对每条子线线程私有的流程变量。

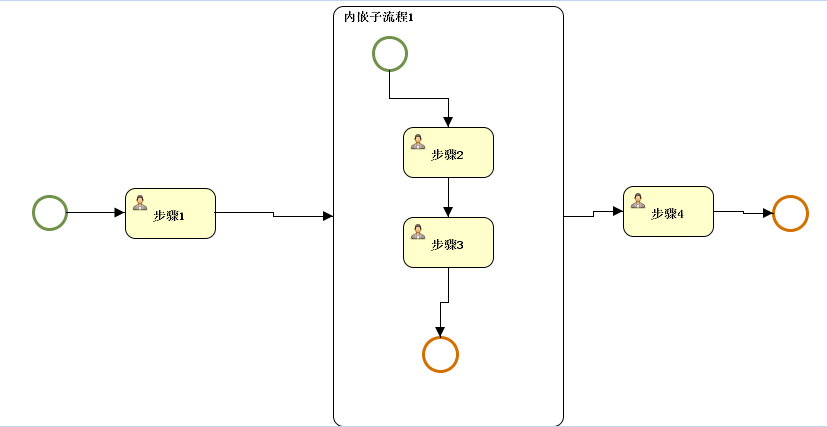
在这里 使用execution和task设置本地变量效果是一样的。



并且变量名称相同也不会冲突。

#### 子流程变量设置

子流程如下



步骤1

设置任务开始监听事件代码如下

task.getExecution().setVariableLocal("excutionLocal", "local");  
task.getExecution().setVariable("excutionGlobal", "global");

步骤2

设置任务开始监听事件代码如下:

在子流程中中设置全局的流程变量：

task.getExecution().setVariableLocal("subLocal", "subLocal");  
task.getExecution().setVariable("subGlobal", "subGlobal");  
System.out.println(task.getExecution().getVariableLocal("excutionLocal"));  
System.out.println(task.getExecution().getVariable("excutionGlobal"));

输出结果：

null

global  
步骤3：

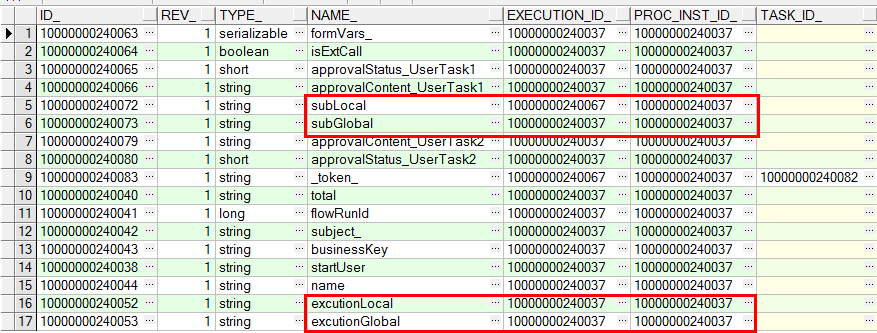
System.out.println(task.getExecution().getVariableLocal("subLocal"));  
System.out.println(task.getExecution().getVariable("subGlobal"));  
System.out.println(task.getExecution().getVariableLocal("excutionLocal"));  
System.out.println(task.getExecution().getVariable("excutionGlobal"));

Act\_ru\_execution



子流程产生了一个子实例。

流程变量如下：



输出结果：

subLocal

subGlobal

null

global  
步骤4：

System.out.println(task.getExecution().getVariableLocal("subLocal"));  
System.out.println(task.getExecution().getVariable("subGlobal));  
System.out.println(task.getExecution().getVariableLocal("excutionLocal"));  
System.out.println(task.getExecution().getVariable("excutionGlobal"));

输出结果：

null

subGlobal

local

global

在外部流程调用：

设置excution的全局变量和局部变量。

在子流程获取外部流程的局部变量为空。

在子流程获取外部的全局变量是可以的。

在子流程中设置全局流程变量，影响全局流程变量。

出了子流程后。

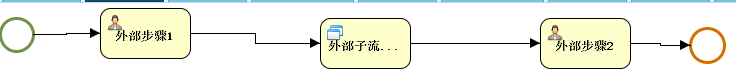
获取子流程局部变量为空。

获取子流程设置的全局变量可以获取到。

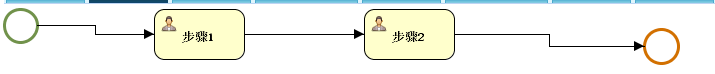
获取前面设置的全局变量可以获取到。

获取前面设置局部变量可以获取到。

#### 外部子流程



父流程



子流程

外部步骤1

节点开始监听器

task.getExecution().setVariableLocal("excutionLocal", "local");  
task.getExecution().setVariable("excutionGlobal", "global");

子流程步骤1

节点开始监听器

task.getExecution().setVariableLocal("subLocal", "subLocal");  
task.getExecution().setVariable("subGlobal", "subGlobal");  
System.out.println(task.getExecution().getVariableLocal("excutionLocal"));  
System.out.println(task.getExecution().getVariable("excutionGlobal"));

子流程步骤2

System.out.println(task.getExecution().getVariableLocal("subLocal"));  
System.out.println(task.getExecution().getVariable("subGlobal"));  
System.out.println(task.getExecution().getVariableLocal("excutionLocal"));  
System.out.println(task.getExecution().getVariable("excutionGlobal"));

外部和内部子流程变量完全是分离的。

子流程不能直接获取到外部流程的流程变量。

外部也不能直接获取内部子流程变量。

需要传递的话，需要通过变量拷贝。

#### 总结

execution 设置流程变量可以设置全局的或者局部的

1. 如果流程没有子实例，全局变量和本地变量作用可以认为是一致的。
2. 产生了子实例的情况，那么子实例需要有自己的私有变量，那么子实例可以通过execution.setVariableLocal(“变量名”,”变量值”);设置子实例变量。
3. 全局变量在流程整个生命周期中都有效。
4. Execution本地变量可以理解为在一条线上有效，如果起了子流程，那么子流程可以有自己的本地变量。

task 设置流程变量可以设置全局的或者局部的

1. 全局的在整个流程中都可以获取到。
2. 局部变量只在某个具体的活动中有效，离开某个活动任务流程变量就会被删除掉。

### 流程事件脚本编写

任务节点生命周期，下图是流程执行事件顺序。



1. 流程启动时，会触发流程启动事件监听器StartEventListener。
2. 如果第一个节点为任务节点。
3. 那么将会调用TaskCreateListener监听器。
4. 产生用户任务。
5. 在流程表单中点击，执行下一步的按钮。
6. 这个时候会触发TaskCompleteListener 监听器。
7. 如果下个节点是任务节点，则会触发StartEventListener监听器。

流程事件脚本编写

1. 开始结束事件脚本。

在脚本中可以使用的流程内置对象为 **“execution”** ,这个变量为ExecutionEntity类的实例。

1. 任务执行事件脚本

任务监听器分为三种：

启动监听器，人员分配监听器，结束任务监听器

在脚本中可以使用的流程内置对象为 **“task”** ,这个变量为TaskEntity类的实例。

task.setVariable(“abc”,”11”);

task.setVariableLocal(“name”,”11”);

### 分支节点条件编写

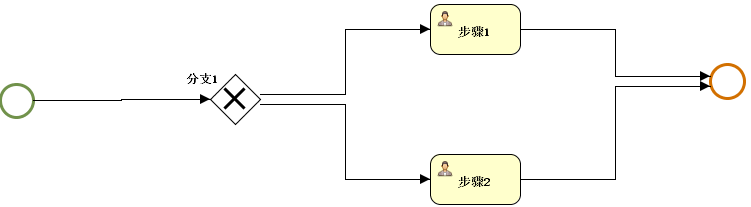
分支节点的条件即为条件表达式。条件表达式为JUEL表达式。

条件表达可以使用

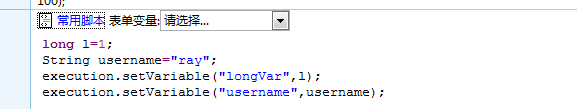
&& 并

|| 或

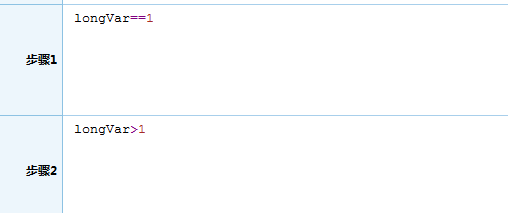
变量为long型



在开始事件节点设置如下脚本

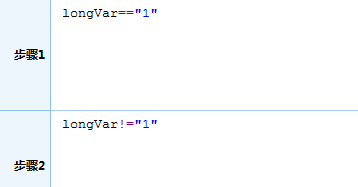


在分支节点写如下条件

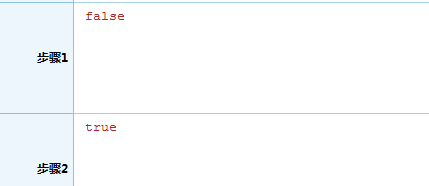


这个流程走步骤1

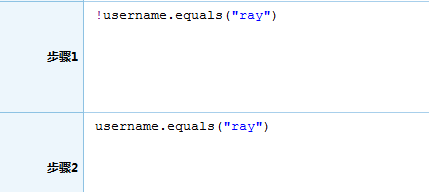
修改条件写法如下



如此写也会走步骤1。



这样写会走步骤2



如此写也可以会走步骤2

注意不能使用以下方式

username.indexOf(“ray”)==-1 这种方式，主要是应为indexOf这个函数在JDK中有多种实现方式。

除了上面写法。这个还支持使用java函数实现的方法。这样实现就可以实现更为复杂的条件判断。

前提是这个类需要使用spring进行管理。

在bpmx3平台中可以使用类的方法包括两类。

1. 实现了IScript接口的类。
2. 继承了BaseService类的类。

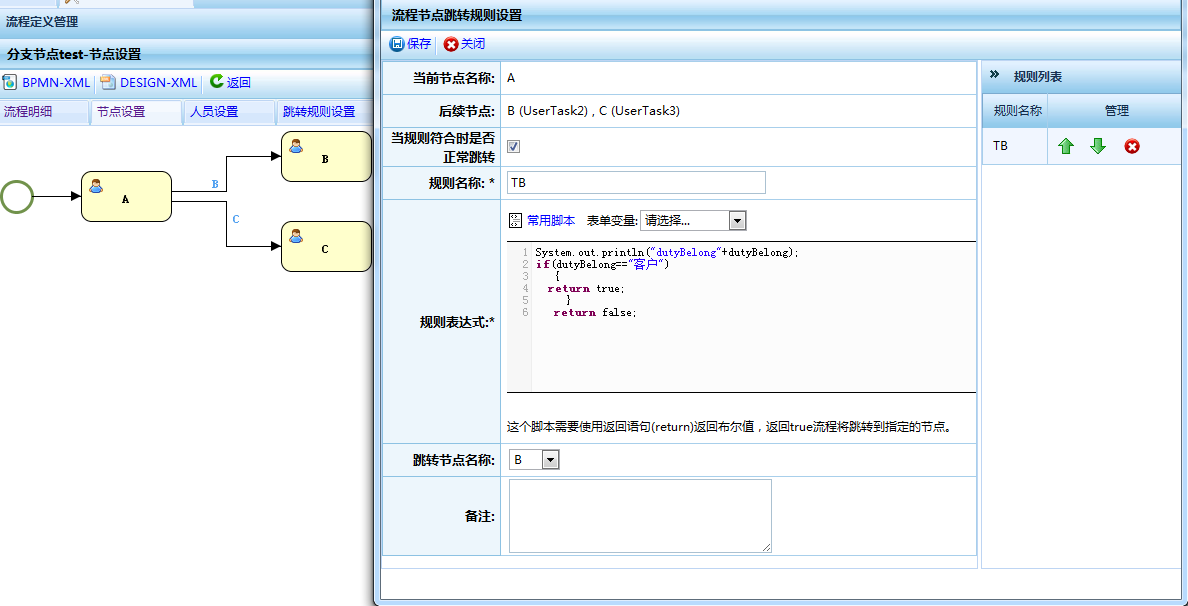
这些类都需要通过spring进行管理。

### 流程规则扩展

可以在节点设置规则，在完成当前任务时，如果这个节点设置了规则，那么流程会执行流程规则条件，如果流程规则返回true，那么就可跳转到符合条件的目标活动节点。

如果设置了多个规则的话那么他会一个个的执行这个规则，只要其中的任何一个返回true，就停止往下执行。获取相应的目标节点进行跳转。

规则使用java脚本，需要注意的是，规则需要有return语句而且必须是返回true和false。



流程跳转规则设置的几种情况：

1. B和C规则都设置，如果B规则成立，而C规则不成立，流程将往B任务跳；如果C规则成立，而B规则不成立，流程将往C任务跳；如果两个规则都成立，流程将默认跳第一个任务(B任务)；如果两个规则都不成立，流程将两个任务（B和C）都跳；

2.只设一个规则，另一个规则为空，如果被设的那个规则成立，流程将跳往该任务，否则流程将跳往两个任务；

3.如果两个规则都不设置，流程将跳往两个任务；

4.至于“当规则符合时是否正常跳转”的选择框，如果该选择框打上钩时，但设置的规则不生效，流程还继续往下跳；如果该选择框去掉，并且设置的规则不生效，流程将抛出异常。

### 流程URL表单开发

#### 实现原理

URL表单就是通过输入对应的URL地址，直接访问该地址就能调用的表单，其表单文件是一个**不带form标签**的jsp或者html页面。在流程表单中通过jquery的load方法将URL表单加入到启动流程或者下一步表单。

设置url表单需要与前置处理器或后置处理器结合才能完成表单数据的处理，前置处理器就是该处理器在流程启动之前处理，后置处理器就是启动后或下一步完成后处理。

前置处理器和后置处理器是负责数据存储的。需要注意的是在启动流程时需要使用前置处理器。前置处理器和后置处理器执行的时间点是：

1. 启动流程

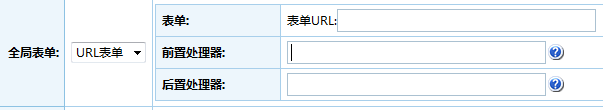
前置处理器发生在调用启动流程之前执行。

后置处理器发生在调用启动流程之后执行。

1. 完成当前任务

前置处理器发生在完成当前任务之前执行。

后置处理器发生在完成当前任务之后执行。



#### URL表单实例

下面是一个设置url表单的个例：

1.在web 目录下建立两个表单文件（例如：/form/ addPersonData.jsp与/form/ getPersonData.jsp），目录如下图1.1。

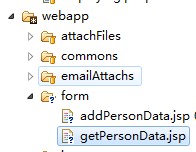


图1.1

说明：addPersonData.jsp：用来第一次提交表单数据，getPersonData.jsp用来查看之前保存的表单数据，同时可以更新表单数据。

addPersonData.jsp文件如下：



注意：由于平台表单提交已经用了form，所有在这里的表单文件中不能存在<form></form>。

getPersonData.jsp文件如下：

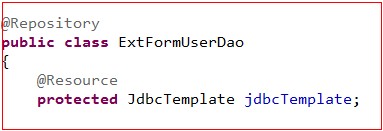


注意：红色框那里是访问后台读取上面addPersonData.jsp保存的数据；

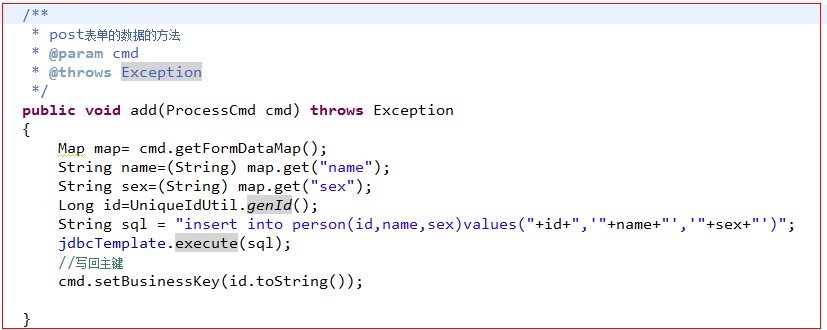
蓝色框那里是展现读取的数据，同时可以修改提交到数据库。

2.在dao目录下建立一个类名字为ExtFormUserDao.java（可以随意命名）的文件，在该类中写三个方法，add(processCmd cmd), Map getById(processCmd cmd) , updById(processCmd cmd)。

该类需要加入:

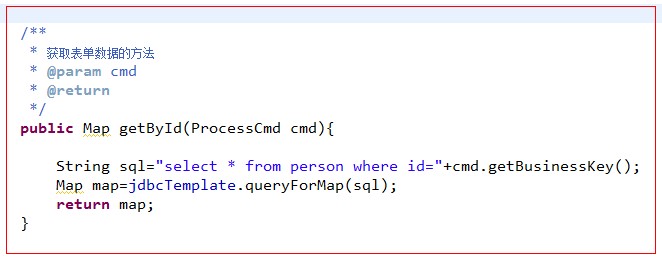
。

add(processCmd cmd):



注意：cmd.setBusinessKey(id.toString()),就是主键必须要写回。这样流程会将业务主键写会到 BPM\_PRO\_RUN 的主键字段中。

Map getById(processCmd cmd):

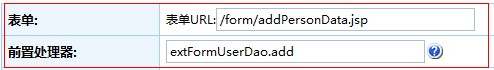


updById(processCmd cmd):

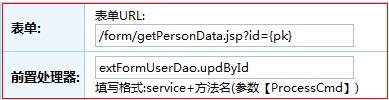


3.流程表单设置。

数据提交页面与方法：



获取数据与更新数据的方法：



数据读取页面

数据更新方法

### 会签使用

#### 概述

会签是流程中很常见的一种需求。

例如如下场景：

优秀员工评选，评选报告提交后需要，相关部门领导给出意见。

比如给5个领导提交意见。

可以是一票否决，也可以是3人同意就可以通过。

这个时候就可以使用会签了。

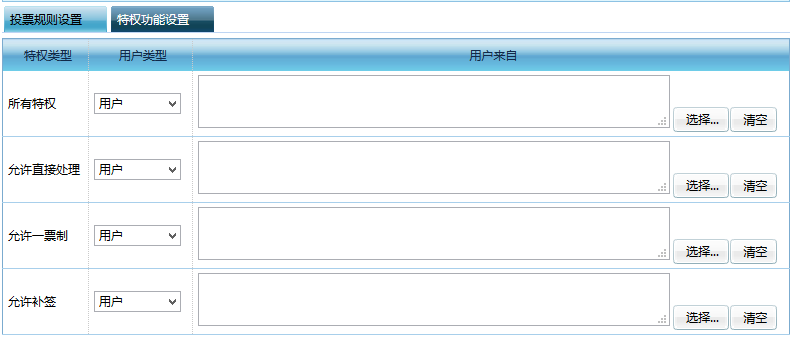
另外还有许多中国特色的会签。

1. 特定的人员参与这个流程后，可以直接审批通过，不用考虑其他的规则。
2. 特定的人员参与这个流程可以一票通过。
3. 特定人员参与这个流程一票否决。
4. 特定人员可以实现会签的补签功能。
5. 流程可以是直接处理，或者等待所有的人投完票。

会签规则设置:



特权功能设置



#### 实现原理

在bpmx3 平台实现方式中我们使用了activiti的多实例任务节点来实现。

1. 可以对该类型节点指定一个会签人员集合，那么流程引擎就会根据这个集合人数生成相应数量的子实例。
2. 对该节点指定一个执行完毕条件，我们在流程设定是可以创建一个会签规则。

流程引擎就会调用我们的会签规则。最终返回一个会签结果。会签结果会保存在全局流程变量当中。

流程变量名称：“signResult\_” + 会签节点名称。

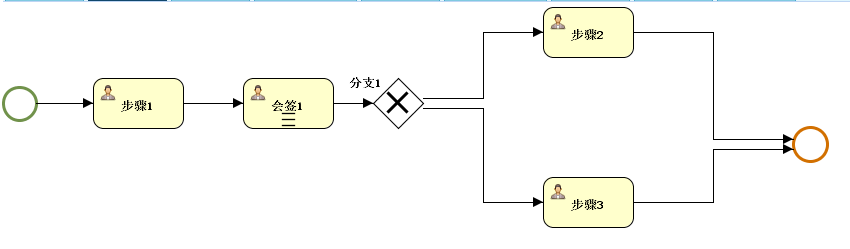
会签结果：

pass 通过

refuse:拒绝

#### 实现细节

会签流程定义



流程定义xml文件说明



相关类：

1.TaskSignCreateListener 会签任务监听器。

1. 添加会签意见。
2. 第一次执行时，这个监听器会执行多次。

添加节点状态。

添加会签执行。

1. 串行和并行

并行

一次产生多个子任务出来。

串行

一次产生一个任务，一个任务结束后，再生成新的任务分配给下一个用户执行。

1. 获取用户集合

TaskUserAssignService类的getSignUser(ActivityExecution execution)方法。

这个方法用于获取用户。具体实现查看相关方法。

1. 完成条件

类：com.hotent.platform.service.bpm.impl.SignComplete

调用isComplete方法。

isComplete(ActivityExecution execution)

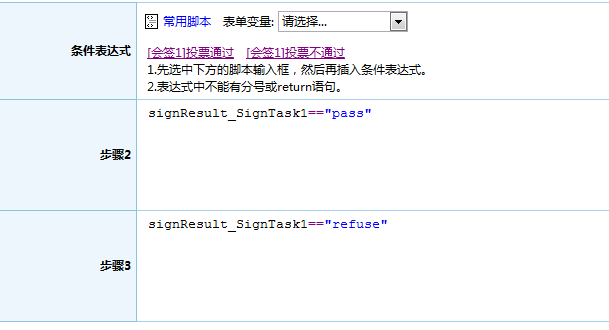
这个方法判定会签是否结束，如果结束，会在全局的流程变量中设置会签结果值。

会签同意：pass

会签否决：refuse

会签变量名称为：

signResult\_ +”会签节点ID”



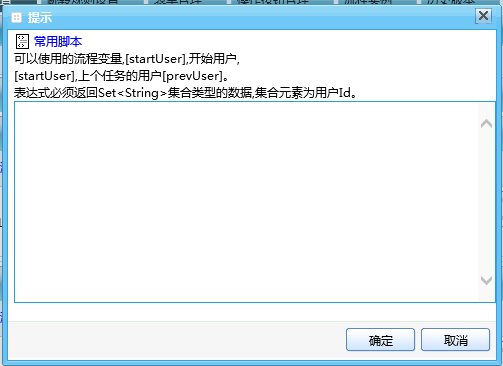
### 使用脚本获取流程人员

#### 概要说明

在设置流程人员的时候,提供了多种人员的选择方法,其中有一种是通过脚本来获取的,这个脚本该如何写,下面就介绍一下这个脚本的写法。

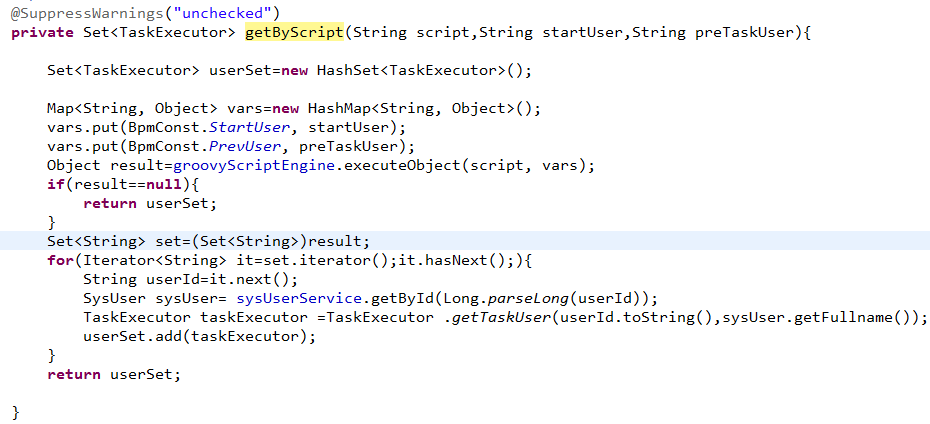
#### 脚本的编写方法

在定义用户时,如果选择脚本,将弹出。



在脚本编写的过程中,提供了两个流程变量。

1. startUser 开始用户。
2. prevUser 上一个任务执行人。这个流程变量表示的是前一个任务的执行者是谁。

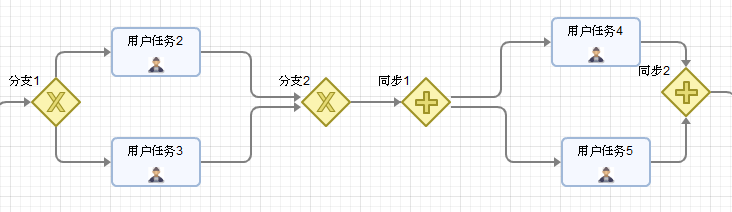


这个代码程序说明了我们是如何通过脚本计算得到用户集合的。

在编写脚本的时候可以直接引用这两个变量。

## 流程设计指导

1. 分支后执行同步网关问题：画法如图

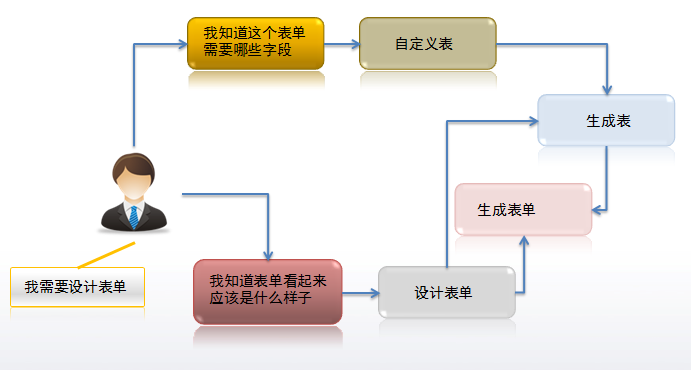


## 表单扩展

### 表单模块概述

表单在系统中主要负责采集业务数据，业务需求千变万化，同样表单也需要足够灵活和方便。

BPMX3提供了多种方式来设计表单，可以根据不同情况选择合适的设计方式，如下图所示：



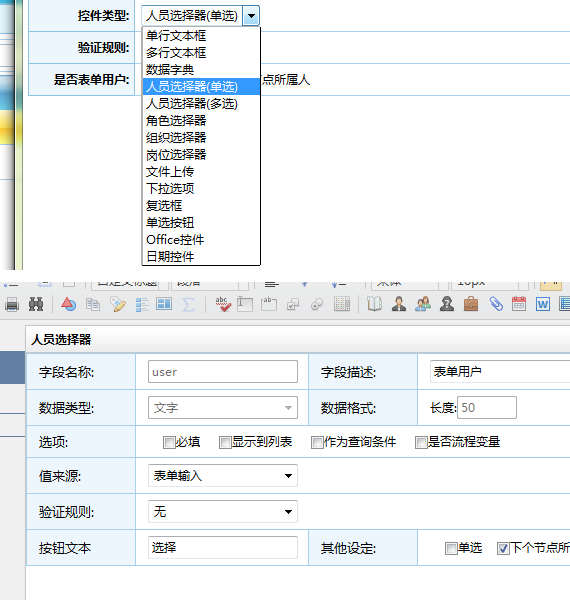
表单在设计完成以后会生成一条自定义表单记录、生成自定义表（物理表）。设计完成的表单可以在流程中使用，在流程中用以记录业务数据；也可以通过创建表单模板直接使用表单记录业务数据。

### 自定义对话框

#### 概述

在表单填写过程中，某些数据可能需要频繁填写，比如用户、所在部门、岗位等等。出于方便、快捷、减少出错方面考虑，我们使用对话框（或称为选择器）来让用户直接选择这些数据。

当然这些用户、部门、岗位等我们的系统已经提供了选择器，在设计表单的时候可以直接添加这些选择器，如下图所示：



但是可能用户需要将另外一些业务数据用作对话框，这里的自定义对话框就能满足这种需求。

#### 编辑

打开页面『流程管理』→『表单管理』→『自定义对话框』，点击【添加】或者在已有自定义对话框的管理列中点击【编辑】按钮可以进入自定义对话框的编辑页面。

编辑页面的属性中，数据源是个下拉列表，其选项在『系统管理』→『系统配置』→『系统数据源』页面进行管理。选择某个数据源以后，可以点击查询表（视图）后面的【查询】按钮，查询的结果会显示在下方。选中要设置为对话框的表（视图），然后点击【设置列】按钮可以打开字段设置页面。

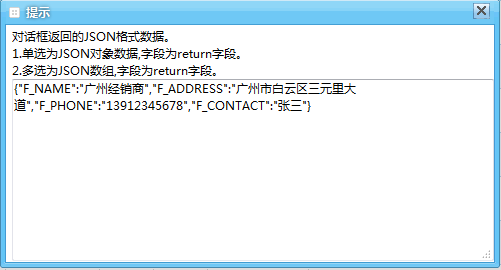
在字段页面中可以设置对话框显示的字段、返回的字段和用作查询的字段，设置完以后点击【确定】按钮即可完成列的设置。点击【保存】按钮就完成了自定义对话框的编辑。

#### 查询、删除和预览

对话框管理列表页面中的【查询】和【删除】都是通用的功能，这里不做介绍。在对话框的管理列中可以点击【预览】按钮打开对话框的预览界面。



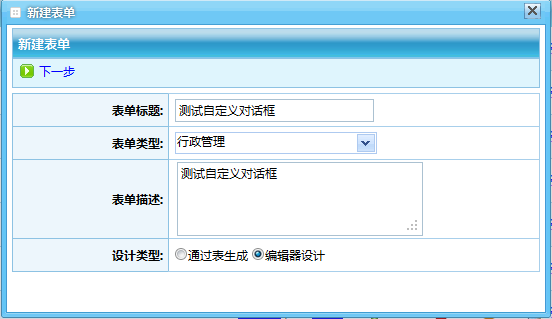
上面的红色框内是查询字段，下面的红框是显示字段，选中某条记录以后点击【确定】按钮会显示返回的字段。



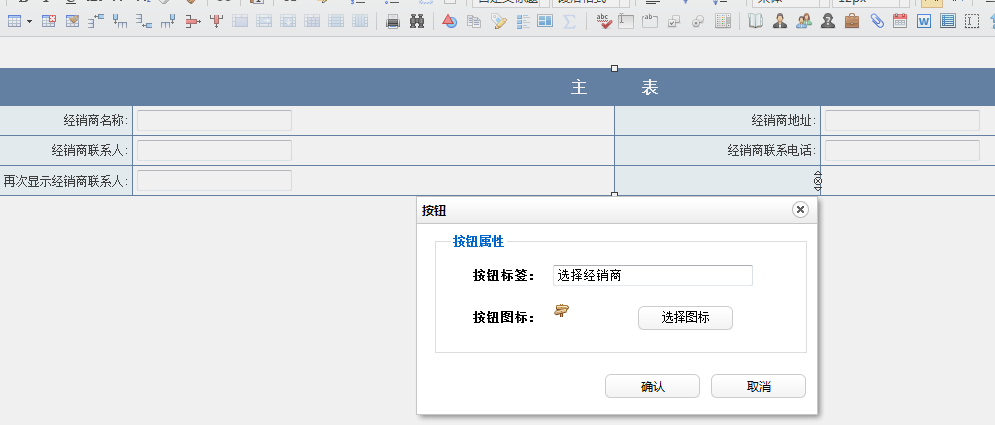
#### 对话框的使用

##### 在自定义表单中的使用

对话框定义好以后，主要是为了在表单中使用。点击『流程管理』→『表单管理』→『自定义表单』页面中的【添加】按钮，打开添加表单的页面。设计类型中选择〖通过表生成〗或者〖编辑器设计〗都行，这里为了方便，我们选择〖编辑器设计〗类型，然后一直点击【下一步】到打开『表单设计』页面。



在设计页面添加一些字段，然后添加一个按钮，如下图所示：



鼠标移到添加的按钮上时会显示工具条，点击工具条上【自定义对话框】按钮可以打开对话框设置窗口，如下图所示：



在选择对话框的下拉列表中选择〖选择经销商〗，在左侧的树状菜单中会显示该对话框的所有返回字段，在右侧是表单的所有字段。想要将返回的某个字段填写到表单的某个字段中的话，只需要将表单的字段拖拽到该返回字段下（之所以这样设置，是因为一个返回值可以填写到多个表单字段中，比如这里F\_CONTACT字段要同时填写到联系人和再次显示经销商联系人这两个表单字段中，就需要把这两个字段都拖拽到F\_CONTACT字段节点下）。拖拽完以后点击【确定】按钮即可完成自定义对话框的设置。

点击页面上方的【预览】按钮，可以打开该表单的预览页面，点击【选择经销商】按钮可以打开对话框，选择经销商以后点击【确定】按钮，改经销商的相关信息就会回填到表单的字段中，如下图所示：



##### 自定义对话框自己开发页面中的使用

自定义对话框除了在自定义表单中使用，也适用在自己开发的页面中进行使用。自定义开发的页面使用方法如下：

引用js代码：

<script type="text/javascript" src="${ctx}/js/hotent/platform/form/CommonDialog.js"></script>

当然需要引用相关的js代码。

可以简单的包含：

<%@include file="/commons/include/get.jsp" %>

就可以了。

在需要使用的页面上编写如下代码：

CommonDialog(alias,**function**(data){

});

这个代码放到按钮事件代码中或其他的事件处理中。

参数说明：

alias ：对话框的别名。

function(data){

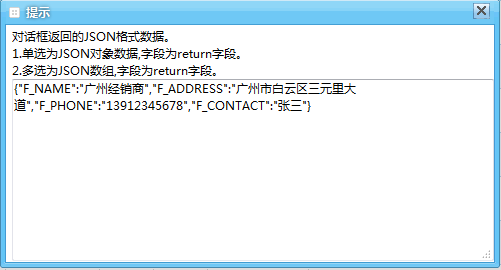
}

data:为对话框的返回数据。

这个data数据就根据配置的来处理。

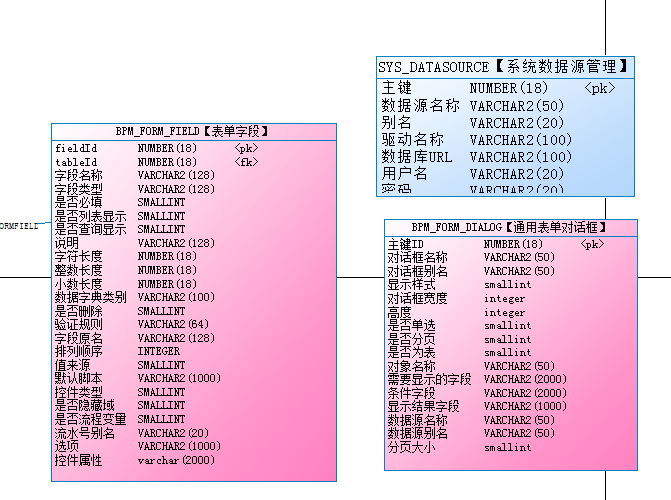
单选的话：返回的数据为一个json对象。

多线的话：为一个json数组。



这样用户就可以很快速的在页面中使用自定义对话框的功能了。

#### 相关的数据库表



* SYS\_DATASOURCE【系统数据源管理】

系统数据源管理，在编辑自定义对话框时选择数据源的下拉列表选项就是来自这个张表。

* BPM\_FORM\_DIALOG【通用表单对话框】

自定义对话框的表，记录对话框的一些属性，以及查询、显示和返回字段的一些设置。

### 表单规则验证

#### 概述

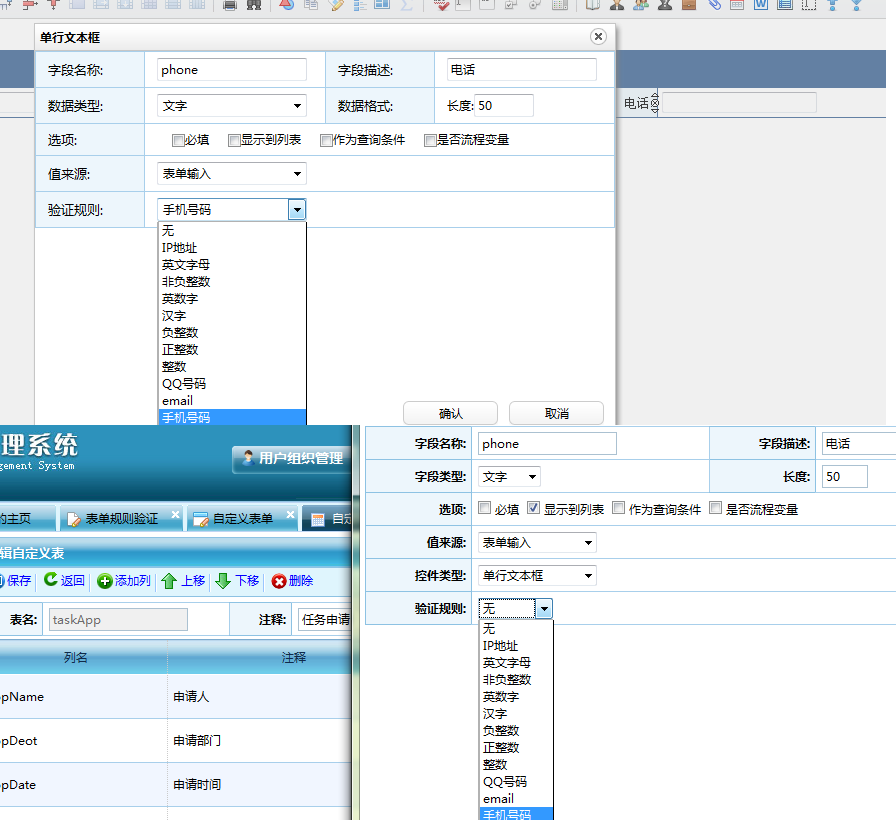
用户在填写表单的时候，对于某些字段的书写规则可能不太了解，可能就会造成输入的内容与预期的不符。为了让某些字段按照特定规则录入数据，可以在表单的字段设计时添加上规则验证。

#### 查询、添加、修改和删除

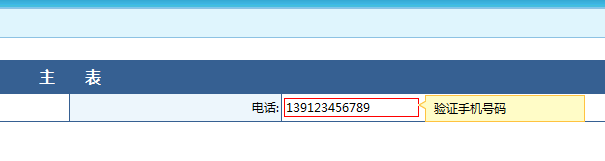
打开『流程管理』→『表单管理』→『表单规则验证』页面，在页面上对于表单规则可以进行【查询】、【添加】、【修改】和【删除】操作，因为功能都比较简单明了，这里就不做介绍。

#### 表单规则的使用

表单规则可以再两个地方使用，一是在『流程管理』→『表单管理』→『自定义表』页面中，添加字段的时候，对于表单输入类型的字段可以添加表单规则验证；二是『流程管理』→『表单管理』→『自定义表单』页面中，通过〖编辑器设计〗的方式添加表单时，为表单添加表单输入类型字段时可以添加表单规则验证。如下图所示：



设置了验证规则的字段，在填写表单时，这个字段填写的内容需要符合验证规则才能提交，如下图所示：



#### 相关的数据库表



* BPM\_FORM\_RULE【表单认证规则】

记录表单规则的基本信息，包括正则表达式、提示信息等。

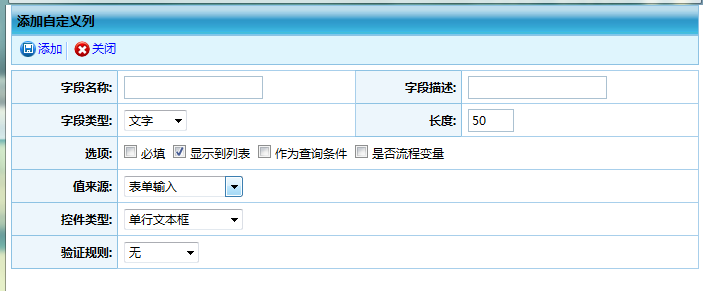
### 自定义表

#### 概述

BPMX3系统可以非常直观的让用户直接在页面上创建自定义表并生成物理表。支持不同类型的数据库，如oracle、sqlserver、mysql,db2,国产达梦等。

#### 编辑

打开『流程管理』→『表单管理』→『自定义表』页面，点击【添加】按钮或【编辑】按钮可以打开自定义表的编辑页面。表的属性有〖表名〗、〖注释〗以及〖是否主表〗，选择从表需要指定其主表（只能指定未生成物理表的主表）。另外就可以通过【添加列】按钮来编辑表的字段，如下图所示：



* 字段类型

可以选择文字、数字、日期和大文本。

* 选项

〖必填〗表示该字段在录入数据时为必填字段，〖显示到列表〗是指在查看该表的业务数据时会显示到列表的字段，〖作为查询条件〗是在查看业务数据时会用作查询条件的字段，〖是否流程变量〗是在流程中可以用作变量进行流程控制或其他用途。

* 值来源

可以选择表单输入、脚本运算（显示）、脚本运算（不显示）、流水号。

表单输入为用户输入；

脚本运算为显示脚本返回的值，显示与不显示即在页面上是显示还是隐藏；

流水号是按照一定的规则生成一串流水号，打开『系统管理』→『流程辅助』→『流水号管理』可以添加流水号，流水号添加完成以后就可以在字段设置中使用。

另外流水号的使用只能在主表字段中，因为流水号是通过后台生成的，而子表添加记录时只通过js添加子项，没有请求后台，所以不能生成新的流水号。

* 控件类型

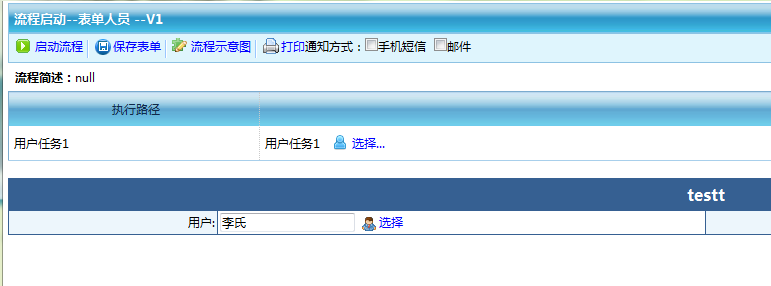
可以选择单行文本、多行文本、数据字典、人员选择器（单选）、人员选择器（多选）、角色选择器、组织选择器、岗位选择器、文件上传、下拉选项、复选框、单选按钮、office控件和日期控件。

数据字典需要指定数据字典分类，打开『系统管理』→『系统配置』→『分类管理』→『数据字典』页面可以添加数据字典分类，以及给某类数据字典添加字典项。

人员选择器（单选）、人员选择器（多选），人员选择器还可以作为下个节点的执行人，如下图所示，在勾选了〖所选人员作为下一节点执行人〗以后。在流程中某一节点填写表单时，选择了该字段，则所选人员将会作为下个节点的执行人（如果选择了多人就作为候选执行人）。



流程节点人员设置的优先级，如下图所示，执行路径上选择的人员>表单中的下一节点执行人>数据库中人员（流程设置中该节点所设置的人员）。



部门选择器也可以作为下一节点的执行人，即所选部门的负责人作为下一节点执行人。

#### 导入导出

自定义表要转移到其他的服务器上时，可以使用导入导出的功能。

#### 生成表

自定义表添加完成以后就可以点【生成表】按钮来生成物理表。

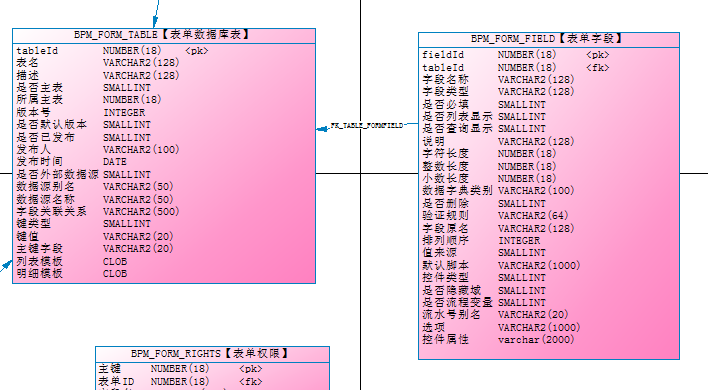
#### 索引管理

每个自定义表会默认生成一个主键索引，点击【索引管理】按钮进入索引管理页面还可以继续添加索引。

#### 自定义表的使用

自定义表主要用作生成自定义表单，在自定义表单的管理页面点击【添加】按钮，然后选择自定义表，点击【下一步】按钮，选择模板以后就会生成该表的表单。

#### 相关的数据库表



* BPM\_FORM\_TABLE【表单数据库表】

记录的自定义表的属性信息。

* BPM\_FORM\_FIELD【表单字段】

记录了自定义表的字段信息。

### 自定义表单

#### 概述

自定义表如上面所述两种生成方式，表单的设计界面也比较简单、方便，设计好的表单主要用在管理业务数据和流程中。

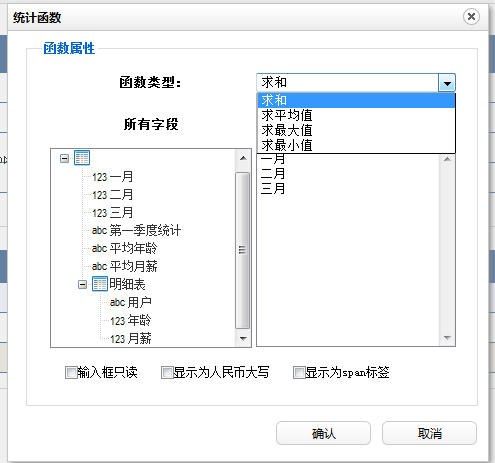
#### 设计表单

表单设计中的大部分功能都比较简单、直观，这里介绍下比较复杂的功能怎么使用。

* 统计函数

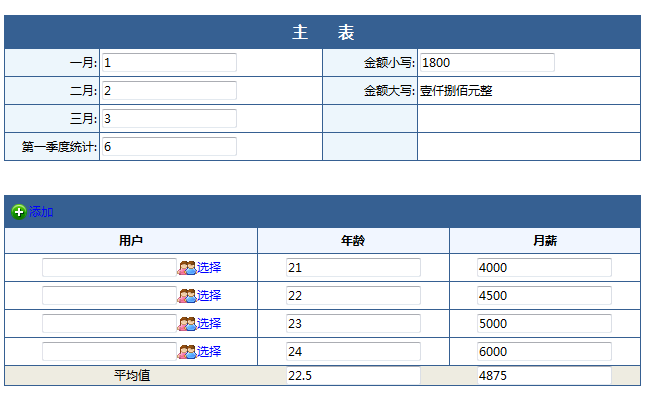
统计函数就是类似excel中的统计功能，比如要统计主表中的一些字段或者子表中的某列，求和、求平均值、求最大值等等；或者某个字段是金额，需要同步显示该金额的大写。

双击选中某个字段，然后右键打开右键菜单点击【统计函数】，或者选择工具按钮上的【统计函数】打开统计函数的编辑界面。编辑界面如下图所示：



函数类型下拉菜单那里可以选择统计函数的类型，这个统计函数可以进行扩展，在/webapp/js/hotent/platform/form/FormData.js这个文件内扩展，扩展的方式可以参照已有的四种函数。

左侧是所有字段的列表，包括主表字段和子表字段，双击字段则选为统计字段（字段类型必须是数字类型）。选择好以后点击【确定】按钮即完成了设置。效果如下所示：



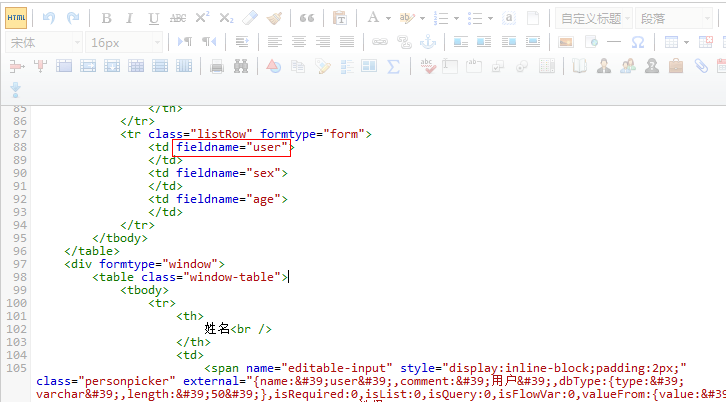
* 子表弹出窗口模式

在通过编辑器设计模式添加表单时，添加的子表选择弹窗模式时，需要进行一些特殊的设置。

编辑完弹出窗的字段以后，需要设置表格中回填的字段，如下图所示：



需要切换到html代码模式下进行设置，如下图所示，在表格的单元格中添加属性fieldname，其值与弹窗中的字段名保持一致。



另外，在子表中添加字段时候，不要选择单选按钮类型，可以用下拉列表代替，因为在两条乃至多条记录中，单选按钮只能选择一个。

#### 预览

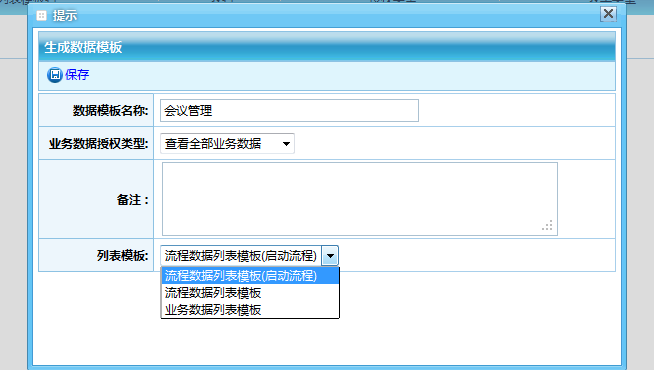
表单设计完成以后可以进行预览。

#### 表单权限

主表的所有字段分别可以设置权限，子表可以整体进行权限设置。

#### 数据模板

设计好的表单可以通过添加数据模板来管理业务数据，如下图所示，有三种类型的数据模板：



* 流程数据列表模板（启动流程）

介绍：该模板在表单用作流程时，在启动界面填写了部分表单数据，然后点【保存表单】按钮，然后在【查看业务数据】页面可以查看到该条记录。此时可以点击【启动流程】来直接启动流程，而且保存时填写的表单数据也能默认填写好。

应用场景：一个流程的表单比较复杂，内容比较多，但是大部分内容是相同的，这个时候我们就添加一个这样的数据模板，然后启动应用了该表单的流程，填写那些通用的字段，填完保存表单。这样下次启动该流程时，这部分数据就不需要填写了。

* 流程数据列表模板

表单应用到流程中，产生了很多业务数据，但是这些业务数据没有页面可以进行查询。这个模板就是提供来查询流程中产生的业务数据的。

* 业务数据列表模板

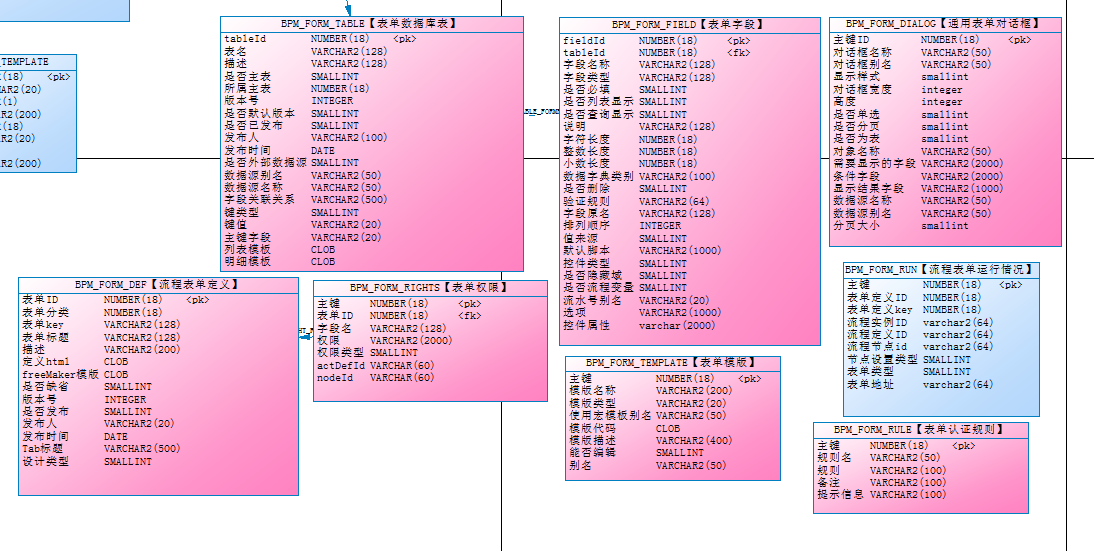
介绍：有时候表单设计出来可能不会应用到流程中去，只是简单的添加、修改、删除和查询业务数据。这是就可以添加业务数据列表模板。

应用场景：设计了一张入库单，在用货物要入库的时候填写一下，然后保存。

#### 自定义表单的使用

自定义表单主要有两个用途，首先通过添加业务数据模板进行业务数据的管理；其次通过应用到流程中来管理数据。

#### 相关的数据库表



* BPM\_FORM\_TABLE【表单数据库表】

记录生成表单的表信息。

* BPM\_FORM\_FIELD【表单字段】

记录自定义表的所有字段信息。

* BPM\_FORM\_DIALOG【通用表单对话框】

记录自定义对话框的信息。

* BPM\_FORM\_DEF【流程表单定义】

自定义表单的信息。

* BPM\_FORM\_RIGHTS【表单权限】

表单的字段和子表的权限设置信息。

* BPM\_FORM\_TEMPLATE【表单模版】

表单的业务数据模板信息。

* BPM\_FORM\_RUN【流程表单运行情况】

应用到流程中的表单运行信息。

* BPM\_FORM\_RULE【表单认证规则】

用作表单规则验证的信息。

### 业务数据模板管理

上一节中介绍了自定义表的数据模板，每个表单都可以添加多个数据模板，该页面可以管理所有的数据模板。

## 使用平台做二次开发

### 系统架构介绍

本系统采用了spring架构开发，web端使用了spring mvc 框架。Spring mvc和众多的mvc框架一样，它基于mvc设计理念。此外它采用了松散耦合和插拔组件结构，比其他的MVC框架更具扩展性和灵活性。Spring mvc 通过一套MVC注解，让pojo成为处理请求的控制器，无需实现任何接口，同时spring mvc 还支持REST风格的的URL请求。注解驱动即REST风格的spring mvc 是SPRING 3.0最出彩的功能之一。此外，spring mvc 在数据绑定，视图解析，本地化处理及静态资源处理上都有许都不俗的表现。

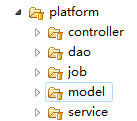
各层级主要涉及的框架技术：

1. 视图层 JSTL

2. 逻辑层 Sqring3.0.3

3. 持久层 Mybatis3.0.4

### 开发目录介绍



platform 业务系统名称

controller 存放控制器类

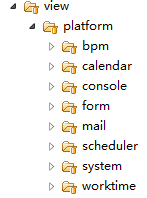
dao 存放数据访问的类

model 存放数据实体的类

service 存放业务代码类



maper 存放数据库表的映射关系以及数据访问语句



view/platform 存放视图页面

### 开发步骤

#### 创建业务数据表

根据业务逻辑需求创建所需的数据表，表名，字段名命名规则如下:

1. 表名均为大写，多个单词用下划线分开，示例：

SYS\_USER表

1. 主键类型，均采用NUMBER (18),并且采用自增类型。
2. 主键命名规则。

Pk\_表名\_主键字段。

示例：表名person

Pk\_person\_id

1. 索引命名规则

Idx\_表名\_字段名称。

1. 外键的命名则需要为FK\_从表缩写\_R\_主表缩写，示例：

SYS\_USER（缩写：SYS\_USR）表引用ORGANIZATION（缩写：ORG）表，

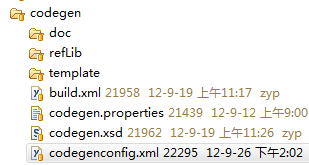
FK\_ORG\_R\_SYS\_USR

若有从表有多个字段引用主表，则命名为: FK\_从表缩写\_字段缩写\_R\_主表缩写

1. 字段命名，采用大写，多个单词用下划线分开，以方便生成类，并且方便开发人员直接的SQL编写。示例：USER\_ID。
2. 对于一些常用的枚举类型字段，数据类型即采用Smallint，并且需要标识每个值的作用，如SYS\_USER表中，是否激活字段，则需要命名isActive,类型为Smallint，其说明如1=激活，0＝禁用
3. 视图命名使用V 开头。

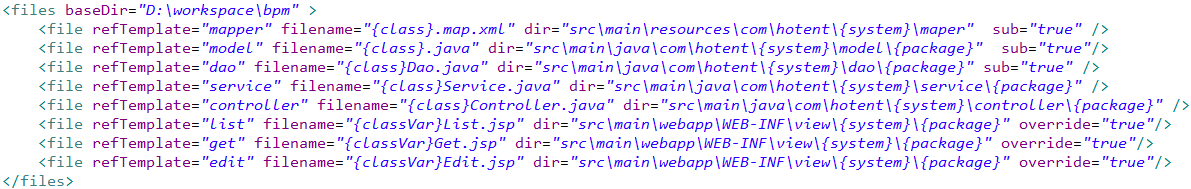
#### 生成业务基本文件

基本文件主要通过代码生成器生成



在codegen.properties属性文件中配置连接的数据库信息，系统名称，编码；

codegenconfig.xml配置文件的作用：

1. 生成文件的相关信息，包括路径，文件名，是否覆盖已有文件，
2. 数据表（单表，主从表）的基本信息，包括表名，主键，实体类类名，实例变量名，包名。

<table tableName=*"SALES\_ORDER"* >

<variable name=*"class"* value=*"SalesOrder"* />

<variable name=*"classVar"* value=*"salesOrder"* />

<variable name=*"package"* value=*"ms"* />

<variable name=*"path"* value=*"ms"* />

<subtable tablename=*"SALES\_DETAIL"*foreignKey=*"ORDER\_ID"*>

<variable name=*"class"* value=*"SalesDetail"* />

<variable name=*"classVar"* value=*"salesDetail"* />

<variable name=*"package"* value=*"ms"* />

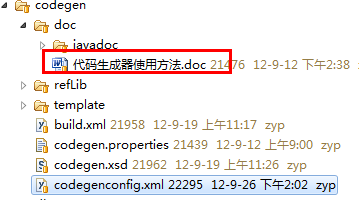
<variable name=*"path"* value=*"ms"* />

</subtable>

</table>

build.xml 可执行ant文件共有两个可执行任务（生成文件(默认)，删除生成的文件）。

详细的生成方法请参考文档：**代码生成器使用方法**



#### 根据具体的业务需要修改

基本文件生成后，开发人员可以根据实际的业务需求做些修改；

#### 开发实例

需求：订单管理系统（用户可以添加订单，查看订单详情，编辑订单，删除订单）

业务表：

SALES\_ORDER\*订单表（ORDER\_ID 主键，ORDER\_NO 订单号，CREATETIME 订单创建时间，TOTAL 总金额，DESCP 备注]）。

SALES\_DETAIL\*订单详情表（DETAIL\_ID 主键，ORDER\_ID 关联订单主键，ITEM\_NAME 商品名，PRICE 价格，AMT 数量，DESCP 备注）。

1. 建表SQL:

**CREATE** **TABLE** SALES\_ORDER

(

ORDER\_ID NUMBER(18) **PRIMARY** **KEY** **NOT** **NULL**,

ORDER\_NO NUMBER(18) ,

CREATETIME **DATE**,

TOTAL NUMBER(18,4)，

DESCP **VARCHAR**(400)

);

**CREATE** **TABLE** SALES\_DETAIL

(

DETAIL\_ID NUMBER(18) **PRIMARY** **KEY** **NOT** **NULL**,

ORDER\_ID NUMBER(18) ,

ITEM\_NAME **VARCHAR**(400),

PRICE NUMBER(18,4),

AMT NUMBER(18,4),

DESCP **VARCHAR**(400)

);

1. 使用代码生成器生成基本文件

实体类：SalesOrder.java,SalesDetail.java;

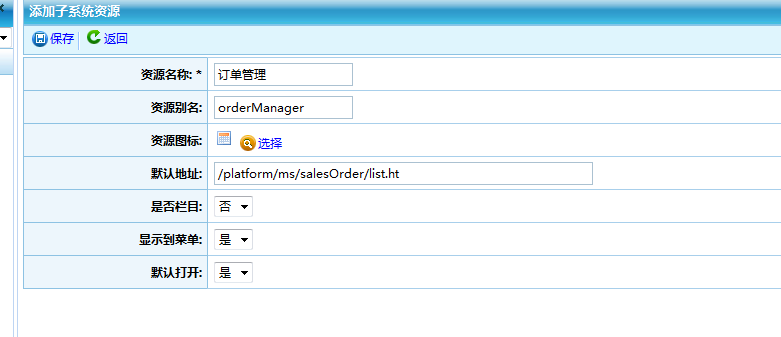
控制器类：SaleOrderController.java;

Service类：SaleOrderService.java;

Dao类：SaleOrderDao.java;

Maper文件：SaleOrder.map.xml,SalesDetail.map.xml;

View 文件：salesOrderList.jsp,salesOrderEdit.jsp,salesOrderGet.jsp;

1. 添加功能至资源表中,如图：

## 定时任务开发介绍

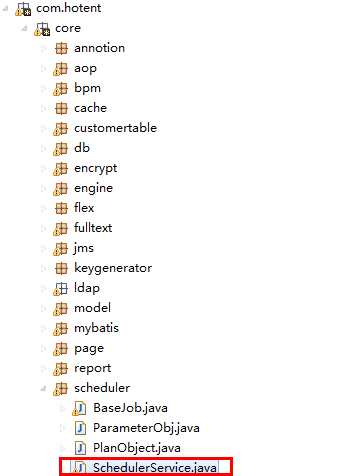
BPMX3 集成了QUARTZ 2.0定时框架。这个任务框架解决了我们任务定时后台执行的问题。在平台中二次开发用户只需要定义自定义任务。通过平台加入到系统，就可以实现任务调度了。

### QUARTZ 集成方法

QUARTZ和spring框架集成，查看app-resources.xml文件。



定时器代码核心类



SchedelerService类

实现了任务的添加，任务的定时管理等。

### 定时任务的开发

在定时任务开发的过程中，如果需要增加自定义执行逻辑，那么就需要自定义任务，实现需要定时的逻辑。

开发方法如下：

继承com.hotent.core.scheduler.BaseJob 。这个类已经帮我们实现了任务的日志记录。

用户只需要实现executeJob方法就可以了。

#### 开发技巧

有时用户在实现和任务整合时需要传入一些参数，例如：有这样的一个需求，需要根据指定的客户定时进行发送短信，在新建任务的时候，系统可以指定哪些客户。那这样开发任务的时候如何取到这些用户呢？

1. 指定用户方法。

使用SchedulerService的addJob(String jobName,String className,Map parameterMap,String description) 方法

jobName:任务名称

className:自定义任务 Job的class路径，

parameterMap:这个就是传入具体的参数了。

Description:任务描述

示例：

List<String> list=new ArrayList<String>();

list.add(“user1”);

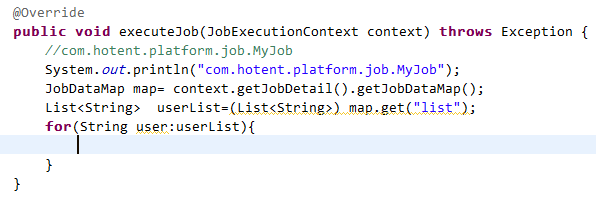
list.add(“user2”);

Map map=new HashMap();

map.put(“list”, list);

schedulerService.addJob(“发送短信”,”com.hotent.platform.job.SendSmsJob”,map,”发送短信任务”);

1. 读取用户的方法



#### 任务调度

任务调度我们构建了多种调度，如果这些还不能满足调度要求，高级用户可以编写cron 表达式，表达式的语法可以参考相关网上的文档。

## 通过WEB服务操作流程

平台中集成了cxf发布WEB服务。Bpmx3平台提供了使用WEB服务接口，对流程的操作。

平台提供了操作流程接口类，接口类地址如下：

com.hotent.platform.webservice.api.ProcessService，具体的操作请参考这个接口的说明。

### 配置服务端

集成相关的地方：

1. web.xml



1. app-cxf-service.xml 文件

定义web服务需要定义接口类和接口实现类。配置到这个xml文件中就好了。



将类配置到cxf，发布成web服务。

Web服务访问地址 /service/ 加上上面配置的地址。

上例的地址为：

<http://serverUrl>:端口/bpm/serv ice/ProcessService 。

<jaxws:inInterceptors>

<ref bean="addressInInterceptor" />

</jaxws:inInterceptors>

为拦截器，在这里我们配置了IP地址拦截器，只有IP地址在配置范围的才可以访问。

具体实现逻辑参考

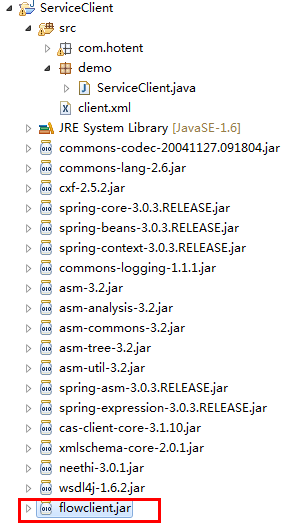
<bean id="addressInInterceptor"

class="com.hotent.platform.webservice.impl.util.AddressInInterceptor" />

参考这个地址拦截类。

### 配置客户端

客户端项目结构如下：



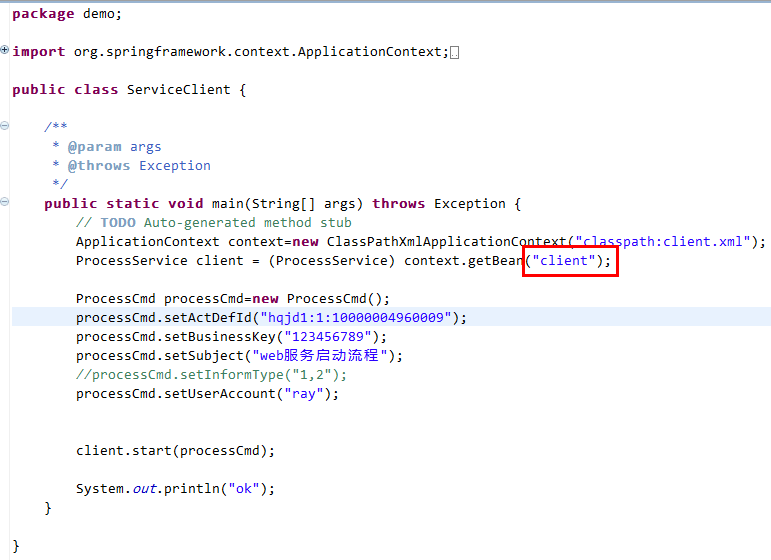
flowclient.jar是客户端调用接口包。

客户段spring文件client.xml



这个是spring调用web服务相关配置。

实现代码如下：



这里指定流程定义ID,业务主键，启动流程人的帐号，启动流程。

## 系统部署

### 配置分布式部署

#### 概述

BPMX3 平台支持系统部署到多台应用服务器上。分布式部署的问题需要解决的一个问题时如何保持ID一致的问题。

解决方案1：

使用guid保证ID的唯一性的问题，这个方法优点是id始终是唯一的，但是ID太长。

解决方案2：

为每台应用服务器指定一个ID前缀。

在我们系统中解决方法是:

前缀加流水号。

具体做法：

1. 为每台服务器配置一个前缀，前缀为数字。
2. 在数据库为每台服务定义一条记录。



ID，代表服务前缀，INCREMENTAL代表ID递增值。

1. 产生id方法。

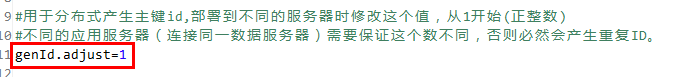
记录两个数字，一个为上限值,一个为当前ID,当前ID每次加1，当当前ID小于上限值，就一直加，当当前id大于上限值时，那么将上限值加一个递增值。

将这个递增值写入数据库，为最大 的边界值。在将当前的id +前缀\*10的14次方。

#### 部署方法

如果只有一台应用服务器，这个部署就不用了。如果程序需要部署到多台应用服务器，则需要做一下步骤。

打开 WEB-INF/classes/conf/app.properties文件。



修改这个值。

有多台服务器，可以从1开始编号，要保证每台服务的值都不同。

### 部署不同类型的数据库

Bpmx3 平台支持多种类型的数据库，目前支持5种主流的数据库。

1. oracle 10g以上。
2. mysql 5.0以上。
3. sqlserver2005 。
4. Db2。
5. 达梦数据库。

其他的如db2需要测试。

具体部署方法：

修改 app.properties 文件。



修改为相应的数据库。

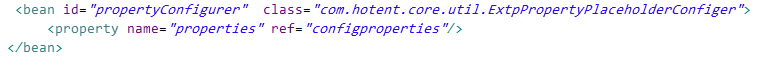
### 数据库密码加密

程序一般部署在服务器端，在配置文件中数据库密码是使用明文的。有这样的需求，密码加密存放。只有知道如何加密的人，才可以修改此密码。

在app-resource.xml将配置:

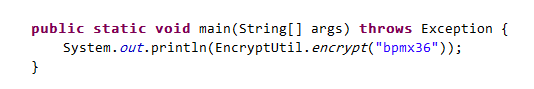


修改为



同时加密码：

管理员可以编写如下代码：



将获取到的密文：



编辑app.properties 文件，替换原有的密码。

### 单个启动定时器

有时我们在部署了多台应用服务器的时候，定时器也会随着启动多个，如何只启动一个定时器呢？我们可以对配置文件app-resources.xml 做如下修改。



这里修改成false。只在需要定时器的服务器，把这个改为true。

## 自定义流程实例示意图的显示颜色

系统默认的流程实例示意图的显示样式,会根据不同的执行状态,节点显示不同的样式。虽然系统提供的默认样式可以满足大多数的需求，但系统也提供了自定义显示样式的的接口。二次开发或用户都可按需要定制自已的样式。

### 实现接口

要自定义运行流程显示图的显示颜色，需要实现接口com.hotent.platform.service.bpm.IFlowStatus

接口IFlowStatus中的方法getStatusByInstanceId的返回值是一个Map：key=<节点的ID>,Value=<颜色>

系统默认的实现是com.hotent.platform.service.bpm.impl.FlowStatus。(见下面的Spring 配置)在FlowStatus实现类，通过bpmProStatusDao到BPM\_PRO\_STATUS表取得流程节点的执行状态，再根据状态到statusColorMap(这个map对象做了状态和颜色的配置，需要注意的是状态值是固定的)取得相应的颜色设置。

二次开发时，可以根据需要，实现接口IFlowStatus，实现自定义的节点状态与颜色关系。也可以直接基于原有的默认实现，能过配置statusColorMap，定义不同的状态的颜色。

### 修改配置

接口IFlowStatus的自定义实现类需要在Spring的配置文件进行依赖注入配置（注意：要删除原来的默认实现）

下面是系统默认的实现，二次开发时只要替换为自己的实现即可。

app-beans.xml：

<bean id=*"iFlowStatus"* class=*"com.hotent.platform.service.bpm.impl.FlowStatus"*>

<property name=*"bpmProStatus"* ref=*"bpmProStatusDao"*></property>

<property name=*"statusColor"* ref=*"statusColorMap"*></property>

</bean>

<bean id=*"statusColorMap"* class=*"java.util.HashMap"*>

<constructor-arg>

<map>

<entry key=*"1"* value=*"#00FF00"*></entry>

<entry key=*"0"* value=*"#FFA500"*></entry>

<entry key=*"-1"* value=*"#FF0000"*></entry>

<entry key=*"2"* value=*"#0000FF"*></entry>

<entry key=*"3"* value=*"#8A0902"*></entry>

<entry key=*"4"* value=*"#023B62"*></entry>

<entry key=*"5"* value=*"#338848"*></entry>

<entry key=*"6"* value=*"#82B7D7"*></entry>

</map>

</constructor-arg>

</bean>

## SpringSecurity配置

### 单一登录配置

#### 概要说明

在平台中我们默认限制一个用户在同一个时刻只允许登录一次，比如USER1在A电脑登录，同时用户在B电脑也使用USER1的帐号登录，那么这个时候，在A电脑的登录将自动退出系统操作。

这样做在的好处在保证用户的误操作。

例如这样的场景：

我们的系统支持多个子系统，这些系统可能是部署在不同的服务器的。我们系统在切换当前用户组织时，我们需要保证给各个系统都能一致切换。这个时候我们用了独立的缓存系统来实现，也就是我们切换系统的时候，修改缓存系统中的当前用户的组织数据，其他的系统都使用缓存来读取当前组织的数据。

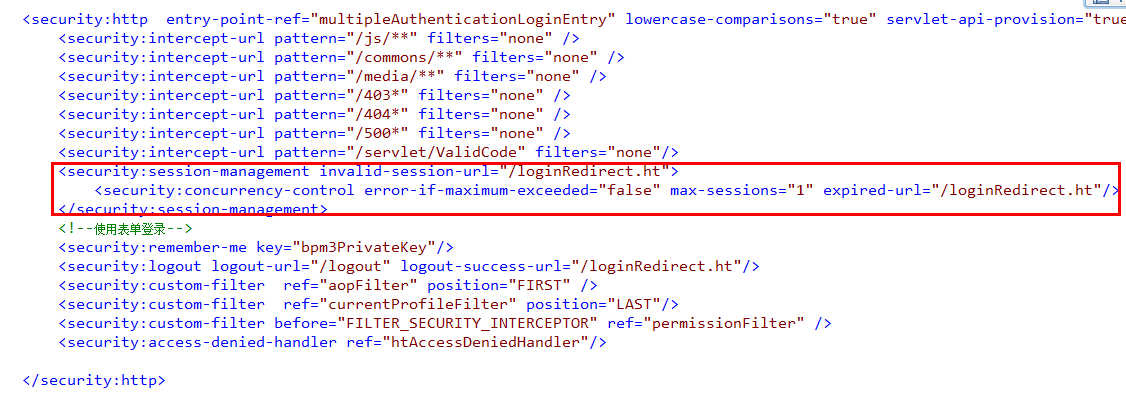
在这种场景中，我们如果允许同一个用户同时在不同的地方进行登录会遇到什么情况呢？

1. 用户user1在A处登录，他的组织为A组织。
2. 用户user2在B处登录，这个他修改了当前组织为B组织。
3. 如果在A处操作的时候他会还以为当前组织为A，因为他并没有切换过组织。

所以这个时候就需要保证用户在同一个时刻，一个帐号在系统中只能登录一次。

#### 实现方法

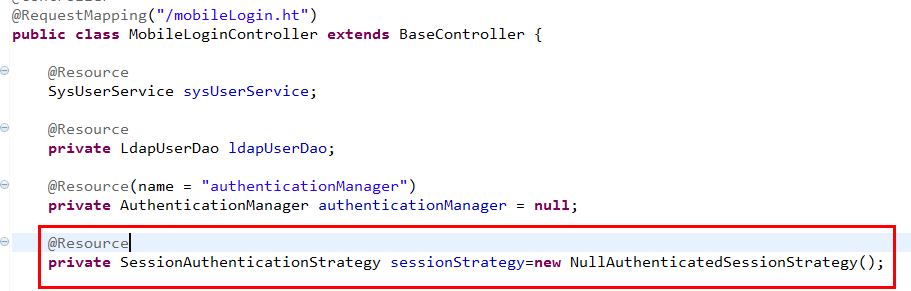
修改app-security.xml配置如下。

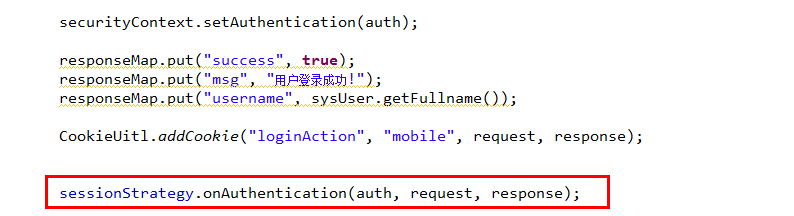


增加这段配置。error-if-maximun-exceeded 为false表示，当前登录后，前面登录的会自动退出，如果设为true的话，那么如果之前有登录，那么后面登录的就会报错登录不了。 max-sessions为1表示只允许登录一次。

需要注意的时候，我们在做用户验证的时候并没有直接使用springsecurity提供的过滤器。也就是我们没有在页面的action没有用j\_spring\_security\_check 做验证。我们自己编写控制器实现，这样可以做得更加灵活。

需要注意的是如果使用自定义控制器来做验证，需要注意的是，需要加入session策略。





在验证完成之后，加上红框的代码，这样才能保证那个session配置有效，否则，即使在配置文件做了配置也是无效的。

### 多入口配置

#### 概要说明

系统有时需要多个入口，比如系统需要使用手机登录。手机登录界面和电脑登录界面不一样，那么我们就需要定义多个登录界面。

实现原理如下：

1. spring security 支持自定义登录入口实现。
2. 实现自定义登录入口，根据输入的URL不同，自定义入口程序跳转到不同的登录界面。
3. 在系统超时的时候，页面需要跳转到相应的登录页面进行处理，比如登录手机页面，页面超时后，还是要能退回登录界面。

#### 实现方法

#### 修改配置

修改app-security.xml

增加如下配置：





修改entry-point-ref:为multipleAuthenticationLoginEntry 。

下面介绍一下配置的作用。



这段配置可以配置多个入口，目前配置了一个手机的，系统本身有一个是默认的，就是直接跳转到login.jsp页面。

如果需要实现其他的，可以参考mobileLoginEntry配置。

pattern : 表示路径中输入的字符，比如在路径中输入 <http://localhost:8080/bpm/mobile>就会自动跳转到<http://localhost:8080/bpm/mobileLogin.jsp>

directUrl:表示实际跳转到的地址。

如何在系统退出时跳转回相应的登录页面，目前是如此处理的。

1. 在系统登录时，自定义登录控制器写入名字为loginAction的cookie，值为先定好。在手机登录控制器中写入的值为 mobile。
2. 在退出时都退出到统一的 loginRedirect.ht页面。

这个页面判断cookie中是否有 loginAction这个值，如果有则取出，根据这个值在actionPageMap中得到页面地址，如果这个地址存在，则跳转到这个页面上。如果不存在则直接跳转到login.jsp上。

需要注意的是，我们没有直接使用springsecurity提供的过滤器对用户进行验证，而是直接通过springsecurity的api实现用户登录验证。

具体实现可以参考代码LoginController。

### 切换用户

#### 概要说明

在系统中经常有这样的需求，经理想查看下属的任务的工作情况，通过直接切换过去，不用在界面上进行登录，并可以切换回来。

Springsecurity 本身支持这种操作的实现。

#### 具体实现方法

1. 继承org.springframework.security.web.authentication.switchuser.SwitchUserFilter类，并重写dofilter方法。

实现方法如下：

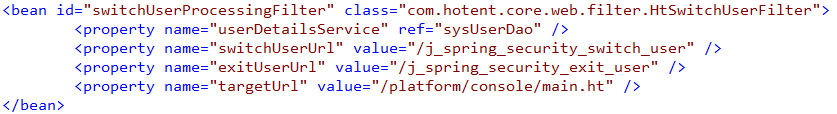


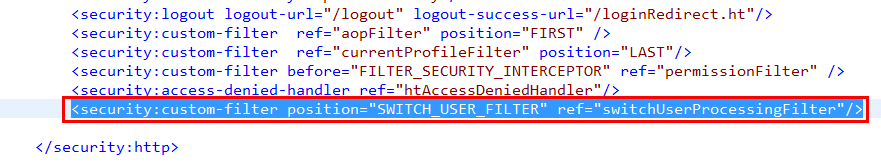
在切换之前在cookie中保存当前登录帐号。

退出切换时删除cookie标识。

1. 修改app-security.xml如下：

增加过滤器配置

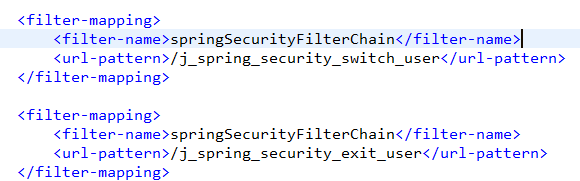




配置切换用户过滤器。

1. 修改web.xml如下

增加如下配置



1. 如何判定当前处于切换用户状态

在cookie中读取 origSwitch值是否为空，如果不为空表示当前处于切换状态。可以切换回去。

1. 如何进行切换？

切换时使用连接 <a href=’${ctx}/j\_spring\_security\_switch\_user?j\_username=帐号名’>切换</a>

账户名为需要切换到的用户帐号。

1. 如何退出切换？

退出切换是使用<a href=’${ctx}/j\_spring\_security\_exit\_user?j\_username=账户名’>退出切换</a>退出切换状态。

帐号为切换之前保留的帐号。

## 系统缓存

#### 概要说明

缓存技术在系统中经常使用，一些很频繁查询的数据，但是这些数据基本不变的情况下，我们就可以通过缓存技术将这些的数据进行缓存起来，减少对数据库的查询，一旦数据发生变化我们可以将这些数据从缓存中清除。以提系统运行的效率。

系统缓存有两种，一种是系统类的缓存，这种缓存和应用服务器相关，另外的一种是独立缓存服务器。

1. 系统内的缓存，优点是配置少，部署方便，性能上更好，缺点是如果系统分布式部署的情况，系统缓存同步将是一个很大的问题，这样就导致各个服务器无法保持同步，或者实现起来很困难。
2. 独立缓存服务，缺点是部署麻烦一点，性能比服务器内缓存要差。优点是适用于分布式部署，所有的应用服务器使用的是同一份缓存，因此不存在同步的问题。

#### 系统实现

我们经过研究发现，缓存的操作也就那么几类，增加，删除，查询，清除。因此我们可以抽象出一个接口，实现上述的几种操作。然后用不同的实现，基于不同的缓存技术实现这个接口，从而实现不同的缓存策略。

在程序中，程序只需对这个接口进行操作而不需要知道，缓存具体是如何实现的。

这样就达到了我们的隔离目的。

在我们的平台中，缓存的实现有两种，一种是基于价简单的hashmap的实现，一种是基于MemcachedCache的实现。

实现1：

com.hotent.core.cache.impl.MemoryCache

实现2:

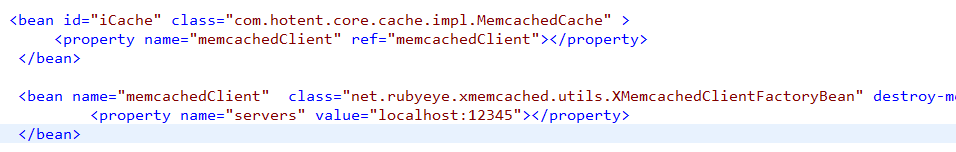
com.hotent.core.cache.impl.MemcachedCache

在app-beans.xml中

如果需要使用MemoryCache

配置如下：

使用MemcachedCache



#### 如何选择缓存策略？

1. 如果bpmx3平台为独立部署，那么使用系统内缓存就好了。
2. 如果涉及到分布式部署请使用独立的缓存。

#### 如何开发自己的缓存？

1. 实现ICache接口。
2. 将实现的接口类替换现有的配置类，**注意ID一定为 “iCache”**。

#### 如何安装Memacached服务器？

##### 下载

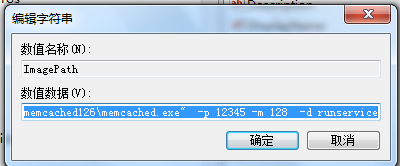
<http://code.jellycan.com/files/memcached-1.2.6-win32-bin.zip>

##### 安装

解压之后放在硬盘的目录下，如：D:\memcached . 然后在运行中输入cmd进入命令行，进入到Memcached.exe 所在的目录，例如：D:\memcached。

使用命令行执行。

到memcached根目录  
1、安装  
memcached.exe –d install   
  
2、启动  
memcached.exe -d start  
  
此时memcached已经注册为开机启动服务  
  
完成安装。  
  
经实验使用命令修改端口无效，相应的解决方案是修改注册表：  
  
通过修改注册表来达到这个修改端口的目的。  
  
在 HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\memcached Server 下面找到一个 ImagePath 的字符串项，正好是服务的执行路径的字符串，双击该串，在后面加入 -l 192.168.1.135 -m 45 -p 12345 （访问ip为：192.168.1.135 使用45M内存，12345为端口），再启动服务。



##### 命令的参数说明

-p 监听的端口  
-l 连接的IP地址, 默认是本机  
-d start 启动memcached服务  
-d restart 重起memcached服务  
-d stop|shutdown 关闭正在运行的memcached服务  
-d install 安装memcached服务  
-d uninstall 卸载memcached服务  
-u 以的身份运行 (仅在以root运行的时候有效)  
-m 最大内存使用，单位MB。默认64MB  
-M 内存耗尽时返回错误，而不是删除项  
-c 最大同时连接数，默认是1024  
-f 块大小增长因子，默认是1.25  
-n 最小分配空间，key+value+flags默认是48  
-h 显示帮助

##### 验证是否安装成功

使用telnet到memcached端口。

例如：

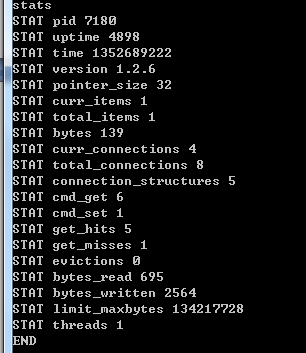
Memcached使用的端口为12345

在命令行窗口中使用如下命令：

telnet localhost 12345

如果能连接则表示memcache已启动。

输入stats 命令：



可以查看memcache的状态信息。

## 消息队列自定义处理

### 概要说明

之前对消息队列处理是硬编码的，对遇到类型的消息就进行处理。需要修改方法，对原来的代码具有依赖性和侵略性。

原来的处理方式如下：

**public** **void** sendMessage(Object model) **throws** Exception {

MessageEngine mSender = (MessageEngine) AppUtil.getBean(MessageEngine.**class**);

//发送邮件

**if** (model **instanceof** MailModel) {

//处理发送邮件消息实现

}

//短消息

**else** **if** (model **instanceof** SmsMobile) {

//处理发送短消息实现

}

//内部消息

**else** **if** (model **instanceof** InnerMessage) {

//处理内部消息实现

}

//如果其它类型的消息又要进行处理 。 这样具有依赖和侵略性

}

这样如果需要新增一种消息的处理不得不修改源码。我们将发送消息使用接口处理，针对不同的消息使用不同的处理类，然后这些处理类可以通过spring的配置，手工加进去。

### 实现方式

如果按照上面说的方式进行处理多种消息会对原来的代码进行修改，现在就只需要留出接口，自己对多种消息可以通过配置、接口进行实现。

现在处理方式：

// 处理消息

**private** Map<String, IJmsHandler> handlers = **new** HashMap<String, IJmsHandler>();

**public** **void** setHandlers(Map<String, IJmsHandler> handlers) {

**this**.handlers = handlers;

}

/\*\*

\* 发送消息

\*

\* **@param** model

\* 发送的对象

\* **@throws** Exception

\*/

**public** **void** sendMessage(Object model) **throws** Exception {

**if** (BeanUtils.*isNotEmpty*(handlers) && BeanUtils.*isNotEmpty*(model)) {

handlers.get(model.getClass().getName()).handMessage(model);

} **else** {

**throw** **new** Exception("Object:[" + model + "] is not entity Object ");

}

}

配置文件修改如下：

/resources/app-jms.xml 该配置文件配置说明：

<!-- 消息消费者 -->

<bean name="messageConsumer" class="com.hotent.core.jms.MessageConsumer">

<property name="handlers">

<map>

<entry key="com.hotent.core.model.MailModel">

<bean class="com.hotent.platform.service.jms.impl.MailHandler"></bean>

</entry>

<entry key="com.hotent.core.model.SmsMobile">

<bean class="com.hotent.platform.service.jms.impl.SmsHandler"></bean>

</entry>

<entry key="com.hotent.core.model.InnerMessage">

<bean class="com.hotent.platform.service.jms.impl.InnerHandler"></bean>

</entry>

<!—可以实现多种类型的发送消息-->

</map>

</property>

</bean>

找到 messageConsumer 这个bean 在<map>下配置 要实现的发送消息类及发送消息的Model。发送消息实现类需要实现IJmsHandler接口。

entry 的key 配置 发送消息的Model

bean 的class配置 发送消息的实现类(该类需要实现IJmsHandler接口)

### 二次开发指导

1. 实现一个接口处理器。

实现一个邮件代码如下：

比如实现发送邮件消息如下：

**public** **class** MailHandler **implements** IJmsHandler {

**private** **final** Log logger = LogFactory.getLog(MailHandler.**class**);

@Resource

**private** MessageEngine messageEngine;

@Override

**public** **void** handMessage(Object model) {

MailModel mailModel = (MailModel) model;

//具体发送邮件的方法

SimpleMailMessage message = **new** SimpleMailMessage();

message.setSubject(mailModel.getSubject());

message.setTo(mailModel.getTo());

message.setCc(mailModel.getCc());

message.setBcc(mailModel.getBcc());

message.setText(mailModel.getContent());

message.setSentDate(mailModel.getSendDate());

messageEngine.sendMail(message);

logger.debug("MailModel");

}

}

1. 在app-jms.xml中增加配置如下。



这样就能够自己处理自定义消息了。

## 如何自定义桌面栏目

### 概要说明

在平台首页，会显示主页信息，主页上显示了许多的栏目，比如我的待办，我的邮件等等。

如果客户希望自己将自己某些重要的信息也显示到页面上，在我们平台中如何实现呢？

在平台中实现方法如下：

1. 编写一个方法获取需要显示的数据。
2. 编写一个模版去展示这个数据。

下面具体举例说明

实现一个去cms系统获取头条信息的栏目。

### 实现步骤

#### 创建查询子系统信息方法

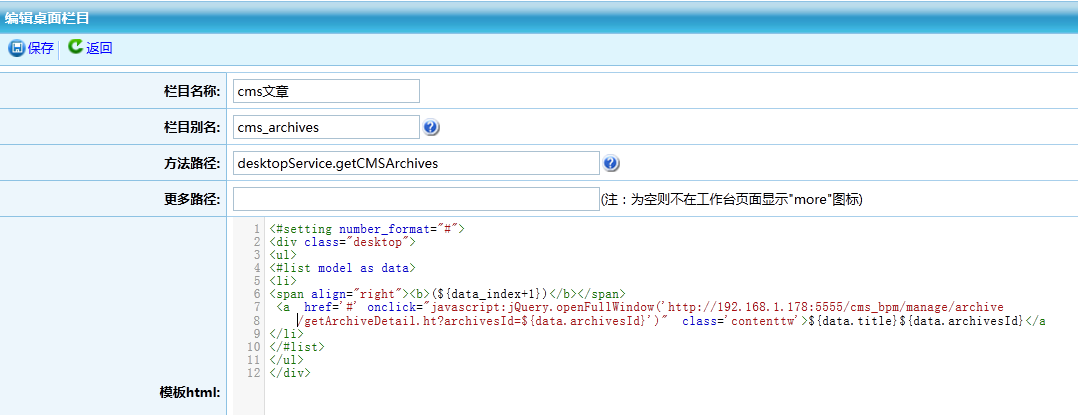
Bpmx3系统spring配置中设置了一个desktopService来处理桌面所需信息的查询，查询子系统信息的方法也也创建在DesktopService这个类中，代码如下：

|  |
| --- |
| **public** List<Map<String,Object>> getCMSArchives() {  JdbcHelper jdbcHelper =JdbcHelper.*getInstance*();  List<Map<String,Object>> archives = **new** ArrayList<Map<String,Object>>();  **try** {  jdbcHelper.init("cms", "oracle.jdbc.OracleDriver", "jdbc:oracle:thin:@localhost:1521:xe", "cms\_bpm", "cms\_bpm");  jdbcHelper.setCurrentDb("cms");  String sql="select title,archivesId from cms\_archives";  archives= jdbcHelper.queryForList(sql, **new** HashMap());  } **catch** (Exception e) {  e.printStackTrace();  }  **return** archives.subList(0, 9);  } |

通过jdbcHelper创建与cms数据库的连接，然后编写查询语句，通过jdbcHelper中得到查询方法查询到需要的数据。

#### 添加桌面栏目

在bpmx3系统的【系统管理】🡪【桌面管理】🡪【桌面栏目】页面中单击【添加】，根据需求填写：



方法路径：8.13.2.1中创建的方法，desktopService为spring配置id。

模板html：

模版是使用freemaker来编写的，编写模版需要遵照freemaker的语法。

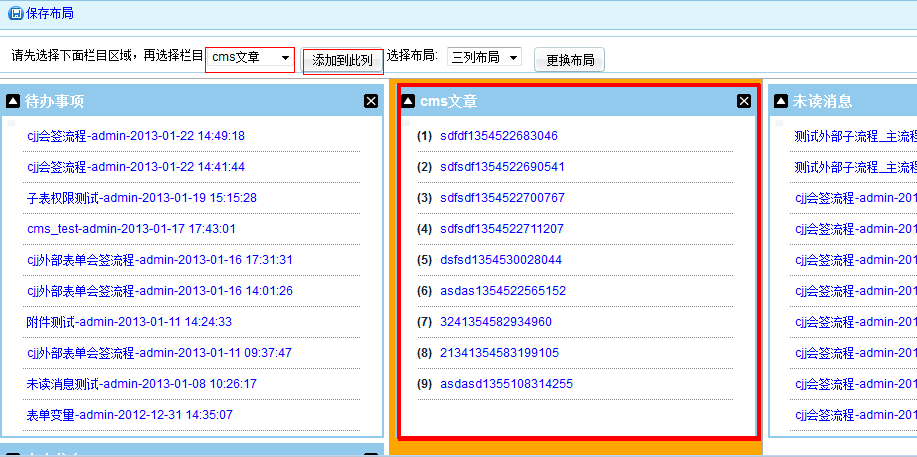
其中**model**为方法返回的数据对象，这个对象可以一个对象实例或者List对象例。

编写模版时需要根据具体的情况进行编写。

<a>标签: 单击显示列表某列时弹出的详细信息页，其url地址为子系统显示数据详细信息的地址。

#### 更改桌面布局

在bpmx3系统的【个人办公】🡪【桌面布局设置】页面，选择新创建的桌面栏目[cms文章]，单击【添加到此列】显示出cms文章列表，单击任一条数据即可以查看文章的详细信息。

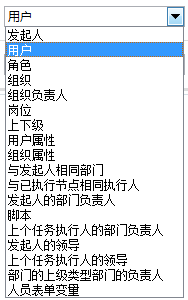




保存布局后即可在主页显示新增的栏目信息。

## 如何自定义开发平台支持的用户类型

平台系统目前支持如下的用户类型选择。

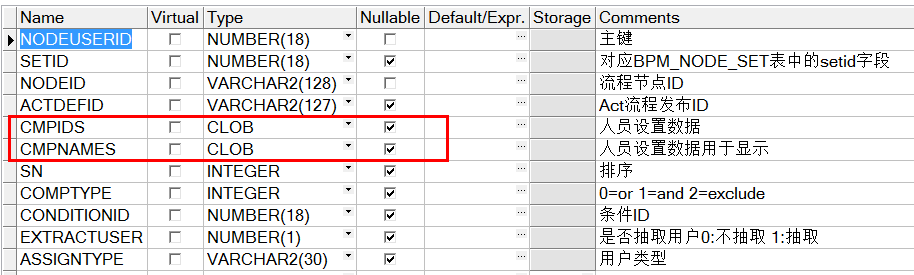


目前以支持十八种类型的选择，这些类型应该说是可以满足大部分的需求。

有些复杂的需求，可以使用脚本编写。但是一般用户还是希望通过界面直接进行设置。系统支持用户通过二次开发来增加用户类型。

实现方式介绍如下：

1.在用户处设置用户类型即相关的参数，这些参数一般存放在表BPM\_NODE\_USER表。



的cmpids,cmpnames 字段中。cmpids 存储设置的值，cmpnames存储显示的数据。

2.用户类型的算法是通过实现com.hotent.platform.service.bpm.IBpmNodeUserCalculation接口来处理的。

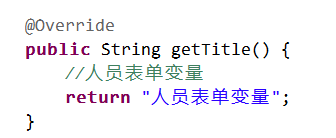
我们在这里使用了策略模式。用户只需实现该接口就可以在下拉列表中增加一种运算类型，这个我们配置在app-beans.xml中。



如果我们实现了该接口，就可以将这个策略通过如上的方式注入进来。那么我们就可以看到下拉框会增加相应的用户类型。

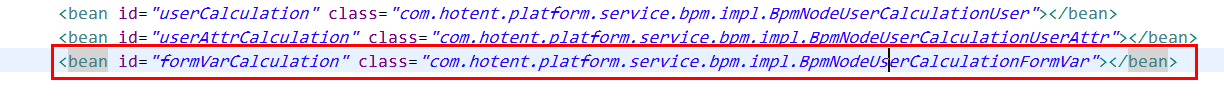
以BpmNodeUserCalculationFormVar为例。

这个接口有个：



这个就是下拉框显示的名称。

增加了这个类在app-beans.xml中注册这个用户类型。



同时注册到：



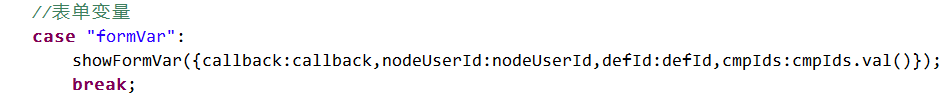
这个key是下拉框的key。这个key会保存到bpm\_node\_user表中。

需要注意的是如果增加的算法，不需要设置参数界面，例如流程发起人。

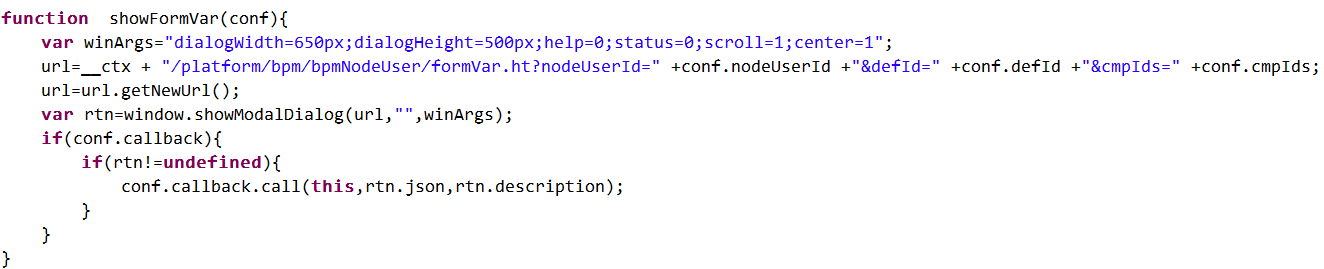
那么我们就完成了增加用户类型的操作。如果需要设置参数那么需要修改js文件如下：

nodeUserSelector.jsp文件的selectCmp函数：

增加如下代码：



和



这样就可以了。

具体接口的实现请参考代码：

