**ECMP（企业云管理平台）**

**工作流集成文档**

四川虹信软件股份有限公司

【2018.7 V1.0】



目录

[一 实体Entity 3](#_Toc519012169)

[1.1 条件实体 3](#_Toc519012170)

[1.1.1 业务单据的条件实体基类BaseOrderCondition 3](#_Toc519012171)

[1.1.2 条件实体StoreRequestCondition 5](#_Toc519012172)

[1.2 注解和继承 7](#_Toc519012173)

[1.2.1 注解 7](#_Toc519012174)

[1.2.2 实现方法 7](#_Toc519012175)

[1.2.3 临时字段 7](#_Toc519012176)

[二 接口Api 9](#_Toc519012177)

[2.1 继承 9](#_Toc519012178)

[2.2 自定义执行人（非必要） 9](#_Toc519012179)

[2.3 流程启动前的检查事件（非必要） 9](#_Toc519012180)

[三 服务Service 11](#_Toc519012181)

[3.1 工作流业务逻辑FlowConditionService 11](#_Toc519012182)

[3.2 单据实体对应的业务逻辑 11](#_Toc519012183)

[3.2.1 注入工作流业务逻辑 11](#_Toc519012184)

[3.2.2 实现对应方法 11](#_Toc519012185)

[3.3 自定义执行人（非必要） 14](#_Toc519012186)

[3.4 流程启动前的检查事件（非必要） 15](#_Toc519012187)

[四 前端Web 16](#_Toc519012188)

[4.1 Controller 16](#_Toc519012189)

[4.2 EUI启动工作流 16](#_Toc519012190)

# 实体Entity

## 条件实体

### 业务单据的条件实体基类BaseOrderCondition

**public abstract class** BaseOrderCondition **implements** Serializable, IConditionPojo {  
 **private static final long *serialVersionUID*** = 1L;  
 */\*\*  
 \* 单据编号  
 \*/* **private** String **businessCode**;  
 */\*\*  
 \* 组织机构代码  
 \*/* **private** String **orgCode**;  
 */\*\*  
 \* 组织机构Id  
 \*/* **private** String **orgId**;  
 */\*\*  
 \* 组织机构路径  
 \*/* **private** String **orgPath**;  
 */\*\*  
 \* 工作说明  
 \*/* **private** String **workCaption**;  
 */\*\*  
 \* 业务名称  
 \*/* **private** String **name**;  
 */\*\*  
 \* 租户代码  
 \*/* **private** String **tenantCode**;  
  
 @ConditionAnnotaion(name = **"业务单号"**, rank = -10000, canSee = **false**)  
 **public** String getBusinessCode() {  
 **return businessCode**;  
 }  
  
 **public void** setBusinessCode(String businessCode) {  
 **this**.**businessCode** = businessCode;  
 }  
  
 @ConditionAnnotaion(name=**"组织机构代码"**,rank = -10000,canSee=**false**)  
 **public** String getOrgCode() {  
 **return orgCode**;  
 }  
  
 **public void** setOrgCode(String orgCode) {  
 **this**.**orgCode** = orgCode;  
 }  
  
 @ConditionAnnotaion(name=**"组织机构id"**,rank = -10000,canSee=**false**)  
 **public** String getOrgId() {  
 **return orgId**;  
 }  
  
 **public void** setOrgId(String orgId) {  
 **this**.**orgId** = orgId;  
 }  
  
 @ConditionAnnotaion(name=**"组织机构路径"**,rank = -10000,canSee=**false**)  
 **public** String getOrgPath() {  
 **return orgPath**;  
 }  
  
 **public void** setOrgPath(String orgPath) {  
 **this**.**orgPath** = orgPath;  
 }  
  
 @ConditionAnnotaion(name=**"工作说明"**,rank = -10000,canSee=**false**)  
 **public** String getWorkCaption() {  
 **return workCaption**;  
 }  
  
 **public void** setWorkCaption(String workCaption) {  
 **this**.**workCaption** = workCaption;  
 }  
  
 @ConditionAnnotaion(name=**"业务名称"**,rank = -10000,canSee=**false**)  
 **public** String getName() {  
 **return name**;  
 }  
  
 **public void** setName(String name) {  
 **this**.**name** = name;  
 }  
  
 @ConditionAnnotaion(name=**"租户代码"**,rank = -10000,canSee=**false**)  
 **public** String getTenantCode() {  
 **return tenantCode**;  
 }  
  
 **public void** setTenantCode(String tenantCode) {  
 **this**.**tenantCode** = tenantCode;  
 }  
  
 */\*\*  
 \* 条件表达式初始化，提供给表达式做初始化验证，  
 \* 结合具体业务实现  
 \*/* @Override  
 **public void** init() {  
 setBusinessCode(**""**);  
 setOrgId(**""**);  
 setOrgCode(**""**);  
 setOrgPath(**""**);  
 setName(**""**);  
 setWorkCaption(**""**);  
 setTenantCode(**""**);  
 }  
  
 */\*\*  
 \* 自定义逻辑方法，  
 \* 场景：应用于条件表达式POJO的额外定义属性值初始化  
 \*/* @Override  
 **public void** customLogic() {  
 }  
}

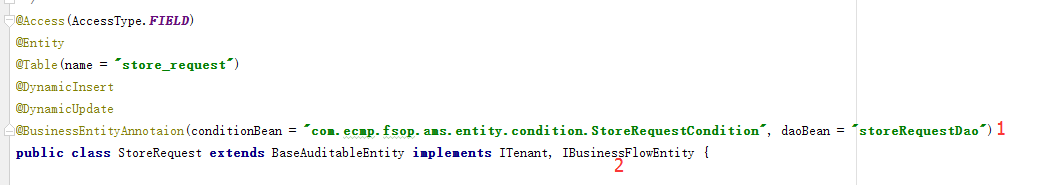
1. 实现序列化接口Serializable和工作流提供的条件Pojo接口
2. 字段是为了与工作流使用的字段相匹配
3. Get方法注解ConditionAnnotaion：工作流提供的条件表达式自定义注解

### 条件实体StoreRequestCondition

继承条件实体基类BaseOrderCondition，同时可以添加自定义的字段，字段的get方法上添加ConditionAnnotaion注解

**public class** StoreRequestCondition **extends** BaseOrderCondition {  
  
 */\*\*  
 \* 申请类型  
 \*/* **private int requestType**;  
  
 */\*\*  
 \* 条件表达式初始化，提供给表达式做初始化验证，  
 \* 结合具体业务实现  
 \*/* @Override  
 **public void** init() {  
 setRequestType(0);  
 **super**.init();  
 }  
  
 */\*\*  
 \* 自定义逻辑方法，  
 \* 场景：应用于条件表达式POJO的额外定义属性值初始化  
 \*/* @Override  
 **public void** customLogic() {  
 **super**.customLogic();  
 }  
  
 @ConditionAnnotaion(name = **"申请类型"**)  
 **public int** getRequestType() {  
 **return requestType**;  
 }  
  
 **public void** setRequestType(**int** requestType) {  
 **this**.**requestType** = requestType;  
 }  
}

## 注解和继承



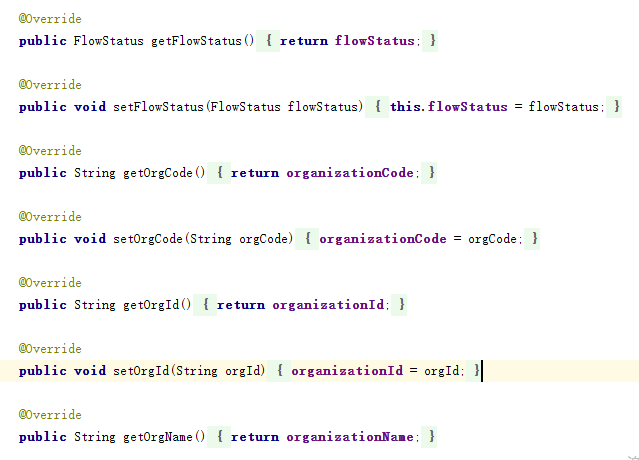
### 注解

在原有注解的基础上新增一个条件表达式自定义注解@BusinessEntityAnnotaion

* 1. conditionBean：条件实体路径
  2. daoBean：填入值为对应dao的小驼峰写法，即首字母小写

### 实现方法

在原本普通实体写法的基础上实现工作流的实体基类IBusinessFlowEntity，并修改实现的get和set方法为实体中对应字段



### 临时字段

如果IBusinessFlowEntity中的部分字段在实体中没有对应的字段，可在实体中创建对应的临时字段，如流程处理优先级字段priority

/\*\*

\* 流程处理优先级(默认0)

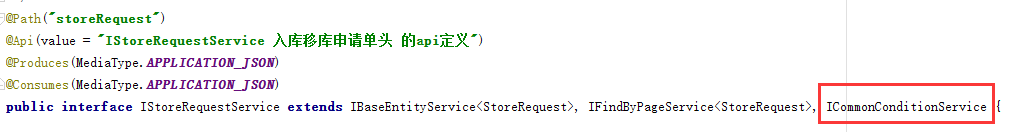
\*/

@Transient

@JsonSerialize

private int priority = 0;

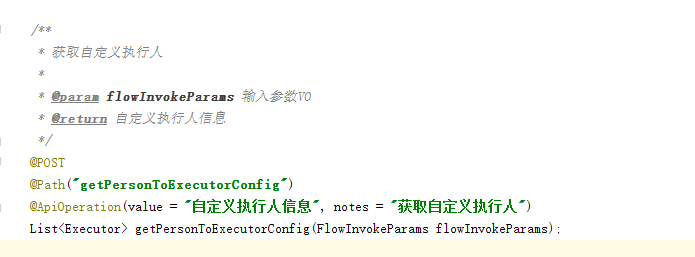
# 接口Api



## 继承

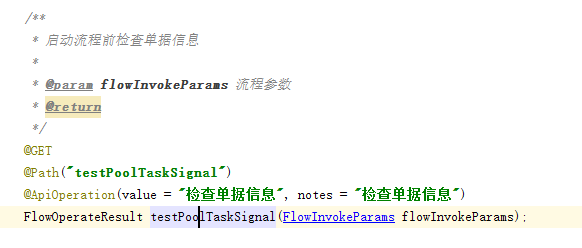
继承工作流提供的业务系统条件通用服务API接口定义ICommonConditionService

## 自定义执行人（非必要）



接口的返回类型和参数类型固定，其它的可以自定义

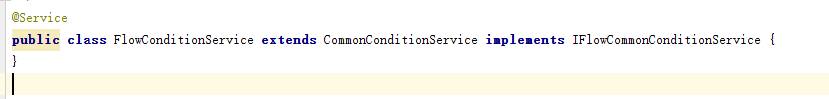
## 流程启动前的检查事件（非必要）



接口的返回类型和参数类型固定，其它的可以自定义

# 服务Service

## 工作流业务逻辑FlowConditionService



1. 继承工作流提供的通用客户端条件表达式服务CommonConditionService
2. 实现工作流提供的条件通用服务API接口IFlowCommonConditionService

## 单据实体对应的业务逻辑

### 注入工作流业务逻辑

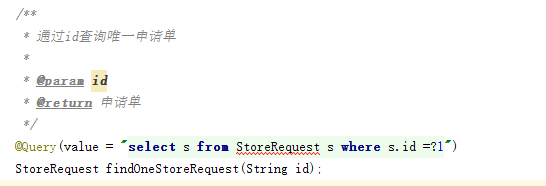


### 实现对应方法

*/\*\*  
 \* 获取条件POJO属性说明  
 \*  
 \** ***@param businessModelCode*** *业务实体代码  
 \** ***@param all*** *是否查询全部  
 \** ***@return*** *POJO属性说明Map  
 \** ***@throws*** *ClassNotFoundException 类找不到异常  
 \*/*@Override  
**public** Map<String, String> properties(String businessModelCode, Boolean all) **throws** ClassNotFoundException {  
 **return flowConditionService**.properties(businessModelCode, all);  
}  
  
*/\*\*  
 \* 获取条件POJO属性初始化值键值对  
 \*  
 \** ***@param businessModelCode*** *业务实体代码  
 \** ***@return*** *POJO属性说明Map  
 \** ***@throws*** *ClassNotFoundException 类找不到异常  
 \** ***@throws*** *InvocationTargetException 目标类解析异常  
 \** ***@throws*** *InstantiationException 实例异常  
 \** ***@throws*** *IllegalAccessException 访问异常  
 \** ***@throws*** *NoSuchMethodException 没有方法异常  
 \*/*@Override  
**public** Map<String, Object> initPropertiesAndValues(String businessModelCode) **throws** ClassNotFoundException, InvocationTargetException, InstantiationException, IllegalAccessException, NoSuchMethodException {  
 **return flowConditionService**.initPropertiesAndValues(businessModelCode);  
}  
  
*/\*\*  
 \* 获取条件POJO属性键值对  
 \*  
 \** ***@param businessModelCode*** *业务实体代码  
 \** ***@param id*** *单据id  
 \** ***@param all*** *\** ***@return*** *POJO属性说明Map  
 \** ***@throws*** *ClassNotFoundException 类找不到异常  
 \** ***@throws*** *InvocationTargetException 目标类解析异常  
 \** ***@throws*** *InstantiationException 实例异常  
 \** ***@throws*** *IllegalAccessException 访问异常  
 \** ***@throws*** *NoSuchMethodException 没有方法异常  
 \*/*@Override  
**public** Map<String, Object> propertiesAndValues(String businessModelCode, String id, Boolean all) **throws** NoSuchMethodException, InvocationTargetException, IllegalAccessException, ClassNotFoundException, InstantiationException {  
 StoreRequestCondition condition = **new** StoreRequestCondition();  
 condition.init();  
 StoreRequest order = **dao**.findOneStoreRequest(id);  
 **if** (Objects.*nonNull*(order)) {  
 BeanUtils.*copyProperties*(order, condition);  
 condition.setRequestType(order.getRequestType().ordinal());  
 }  
 **return new** ExpressionUtil<StoreRequestCondition>().getPropertiesAndValues(condition, all);  
}  
  
*/\*\*  
 \* 重置单据状态  
 \*  
 \** ***@param businessModelCode*** *业务实体代码  
 \** ***@param id*** *单据id  
 \** ***@param status*** *状态  
 \** ***@return*** *返回结果  
 \** ***@throws*** *ClassNotFoundException 类找不到异常  
 \** ***@throws*** *InvocationTargetException 目标类解析异常  
 \** ***@throws*** *InstantiationException 实例异常  
 \** ***@throws*** *IllegalAccessException 访问异常  
 \** ***@throws*** *NoSuchMethodException 没有方法异常  
 \*/*@Override  
**public** Boolean resetState(String businessModelCode, String id, FlowStatus status) **throws** NoSuchMethodException, InvocationTargetException, IllegalAccessException, ClassNotFoundException, InstantiationException {  
 **return flowConditionService**.resetState(businessModelCode, id, status);  
}  
  
*/\*\*  
 \* 获取条件POJO属性键值对  
 \*  
 \** ***@param businessModelCode*** *业务实体代码  
 \** ***@param id*** *单据id  
 \** ***@return*** *POJO属性说明Map  
 \*/*@Override  
**public** Map<String, Object> businessPropertiesAndValues(String businessModelCode, String id) **throws** Exception {  
  
 **return null**;  
}

**说明：**

1. 大部分实现的方法通过注入的工作流业务逻辑类flowConditionService均可直接调用，无需写额外的代码逻辑
2. propertiesAndValues方法中需要进行各个单据的自定义处理，通过单据id查询出对应的单据，再给条件实体进行赋值，最后构造条件POJO属性键值对。其中，需要注意的是，经过工作流之后，工作流在调用当前模块的实现类时，不会带上租户代码，因此不能直接使用平台封装的findOne方法，需要自己重写对应的查询方法，例如在dao中自定义查询：

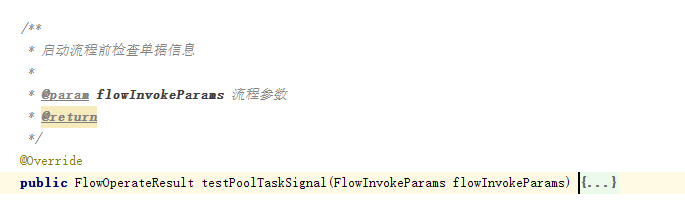


## 自定义执行人（非必要）

*/\*\*  
 \* 获取自定义执行人  
 \*  
 \** ***@param flowInvokeParams*** *输入参数VO  
 \** ***@return*** *自定义执行人信息  
 \*/* @Override  
 @Transactional(propagation = Propagation.***REQUIRES\_NEW***)  
 **public** List<Executor> getPersonToExecutorConfig(FlowInvokeParams flowInvokeParams) {  
 String businessId = flowInvokeParams.getId();  
 List<Executor> result = **new** ArrayList<Executor>();  
 **if** (StringUtils.*isNotEmpty*(businessId)) {  
 StoreRequest storeRequest = findOne(businessId);  
 **if** (storeRequest != **null**) {  
 *//获取申请单上的目标档案室id* String roomId = storeRequest.getTargetArchiveRoomId(); *//通过目标档案室id获取档案室管理员配置* List<ArchivesRoomManagers> archivesRoomManagersList = **archivesRoomManagersDao**.findByRoomId(roomId);  
 *//获取档案室管理员的id集合* List<String> employeeIds = **new** ArrayList<String>();  
 archivesRoomManagersList.forEach(m -> {  
 employeeIds.add(m.getManagerId());  
 });  
  
 Map<String, Object> paramsV2 = **new** HashMap();  
 paramsV2.put(**"employeeIds"**, employeeIds);  
 String url = Constants.*getBasicEmployeeGetexecutorsbyemployeeidsUrl*();  
 result = ApiClient.*getEntityViaProxy*(url, **new** GenericType<List<Executor>>() {  
 }, paramsV2);  
 }  
 }  
 **return** result;  
 }

参数的来源是工作流，我们需要用参数中的单据id来展开我们的逻辑

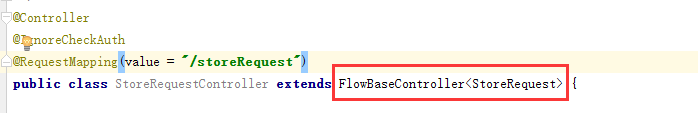
## 流程启动前的检查事件（非必要）



参数的来源是工作流，我们需要用参数中的单据id来展开我们的逻辑

# 前端Web

## Controller



需要继承FlowBaseController来调用其中工作流相关的方法

## EUI启动工作流

EUI已封装好工作流启动的方法，可直接调用



1. businessId：选择的单据的id
2. businessModelCode：单据实体的路径
3. url：Controller的相对路径（实质是调用FlowBaseController的startFlow方法）
4. afterSubmit：提交后的操作（多为刷新页面）