业务系统整合流程方案

# 需求

在业务系统开发时，业务需要和流程挂钩。这个时候我们希望对业务系统做最小的修改，就能实现和流程的整合。比如我们需要开发CRM管理系统。合同希望走流程，我们之前已经将合同部分开发完成，希望加入流程部分。

这个时候有两种方案：

1.业务系统独立部署，我们使用web服务的方案处理。

2.业务系统和平台部署到一起，在业务系统中调用流程的API，这是本文档需要介绍的内容。

我们在业务代码中出现的流程的部分越少越好，整合更加简洁。

# 实现方式

业务系统整合流程其实主要分为3部分：

1.流程启动

2.流程审批

3.业务数据查看

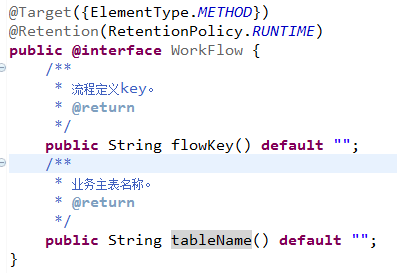
在业务系统中，我们一般调用业务代码的做数据保存。

比如在添加数据，这个时候我们启动流程。

在更新数据的时候我们审批流程。

我们使用统一的方式处理流程。我们定义一个注解，定义在需要启动或审批流程方法上。

注解定义为：



我们使用SPRING AOP拦截这些业务方法，只有定义了WorkFlow注解我们才对流程进行处理。

拦截器切面定义为：

com.hotent.platform.service.bpm.BpmAspect

拦截器配置为，配置到app-resource.xml

<bean id="bpmAspect" class="com.hotent.platform.service.bpm.BpmAspect"/> <aop:aspect ref="bpmAspect" order="100">

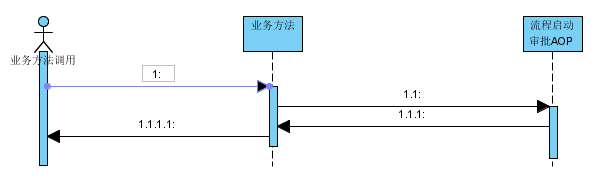
<aop:pointcut expression="execution(\* com.hotent.platform.service..\*.\*(..))" id="pointCutBpm"/>

<aop:after method="doAfter" pointcut-ref="pointCutBpm" />

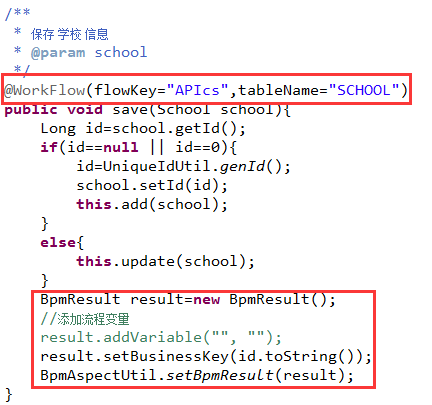
</aop:aspect>

拦截器切面的原理是：

我们通过拦截所有业务方法，使用BpmAspect进行切入，切面判断业务方法是否有WorkFlow注解，如果有此注解则做相关流程的相关操作。此切面是在业务方法之后执行操作。



这个AOP调用，业务方方法开发人员不用知道。当然这不是说开发人员什么都不用做。开发人员需要编写的业务代码如下：



我们在请求上下文获取参数：

我们从线程变量中获取request对象。

HttpServletRequest request= RequestContext.getHttpServletRequest();

在同一个方法上，我们怎么判断是启动流程，还是审批流程呢？

String executeType= request.getParameter("executeType");

我们在提交表单的时候，我们获取executeType：

这个参数有两个值：

|  |  |
| --- | --- |
| 值名称 | 备注 |
| start | 启动流程 |
| doNext | 审批流程 |

业务方法执行完成后需要一个BpmResult实例。提交给切面处理流程使用。

|  |  |
| --- | --- |
| 属性 | 备注 |
| businessKey | 业务主键 |
| tableName | 表名 |
| dataType | 业务主键数据类型  NUMBER  STRING |
| vars | 在业务执行过程中需要使用的业务变量 。 |

## 启动流程

在启动流程时，我们可以在注解上定义流程KEY。

|  |  |
| --- | --- |
| 获取数据 | 调用代码 |
| defKey | 在获取流程key时，我们可以优先从请求上下文获取，如果为空，我们可以从注解获取到。  从上下文获取:  String defKey= request.getParameter("flowKey")  WorkFlow. flowKey(); |
| 在切面中获取BpmResult | 在业务方法中设置流程执行结果。  BpmAspectUtil.setBpmResult  在切面中获取结果：  BpmAspectUtil. getBpmResult(); |
| startNode | 这个是流程直接跳转到的节点。 |

## 审批流程

|  |  |
| --- | --- |
| 获取数据 | 调用代码 |
| taskId | 在切面中  String taskId= request.getParameter("taskId");  在表单中需要传入任务id。  必填 |
| voteAgree | short voteAgree=RequestUtil.getShort(request, "voteAgree",(short)1);  投票意见。  0=弃权， 1=同意  2=反对， 3=驳回  4=追回  5=会签通过  6=会签不通过  默认为1 |
| opinion | 投票意见  String opinion=RequestUtil.getString(request, "opinion",""); |
| executors | String executors=RequestUtil.getString(request, "executors");  执行人。  格式如下:  [{nodeId:"userTask1",executors:[{id:"",name:""},{id:"",name:""}]}]  为一个JSON数组。数组每一个值为:  nodeId:节点ID  executors:执行人. |
| destTask | String destTask=RequestUtil.getString(request, "destTask","");  跳转的目标节点。 |
| isBack | Integer isBack=RequestUtil.getInt(request, "isBack",0);  0，正常跳转。  1，驳回  2，驳回到发起人。 |

## 保存草稿

在启动流程时，很多情况下需要保存草稿。

保存草稿动作在前台提交表单

int saveData=RequestUtil.getInt(request,"saveData",0);进行处理。

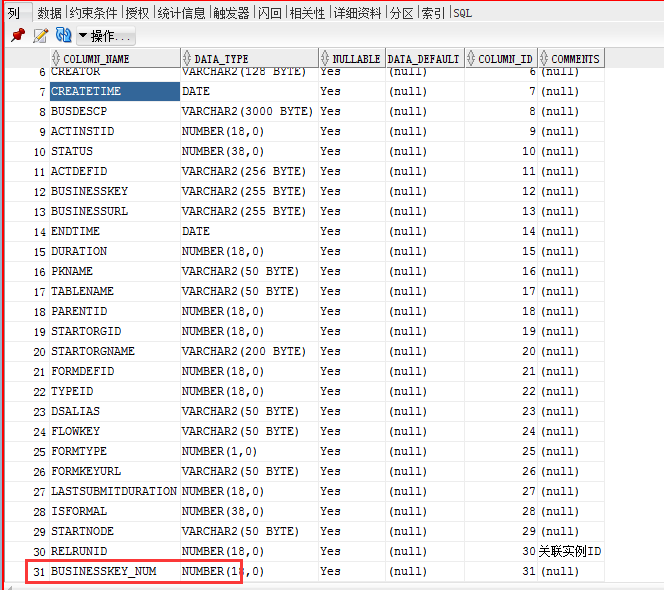
保存草稿动作：

1.将数据保存到自己的业务表。

2.将流程相关数据保存到BPM\_PRO\_RUN表中。

## 待办数据查询

流程实例表：BPM\_PROCESS\_RUN



查询代办

SELECT \* FROM 业务表 a,BPM\_PRO\_RUN b,ACT\_RU\_TASK c where a.主键=b.BUSINESSKEY\_NUM

AND c.PROC\_INST\_ID\_=b.BUS\_PROC\_INST\_ID

查询完成

SELECT \* FROM 业务表 a,BPM\_PRO\_RUN\_HIS b,BPM\_TASK c where a.主键=b.BUSINESSKEY\_NUM

AND c.PROC\_INST\_ID\_=b.BUS\_PROC\_INST\_ID AND B.status=2

AND B.CREATORID=用户ID

查询草稿

SELECT \* FROM 业务表 a,BPM\_PRO\_RUN b,BPM\_TASK c where a.主键=b.BUSINESSKEY\_NUM

AND c.PROC\_INST\_ID\_=b.BUS\_PROC\_INST\_ID AND B.status=4

AND B.CREATORID=用户ID

# 代码生成器

代码生成器针对新的整合方式进行了调整。

调整主要包括：

1.map文件的生成。

2.model 文件的生成。

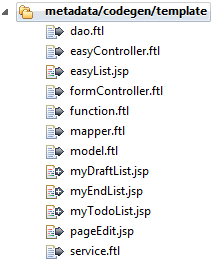
3.dao代码生成。

4.service生成。

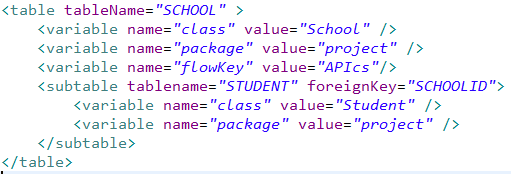
5.controller生成。

6.编辑页面的生成。

7.列表页面的生成包括（草稿，代办，结束）等页面。



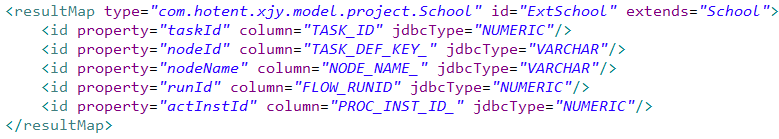
流程代码生成在配置文件中只需要添加一个变量。

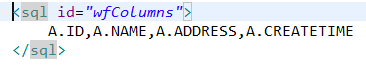


# 代码的调整

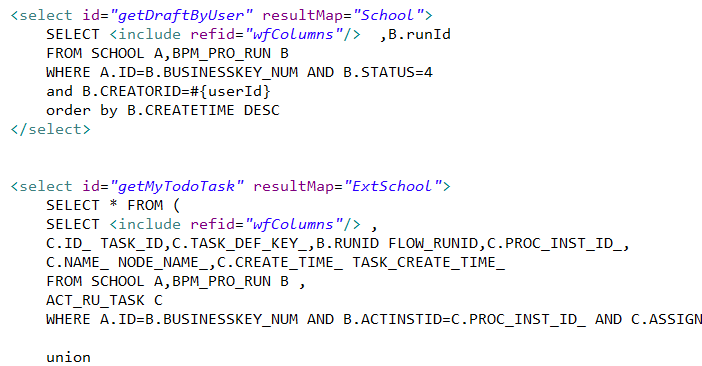
## mapper文件

流程扩展部分



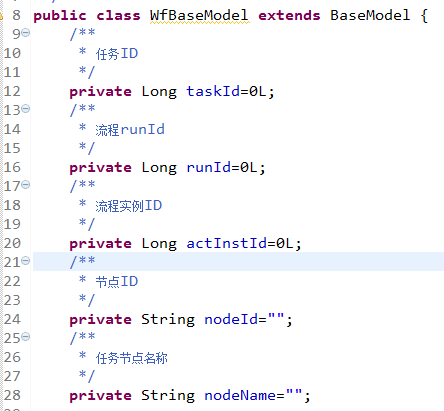


扩展了我的草稿，我的代办，我的结束流程。



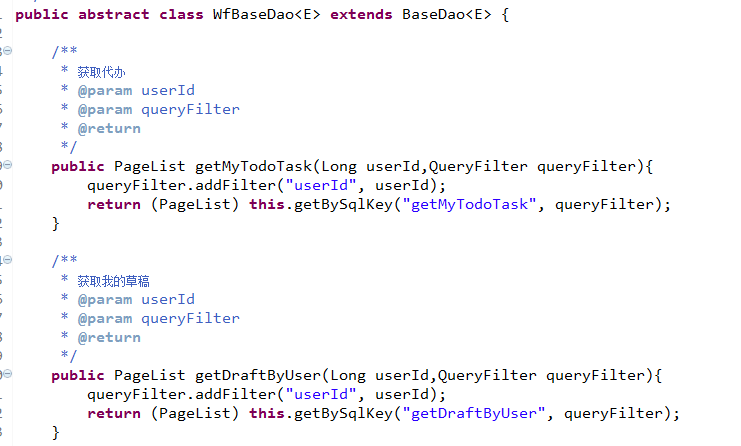
## model 层的扩展

普通的业务model继承BaseModel,工作流业务继承WfBaseModel。



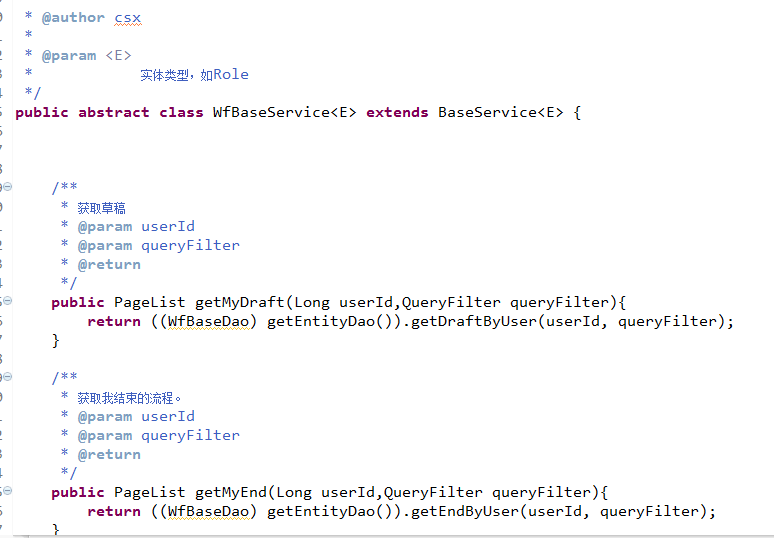
## Dao层扩展

WfBaseDao 继承baseDao，需要使用工作流的业务扩展此类。

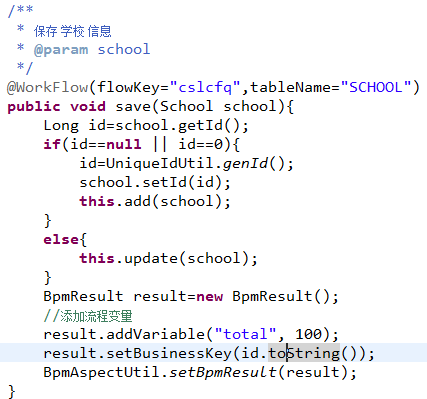


## SERVICE层代码扩展

涉及到流程业务的可以集成此类。



流程发起审批交互关键代码

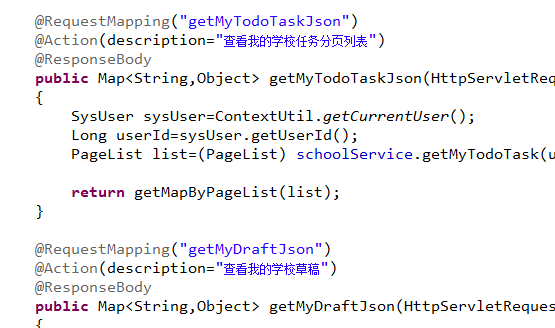


## 控制器层代码修改

编辑页面扩展了流程的东西。



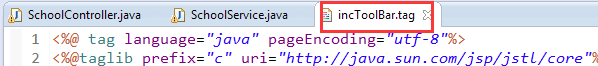
获取代办，已办，草稿等内容。



## 页面展示的修改

编辑页面，出了表单原本的业务数据，我们也会在流程中添加一些流程相关的必要的数据。

这些参数可以参考：



为了快速开发添加了

草稿，代办，终结等三个模版。