EIPSAAS 脚本事件说明

广州宏天软件股份有限公司

2020 年 11 月 03 日

修订记录

序号	修订人	修订时间	修订说明
1	何一帆	2020-11-03	创建

目 录

1 背景介绍

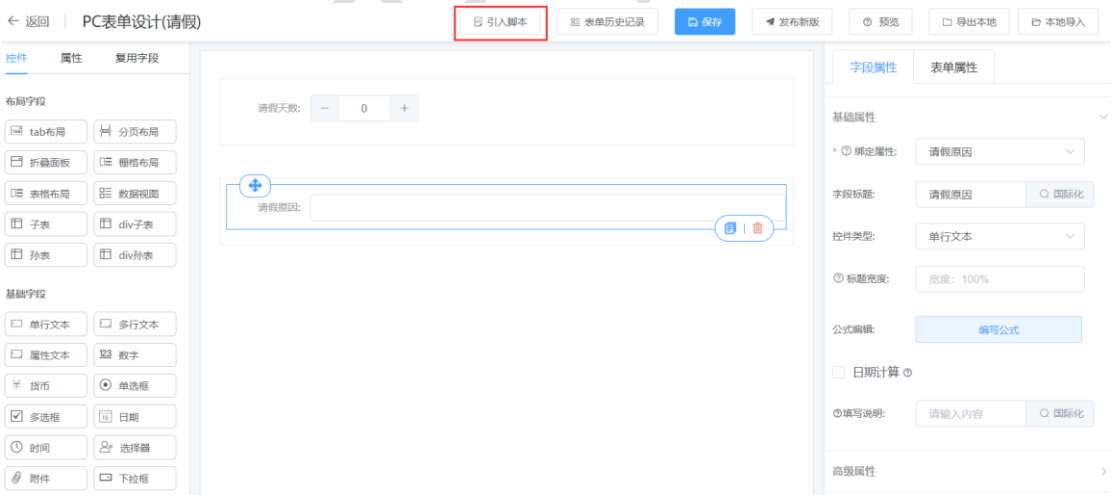
本产品为快速开发平台，为了用户可以灵活的实现自己的业务需求，平台内部很多很多环节都支持撰写脚本，但是这些脚本的执行顺序，脚本语言与规范，脚本撰写的注意事项都无法准确的通过系统配置页面了解到，故而撰写该文档协助产品购买方了解脚本与事件的使用。

2 脚本事件使用说明

2.1 表单相关

2.1.1 引入脚本

设计的表单支持编写 JS 逻辑代码，如下图所示，通过【引入脚本】按钮可以打开脚本编辑界面。

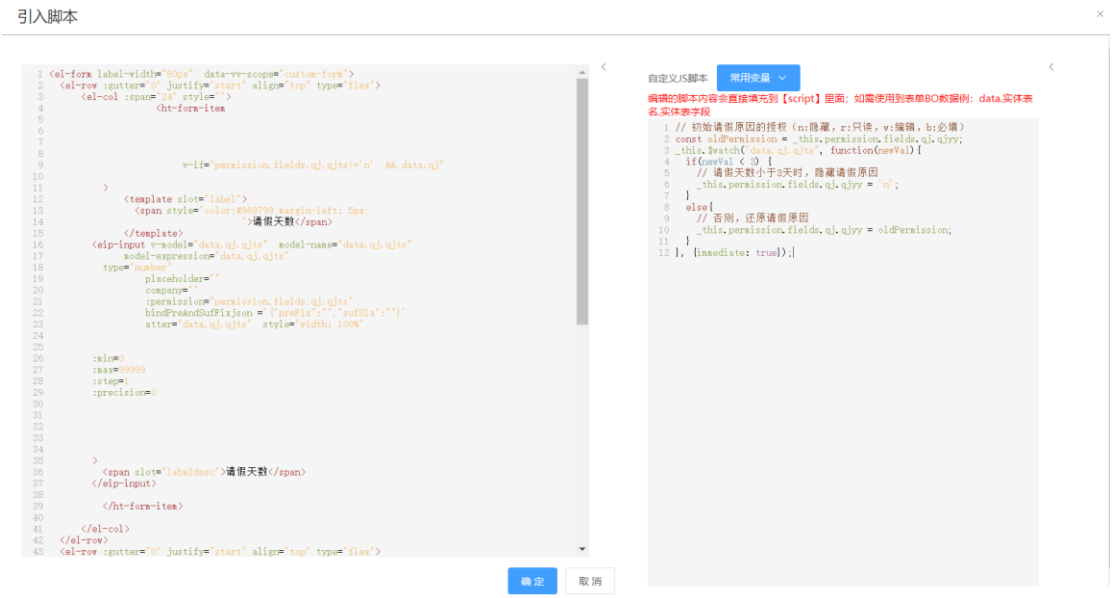


在脚本编辑界面，左侧为当前表单的页面 HTML 代码（可以理解为 Vue 开发时 template 标签中的部分），右侧为 JS 代码，可以按照 Vue 的语法编写表单逻辑代码，_this 为当前表单的 Vue 实例对象。

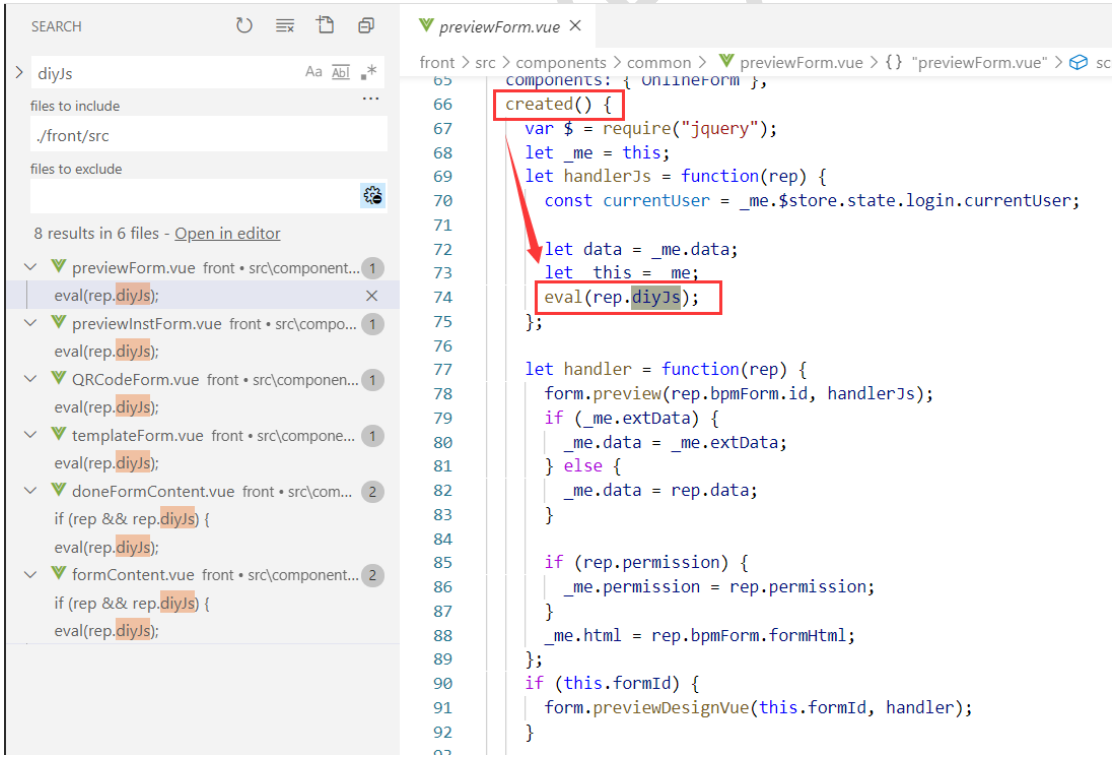
特别注意，在进行表单保存/发布新版这样的操作时，HTML 内容会通过模板重新生成，所以如果修改了 HTML 内容，保存表单后会导致之前修改的 HTML 丢失；JS 不会丢失。

另外我们也引入了 jQuery 库，可以在这里通过$编写相应代码，不过我们的建议是尽量

不要使用 jQuery。



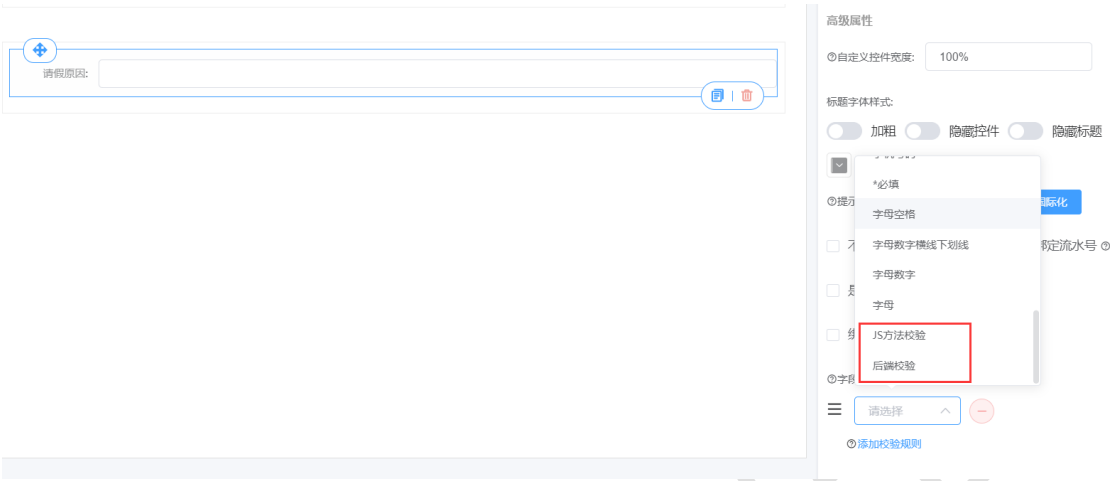
在表单渲染时，我们会如下图所示在 Vue 的 created 事件中挂载自定义 JS 代码，在上下文中定义了参数 data,_this，所以编写的 JS 代码中可以用到这些参数。



2.1.2 字段校验

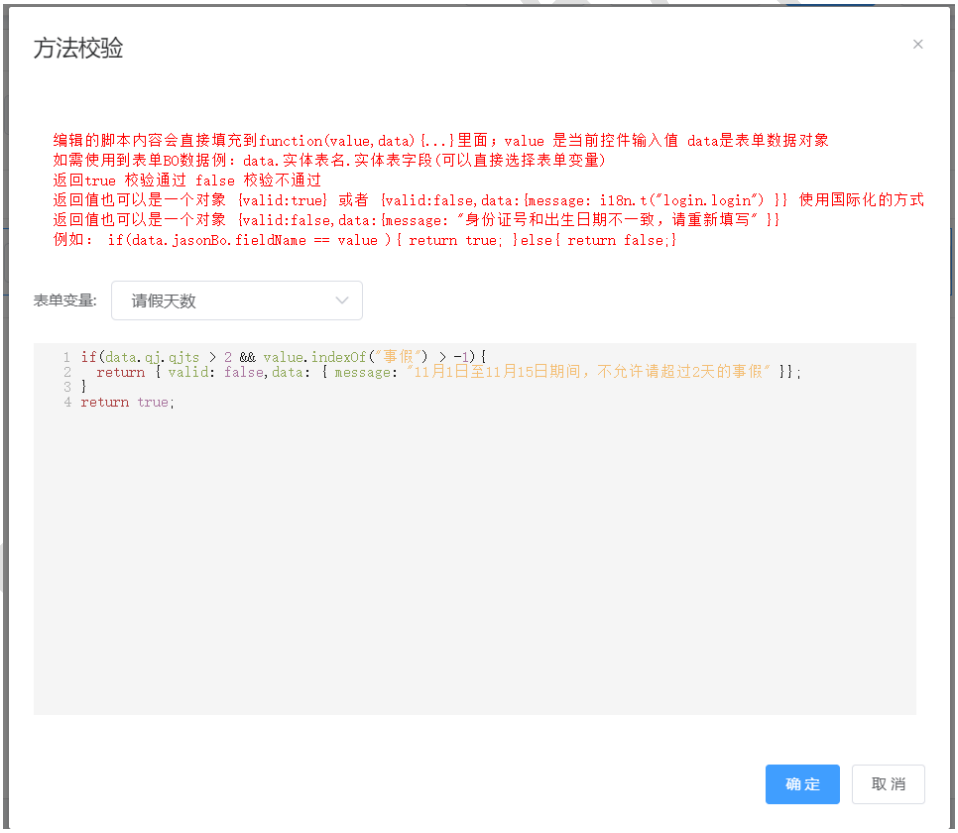
字段的校验集成了 vee-validate 框架，如下图所示，提供的校验规则大多配置起来比较

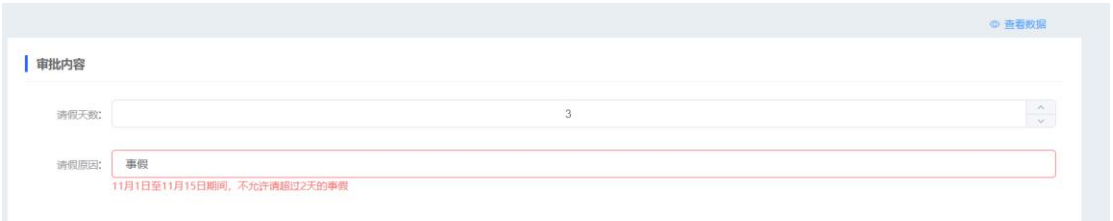
简单易懂，只有 JS 方法校验和后端校验配置较为复杂。



➢ JS 校验方法

在配置界面已经有详细的说明，可以参照示例进行配置。





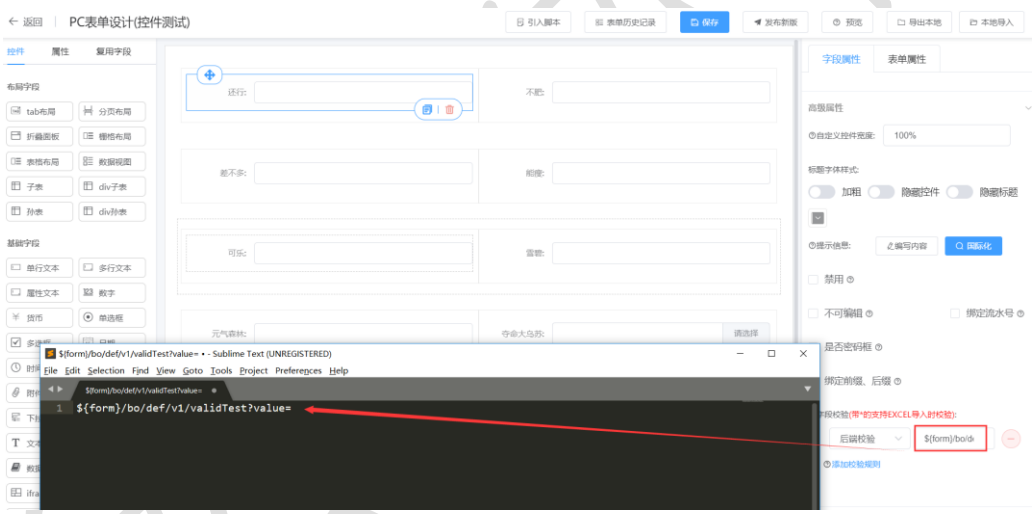
➢ 后端校验

后端校验可以将字段值传递给后端进行校验，校验结果再返回到前端。如下图所示，后端需要以 POST 格式发布为 Restful 接口，取值时从 URL 地址参数中以 value 为 key 进行取值，返回 Boolean 类型的校验结果。

BoDefController.java

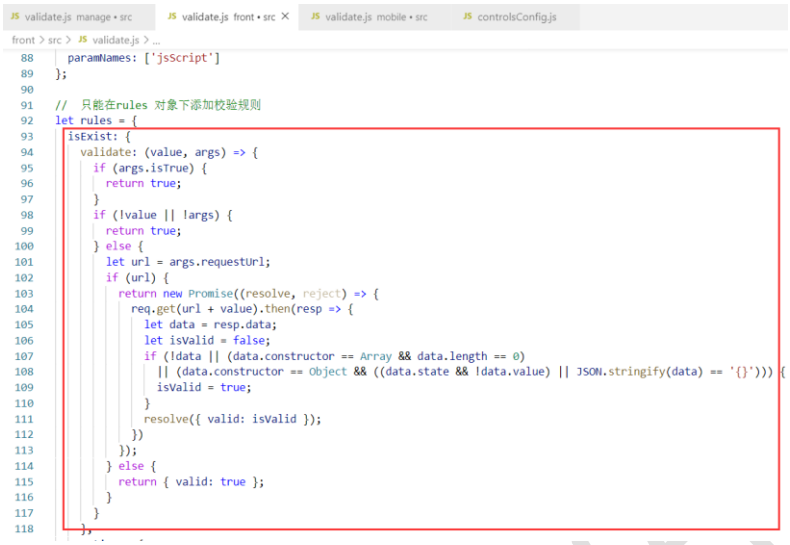
```
@RestController
@RequestMapping("/bo/def/v1/")
@Api(tags="业务对象定义")
@ApiGroup(group= {ApiGroupConsts.GROUP_FORM})
public class BoDefController extends BaseController<BoDefManager, BoDef> {
    @Resource
    BoEntManager boEntManager;
    @Resource
    BoDataHandler boDataHandler;
    @Resource
    FormMetaManager  bpmFormDefManager;
    @Resource
    BoAttributeManager boAttributeManager;

    @RequestMapping(value="validTest",method=RequestMethod.POST, produces = { "application/json; charset=utf-8" })
    @ApiOperation(value = "后端校验测试", httpMethod = "POST", notes = "后端校验测试")
    public Boolean validTest(HttpServletRequest request) throws Exception{
        String value = HttpUtil.getRequestParameter("value");
        return value.indexOf("1") > -1;
    }

```

前端配置时，添加 URL 参数 key 为 value，等号后面留空，在使用表单时会自动将字段的值追加到 URL 地址后面并请求后端的 Restful 接口。



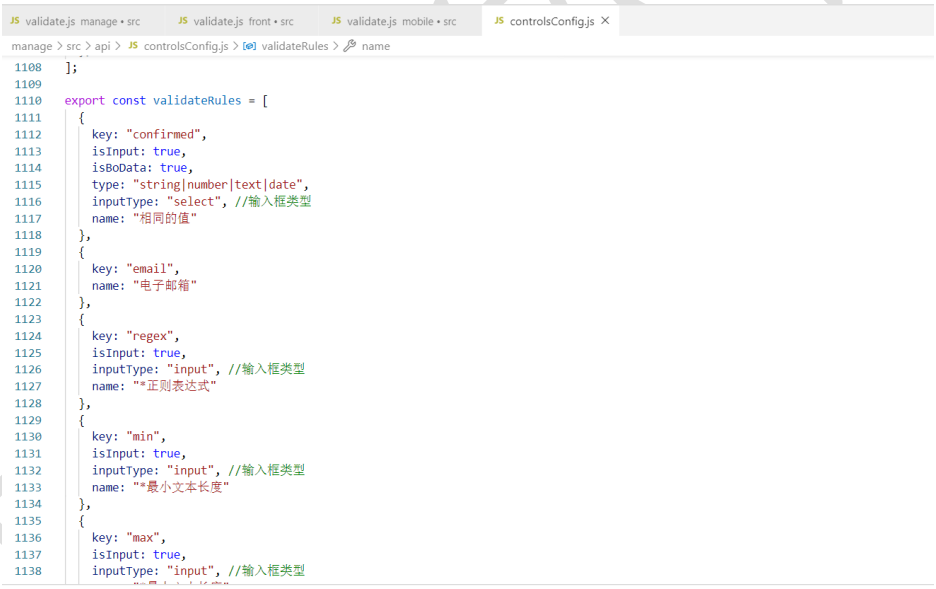
➢ 校验规则的扩展

如下图所示，添加新的校验规则需要在 validate.js 中在 rules 对象下扩展，新规则的定义可以参照原有规则。



```
validate.js manage • src | validate.js front • src × | validate.js mobile • src | controlsConfig.js
front > src > validate.js > ...
88    paramNames: ['jsScript']
89  };
90
91  //  只能在rules 对象下添加校验规则
92  let rules = {
93    isExist: {
94      validate: (value, args) => {
95        if (args.isTrue) {
96          return true;
97        }
98        if (!value || !args) {
99          return true;
100       } else {
101         let url = args.requestUrl;
102         if (url) {
103           return new Promise((resolve, reject) => {
104             req.get(url + value).then(resp => {
105               let data = resp.data;
106               let isValid = false;
107               if (!data || (data.constructor == Array && data.length == 0)
108                 || (data.constructor == Object && ((data.state && !data.value) || JSON.stringify(data) == '{}'))) {
109                 isValid = true;
110               }
111               resolve({ valid: isValid });
112             })
113           });
114         } else {
115           return { valid: true };
116         }
117       }
118     },
```

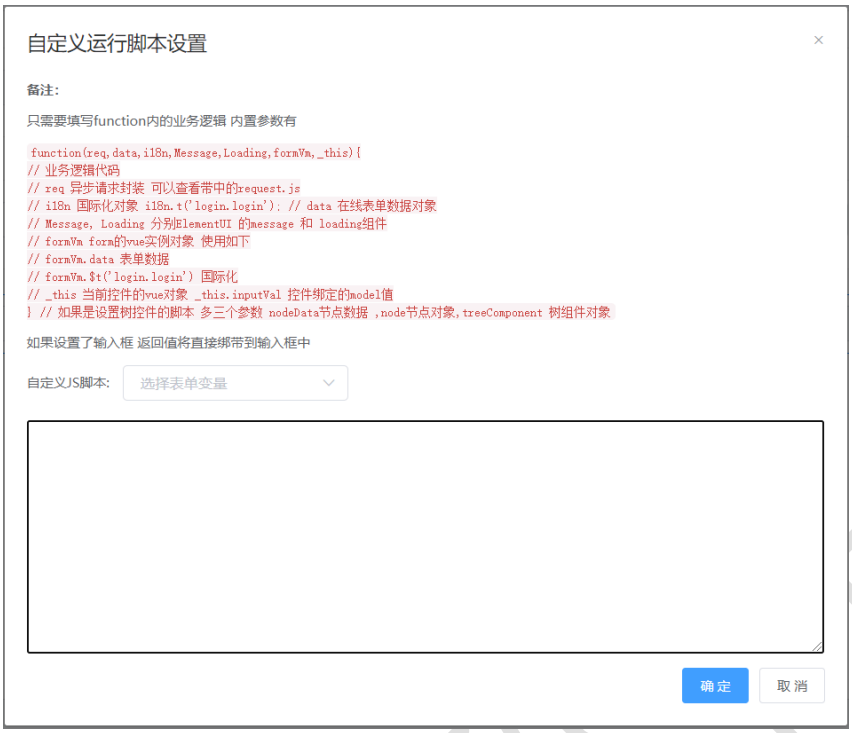
新增的规则还需要在表单设计时，在【字段校验】的下拉框中添加新的下拉选项，该选项的扩展需要在如下所示的 js 文件中添加选项，注意：添加的选项的 key 必须和 rules 中定义的规则名称一致。



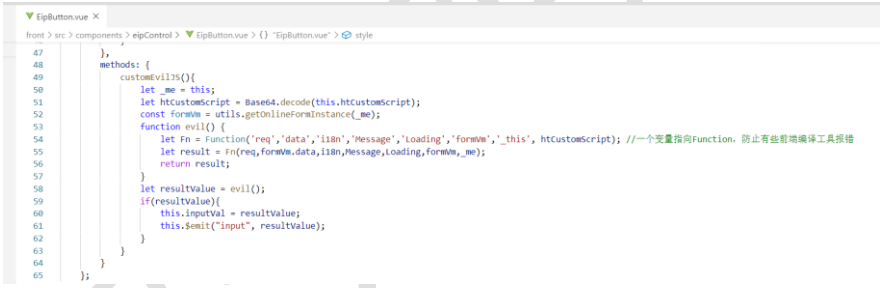
```
validate.js manage • src | validate.js front • src | validate.js mobile • src | controlsConfig.js ×
manage > src > api > controlsConfig.js > validateRules > name
1108  ];
1109
1110  export const validateRules = [
1111    {
1112      key: "confirmed",
1113      isInput: true,
1114      isBoData: true,
1115      type: "string|number|text|date",
1116      inputType: "select", //输入框类型
1117      name: "相同的值"
1118    },
1119    {
1120      key: "email",
1121      name: "电子邮箱"
1122    },
1123    {
1124      key: "regex",
1125      isInput: true,
1126      inputType: "input", //输入框类型
1127      name: "*正则表达式"
1128    },
1129    {
1130      key: "min",
1131      isInput: true,
1132      inputType: "input", //输入框类型
1133      name: "*最小文本长度"
1134    },
1135    {
1136      key: "max",
1137      isInput: true,
1138      inputType: "input", //输入框类型
```

2.1.3 按钮 js 方法

设计表单时可以添加按钮，按钮的 js 方法可以按照说明示例来配置。



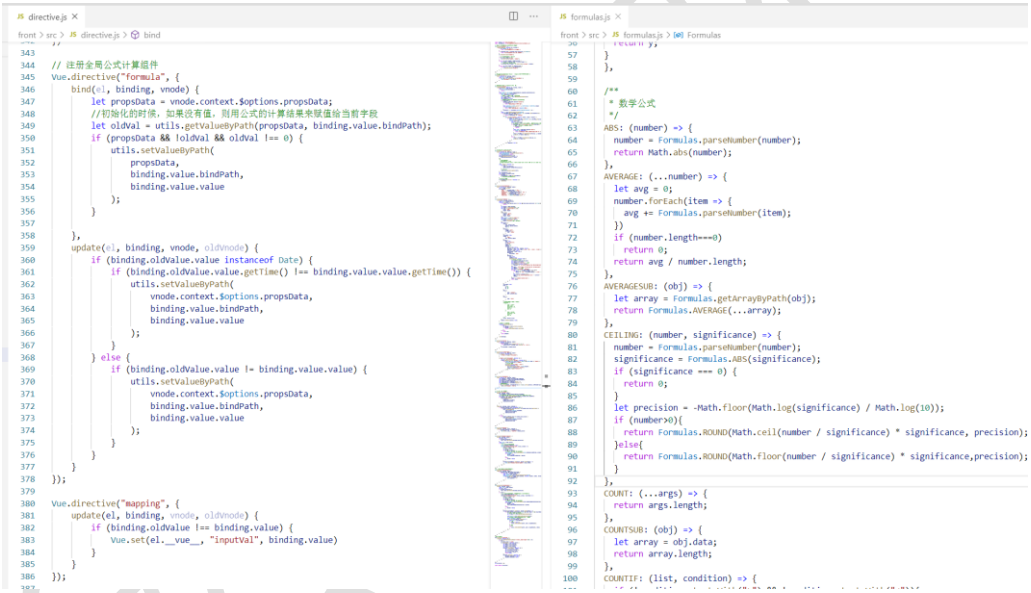
代码中执行这段 js 的切入如下图所示。



2.1.4 公式编辑

如下图所示，公式编辑按照每个函数的帮助说明进行配置即可完成各种数学计算、文本处理、日期处理等等。

这些函数的定义及如何在表单中生效的可以查看下图中的代码。

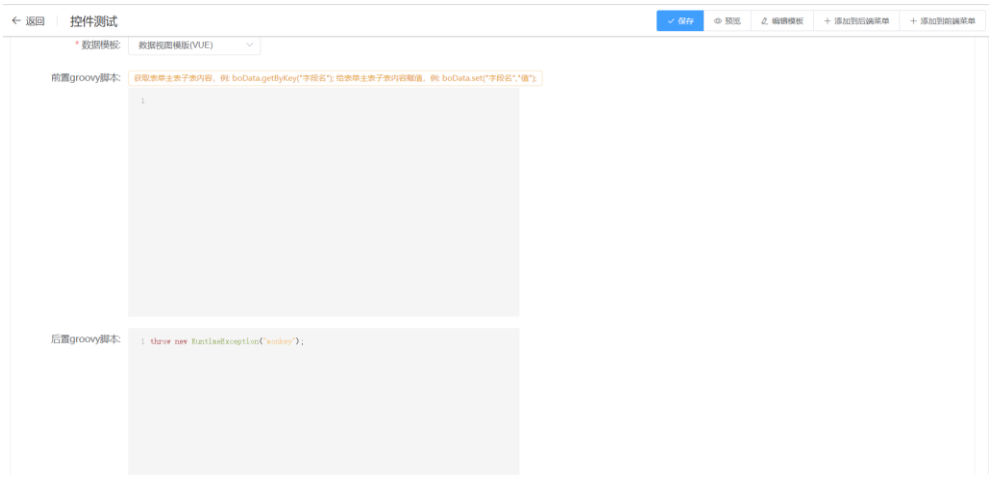


2.2 表单列表相关

2.2.1 前后置 groovy 脚本

如下图所示，在表单列表配置界面中可以配置前后置 Groovy 脚本，在添加/编辑数据之前会执行前置脚本，之后会执行后置脚本，脚本的上下文中有 boData 变量可以获取到表单数据（注意在前置脚本中修改数据才会保存到数据库中，后置脚本中修改无效）。

boData 对应的 Java 类为 com.hotent.bo.model.BoData。



执行前后置脚本的代码如下图所示。

GroovyScriptEngine.java | FormDataTemplateManagerImpl.java | BoData.java

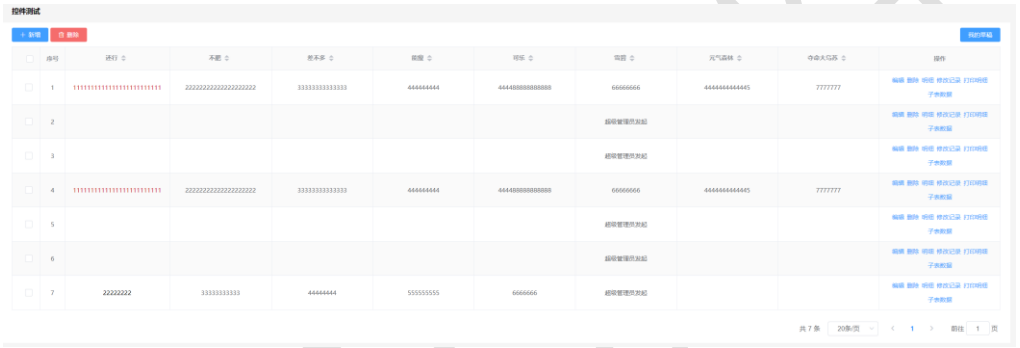
```
906         }
907         Map<String,Object> param = new HashMap<String, Object>();
908         param.put("boData", boData);
909         //业务数据模板别名
910         FormDataTemplate template = null;
911         if(jsonObject.has("templateKey")) {
912             String templateKey = jsonObject.get("templateKey").asText();
913             template = getByAlias(templateKey);
914             //判断是否有前置groovy脚本
915             if (BeanUtils.isNotEmpty(template.getBeforeScript())){
916                 groovyScriptEngine.execute(template.getBeforeScript(), param);
917             }
918         }
919
920         List<BoResult> resultList = handler.save("", "", boData);
921         if(BeanUtils.isNotEmpty(resultList)) {
922             //添加修改记录
923             handleBoResult(resultList, jsonObject);
924         }
925         String pk=StringUtil.isEmpty(boEnt.getPk())?"id_":boEnt.getPk();
926         if(jsonObject.has("formKey")){
927             String formKey=jsonObject.get("formKey").asText();
928             String boDataId="";
929             if(jsonData.has(pk)){
930                 boDataId=jsonData.get(pk).asText();
931             }else{
```

2.2.2 显示字段设置

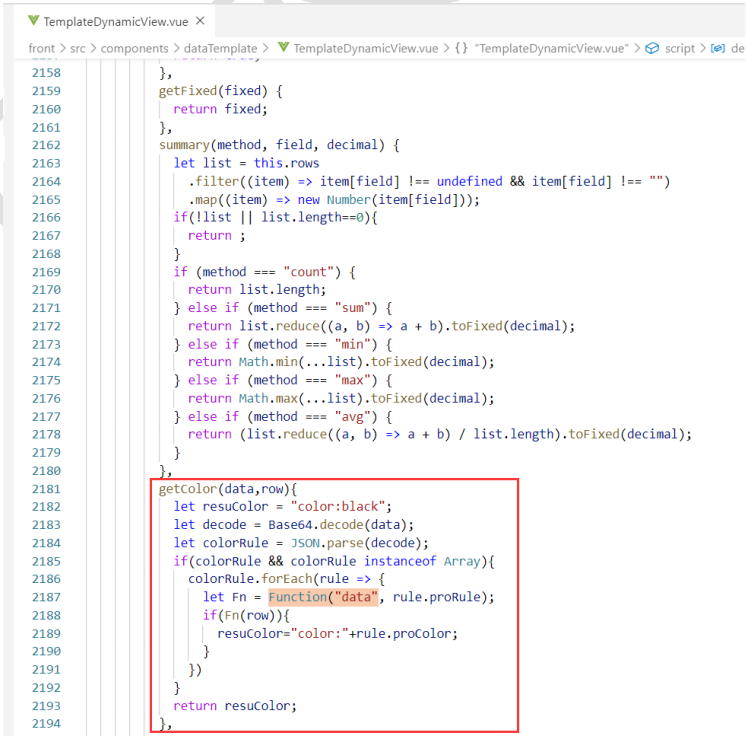
显示字段中可以对某一列进行文字颜色设置，例如金额在不同范围时显示不同的颜色，其配置如下图所示，首先指定文字颜色，在配置验证规则，返回 true/false 来表示是否应用该文字颜色。

当配置了多条规则，而数据同时符合多条规则时，以最后一条规则的颜色为准。

注意：因为 data 中某一字段可能为空，所以代码中最好对该字段的值是否存在进行判断（还要注意该字段如果为数字类型，且正好值为 0 时的判断）。



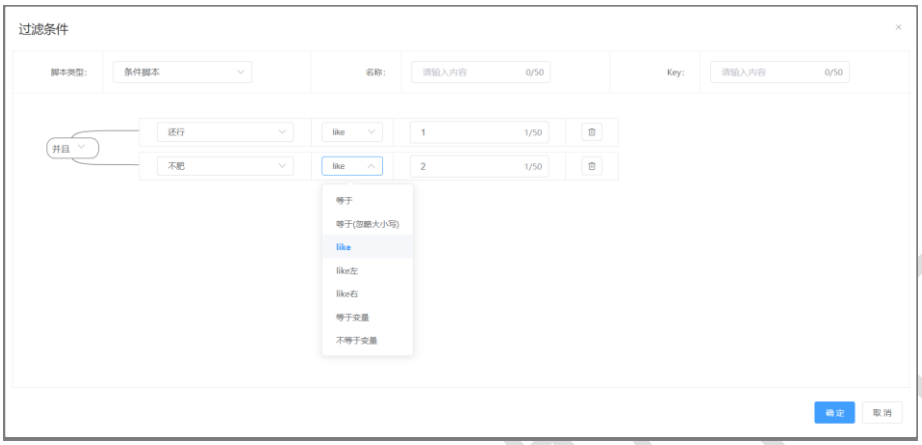
执行该规则的代码如下图所示。



2.2.3 数据过滤

- 条件脚本

  条件脚本的配置都非常直观易懂，按照业务规则进行相应的过滤配置即可。



- 追加 SQL

  如下图所示，可以通过追加 SQL 的方式对数据进行更个性化的过滤，配置的 SQL 语句会追加到 where 后面。

  执行数据过滤时，应用追加 SQL 的代码如下图所示。

FormDataTemplateManagerImpl.java

```
1157        }
1158        return bpmDataTemplate;
1159    }
1160
1161    @Override
1162    public String getFilterSql(String filterField,String dsName,Map<String, Object> param) throws IOException {
1163        StringBuffer sb = new StringBuffer();
1164        String sql = "";
1165        Map<String, Set<String>> curProfiles =  permssionCalc.getCurrentProfiles();
1166        List<Map<String,String>> filters = getFilterPermission(filterField,curProfiles);
1167        ArrayNode jsonArray = (ArrayNode) JsonUtil.toJsonNode(filterField);
1168        ObjectNode json = JsonUtil.arrayToObject(jsonArray, "key");
1169        if(BeanUtils.isEmpty(filters)) return sb.toString();
1170        for (Map<String,String> map : filters) {
1171            ObjectNode jsonObject = (ObjectNode) json.get(map.get("filterKey"));
1172            int type = JsonUtil.getInt(jsonObject, "type", 0);
1173            switch(type) {
1174                case 1:// 条件脚本
1175                    String dbType = databaseContext.getDbTypeByAlias(dsName);
1176                    sql = FilterJsonStructUtil.getSql(JsonUtil.getString(jsonObject, "condition"), dbType);
1177                    break;
1178                case 3:// 追加SQL
1179                    sql = executeScript(jsonObject.get("condition").asText(), param);
1180                    break;
1181                case 4://数据权限
1182                    sql = getDataPermissionSql(jsonObject.get("condition").asText(), "");
1183                    break;
1184            }
1185            if(StringUtil.isNotEmpty(sql)){
1186                if(4!=type && !sql.trim().toUpperCase().startsWith("AND")){
1187                    sb.append(" AND ");
1188                }
1189                sb.append(sql);
1190            }
1191        }
1192        return sb.toString();
1193    }
1194
1195    @Override
```

➢ 数据权限

数据权限的过滤也比较直观易懂，按照表单中字段实际存放的数据与当前用户、当前组织进行匹配来实现数据过滤。

2.3 流程相关

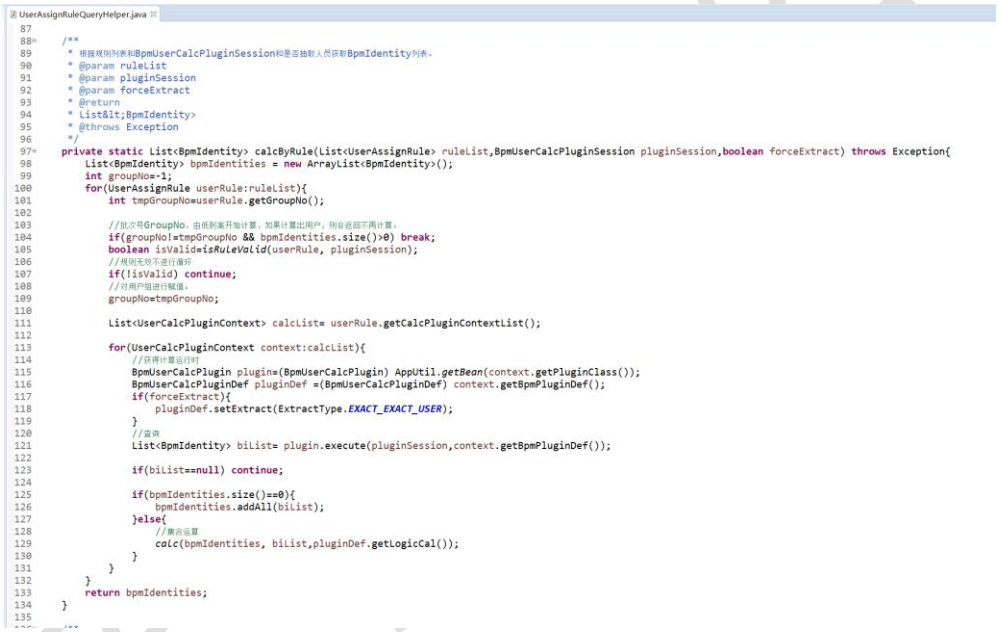
2.3.1 节点审批人的人员规则设置

如下图所示，节点审批人员可以配置为多个批次，按照顺序逐个批次查找执行人，直到该批次返回的审批人不为空为止。

每个批次配置节点审批人的时候可以配置一个人员规则，该规则运算结果返回 true 或 false，为 true 时该人员设置才生效，为 false 时继续找下一个批次的人员设置。

人员规则设置是一个可视化的配置界面，可以自由组合多种条件，包括规则和脚本，规则的配置比较直观易懂，脚本则支持通过 Groovy 代码进行逻辑判断，如下图所示，脚本中需要返回 true 或者 false。

对多个批次及人员规则进行计算的代码如下图所示：



```
UserAssignRuleQueryHelper.java
87
88*     /**
89       * 根据规则列表和BpmUserCalcPluginSession和是否抽取人员获取BpmIdentity列表。
90       * @param ruleList
91       * @param pluginSession
92       * @param forceExtract
93       * @return
94       * List&lt;BpmIdentity>
95       * @throws Exception
96       */
97*     private static List<BpmIdentity> calcByRule(List<UserAssignRule> ruleList,BpmUserCalcPluginSession pluginSession,boolean forceExtract) throws Exception{
98          List<BpmIdentity> bpmIdentities = new ArrayList<BpmIdentity>();
99          int groupNo=-1;
100         for(UserAssignRule userRule:ruleList){
101             int tmpGroupNo=userRule.getGroupNo();
102
103             //批次号GroupNo，由低到高开始计算，如果计算出用户，则会返回不再计算。
104             if(groupNo!=tmpGroupNo && bpmIdentities.size()>0) break;
105             boolean isValid=isRuleValid(userRule, pluginSession);
106             //规则无效不进行循环
107             if(!isValid) continue;
108             //对用户组进行赋值。
109             groupNo=tmpGroupNo;
110
111             List<UserCalcPluginContext> calcList= userRule.getCalcPluginContextList();
112
113             for(UserCalcPluginContext context:calcList){
114                 //获得计算运行时
115                 BpmUserCalcPlugin plugin=(BpmUserCalcPlugin) AppUtil.getBean(context.getPluginClass());
116                 BpmUserCalcPluginDef pluginDef =(BpmUserCalcPluginDef) context.getBpmPluginDef();
117                 if(forceExtract){
118                     pluginDef.setExtract(ExtractType.EXACT_EXACT_USER);
119                 }
120                 //查询
121                 List<BpmIdentity> biList= plugin.execute(pluginSession,context.getBpmPluginDef());
122
123                 if(biList==null) continue;
124
125                 if(bpmIdentities.size()==0){
126                     bpmIdentities.addAll(biList);
127                 }else{
128                     //兼容运算
129                     calc(bpmIdentities, biList,pluginDef.getLogicCal());
130                 }
131             }
132         }
133         return bpmIdentities;
134     }
135     /**
```

2.3.2 节点属性中的前后置处理器

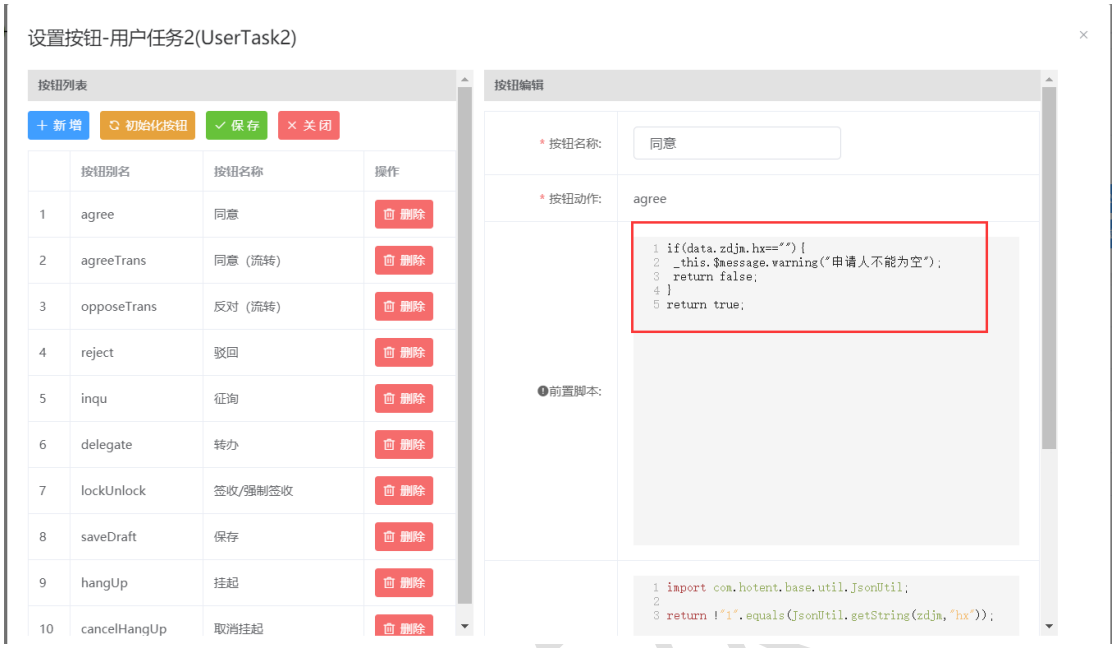
如下图所示，在某个节点上可以配置前后置处理器，配置内容为指定 Spring 容器中的 BeanId+方法名，该方法需要配置一个 ActionCmd 的对象作为入参。

如下图所示，在该节点上用户进行审批操作时会先触发前置处理器，后触发后置处理器，如下面的代码断点所示。

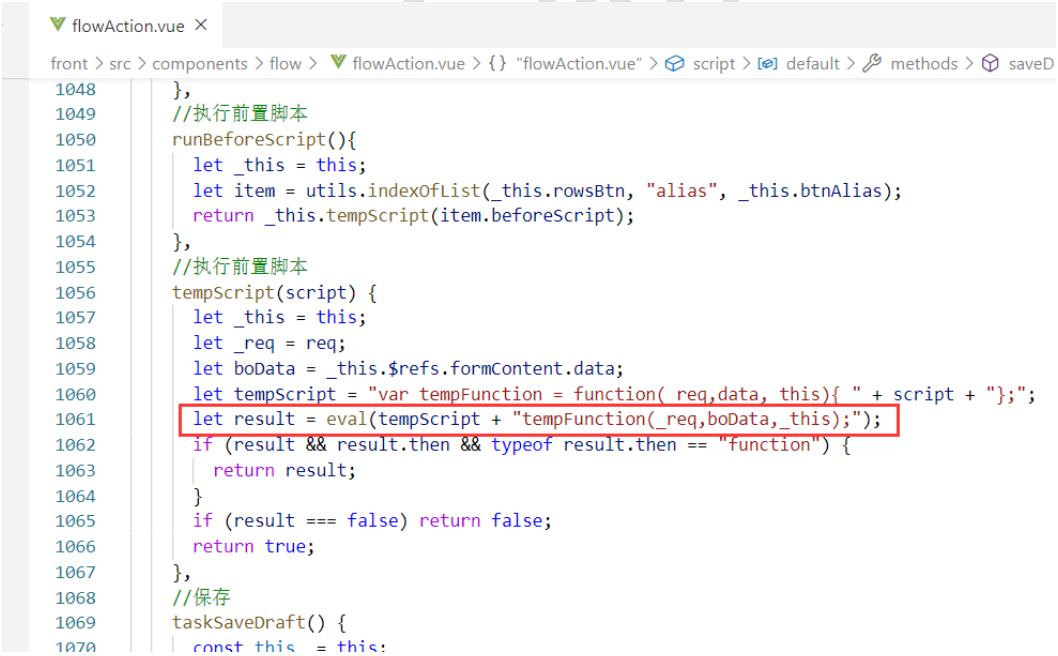
2.3.3 节点按钮中的前置脚本和 groovy 脚本

- 前置脚本

前置脚本可以通过 JavaScript 编写，该脚本在用户点击该按钮时触发，返回 false 时阻止按钮后续动作，返回 true 时继续执行。

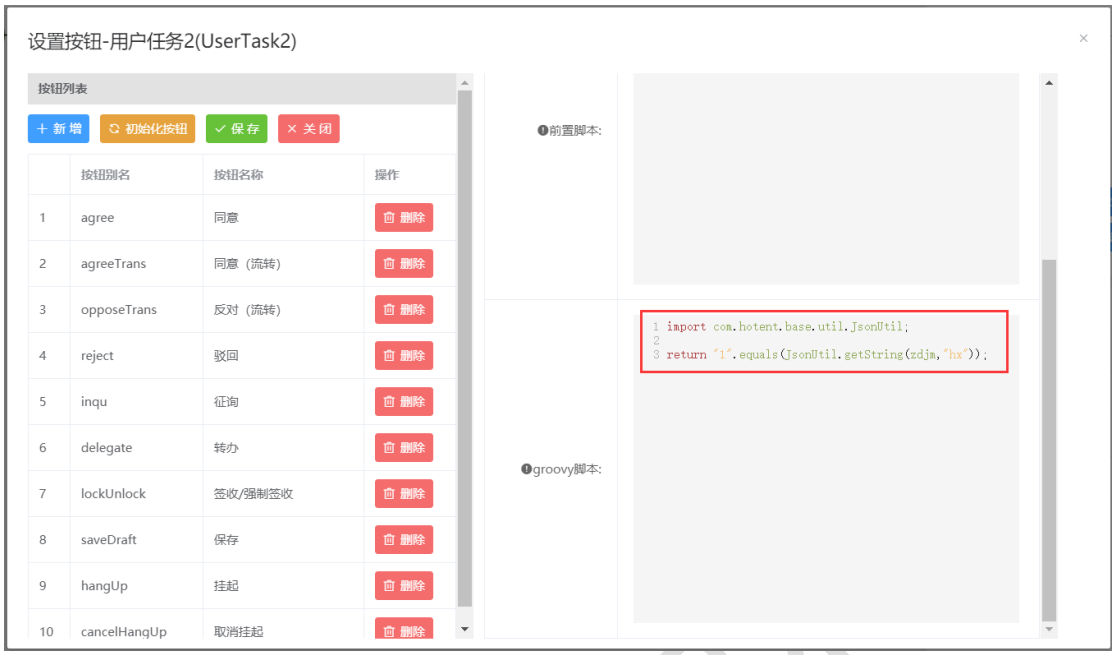


执行这段 JS 代码在如下图所示位置，可以使用_req,data,_this 这三个参数。

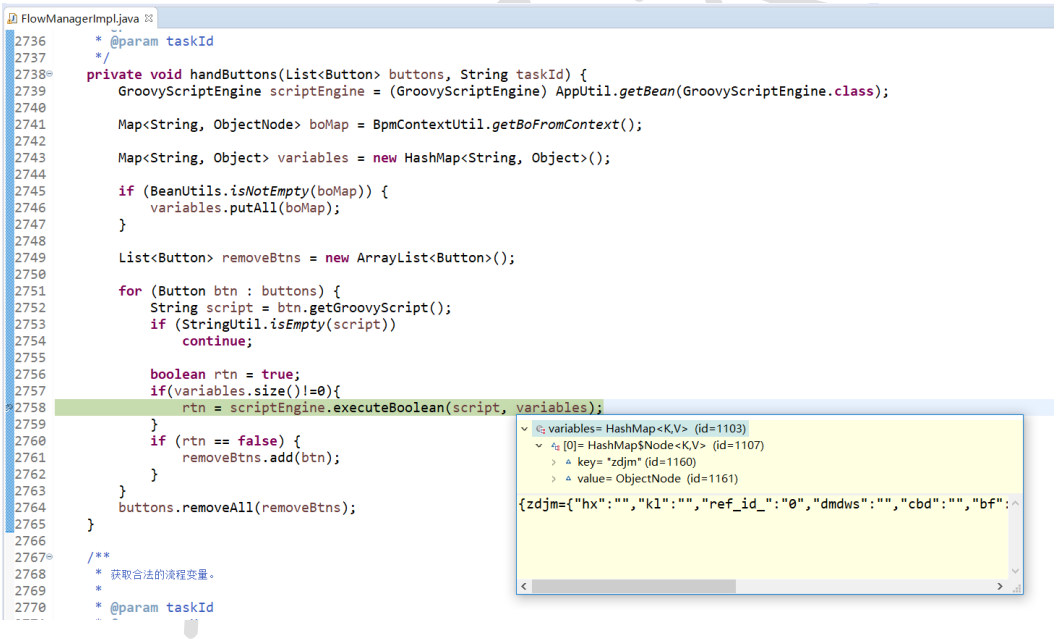


➢ Groovy 脚本

如下图所示，在 Groovy 脚本中可以通过**逻辑判断是否显示当前按钮**，返回 true 则显示，返回 false 则不显示。



脚本中可以使用当前流程变量、表单变量，表单变量以 JsonNode 的格式提供，可以通过 JsonUtil 访问其中的属性，执行脚本的代码如下所示。



2.3.4 节点事件（设置接口事件）

➢ 接口事件的配置

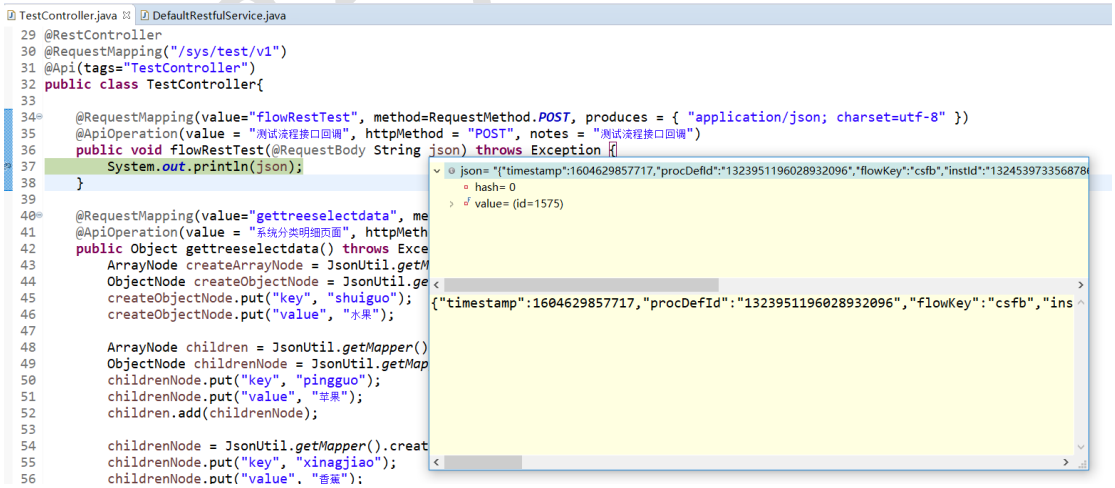
设置接口事件是指在流程启动、结束，或者指定环节的任务创建、任务结束时去调用 Restful 接口的功能，如下图所示，需要配置接口地址、调用类型、触发时机等信息。

如果接口的调用需要授权可以在接口头部添加一些校验信息，现在系统仅支持配置静态

的头部内容，当接口采用类似 auth0 时可以支撑授权校验。更复杂的场景下可能需要针对具体授权校验方式做相应的改造。

➢ 提供的 Restful 接口

如下图所示，提供了一个 Restful 接口的示例，以 POST 方式发布接口，入参为字符串类型的参数，EIP 在调用这个接口时会将当前流程的上下文参数以 JSON 格式的字符串作为请求参数。



➢ 调用接口时传递的参数

TestController.java DefaultRestfulService.java

```
555     }
556
557     class RestfulParam{
558         private long timestamp;                          /*时间戳*/
559         private String procDefId;                        /*流程定义ID*/
560         private String flowKey;                          /*流程定义KEY*/
561         private String instId;                           /*流程实例ID*/
562         private String taskId;                           /*任务ID*/
563         private String nodeId;                           /*节点ID*/
564         private String nodeName;                         /*节点名称*/
565         private String eventType;                        /*事件类型*/
566         private String businesskey;                      /*业务主键*/
567         private String sysCode;                          /*业务系统编码*/
568         private String procDefName;                      /*流程名称*/
569         private BpmIdentityResult creator;               /*实例发起人*/
570         private BpmIdentityResult assignee;              /*任务执行人*/
571         private List<BpmIdentityResult> candidate;       /*任务候选人*/
572         private String actionName;                       /*节点处理类型*/
573         private String nodeType;                         /*节点类型*/
574         private LocalDateTime createTime;                /*创建时间*/
575         private LocalDateTime completeTime;              /*完成时间*/
576         private String subject;                          /*实例标题*/
577         private String boData;                           /*表单数据*/
578         private Map<String, Object> vars;                /*流程变量*/
579
580         public String getFlowKey() {
581             return flowKey;
582         }
583         public long getTimestamp() {
584             return timestamp;
585         }
```

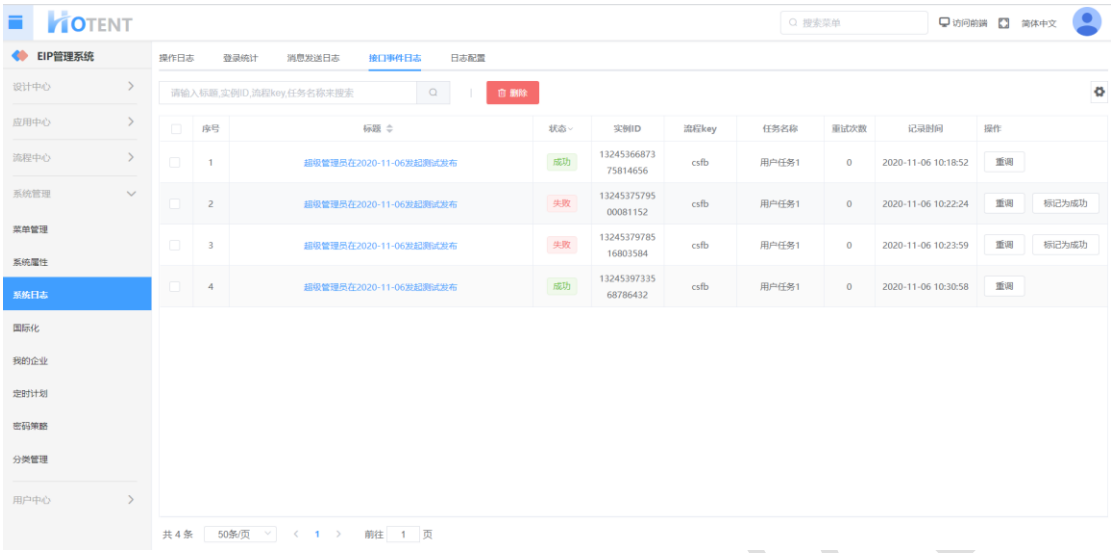
- 接口调用代码

TestController.java DefaultRestfulService.java

```
81      public Void taskPluginExecute(BpmTaskPluginSession pluginSession,
82              BpmTaskPluginDef pluginDef,List<Restful> restfuls) {
83          try{
84              EventType eventType = pluginSession.getEventType();
85              BpmDelegateTask bpmDelegateTask = pluginSession.getBpmDelegateTask();
86              for (Restful restful : restfuls) {
87                  String callTime = restful.getCallTime();
88                  if(!StringUtils.isEmpty(callTime)&&callTime.contains(eventType.getKey())){
89                      RestfulParam param = null;
90                      param = genartor(bpmDelegateTask,eventType);
91                      param.setTimestamp(System.currentTimeMillis());
92                      param.setEventType(eventType.getKey());
93                      String url = restful.getUrl();
94                      if(BeanUtils.isNotEmpty(url)){
95                          BpmCallLog callLog = getCallLog(param,restful);
96                          if(restful.getInvokeMode()==1){
97                              if(BeanUtils.isNotEmpty(param.getVars()) && BeanUtils.isNotEmpty(param.getVars().get("restful_task"))){
98                                  param.getVars().remove("restful_task");
99                              }
100                             // 异步调用Restful接口
101                             postAsync(url, JsonUtil.toJson(param),restful.getHeader(),callLog);
102                         }else{
103                             Boolean isSuccess = true;
104                             try {
105                                 // 同步调用Restful接口
106                                 String post = post(url, JsonUtil.toJson(param),restful.getHeader(),callLog);
107                                 callLog.setResponse(post);
108                             } catch (Exception e) {
109                                 isSuccess = false;
110                                 callLog.setResponse(ExceptionUtils.getRootCauseMessage(e));
111                             }
112                             callLog.setIsSuccess(isSuccess?BpmCallLog.SUCCESS_YES:BpmCallLog.SUCCESS_NO);
113                             buildCallLog(callLog);
114                         }
115                     }
116                 }
```

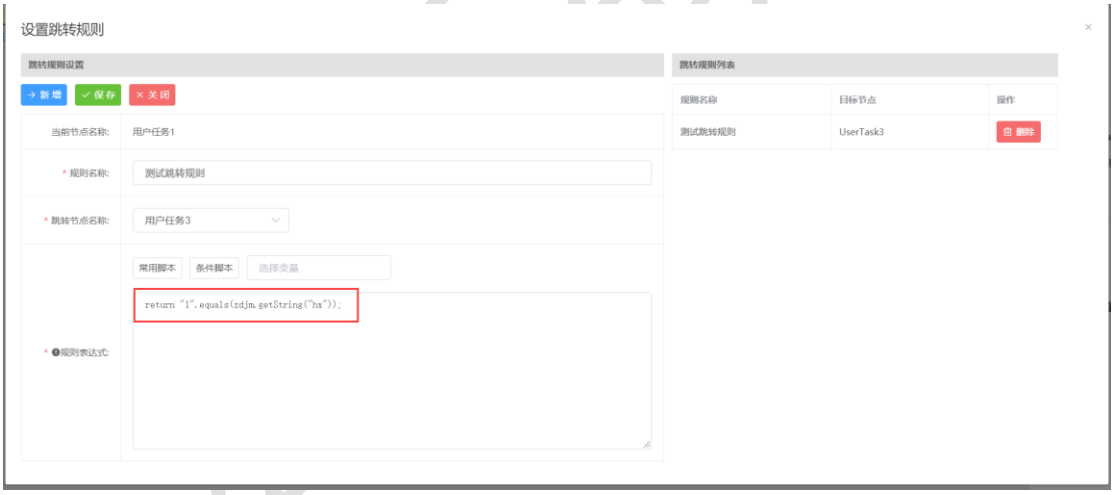
- 接口调用日志与接口重调

接口的调用不论成功还是失败都会记录到调用日志中，日志中记录了调用时间、调用参数、调用地址、调用结果等，而且提供了重调的功能可以手动触发再次调用。

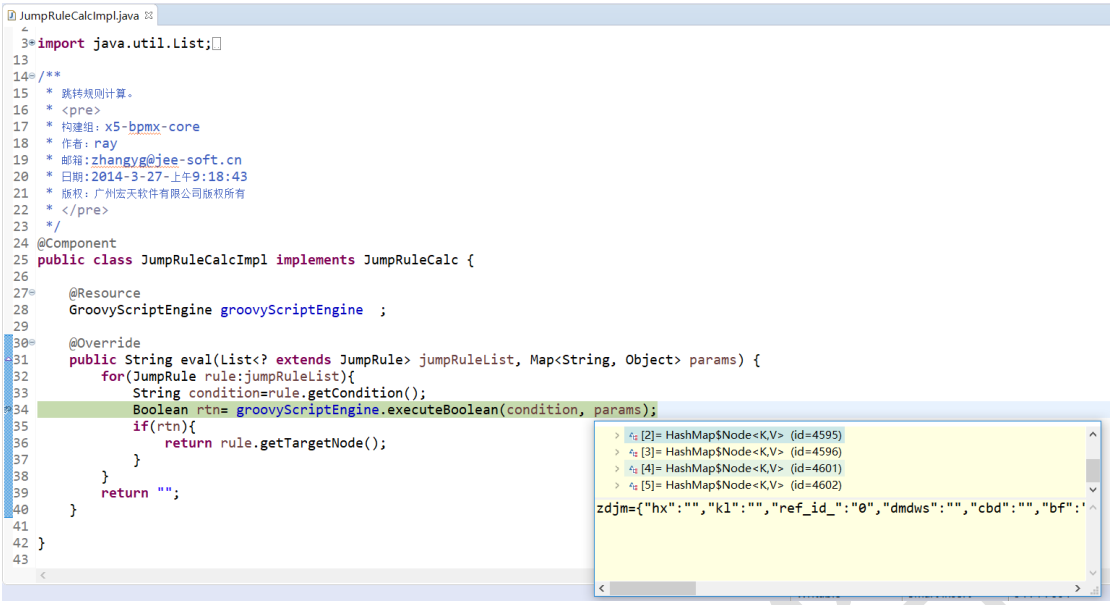


2.3.5 跳转规则的规则表达式

跳转规则的配置中可以配置一个规则表达式，该表达式返回 true 时则流程直接跳转到指定环节。



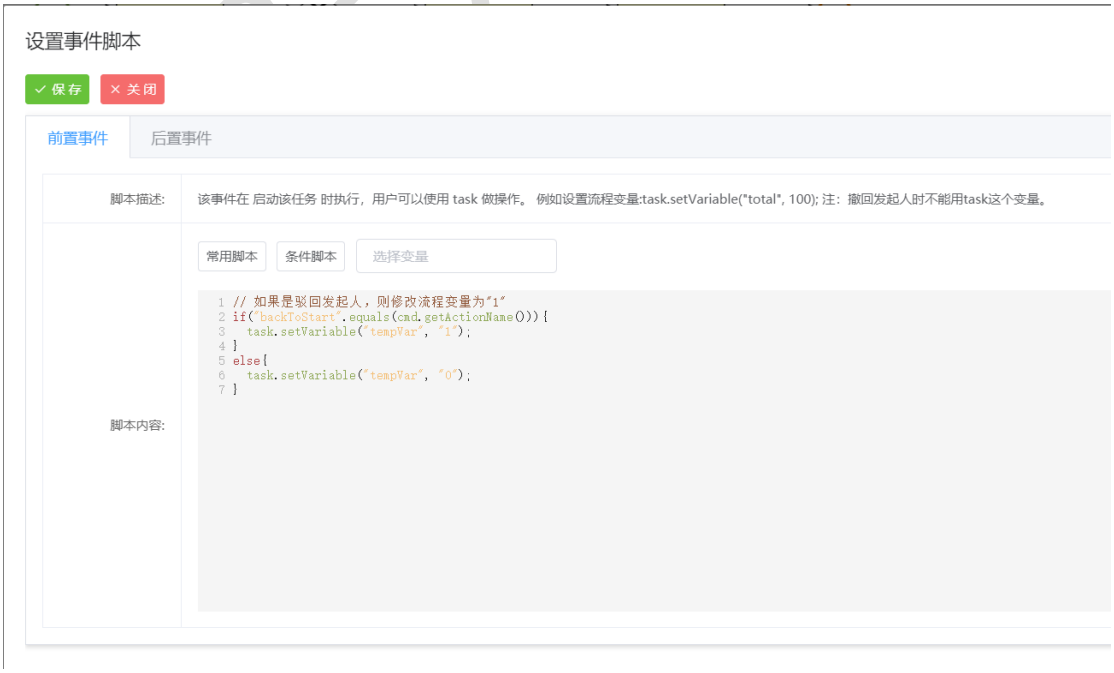
执行跳转规则脚本的代码如下图所示。



2.3.6 流程节点前后置事件

如下图所示，在流程某个环节可以配置前后置事件，前置事件在该环节要产生任务时触发，后置事件在该环节的任务被办理时触发。

如下面的例子所示，前置事件中可以判断当前流程是否被驳回到发起人了，如果是则修改一个流程变量的值为“1”，在后续的分支中可以根据该变量走不通的分支，从而实现正常审批时走分支 1，被驳回到发起人以后走分支 2。



执行前后置事件的代码如下图所示。

AbstractTaskListener.java

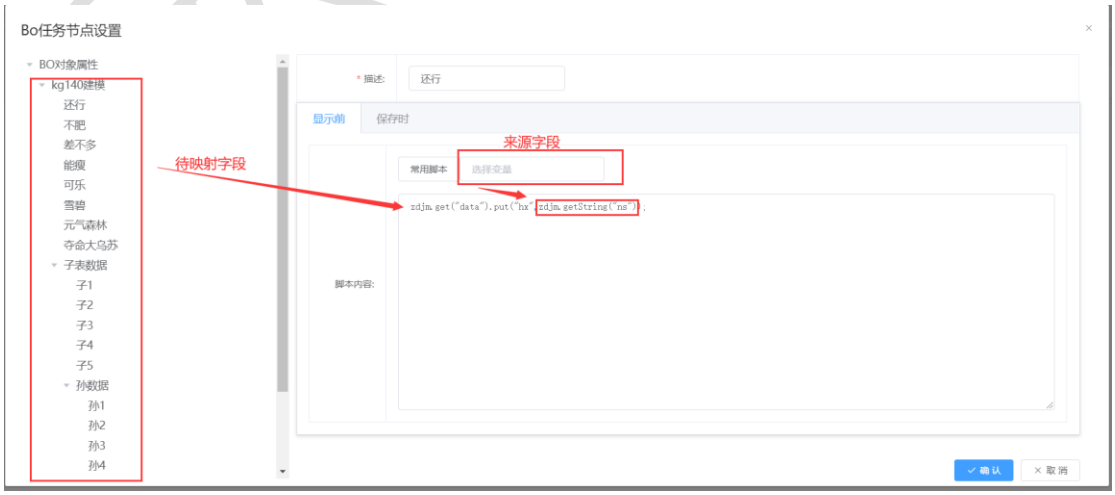
```
175    private void exeEventScript(BpmDelegateTask delegateTask) throws Exception{
176        String bpmnDefId=delegateTask.getBpmnDefId();
177        String defId =bpmDefinitionService.getDefIdByBpmnDefId(bpmnDefId);
178        String nodeId=delegateTask.getTaskDefinitionKey();
179        BpmNodeDef nodeDef= bpmDefinitionAccessor.getBpmNodeDef(defId, nodeId);
180
181        ScriptType scriptType= getScriptType();
182        String script=nodeDef.getScripts().get(scriptType);
183        if(StringUtil.isEmpty(script)) return;
184
185
186        Map<String, Object> vars=delegateTask.getVariables();
187        ActionCmd cmd= ContextThreadUtil.getActionCmd();
188        Map<String,ObjectNode> boMap= BpmContextUtil.getBoFromContext();
189        if (BeanUtils.isEmpty(boMap)) {
190            BpmProcessInstance bpmProcessInstance = bpmInstService.getProcessInstance(delegateTask.getProcessInsta
191            //1.获取BO数据
192            List<ObjectNode> boDatas = boDataService.getDataByInst(bpmProcessInstance);
193            //2.设置bo数据到上下文。
194            BpmContextUtil.setBoToContext(boDatas);
195            boMap = BpmContextUtil.getBoFromContext();
196        }
197        if(BeanUtils.isNotEmpty(boMap)){
198            Map<String, HtObjectNode> newMap =new HashMap<>();
199            for (Iterator<Entry<String, ObjectNode>> iterator = boMap.entrySet().iterator(); iterator.hasNext();)
200                Entry<String, ObjectNode> next = iterator.next();
201                newMap.put(next.getKey(),HtJsonNodeFactory.build().htObjectNode(next.getValue()));
202            }
203            vars.putAll(newMap);
204        }
205        vars.put("nodeDef", nodeDef);
206        vars.put("task", delegateTask);
207        vars.put("cmd", cmd);
208        try {
209            groovyScriptEngine.execute(script, vars);
210        } catch (BusinessException e) {
211            throw new WorkFlowException(Excepti
212        } catch (Exception e) {
213            StringBuffer sb = new StringBuffer(
214            sb.append("<br/><br/>流程在节点："+node
215            sb.append("<br/>请联系管理员！");
216            sb.append("<br/>可能原因为："+e.getMess
217            sb.append("<br/>执行脚本为："+script);
218            sb.append("脚本变量："+vars.toString()
219            throw new WorkFlowException(sb.toSt
```

vars= HashMap<K,V> (id=6202)
[0]= HashMap$Node<K,V> (id=6206)
[1]= HashMap$Node<K,V> (id=6207)
[2]= HashMap$Node<K,V> (id=6208)

```
{subject_=超级管理员在2020-11-06发起测试发布, task=com.hotent.
```

2.3.7 初始赋值

在流程的指定环节可以实现业务数据的映射，如下图所示，可以将来源字段或者通过 Groovy 脚本计算出来的值赋值给某个待映射字段。



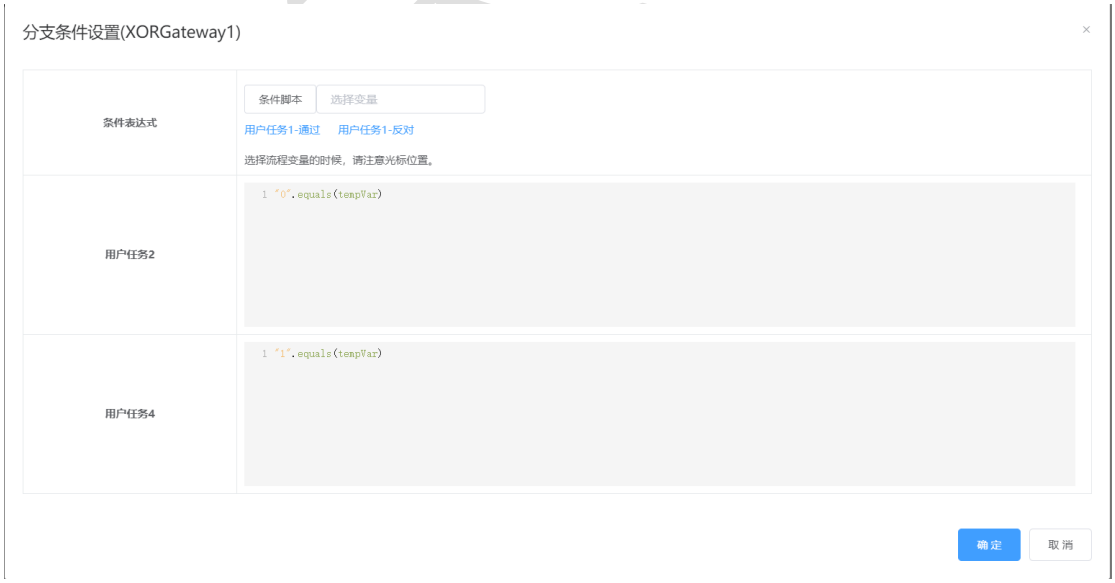
如下所示为执行赋值脚本的代码。

DefaultDataObjectHandler.java

```
119      * @param fieldInitSettings
120      * @param dataObject
121      * void
122      * @throws IOException
123      */
124     private void setDataObject(List<FieldInitSetting> fieldInitSettings, List<ObjectNode> boDatas) throws IOException{
125         Map<String, Object> vars=new HashMap<String, Object>();
126 
127         for(ObjectNode boData:boDatas){
128             String boDefCode = "";
129             if(boData.has("boDef")){
130                 boDefCode = boData.get("boDef").get("alias").asText();
131             }else{
132                 boDefCode = boData.get("boDefAlias").asText();
133             }
134             HtJsonNodeFactory factory = new HtJsonNodeFactory();
135             HtObjectNode htBodata =factory.htObjectNode(boData);
136             vars.put(boDefCode, htBodata);
137         }
138         ActionCmd cmd = ContextThreadUtil.getActionCmd();
139         if(BeanUtils.isNotEmpty(cmd)){
140             vars.putAll(cmd.getVariables());
141         }
142 
143         for(FieldInitSetting setting:fieldInitSettings){
144             String script = setting.getSetting();
145             if(StringUtil.isEmpty(script)) continue;
146             groovyScriptEngine.execute(script, vars);
147         }
148         if(BeanUtils.isNotEmpty(cmd)){
149             cmd.setBusData(BoDataUtil.hanlerData(boDatas).toString());
150         }
151     }
152 
153 
```

2.3.8 分支网关

如下图所示，为分支网关的设置界面，通过 Groovy 脚本的执行返回 true 或者 false，分支判断按照从上至下一次匹配的方式执行，如果某个分支符合判定，则不再往下执行，如果执行完所有的判断均没有符合的分支时则抛出找不到可用分支的异常。



分支网关的脚本执行如下图的代码所示。

GroovyCondition.java

```
63
64      public boolean evaluate(DelegateExecution execution) {
65          Map<String,Object> maps=execution.getVariables();
66          //添加execution。
67          maps.put(VariableScopeElResolver.EXECUTION_KEY, execution);
68
69          ActionCmd cmd=ContextThreadUtil.getActionCmd();
70          if(cmd instanceof TaskFinishCmd){
71              TaskFinishCmd taskCmd=(TaskFinishCmd)cmd;
72              //如果cmd包含会签结果的变量。则表明这个是一个已经完成了的会签任务。则用会签节点的投票结果，作为该节点的actionName
73              SignResult signResult = (SignResult) cmd.getTransitVars(BpmConstants.TASK_SIGN_RESULT);
74              if (BeanUtils.isEmpty(signResult) || taskCmd.getActionName().equals(signResult.getNodeStatus().getKey())) {
75                  maps.put("taskCmd", taskCmd);
76              }else {
77                  try {
78                      DefaultTaskFinishCmd defaultTaskFinishCmd = (DefaultTaskFinishCmd) taskCmd;
79                      DefaultTaskFinishCmd clone = defaultTaskFinishCmd.clone();
80                      clone.setActionName(signResult.getNodeStatus().getKey());
81                      maps.put("taskCmd", clone);
82                  } catch (Exception e) {
83                      e.printStackTrace();
84                      maps.put("taskCmd", taskCmd);
85                  }
86              }
87          }else if (cmd instanceof ProcessInstCmd ) {
88              maps.put("taskCmd", cmd);
89          }
90          Map<String,ObjectNode> boMap= BpmContextUtil.getBoFromContext();
91
92          if(BeanUtils.isNotEmpty(boMap)){
93              Map<String, HtObjectNode> newMap =new HashMap<>();
94              for (Iterator<Entry<String, ObjectNode>> iterator = boMap.entrySet().iterator(); iterator.hasNext();) {
95                  Entry<String, ObjectNode> next = iterator.next();
96                  ObjectNode obj= next.getValue();
97                  if (obj.hasNonNull("data") && (obj.get("data") instanceof ObjectNode)) {
98                      obj = (ObjectNode) obj.get("data");
99                  }
100                 HtObjectNode htObjectNode = HtJsonNodeFactory.build().htObjectNode(obj);
101                 newMap.put(next.getKey(),htObjectNode);
102                 maps.put(next.getKey(),htObjectNode);
103             }
104             maps.putAll(newMap);
105         }
106
107         maps.put("boMap", boMap);
108
109         GroovyScriptEngine engine=AppUtil.getBean(GroovyScriptEngine.class);
110         String newScript = replaceSpecialChar(script);
111         try {
112             return engine.executeBoolean(newScript, maps);
113         }catch(BusinessException e){
114             throw new WorkFlowException(ExceptionUtils.getRootCauseMessage(e));
115         }catch (Exception e) {
```

2.3.9 脚本节点

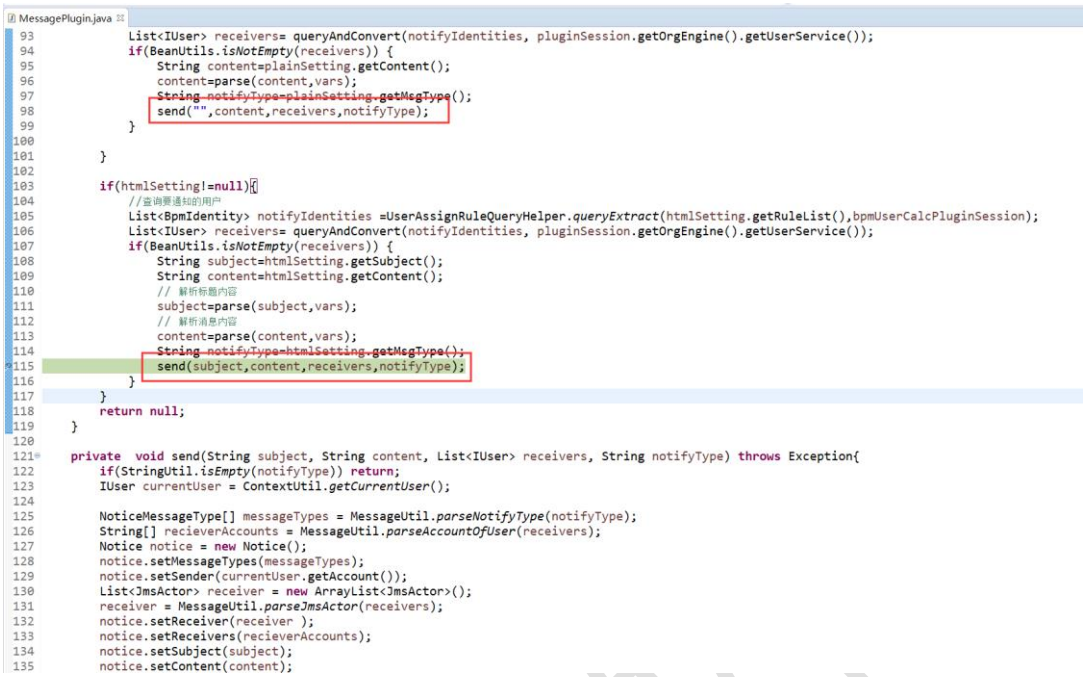
脚本节点属于自动任务环节，流程会自动执行该环节的任务，可以通过 Groovy 脚本来实现一些逻辑处理，脚本的执行如下图所示。

ScriptNodePlugin.java

```java
package com.hotent.bpm.plugin.execution.script.plugin;

import java.util.HashMap;

/**
 * 脚本节点插件运行时
 *
 * @company 广州宏天软件股份有限公司
 * @author heyifan
 * @email heyf@jee-soft.cn
 * @date 2020年11月6日
 */
public class ScriptNodePlugin extends AbstractBpmExecutionPlugin{

    @Resource
    GroovyScriptEngine groovyScriptEngine  ;

    public Void execute(BpmExecutionPluginSession pluginSession,
            BpmExecutionPluginDef pluginDef) {
        ScriptNodePluginDef nodeDef=(ScriptNodePluginDef)pluginDef;
        BpmDelegateExecution execution= pluginSession.getBpmDelegateExecution();

        Map<String, Object> vars=new HashMap<String, Object>();
        vars.put(BpmConstants.BPMN_EXECUTION_ID, execution.getId());
        vars.put(BpmConstants.BPMN_INST_ID, execution.getBpmnInstId());
        vars.putAll( execution.getVariables());
        vars.put("execution",execution);
        //从上下文获取bo实体数据。
        Map<String,ObjectNode> boDatas= BpmContextUtil.getBoFromContext();
        if(BeanUtils.isNotEmpty(boDatas)){
            vars.putAll(boDatas);
        }

        String script=nodeDef.getScript();
        groovyScriptEngine.execute(script, vars);
        return null;
    }

}
```

2.3.10消息节点

消息节点属于自动任务环节，流程会自动执行该环节，按照消息配置进行消息的发送。消息节点对应的处理代码如下图所示。



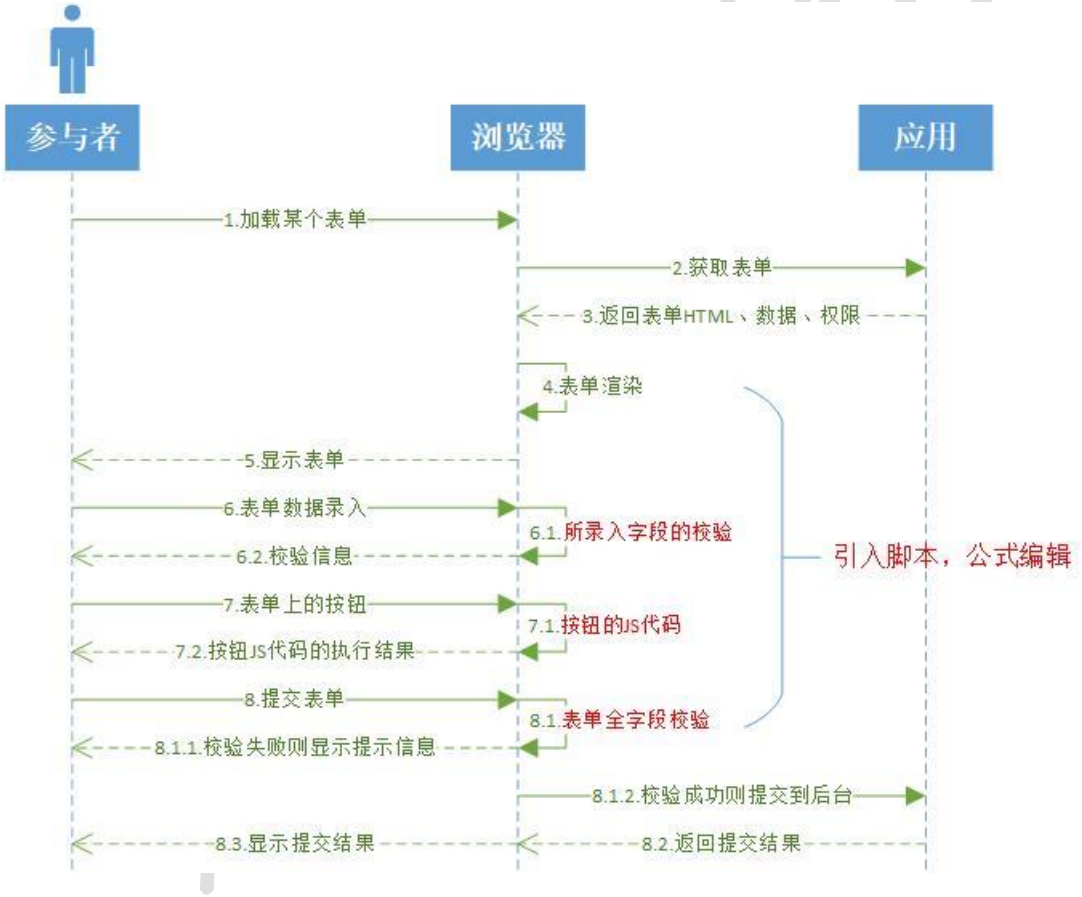
```
MessagePlugin.java
93            List<IUser> receivers= queryAndConvert(notifyIdentities, pluginSession.getOrgEngine().getUserService());
94            if(BeanUtils.isNotEmpty(receivers)) {
95                String content=plainSetting.getContent();
96                content=parse(content,vars);
97                String notifyType=plainSetting.getMsgType();
98                send("",content,receivers,notifyType);
99            }
100
101       }
102
103       if(htmlSetting!=null){
104           //查询要通知的用户
105           List<BpmIdentity> notifyIdentities =UserAssignRuleQueryHelper.queryExtract(htmlSetting.getRuleList(),bpmUserCalcPluginSession);
106           List<IUser> receivers= queryAndConvert(notifyIdentities, pluginSession.getOrgEngine().getUserService());
107           if(BeanUtils.isNotEmpty(receivers)) {
108               String subject=htmlSetting.getSubject();
109               String content=htmlSetting.getContent();
110               // 解析标题内容
111               subject=parse(subject,vars);
112               // 解析消息内容
113               content=parse(content,vars);
114               String notifyType=htmlSetting.getMsgType();
115               send(subject,content,receivers,notifyType);
116           }
117       }
118       return null;
119   }
120
121   private  void send(String subject, String content, List<IUser> receivers, String notifyType) throws Exception{
122       if(StringUtil.isEmpty(notifyType)) return;
123       IUser currentUser = ContextUtil.getCurrentUser();
124
125       NoticeMessageType[] messageTypes = MessageUtil.parseNotifyType(notifyType);
126       String[] recieverAccounts = MessageUtil.parseAccountOfUser(receivers);
127       Notice notice = new Notice();
128       notice.setMessageTypes(messageTypes);
129       notice.setSender(currentUser.getAccount());
130       List<JmsActor> receiver = new ArrayList<JmsActor>();
131       receiver = MessageUtil.parseJmsActor(receivers);
132       notice.setReceiver(receiver );
133       notice.setReceivers(recieverAccounts);
134       notice.setSubject(subject);
135       notice.setContent(content);
```

3 时序图

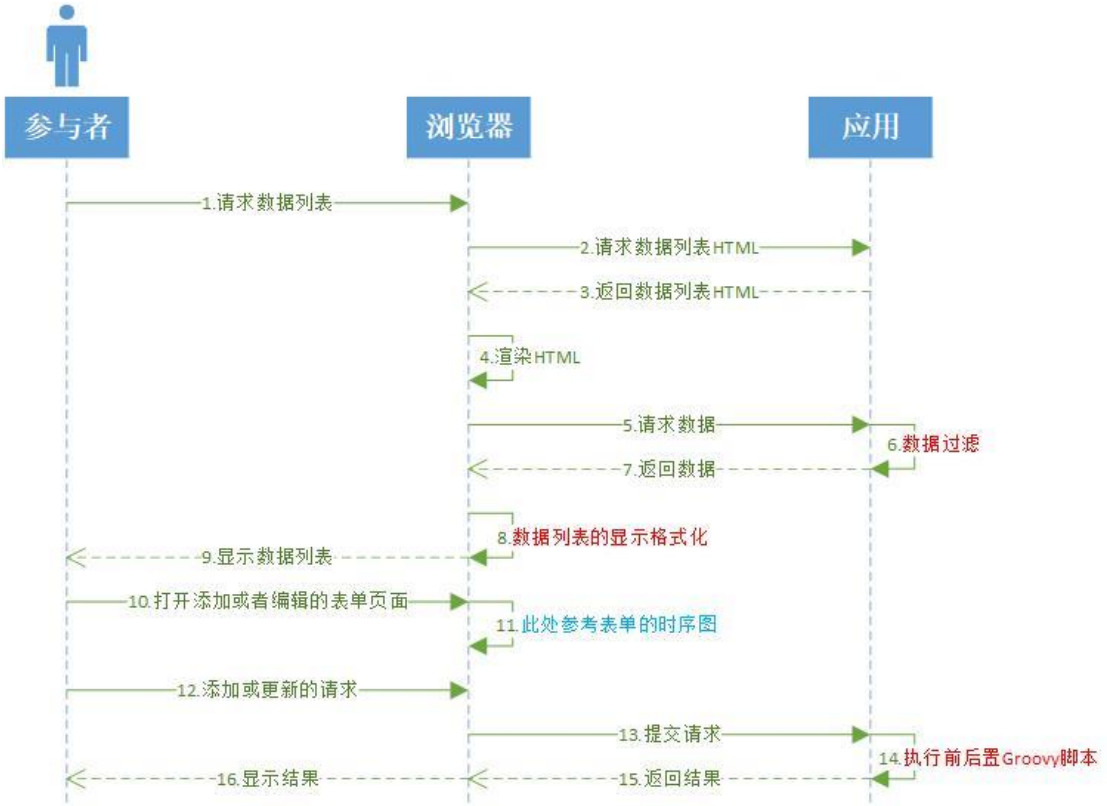
分类	名称	类型	触发时机
表单	引入脚本	JS 脚本	表单使用的整个过程
	字段校验	前后端校验	1. 指定字段的值变化或失去焦点时（校验当前字段）； 2. 提交表单时（校验表单所有字段）。
	按钮 js 方法	JS 脚本	点击按钮时
	公式编辑		组成公式的字段值变化时
数据列表	前后置 Groovy 脚本	Groovy 脚本	添加或更新表单数据时
	显示字段中对字段的设置	JS 脚本	显示数据列表时
	数据过滤	Groovy 脚本&追加 SQL	
流程	节点审批人的人员规则设置	Groovy 脚本	产生任务并分配任务的处理人时
	节点按钮前置脚本	JS 脚本	点击按钮时
	节点按钮 Groovy 脚本	Groovy 脚本	显示按钮之前
	节点事件（设置接口事件）	配置 Restful 接口	1. 流程启动时； 2. 流程结束时； 3. 任务创建时； 4. 任务结束时。
	跳转规则的规则表达式	Groovy 脚本	配置所在节点的任务审批时
	流程节点的前后置事件		1. 前置事件：配置所在节点产生任务时；

			2. 后置事件：配置所在节点的任务被办理时。
	初始赋值		1. 显示前：配置所在节点打开办理页面时； 2. 保存后：配置所在节点的任务被办理时。
	分支网关		流程审批到该节点时
	脚本节点		
	消息节点		

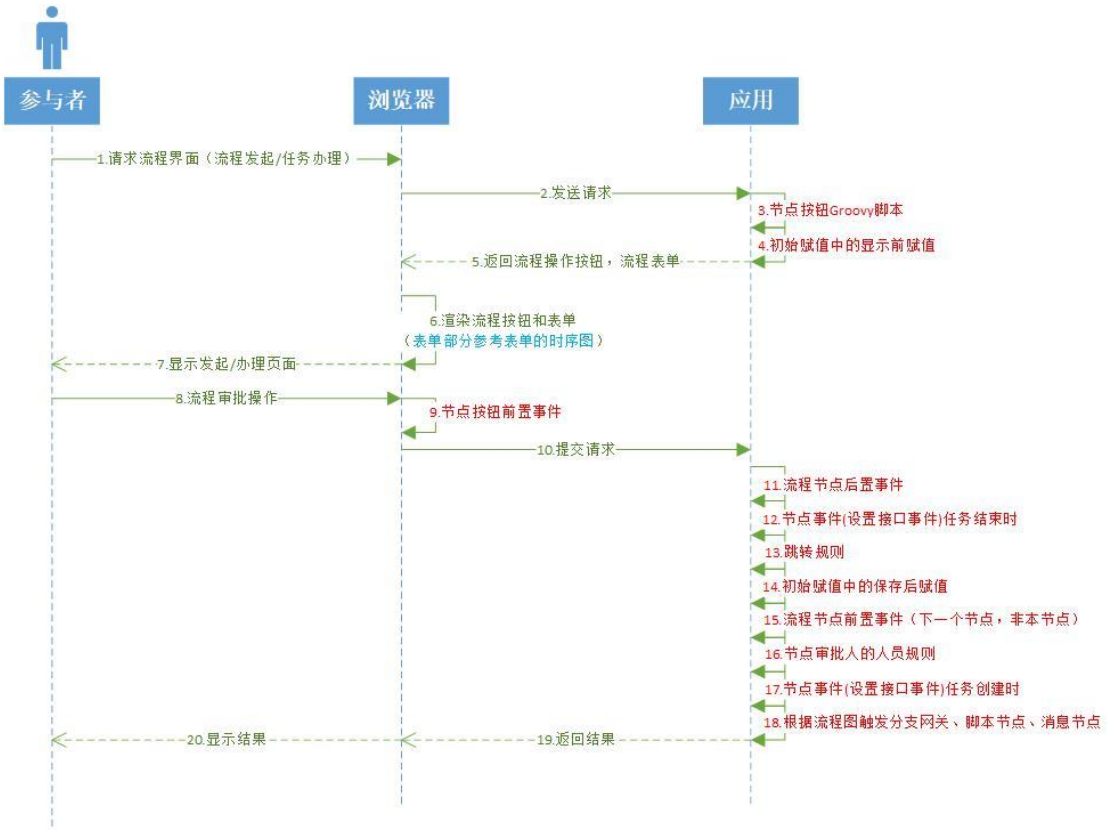
3.1 表单模块



3.2 数据列表模块


参与者
浏览器
应用
1.请求数据列表
2.请求数据列表HTML
3.返回数据列表HTML
4.渲染HTML
5.请求数据
6.数据过滤
7.返回数据
8.数据列表的显示格式化
9.显示数据列表
10.打开添加或者编辑的表单页面
11.此处参考表单的时序图
12.添加或更新的请求
13.提交请求
14.执行前后置Groovy脚本
15.返回结果
16.显示结果

3.3 流程模块


参与者
浏览器
应用
1.请求流程界面（流程发起/任务办理）
2.发送请求
3.节点按钮Groovy脚本
4.初始赋值中的显示前赋值
5.返回流程操作按钮，流程表单
6.渲染流程按钮和表单
（表单部分参考表单的时序图）
7.显示发起/办理页面
8.流程审批操作
9.节点按钮前置事件
10.提交请求
11.流程节点后置事件
12.节点事件(设置接口事件)任务结束时
13.跳转规则
14.初始赋值中的保存后赋值
15.流程节点前置事件（下一个节点，非本节点）
16.节点审批人的人员规则
17.节点事件(设置接口事件)任务创建时
18.根据流程图触发分支网关、脚本节点、消息节点
19.返回结果
20.显示结果