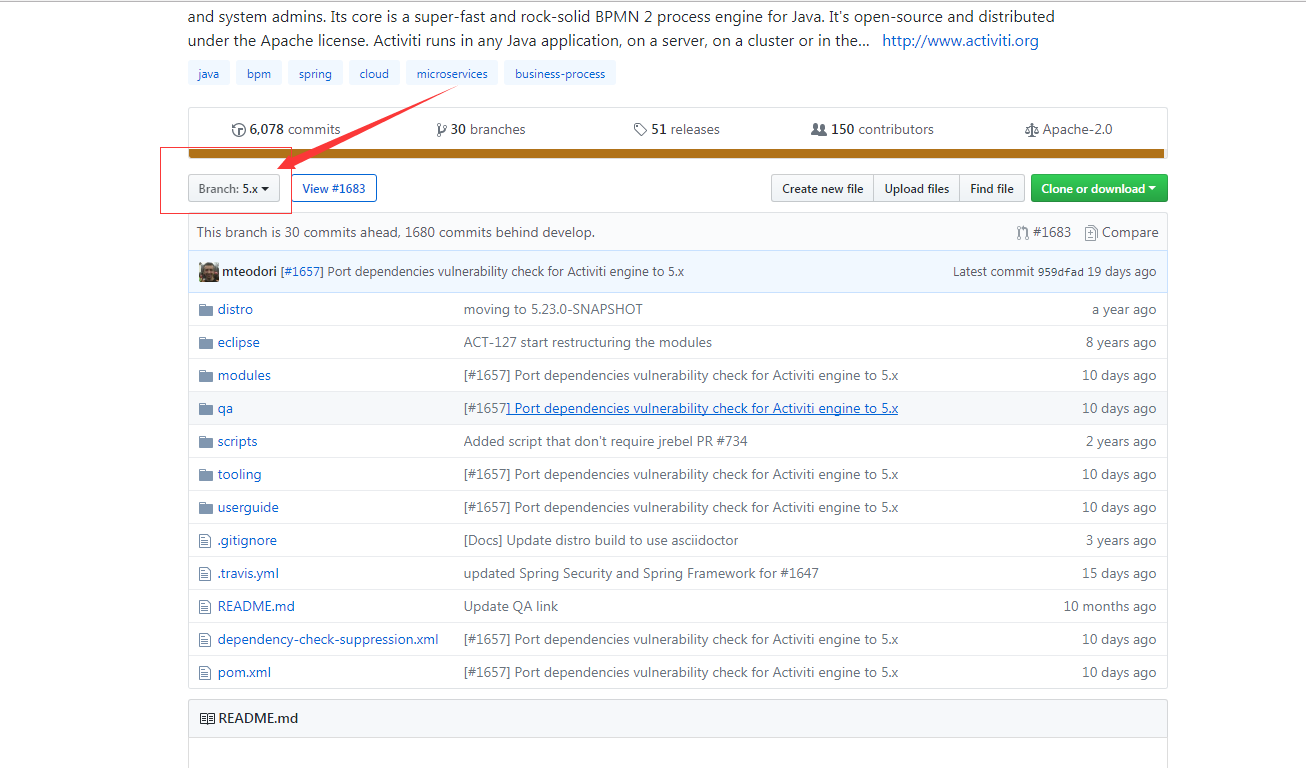
## 以下所有都是要建立在ssm+activiti已经能运行起来:spring至少要4要不然不支持@RestController

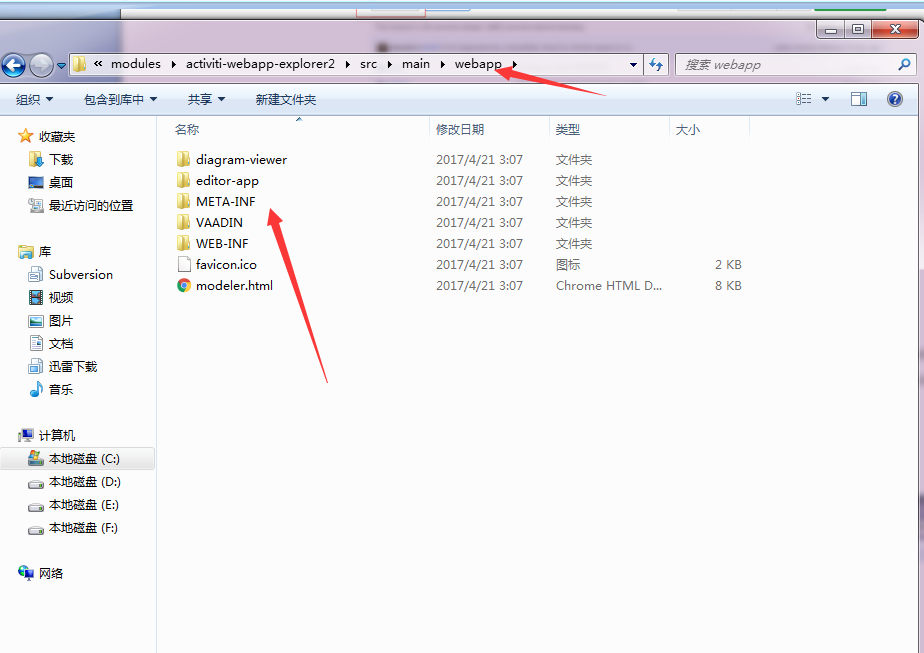
首先需要到github：<https://github.com/Activiti/Activiti>下载源码包：可以本地git或者下载zip包



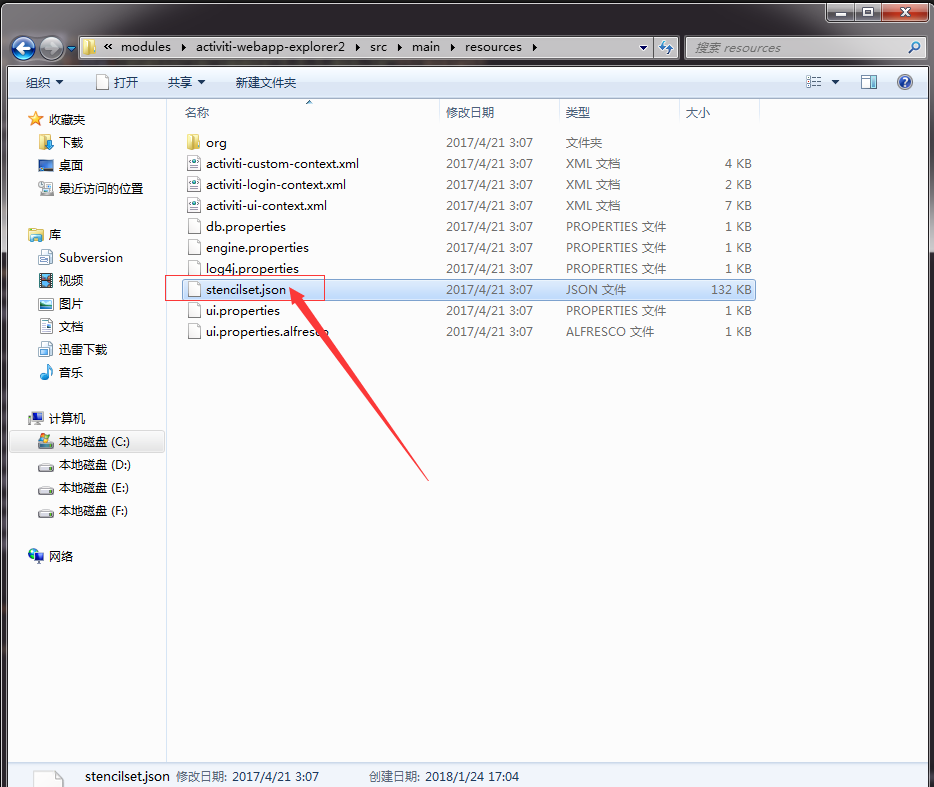
一、获得前端界面显示所需要的：

1、解压Activiti-master.zip

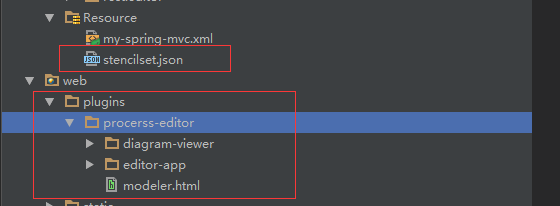
2、找到modules\activiti-webapp-explorer2\src\main\webapp

diagram-viewer、editor-app、modeler.html，将其加到项目的webapp中。

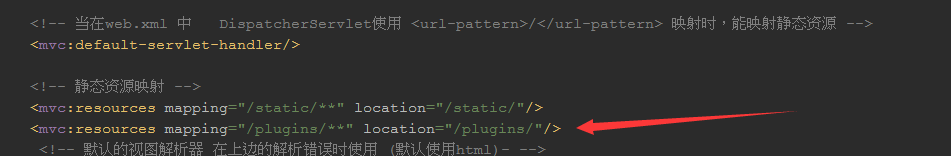
从activiti-webapp-explorer2 的 main/resource下找到stencilset.json放到项目的配置文件中



拷到项目web目录中：idea和eclipse名称也许不一样：我放到了plugins目录下

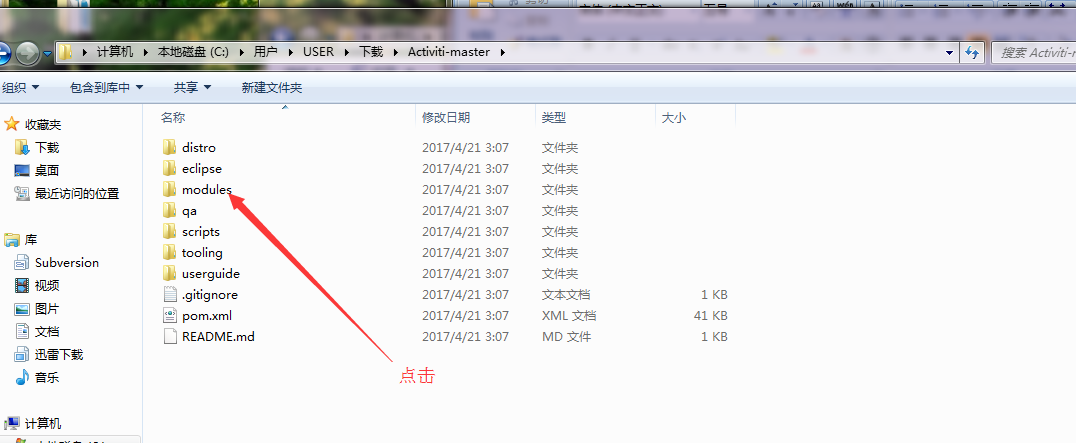


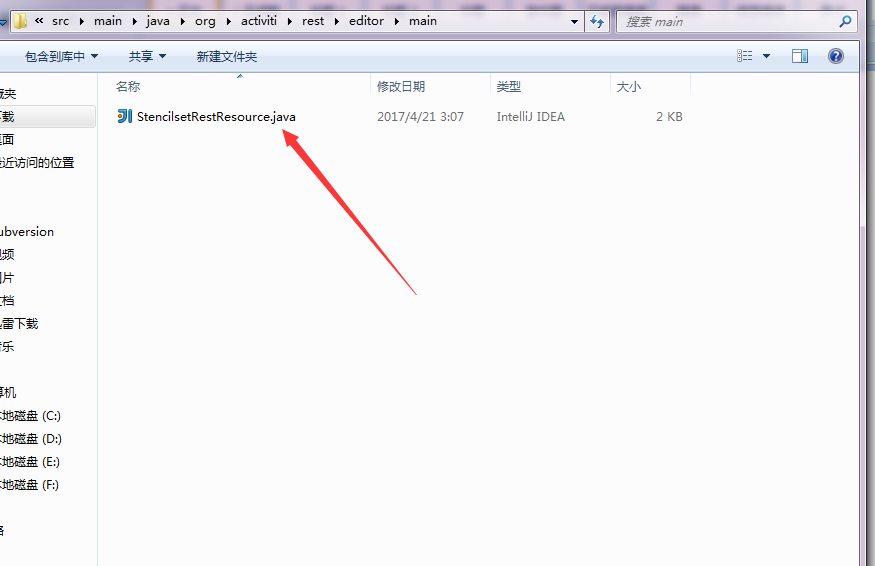
注意：如果springmvc配置的是/，那么需要开启不拦截该目录：



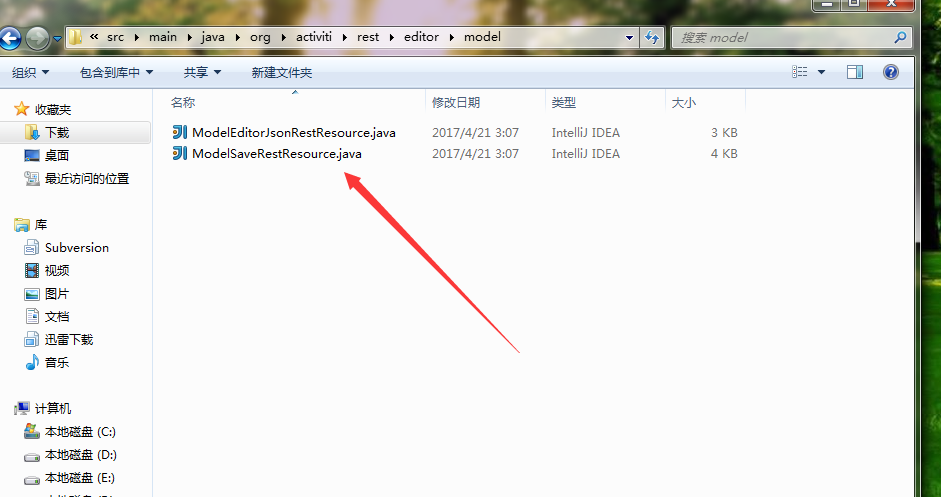
## 二、获取到运行所需要的controller

在Activiti-master下的modules文件夹下有一个activiti-modeler文件

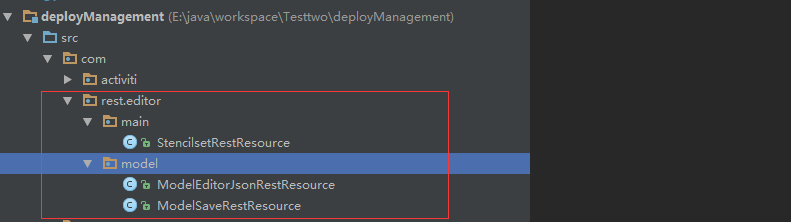


->activiti-modeler\src\main\java\org\activiti\rest\editor\main

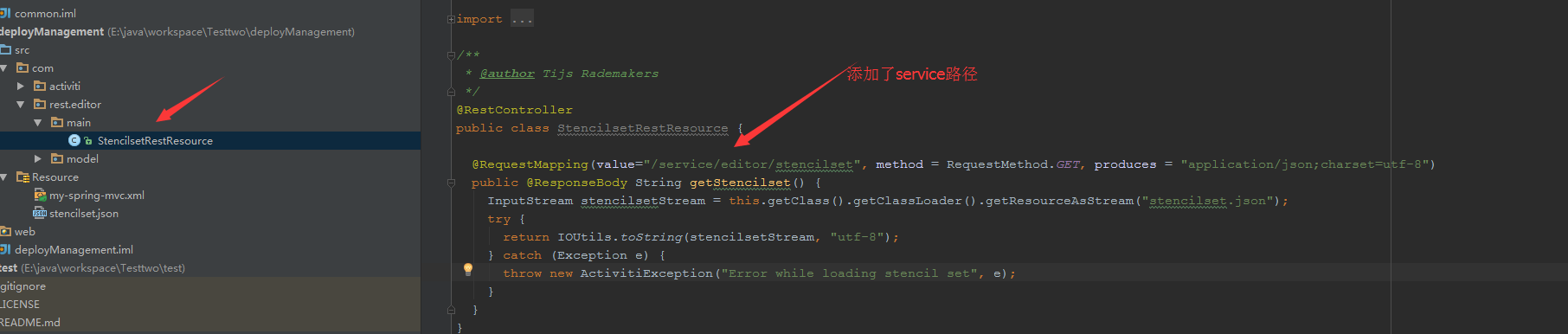
->activiti-modeler\src\main\java\org\activiti\rest\editor\model



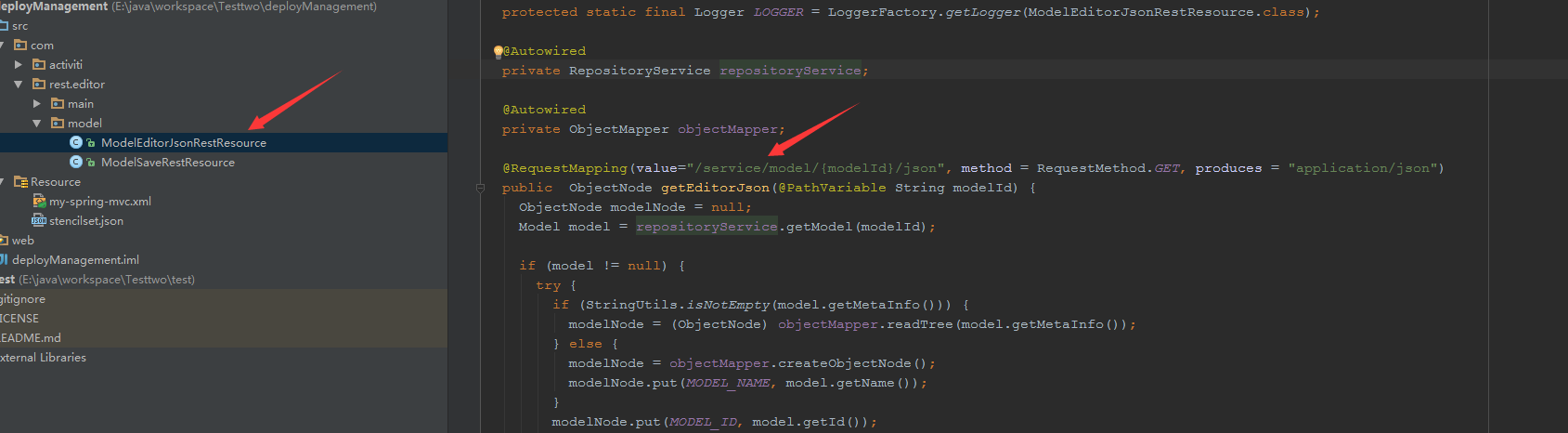
在我们项目下创建com.rest.editor.main来放StencilsetRestResource.java;com.rest.editor.model: ModelEditorJsonRestResource.java 和ModelSaveRestResource.java



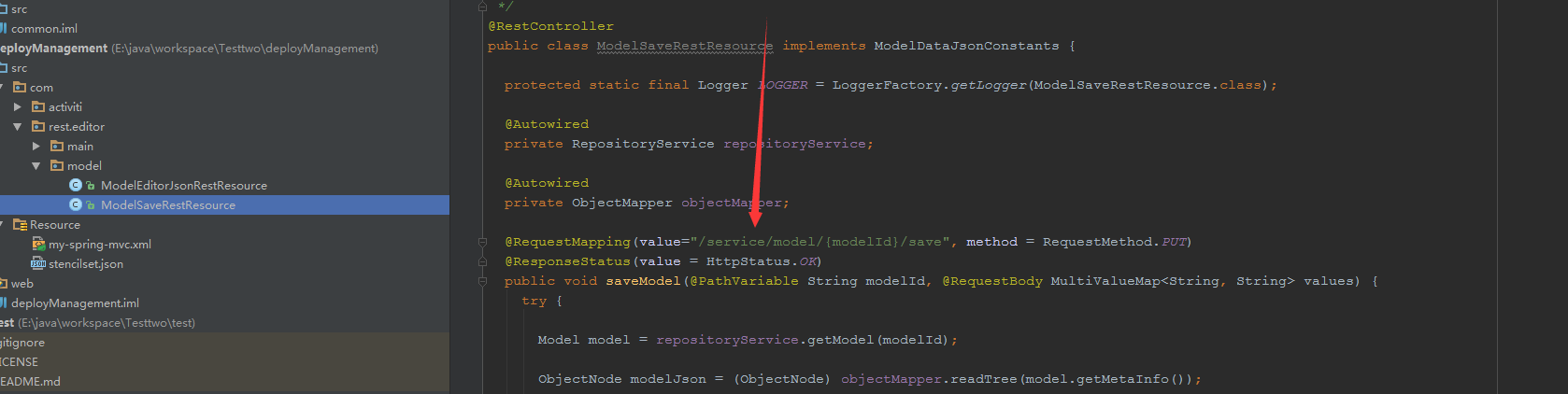
### 修改StencilsetRestResource的映射url:



### ModelEditorJsonRestResource.java url:



### ModelSaveRestResource.java url:



### 根据controller来配置：app-cfg.js

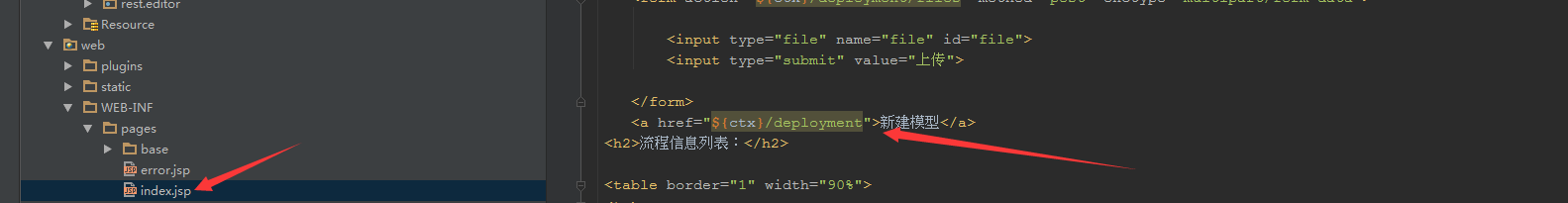


### 在自己的controller添加跳转操作：这里操作的是act\_re\_model这张表

添加如下代码：

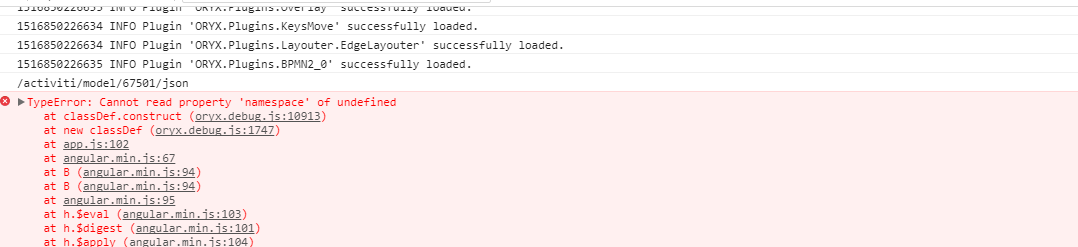
|  |
| --- |
| /\* \* \* 跳转到新建模型对象，这里需要创建一个modelId \* \*/ @RequestMapping(value = "/deployment",method = RequestMethod.*GET*) public void toCreate(HttpServletRequest request, HttpServletResponse response){  try {    ObjectMapper mapper = new ObjectMapper();//创建jackson对象  ObjectNode editorNode = mapper.createObjectNode();//创建节点  editorNode.put("id", "canvas"); //添加数据  editorNode.put("resourceId", "canvas");  ObjectNode stencilSetNode = mapper.createObjectNode();  stencilSetNode.put("namespace", "http://b3mn.org/stencilset/bpmn2.0#");  editorNode.put("stencilset", stencilSetNode);  org.activiti.engine.repository.Model modelData = repositoryService.newModel(); //使用repositoryService创建model   ObjectNode modelObjectNode = mapper.createObjectNode();  modelObjectNode.put(ModelDataJsonConstants.*MODEL\_NAME*, "hello1111");  modelObjectNode.put(ModelDataJsonConstants.*MODEL\_REVISION*, 1);  String description = "hello1111";  modelObjectNode.put(ModelDataJsonConstants.*MODEL\_DESCRIPTION*, description);  modelData.setMetaInfo(modelObjectNode.toString());  modelData.setName("hello1111");  modelData.setKey("12313123");   //保存模型  repositoryService.saveModel(modelData);  repositoryService.addModelEditorSource(modelData.getId(), editorNode.toString().getBytes("utf-8"));  //跳转到modeler.html并传递刚刚生成的modelId  response.sendRedirect(request.getContextPath() + "/plugins/procerss-editor/modeler.html?modelId=" + modelData.getId());  }catch (Exception e){  System.*err*.println("出现了异常");  } } |

### 在jsp界面中添加跳转的链接：

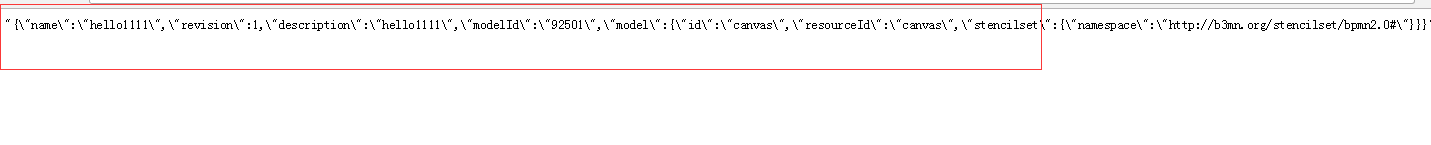


## 也许会出现的问题：

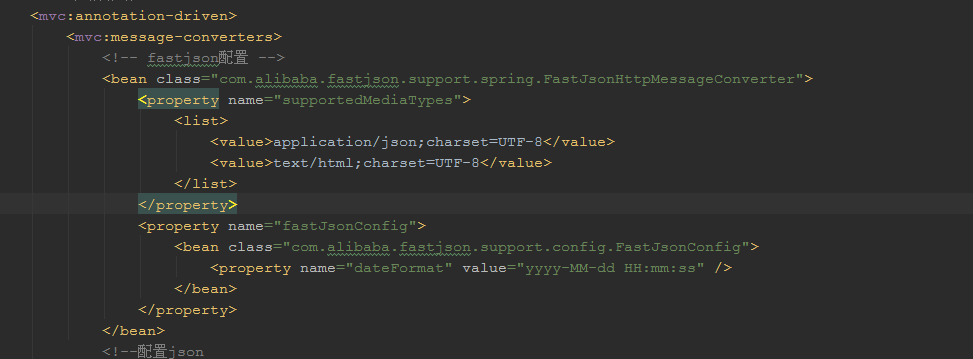
### 报下面这个错：



### 产生的原因：返回了带斜杠的json格式的数据：



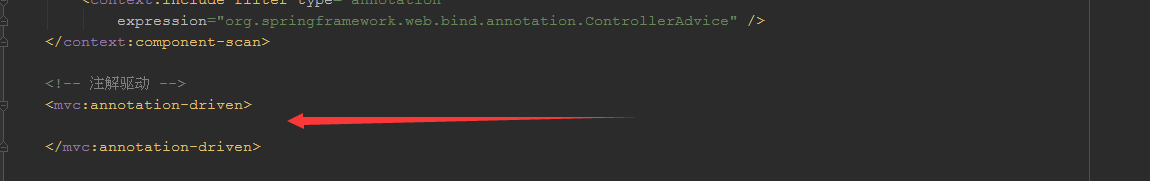
### 可是在spring中已经配置了json解析器啊？，why



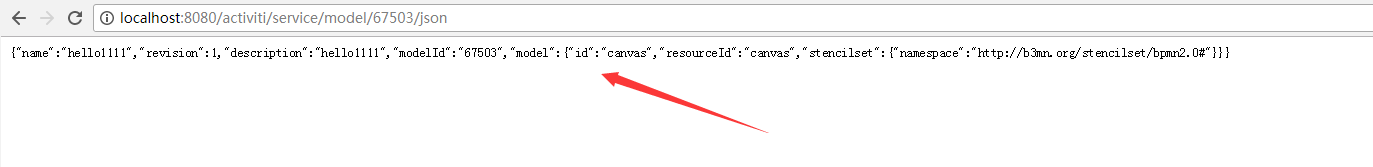
原因是json文件被二次装换了，controller返回过来的已经是json格式的文件了，在返回数据的时候又被转换了一次，导致了格式不对：controller已经返回了json数据：



## 解决办法一：删除spring mvc配置文件下的json解析：



### 重启服务在访问：得到了正确的数据



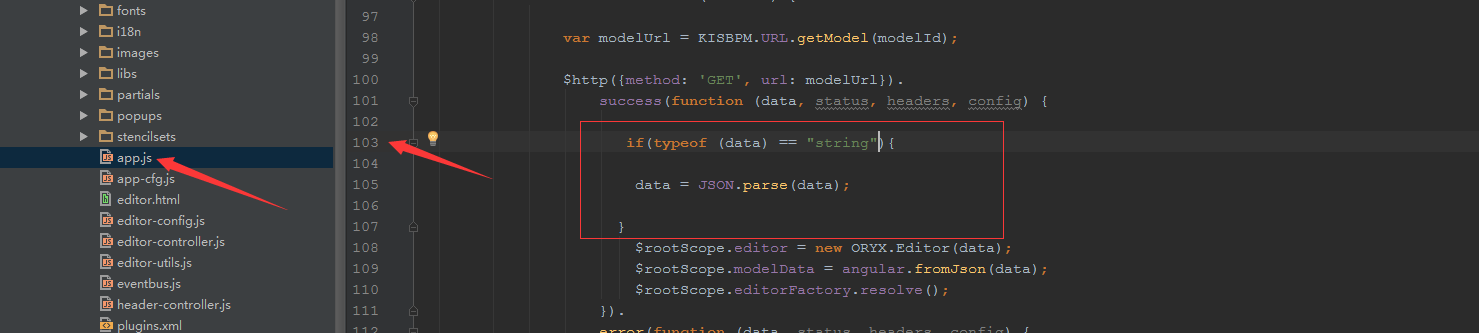
## 解决方法二：修改js文件

修改oryx.debug.js：





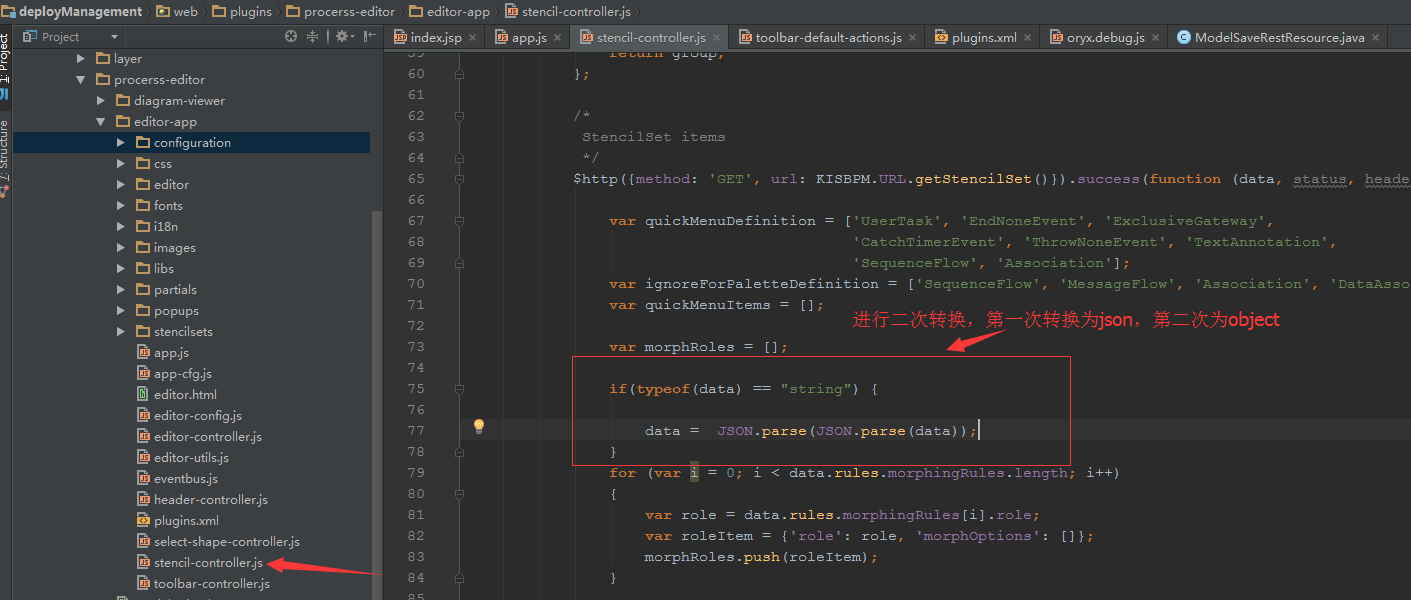
修改：app.js



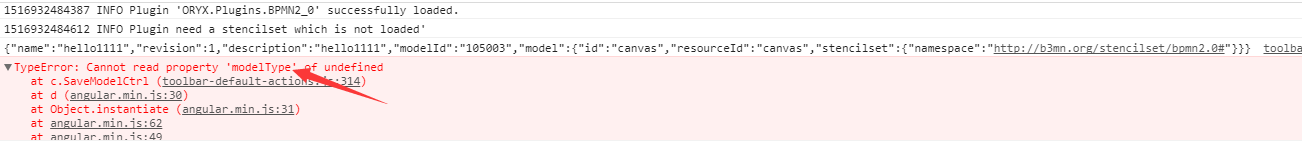
Js中如果报这个错：

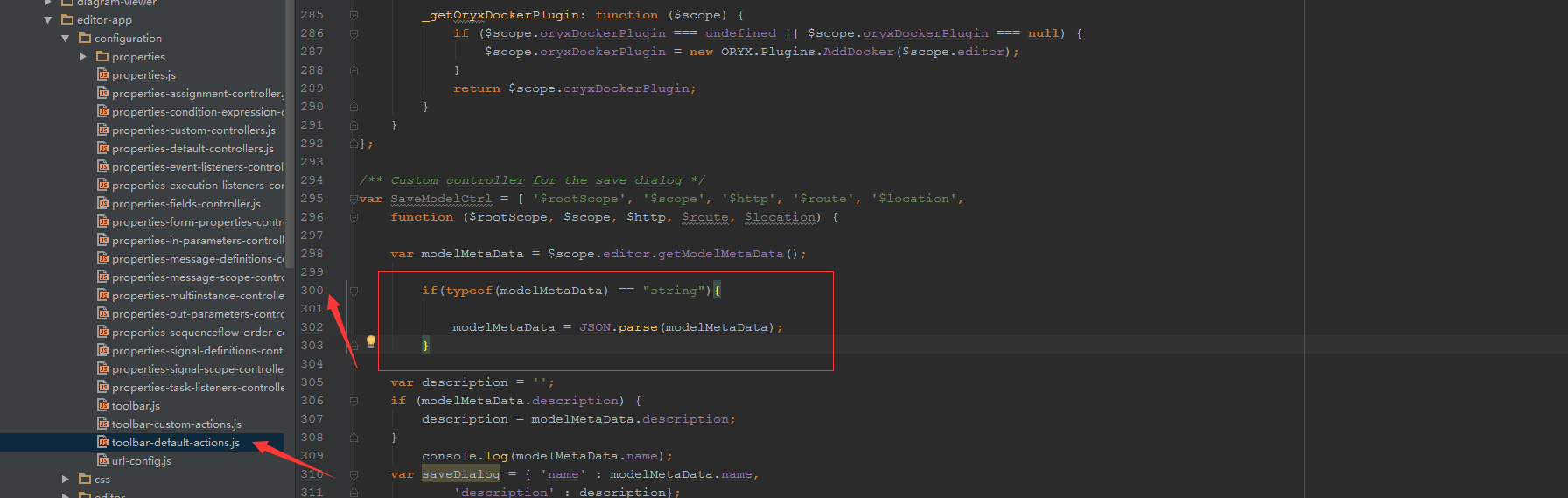
|  |
| --- |
| TypeError: Cannot read property 'split' of undefined     at Object.ORYX.Core.StencilSet.stencilSet (oryx.debug.js:8640)     at oryx.debug.js:8619     at prototype-1.5.1.js:446     at Array.Object.extend.\_each (prototype-1.5.1.js:672)     at Array.Enumerable.each (prototype-1.5.1.js:445)     at Object.ORYX.Core.StencilSet.stencilSets (oryx.debug.js:8618)     at ORYX.Editor.getStencilSets (oryx.debug.js:11705)     at ORYX.Editor.\_createCanvas (oryx.debug.js:11192)     at ORYX.Editor.construct (oryx.debug.js:10921) |

stencil-controller.js：



报下面那个错：





总结：如果报属性没有识别到的都是由于数据无法被解析，在js中转换即可

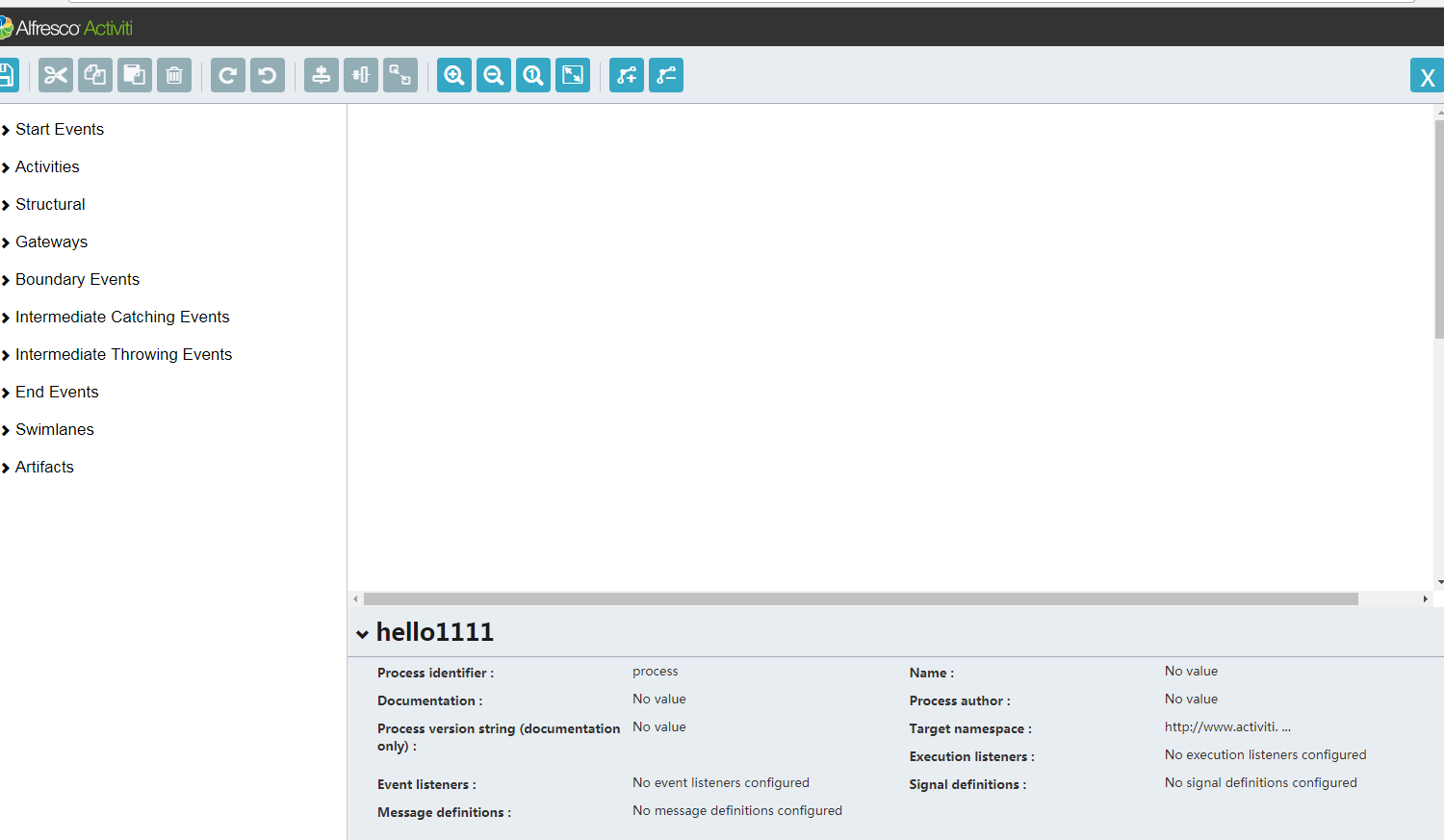
保存的js在editor-app/configuration/properties



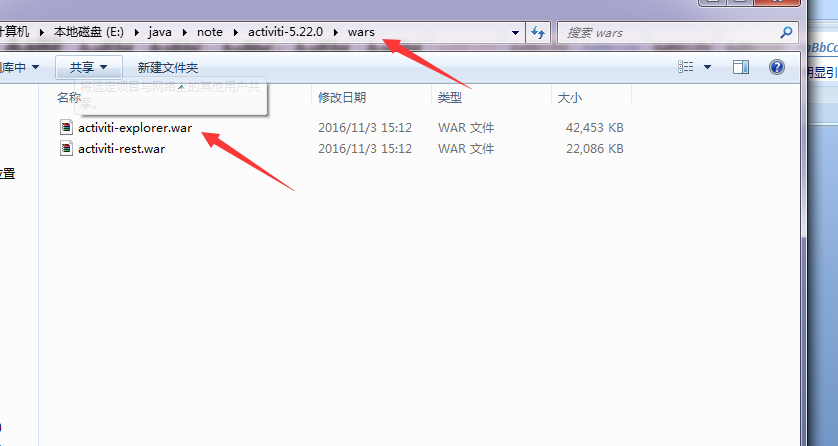
## 改了这些地方之后重新访问界面：



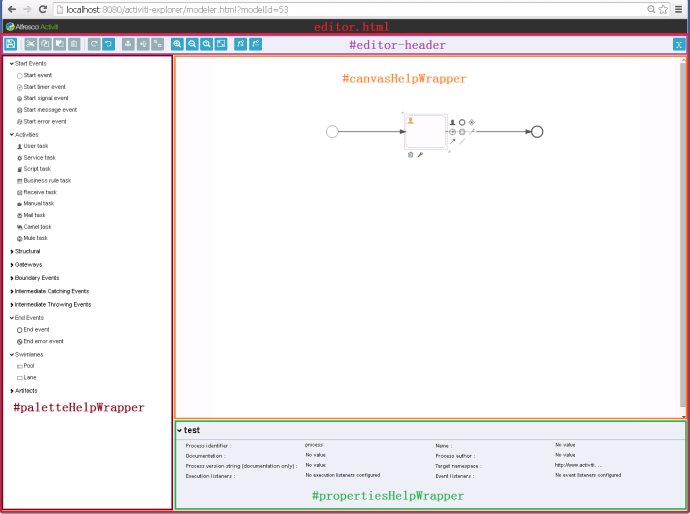
点击成功进入：



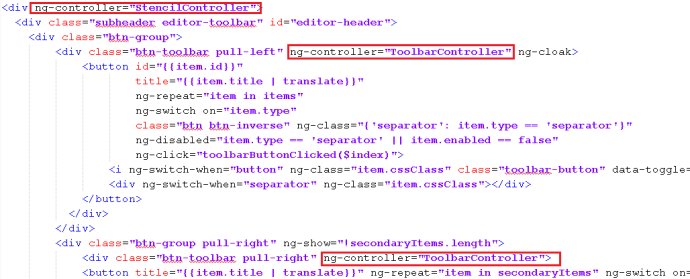
提醒：如果你使用导入jar的方式可以到activiti的开发包那里解压activiti-explorer.war找到对应的jar



## Activit modeler界面介绍：

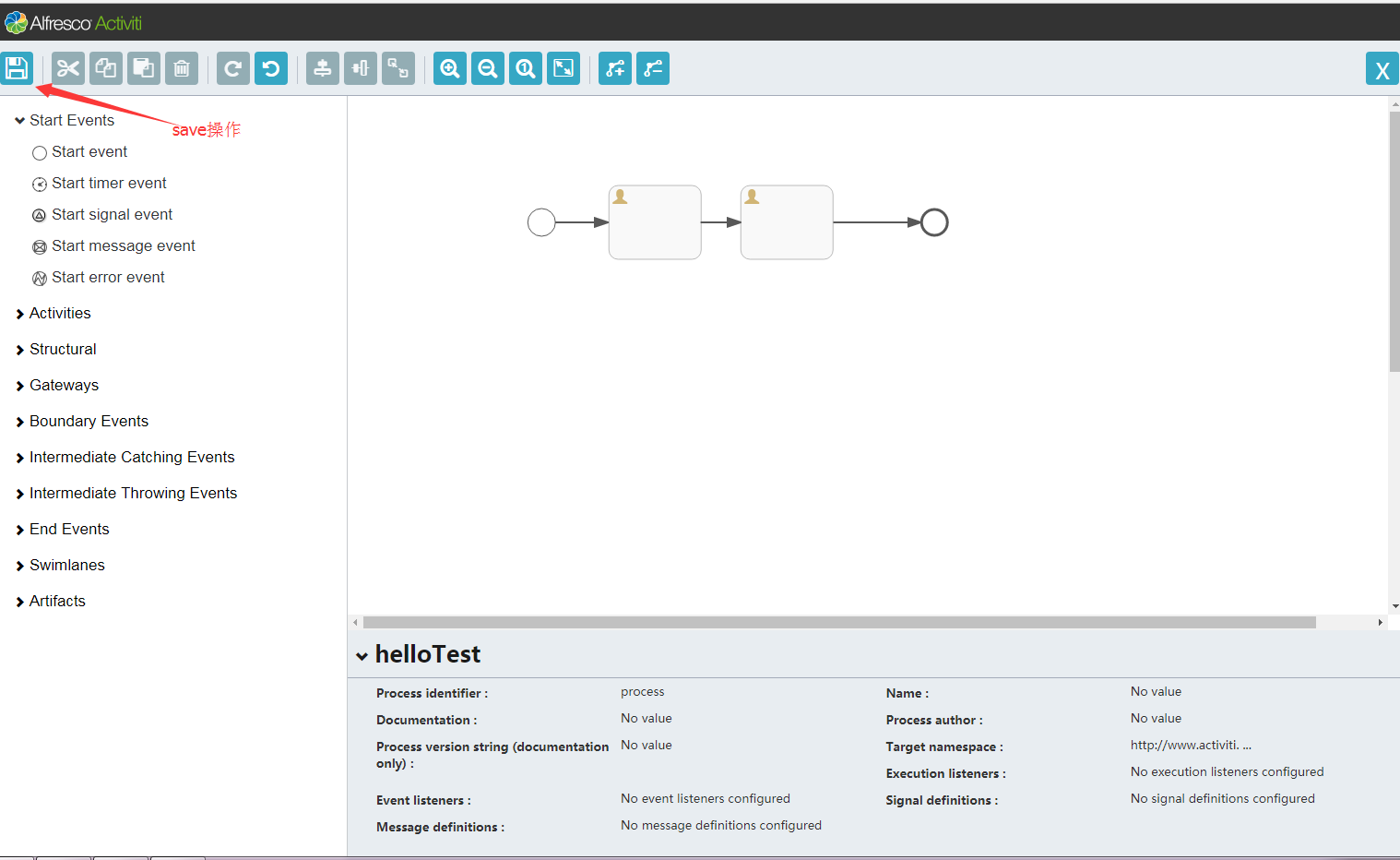
该目录下的editor.html是Activiti Modeler Editor的主界面HTML代码  
[](http://photo.blog.sina.com.cn/showpic.html#blogid=72ef7bea0102vjb8&url=http://album.sina.com.cn/pic/0026uWfMgy6Qex1KLT311)  
其中palette区是通过Angular.JS使用stencilsets\bpmn2.0\icons下多个子目录内的PNG图像形成的多组列表。其节点层次关系获取相关代码为：

1. stencil-controller.js
2. Activiti\modules\activiti-modeler\src\main\java\org\activiti\rest\editor\main\StencilsetRestResource.java
3. Activiti\modules\activiti-webapp-explorer2\src\main\resources\stencilset.json

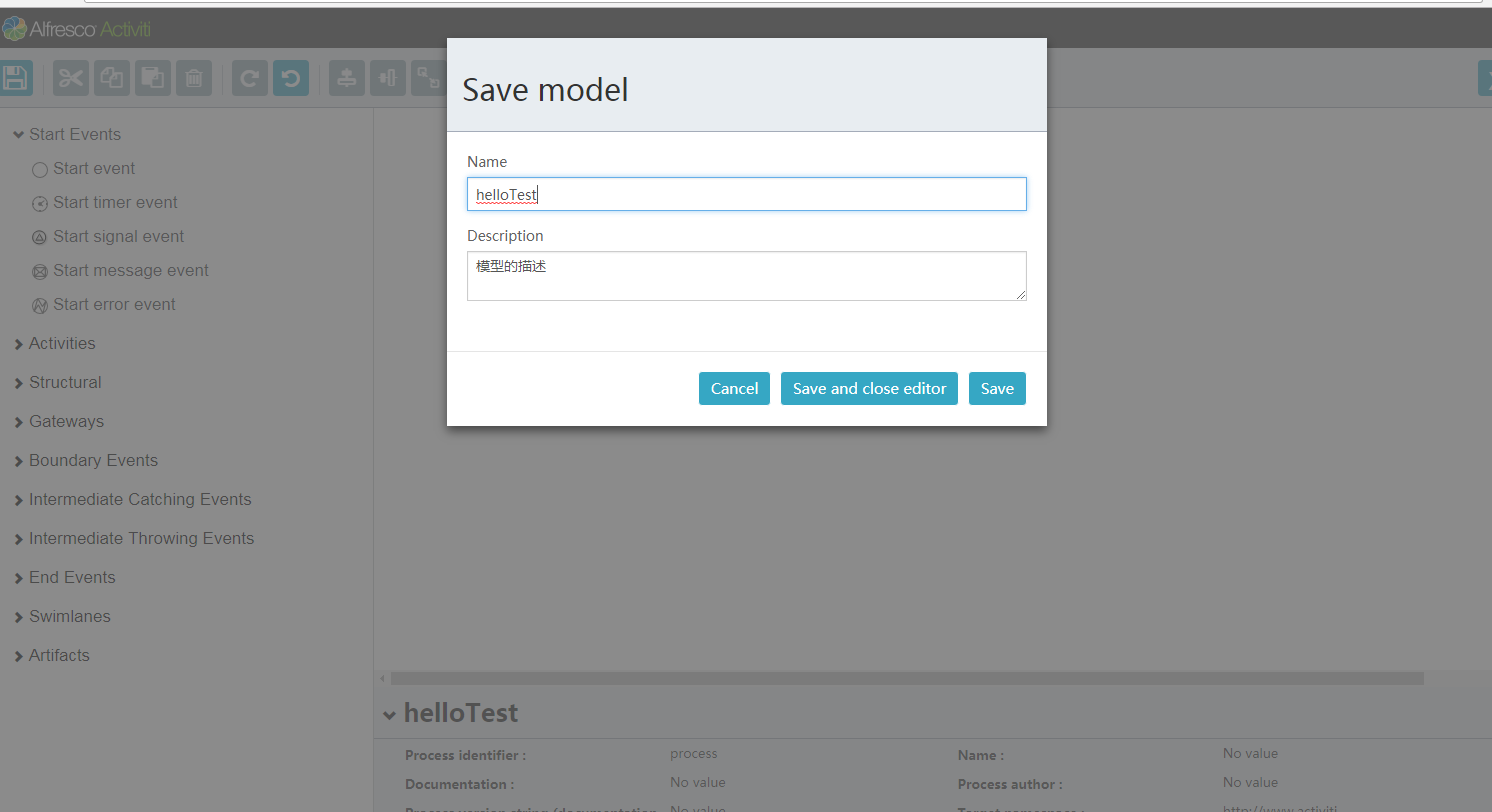
[](http://photo.blog.sina.com.cn/showpic.html#blogid=72ef7bea0102vjb8&url=http://album.sina.com.cn/pic/0026uWfMgy6QeECefVIa6)  
editor.html中的视图与两个控制器进行了绑定:

* stencil-controller.js：处理对canvas中BPMN元素的操作，很多处理是通过editor目录下的QRYX库完成的
* toolbar-controller.js：处理对工具栏的操作，很多处理由configuration\toolbar-default-actions.js完成

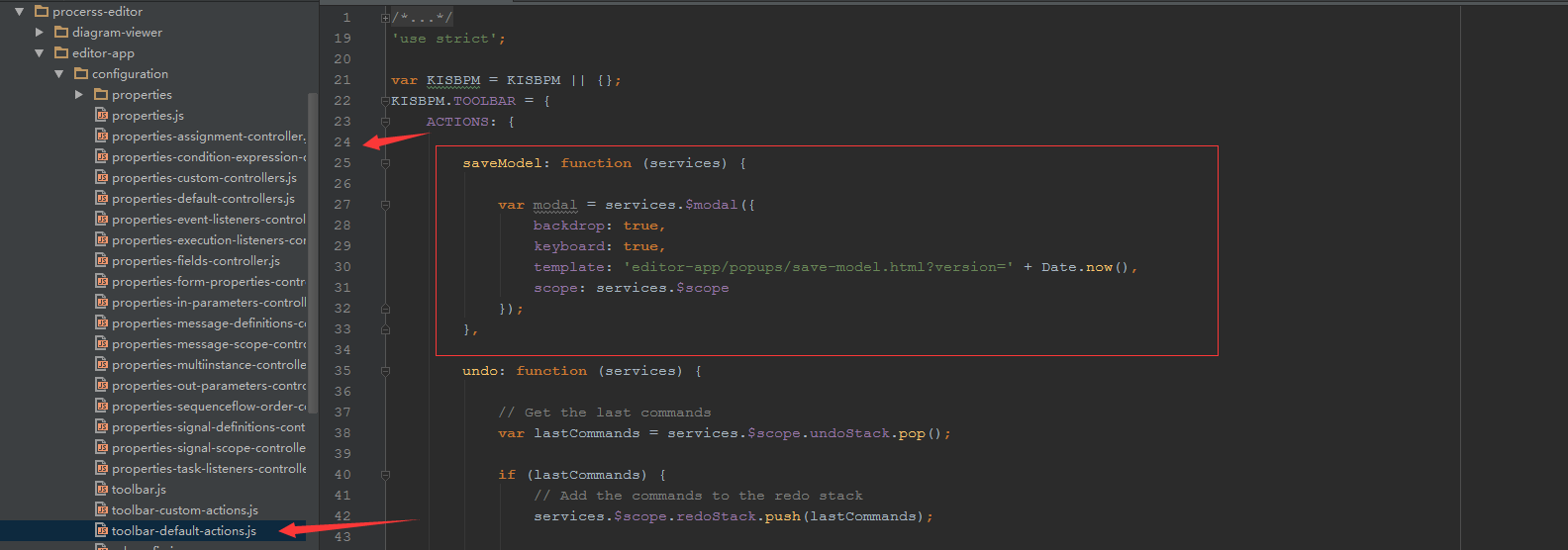
## 把编辑好的模型保存到数据库中：点击save操作



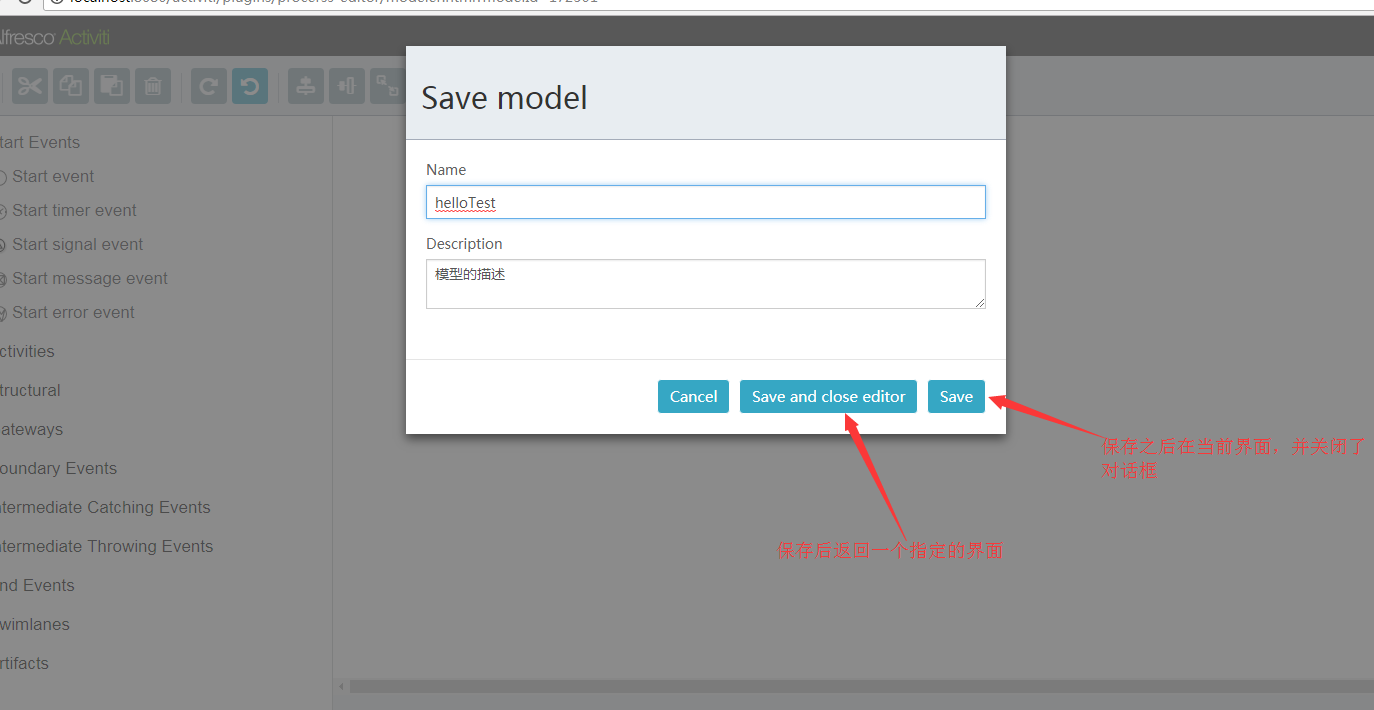
弹出对话框：

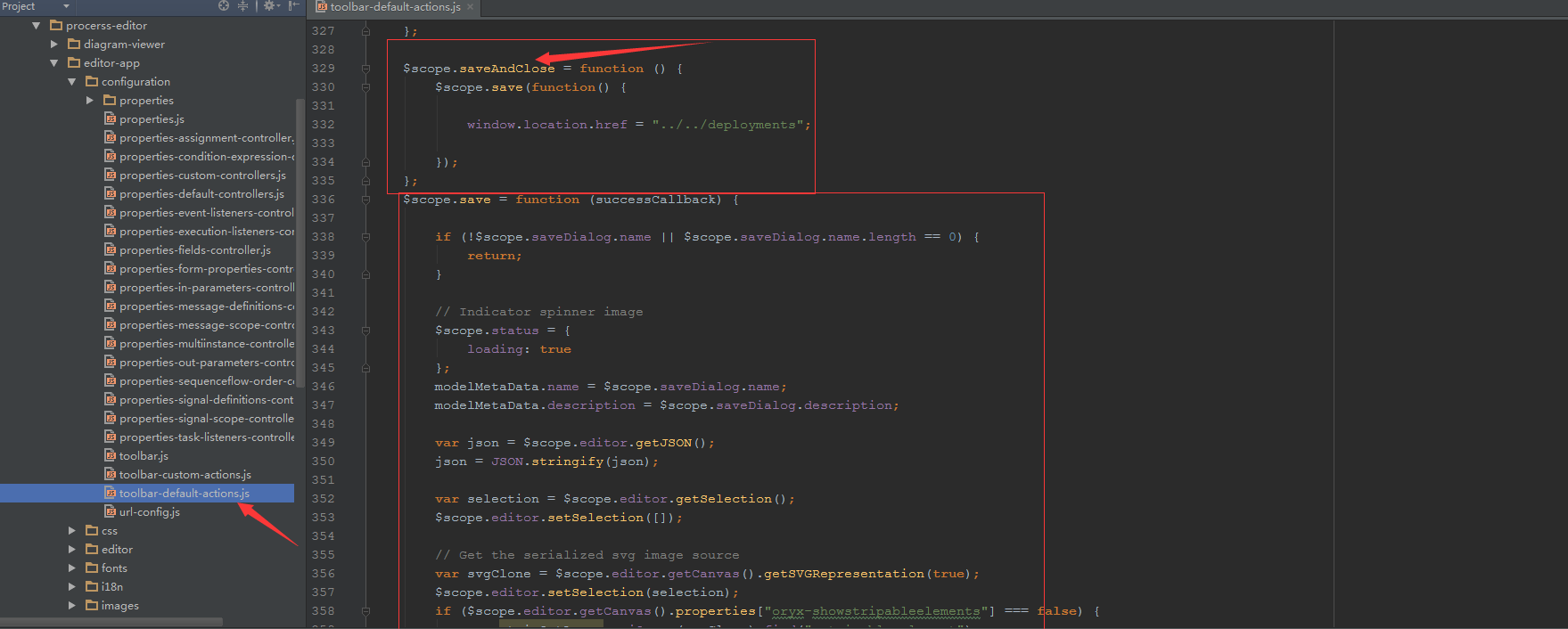


该对话框的js主要在configuration\toolbar-default-actions.js



### 按钮介绍：

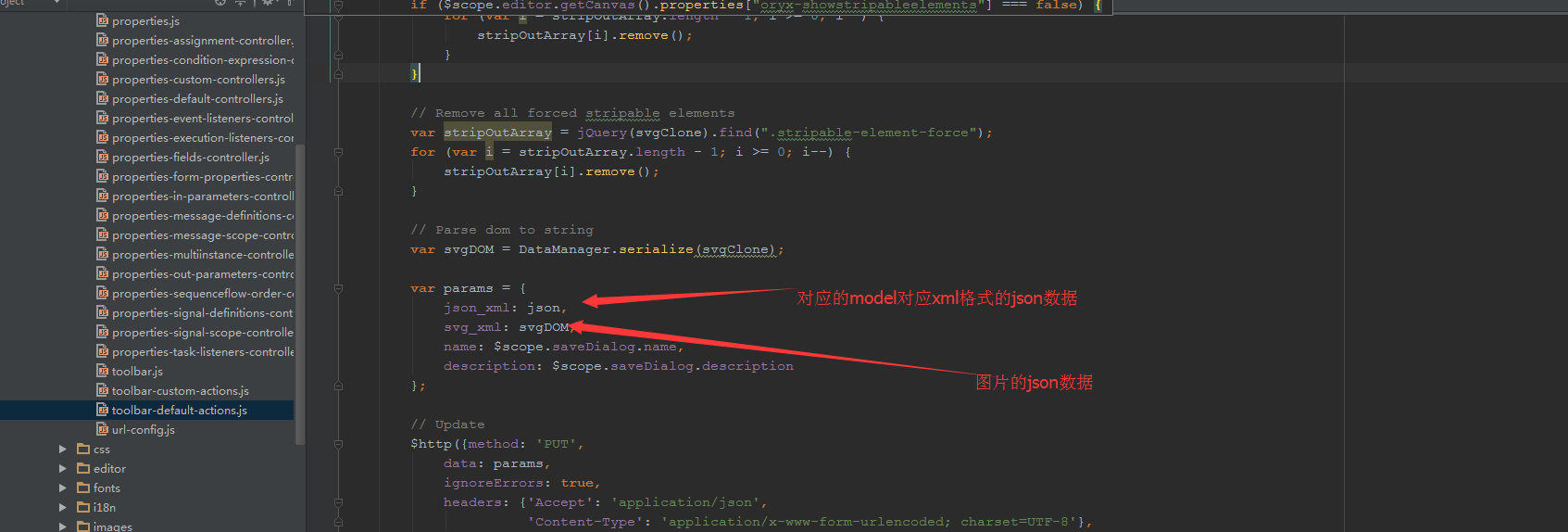




### 保存的URL主要配置在：url-config.js中



### 传递到后台的参数：

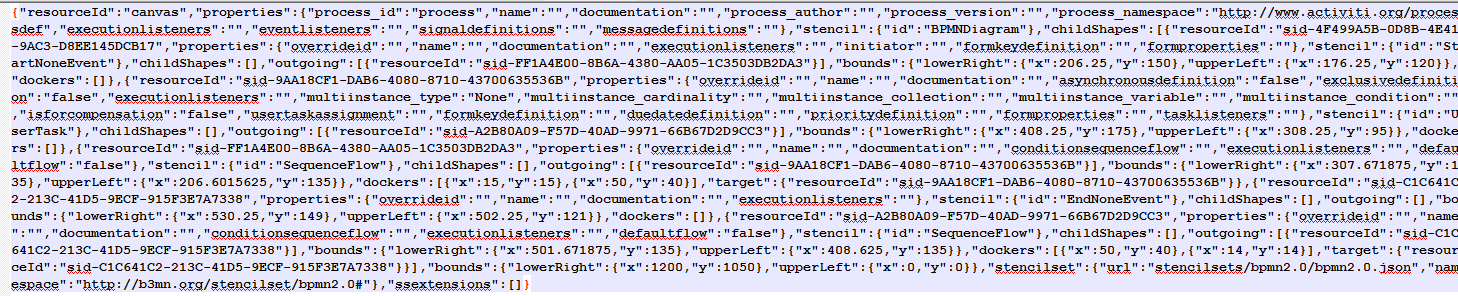


### 当点击提交数据到数据库的时候，

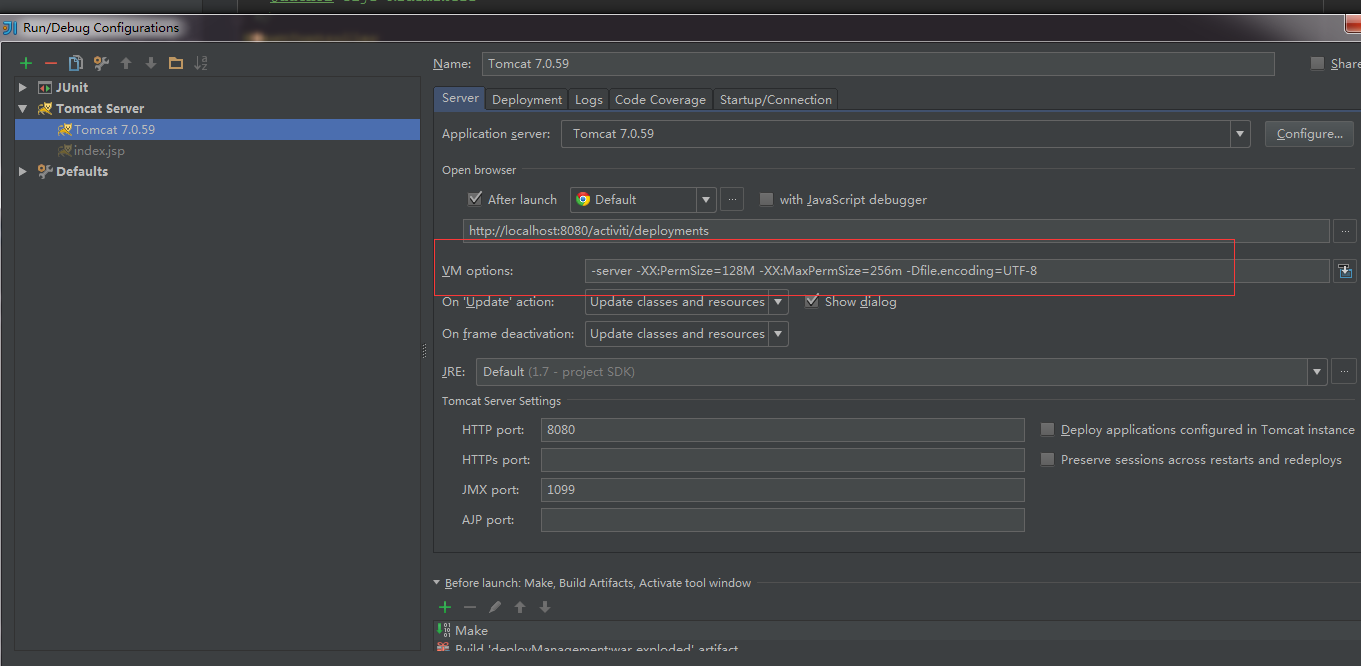
#### 提交到的后台controller是：



#### 获得的Json\_xml数据：



注意：如果你的IDEA内存配置的不够，在进行数据入库的时候，也许入到一半会出现内存溢出异常，这个时候，可以调整IDEA的tomcat运行内存的大小：

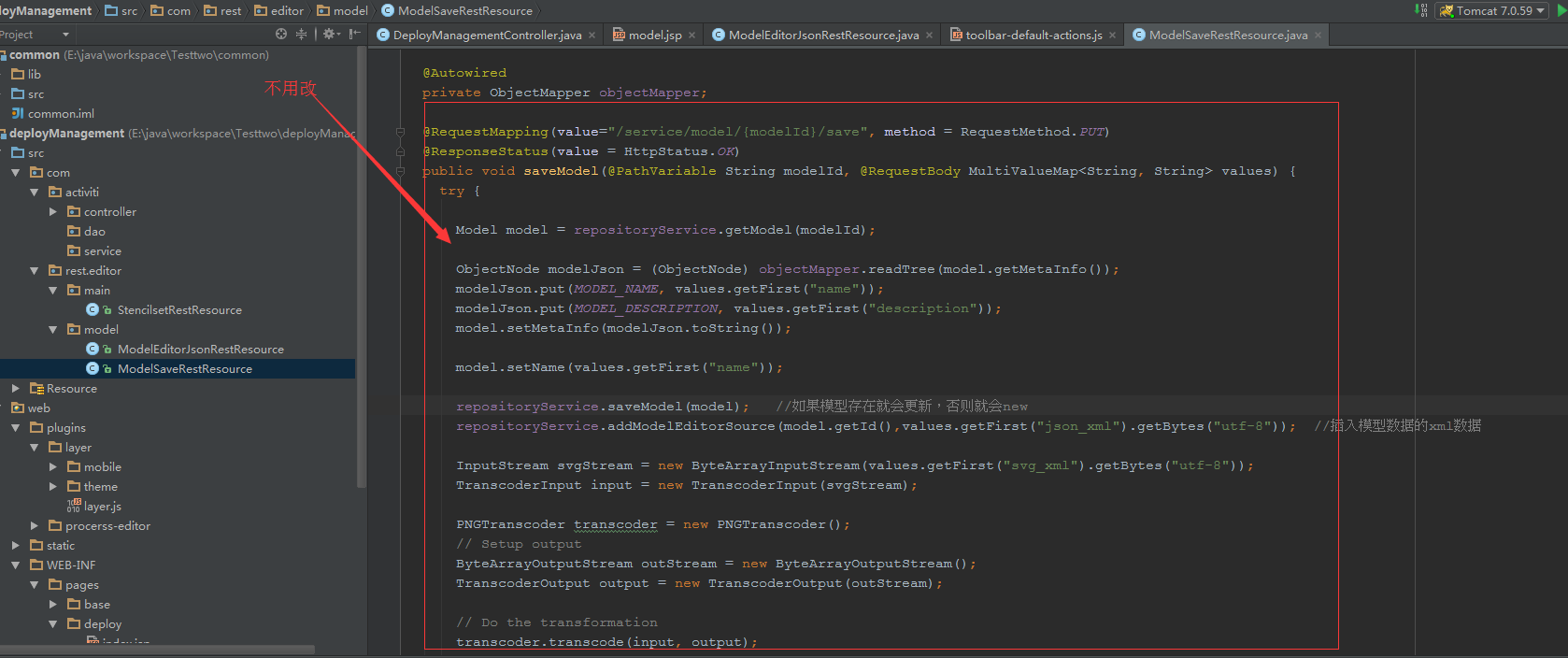


## 在模型保存的时候，数据库可以保存两种形式的数据：

##### 第一种方式在数据库保存的是json：



##### 处理保存的controller可以直接使用官方的ModelSaveRestResource



##### 输出model对应的xml的时候，再把json进行转换：

代码如下：

|  |
| --- |
| /\*  \* @Author:王钟鑫  \* @Description:获取到模型的xml信息或者png的数据  \* @Param:modelId图片资源id,editorSourceValueId 获取xml，editorSourceExtraValueId 获取 png图片  \* @Date:2018/1/27\_14:30  \* \*/ @RequestMapping(value = "/deployment/model/resources",method = RequestMethod.*GET*) public void getModelXmlResource(@RequestParam("id") String modelId,@RequestParam(value = "vid" ,defaultValue = "",required = false) String editorSourceValueId,@RequestParam(value = "evid" ,defaultValue = "",required = false) String editorSourceExtraValueId, HttpServletResponse response){   //获取到对应的模型  org.activiti.engine.repository.Model model = repositoryService.getModel(modelId);  byte[] modelEditorSourceExtra = null;   //查询模型的原数据 xml  if(!editorSourceValueId.equals("")){  modelEditorSourceExtra = repositoryService.getModelEditorSource(model.getId());  JsonNode editNode = null;  try {  editNode = new ObjectMapper().readTree(modelEditorSourceExtra); //读取节点  BpmnJsonConverter jsonConverter = new BpmnJsonConverter();//  BpmnModel bpmnModel = jsonConverter.convertToBpmnModel(editNode);  modelEditorSourceExtra = new BpmnXMLConverter().convertToXML(bpmnModel);//转化为对应的xml  } catch (IOException e) {  e.printStackTrace();  }   }else{ //获取到png图片   modelEditorSourceExtra = repositoryService.getModelEditorSourceExtra(model.getId());   }   //输出到界面  InputStream put = new ByteArrayInputStream(modelEditorSourceExtra);   byte[] b = new byte[1024];  int len = -1;  try {  while ((len = put.read(b,0,1024)) != -1){   response.getOutputStream().write(b,0,len);  }   } catch (IOException e) {  e.printStackTrace();  }  } |

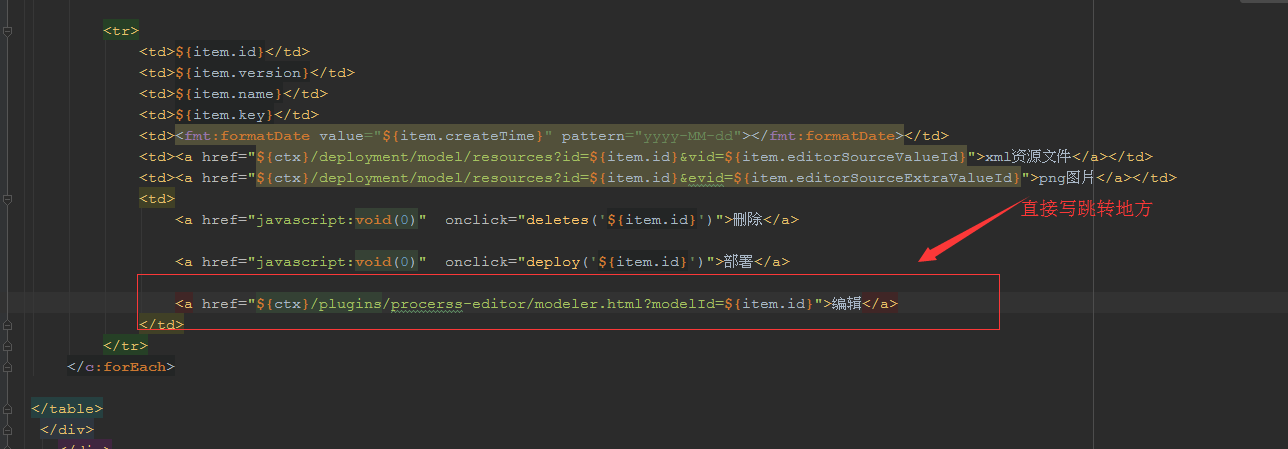
##### 在部署model的时候：也要进行json转化为xml



代码：

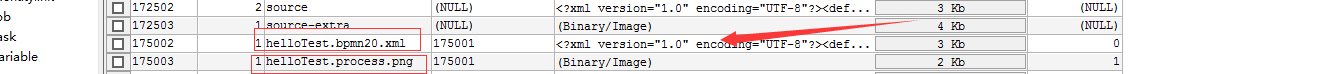
|  |
| --- |
| /\*  \* @Author:王钟鑫  \* @Description:部署流程  \* @Param:modelId 模型编号  \* @Date:2018/1/27\_15:21  \*  \* \*/ @RequestMapping(value = "/deployment/model/deployment/{id}",method = RequestMethod.*PUT*) public @ResponseBody Object deployByModel(@PathVariable("id") String modelId){   try {  org.activiti.engine.repository.Model model = repositoryService.getModel(modelId);   byte[] modelEditorSource = repositoryService.getModelEditorSource(model.getId());   //把json进行转化为xml  JsonNode editNode = null;  editNode = new ObjectMapper().readTree(modelEditorSource); //读取节点  BpmnJsonConverter jsonConverter = new BpmnJsonConverter();//  BpmnModel bpmnModel = jsonConverter.convertToBpmnModel(editNode);  modelEditorSource = new BpmnXMLConverter().convertToXML(bpmnModel);//转化为对应的xml  String processName = model.getName() + ".bpmn20.xml";  //部署  repositoryService.createDeployment().name(model.getName())  .addString(processName,new String(modelEditorSource,"utf-8"))  .deploy();  return R.*success*();   } catch (UnsupportedEncodingException e) {   e.printStackTrace();  }catch (Exception e){   e.printStackTrace();  }  return R.*failure*();  } |

##### 编辑的时候直接调整：数据库保存json的这种方式，好处在于编辑的时候可以直接通过连接进行跳转，不用在转化：

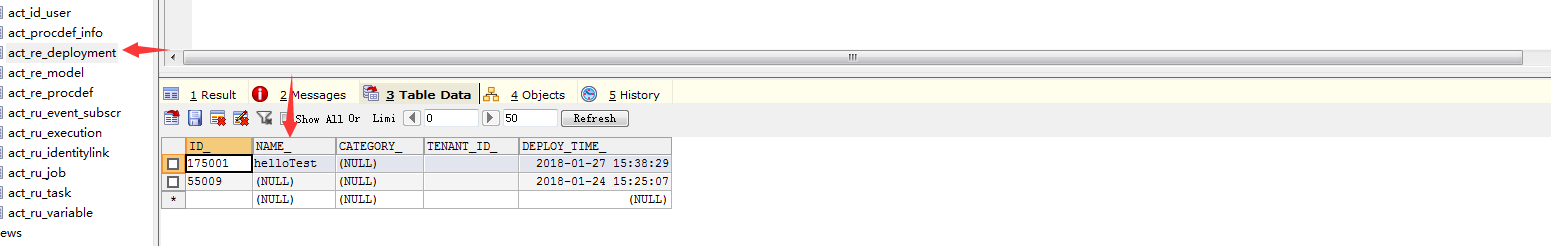


# 数据库的变化：

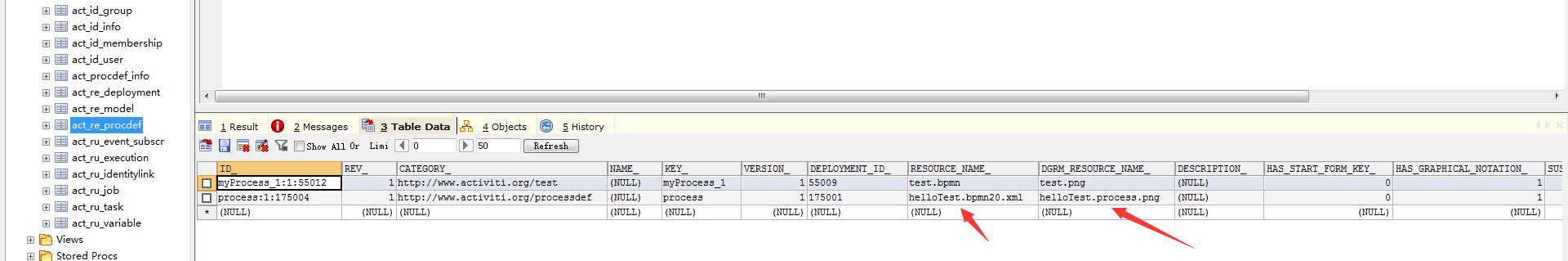
##### 当你部署模型的时候，act\_ge\_bytearray表会重新添加两条记录：



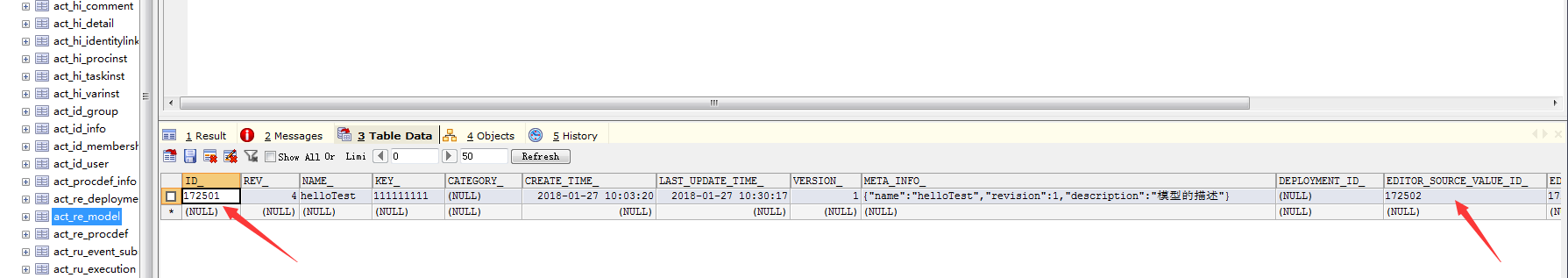
##### act\_re\_deployment表也会添加记录：



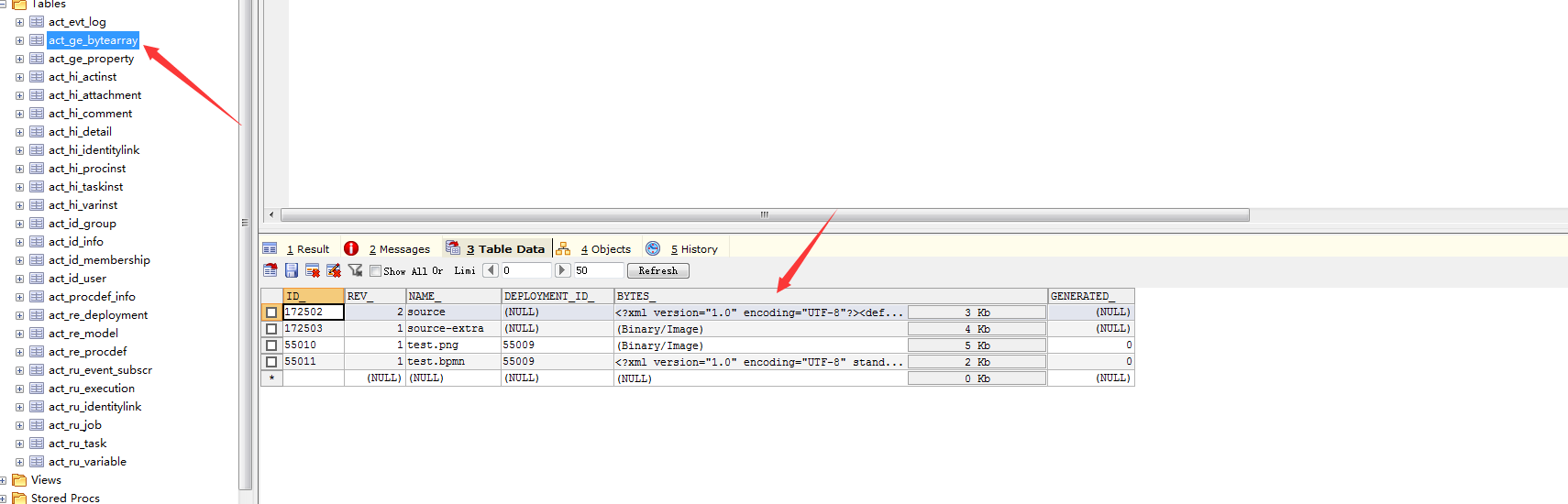
##### Act\_re\_procdef表也会添加对应的记录：



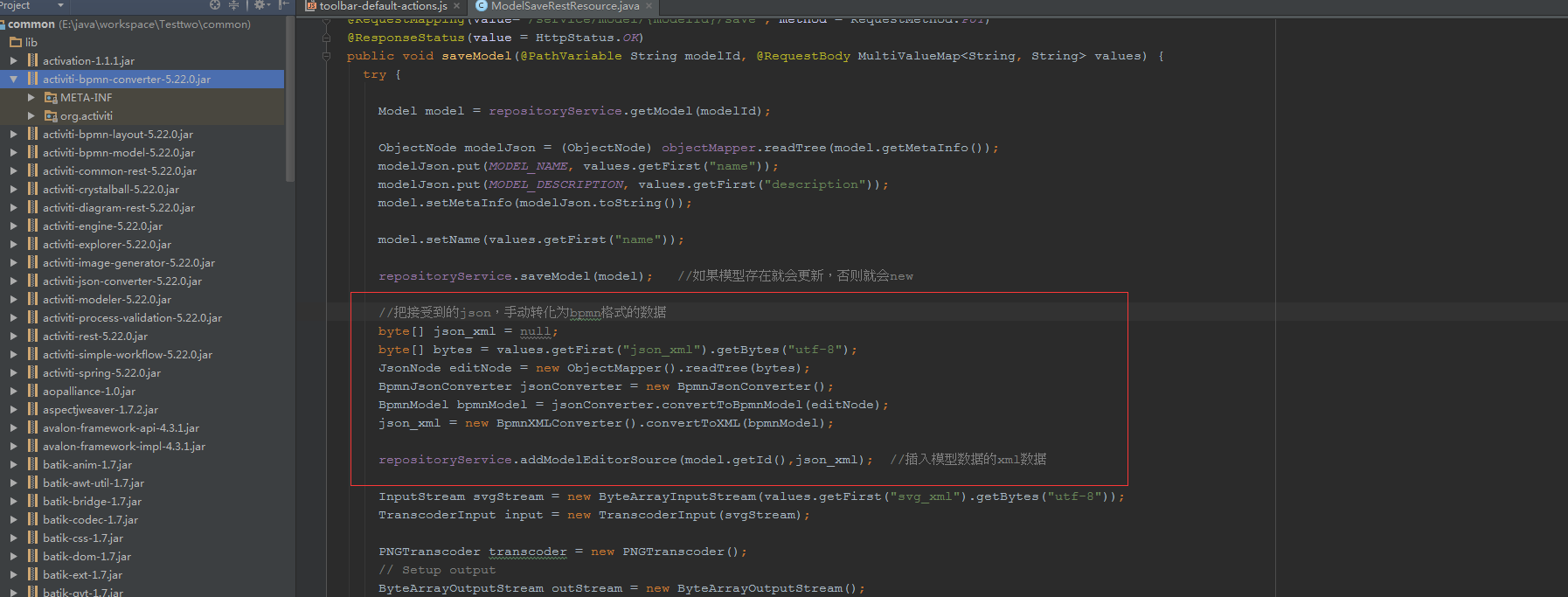
act\_re\_model表保存的是模型的一些基本定义：和对应存储在字节表中的数据所对应的数据



act\_ge\_bytearray表 用来存储对应的xml和图片



## 方法二，当你在数据库保存的是xml的时候：就是在保存数据的时候直接把json转化成xml进行保存



PS：好处在于你在部署模型，保存模型的时候，不需要再进行转化

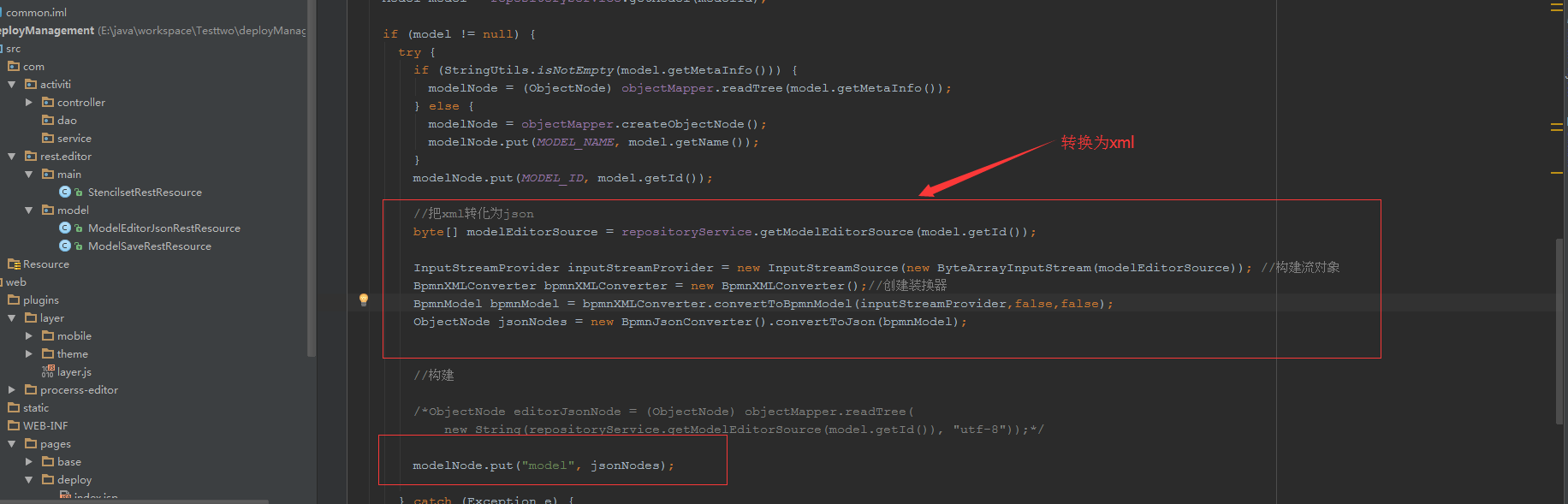
###### 缺点：就是如果你在编辑的时候，直接跳转链接就会报下面的错：



这个时候我们可以修改ModelEditorJsonRestResource.java在返回对象的时候，对数据进行转化为json数据



修改代码如下：

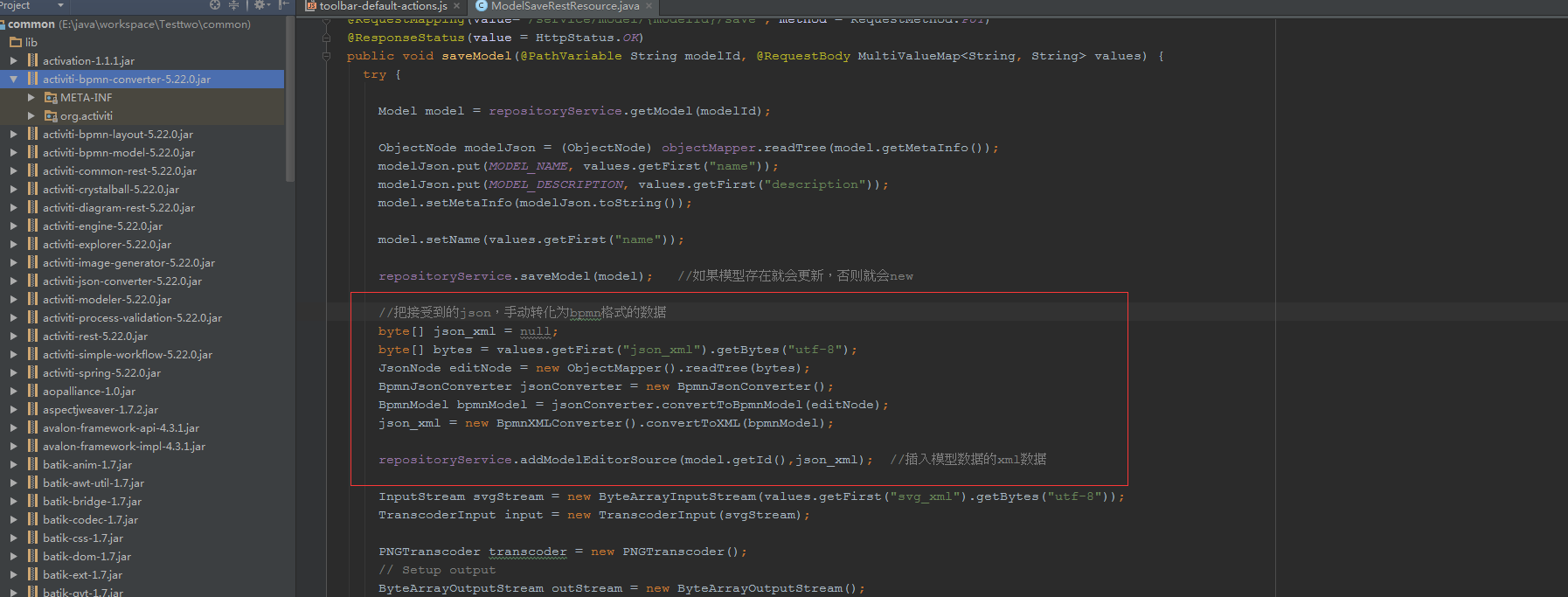


|  |
| --- |
| */\*\*  \** ***@author*** *Tijs Rademakers  \*/* @RestController public class ModelEditorJsonRestResource implements ModelDataJsonConstants {    protected static final Logger *LOGGER* = LoggerFactory.*getLogger*(ModelEditorJsonRestResource.class);    @Autowired  private RepositoryService repositoryService;    @Autowired  private ObjectMapper objectMapper;    @RequestMapping(value="/service/model/{modelId}/json", method = RequestMethod.*GET*, produces = "application/json")  public ObjectNode getEditorJson(@PathVariable String modelId) {  ObjectNode modelNode = null;  Model model = repositoryService.getModel(modelId);   if (model != null) {  try {  if (StringUtils.*isNotEmpty*(model.getMetaInfo())) {  modelNode = (ObjectNode) objectMapper.readTree(model.getMetaInfo());  } else {  modelNode = objectMapper.createObjectNode();  modelNode.put(*MODEL\_NAME*, model.getName());  }  modelNode.put(*MODEL\_ID*, model.getId());   //把xml转化为json  byte[] modelEditorSource = repositoryService.getModelEditorSource(model.getId());   InputStreamProvider inputStreamProvider = new InputStreamSource(new ByteArrayInputStream(modelEditorSource)); //构建流对象  BpmnXMLConverter bpmnXMLConverter = new BpmnXMLConverter();//创建装换器  BpmnModel bpmnModel = bpmnXMLConverter.convertToBpmnModel(inputStreamProvider,false,false);  ObjectNode jsonNodes = new BpmnJsonConverter().convertToJson(bpmnModel);    //构建   /\*ObjectNode editorJsonNode = (ObjectNode) objectMapper.readTree(  new String(repositoryService.getModelEditorSource(model.getId()), "utf-8"));\*/   modelNode.put("model", jsonNodes);    } catch (Exception e) {  *LOGGER*.error("Error creating model JSON", e);  throw new ActivitiException("Error creating model JSON", e);  }  }   return modelNode;  } } |

## 保存方法时的两种方式如下：

解决方法：使用官方的activiti-bpmn-converter-5.22.0.jar包下的文件进行手动装换数据，把数据变成对应的xml数据的文件，在保存到数据库中：

方法一：转完再保存到数据库，这个时候里面保存的是xml，当你需要编辑的时候需要把xml转化为json



方法二：在部署模型的时候再进行装换为xml：



总结：如果你表里面存的是xml，那么你在重新编辑模型的时候，就需要把xml转为json。如果你表里面存的是json，直接取出来进行编辑。不过当你在保存模型部署的时候，那么你需要转化为xml在进行部署

### 最终修改后的代码如下：ModelSaveRestResource.java

|  |
| --- |
| @RestController public class ModelSaveRestResource implements ModelDataJsonConstants {    protected static final Logger *LOGGER* = LoggerFactory.*getLogger*(ModelSaveRestResource.class);   @Autowired  private RepositoryService repositoryService;    @Autowired  private ObjectMapper objectMapper;    @RequestMapping(value="/service/model/{modelId}/save", method = RequestMethod.*PUT*)  @ResponseStatus(value = HttpStatus.*OK*)  public void saveModel(@PathVariable String modelId, @RequestBody MultiValueMap<String, String> values) {  try {  Model model = repositoryService.getModel(modelId);  ObjectNode modelJson = (ObjectNode) objectMapper.readTree(model.getMetaInfo());  modelJson.put(*MODEL\_NAME*, values.getFirst("name"));  modelJson.put(*MODEL\_DESCRIPTION*, values.getFirst("description"));  model.setMetaInfo(modelJson.toString());   model.setName(values.getFirst("name"));   repositoryService.saveModel(model); //如果模型存在就会更新，否则就会new   //把接受到的json，手动转化为bpmn格式的数据  /\* byte[] json\_xml = null;  byte[] bytes = values.getFirst("json\_xml").getBytes("utf-8");  JsonNode editNode = new ObjectMapper().readTree(bytes);  BpmnJsonConverter jsonConverter = new BpmnJsonConverter();  BpmnModel bpmnModel = jsonConverter.convertToBpmnModel(editNode);  json\_xml = new BpmnXMLConverter().convertToXML(bpmnModel);\*/   repositoryService.addModelEditorSource(model.getId(),values.getFirst("json\_xml").getBytes("utf-8")); //插入模型数据的xml数据    InputStream svgStream = new ByteArrayInputStream(values.getFirst("svg\_xml").getBytes("utf-8"));  TranscoderInput input = new TranscoderInput(svgStream);    PNGTranscoder transcoder = new PNGTranscoder();  // Setup output  ByteArrayOutputStream outStream = new ByteArrayOutputStream();  TranscoderOutput output = new TranscoderOutput(outStream);    // Do the transformation  transcoder.transcode(input, output);  final byte[] result = outStream.toByteArray();  repositoryService.addModelEditorSourceExtra(model.getId(), result); //插入模型的svg图片  outStream.close();  } catch (Exception e) {  e.printStackTrace();  *LOGGER*.error("Error saving model", e);  throw new ActivitiException("Error saving model", e);  }  } } |

Json转化为xml代码如下:

|  |
| --- |
| byte[] json\_xml = null;  byte[] bytes = values.getFirst("json\_xml").getBytes("utf-8");  JsonNode editNode = new ObjectMapper().readTree(bytes);  BpmnJsonConverter jsonConverter = new BpmnJsonConverter();  BpmnModel bpmnModel = jsonConverter.convertToBpmnModel(editNode);  json\_xml = new BpmnXMLConverter().convertToXML(bpmnModel); |

Xml转化为json代码如下：

|  |
| --- |
| byte[] modelEditorSource = repositoryService.getModelEditorSource(model.getId());  InputStreamProvider inputStreamProvider = new InputStreamSource(new ByteArrayInputStream(modelEditorSource)); //构建流对象  BpmnXMLConverter bpmnXMLConverter = new BpmnXMLConverter();//创建装换器  BpmnModel bpmnModel = bpmnXMLConverter.convertToBpmnModel(inputStreamProvider,false,false);  ObjectNode jsonNodes = new BpmnJsonConverter().convertToJson(bpmnModel); |

### 删除model

|  |
| --- |
| /\* \* 删除model \* @Param modelId 部署Id \* \*/  @ResponseBody  @RequestMapping(value = "/deployment/model/{id}",method = RequestMethod.*DELETE*)  public Object deleteModel(@PathVariable("id")String modelId){   try {  repositoryService.deleteModel(modelId);  return R.*success*();  }catch (Exception e){   e.printStackTrace();   }   return R.*failure*();    } |

###### 界面效果截图：







Controller代码：

|  |
| --- |
| package com.activiti.controller;  import com.demo.util.R; import com.fasterxml.jackson.databind.JsonNode; import com.fasterxml.jackson.databind.ObjectMapper; import com.fasterxml.jackson.databind.introspect.MemberKey; import com.fasterxml.jackson.databind.node.ObjectNode; import org.activiti.bpmn.converter.BpmnXMLConverter; import org.activiti.bpmn.model.BpmnModel; import org.activiti.editor.constants.ModelDataJsonConstants; import org.activiti.editor.language.json.converter.BpmnJsonConverter; import org.activiti.engine.RepositoryService; import org.activiti.engine.impl.cmd.AddCommentCmd; import org.activiti.engine.repository.DeploymentBuilder; import org.activiti.engine.repository.ProcessDefinition; import org.apache.commons.io.FilenameUtils; import org.apache.ibatis.javassist.compiler.MemberResolver; import org.codehaus.groovy.tools.shell.util.Logger; import org.springframework.stereotype.Controller; import org.springframework.ui.Model; import org.springframework.web.bind.annotation.\*; import org.springframework.web.multipart.MultipartFile;  import javax.annotation.Resource; import javax.servlet.http.HttpServletRequest; import javax.servlet.http.HttpServletResponse; import java.io.\*; import java.util.List; import java.util.zip.ZipInputStream;  */\*\*  \* Created by @author 王钟鑫  \*  \** ***@create*** *2018/1/20.  \*/* @Controller public class DeployManagementController {   private Logger log = Logger.*create*(DeployManagementController.class);   //流程部署实例  private RepositoryService repositoryService;   @Resource  public void setRepositoryService(RepositoryService repositoryService) {  this.repositoryService = repositoryService;  }   @RequestMapping(value = "/deployments",method = RequestMethod.*GET*)  public String index(Model model){   /\* repositoryService.createDeployment().addClasspathResource("diagram/HelloWorld.bpmn")  .addClasspathResource("diagram/HelloWorld.png").deploy();\*/   //获得流程定义对象  List<ProcessDefinition> dataList = repositoryService.createProcessDefinitionQuery().list();   model.addAttribute("dataList",dataList);   return "deploy/index";  }  /\*  \*  \* 通过上传的文件zip,bpmn,png文件来部署流程  \* @Param 前端form表单的file对象  \*  \* \*/   @RequestMapping(value = "/deployment/files",method = RequestMethod.*POST*)  public String deploy(@RequestParam("file")MultipartFile file){   if(file != null){   String originName = file.getOriginalFilename();  try {   InputStream inputStream = file.getInputStream();  //获取到扩展名  String type = FilenameUtils.*getExtension*(originName);   //创建部署对象  DeploymentBuilder deployment = repositoryService.createDeployment();   //如果是zip，bar类型需要使用Zip对象来进行部署  if(type.equals("zip") || type.equals("bar")){   ZipInputStream zip = new ZipInputStream(inputStream);  deployment.addZipInputStream(zip);   //其他文件直接部署  }else{   deployment.addInputStream(originName,inputStream);   }  //部署  deployment.deploy();   return "redirect:deployments";   } catch (IOException e) {   e.printStackTrace();   }   }  return "error";   }  /\*  \*  \* 获取到部署的xml或者是png图片  \* @Param deploymentId 部署Id  @Param resourceName 资源的名称  \*  \* \*/  @RequestMapping(value = "/deployment/resources",method = RequestMethod.*GET*)  public void getXmlResource(@RequestParam("id") String deploymentId,@RequestParam("name") String resourceName, HttpServletResponse response){   //获取到查询对象  InputStream resourceAsStream = repositoryService.getResourceAsStream(deploymentId, resourceName);   byte[] b = new byte[1024];  int len = -1;  try {  while ((len = resourceAsStream.read(b,0,1024)) != -1){   response.getOutputStream().write(b,0,len);  }   } catch (IOException e) {  e.printStackTrace();  }   }  /\*  \* 删除部署的流程  \* @Param deploymentId 部署Id  \*  \* \*/  @ResponseBody  @RequestMapping(value = "/deployment/{id}",method = RequestMethod.*DELETE*)  public Object delete(@PathVariable("id")String deploymentId){   try {  repositoryService.deleteDeployment(deploymentId,true);  return R.*success*();  }catch (Exception e){   e.printStackTrace();   }   return R.*failure*();    }   /\*  \*  \* 跳转到新建模型对象，这里需要创建一个modelId  \* \*/  @RequestMapping(value = "/deployment/model/management",method = RequestMethod.*POST*)  public void toCreate(@RequestParam("name") String name,@RequestParam("description") String description,@RequestParam("key") String key,HttpServletRequest request, HttpServletResponse response){    try {   ObjectMapper mapper = new ObjectMapper();//创建jackson对象  ObjectNode editorNode = mapper.createObjectNode();//创建节点  editorNode.put("id", "canvas"); //添加数据  editorNode.put("resourceId", "canvas");  ObjectNode stencilSetNode = mapper.createObjectNode();  stencilSetNode.put("namespace", "http://b3mn.org/stencilset/bpmn2.0#");  editorNode.put("stencilset", stencilSetNode);  org.activiti.engine.repository.Model modelData = repositoryService.newModel(); //使用repositoryService创建model   ObjectNode modelObjectNode = mapper.createObjectNode();  modelObjectNode.put(ModelDataJsonConstants.*MODEL\_NAME*, name);  modelObjectNode.put(ModelDataJsonConstants.*MODEL\_REVISION*, 1);  modelObjectNode.put(ModelDataJsonConstants.*MODEL\_DESCRIPTION*, description);  modelData.setMetaInfo(modelObjectNode.toString());  modelData.setName(name);  modelData.setKey(key);   //保存模型  repositoryService.saveModel(modelData);  repositoryService.addModelEditorSource(modelData.getId(), editorNode.toString().getBytes("utf-8"));  //跳转到modeler.html并传递刚刚生成的modelId  response.sendRedirect(request.getContextPath() + "/plugins/procerss-editor/modeler.html?modelId=" + modelData.getId());  }catch (Exception e){   System.*err*.println("出现了异常");  }   }  /\*  \* @Author:王钟鑫  \* @Description:获取到所有的模型  \* @Param:  \* @Date:2018/1/27\_10:42  \*  \* \*/  @RequestMapping(value="/deployment/model/management",method = RequestMethod.*GET*)  public String manage(Model model){   List<org.activiti.engine.repository.Model> dataList = repositoryService.createModelQuery()  .orderByCreateTime().desc().list();   model.addAttribute("dataList",dataList);   return "deploy/model";   }   /\*  \* @Author:王钟鑫  \* @Description:获取到模型的xml信息或者png的数据  \* @Param:modelId图片资源id,editorSourceValueId 获取xml，editorSourceExtraValueId 获取 png图片  \* @Date:2018/1/27\_14:30  \*  \* \*/  @RequestMapping(value = "/deployment/model/resources",method = RequestMethod.*GET*)  public void getModelXmlResource(@RequestParam("id") String modelId,@RequestParam(value = "vid" ,defaultValue = "",required = false) String editorSourceValueId,@RequestParam(value = "evid" ,defaultValue = "",required = false) String editorSourceExtraValueId, HttpServletResponse response){   //获取到对应的模型  org.activiti.engine.repository.Model model = repositoryService.getModel(modelId);  byte[] modelEditorSourceExtra = null;   //查询模型的原数据 xml  if(!editorSourceValueId.equals("")){    modelEditorSourceExtra = repositoryService.getModelEditorSource(model.getId());  JsonNode editNode = null;  try {  editNode = new ObjectMapper().readTree(modelEditorSourceExtra); //读取节点  BpmnJsonConverter jsonConverter = new BpmnJsonConverter();//  BpmnModel bpmnModel = jsonConverter.convertToBpmnModel(editNode);  modelEditorSourceExtra = new BpmnXMLConverter().convertToXML(bpmnModel);//转化为对应的xml  } catch (IOException e) {  e.printStackTrace();  }   }else{ //获取到png图片   modelEditorSourceExtra = repositoryService.getModelEditorSourceExtra(model.getId());   }   //输出到界面  InputStream put = new ByteArrayInputStream(modelEditorSourceExtra);   byte[] b = new byte[1024];  int len = -1;  try {  while ((len = put.read(b,0,1024)) != -1){   response.getOutputStream().write(b,0,len);  }   } catch (IOException e) {  e.printStackTrace();  }   }  /\*  \* @Author:王钟鑫  \* @Description:部署流程  \* @Param:modelId 模型编号  \* @Date:2018/1/27\_15:21  \*  \* \*/  @RequestMapping(value = "/deployment/model/deployment/{id}",method = RequestMethod.*PUT*)  public @ResponseBody Object deployByModel(@PathVariable("id") String modelId){   try {  org.activiti.engine.repository.Model model = repositoryService.getModel(modelId);   byte[] modelEditorSource = repositoryService.getModelEditorSource(model.getId());   //把json进行转化为xml  JsonNode editNode = null;  editNode = new ObjectMapper().readTree(modelEditorSource); //读取节点  BpmnJsonConverter jsonConverter = new BpmnJsonConverter();//  BpmnModel bpmnModel = jsonConverter.convertToBpmnModel(editNode);  modelEditorSource = new BpmnXMLConverter().convertToXML(bpmnModel);//转化为对应的xml    String processName = model.getName() + ".bpmn20.xml";  //部署    repositoryService.createDeployment().name(model.getName())  .addString(processName,new String(modelEditorSource,"utf-8"))  .deploy();  return R.*success*();   } catch (UnsupportedEncodingException e) {   e.printStackTrace();  }catch (Exception e){   e.printStackTrace();  }   return R.*failure*();   }  /\*  \* 删除model  \* @Param deploymentId 部署Id  \*  \* \*/  @ResponseBody  @RequestMapping(value = "/deployment/model/{id}",method = RequestMethod.*DELETE*)  public Object deleteModel(@PathVariable("id")String modelId){   try {  repositoryService.deleteModel(modelId);  return R.*success*();  }catch (Exception e){   e.printStackTrace();   }   return R.*failure*();    }   } |

Index.jsp

|  |
| --- |
| <%@ **page** contentType="text/html;charset=UTF-8" language="java" %> <%@**include** file="../base/common.jsp"%> <html> <head>  <title>流程部署</title>  <script src="**${**ctx**}**/static/js/jquery-3.2.1.min.js"></script>  <link rel="stylesheet" href="**${**ctx**}**/static/css/bootstrap.min.css">   </head> <body>  <div style="width:900px;margin:50px auto;">  <div class="jumbotron">  <h1>上传压缩包部署流程</h1>  <form action="**${**ctx**}**/deployment/files" method="post" enctype="multipart/form-data">  <div class="form-group">  <input type="file" name="file" id="file" class="form-control">  </div>  <div class="form-group">  <input type="submit" class="btn btn-primary btn-lg" value="上传">  </div>  </form>  </div>  <a class="btn btn-primary" href="**${**ctx**}**/deployment/model/management">模型管理</a>  <h2>流程信息列表：</h2>  <div class="table-responsive">  <table class="table">  <tr>  <th>流程定义ID</th>  <th>流程部署Id</th>  <th>流程定义名称</th>  <th>流程定义KEY</th>  <th>版本号</th>  <th>XML资源文件</th>  <th>图片资源文件</th>  <th>操作</th>    </tr>  <**c:forEach** items="**${**dataList**}**" var="item">   <tr>  <td>**${**item.id**}**</td>  <td>**${**item.deploymentId**}**</td>  <td>**${**item.name**}**</td>  <td>**${**item.key**}**</td>  <td>**${**item.version**}**</td>  <td><a href="**${**ctx**}**/deployment/resources?id=**${**item.deploymentId**}**&name=**${**item.resourceName**}**">**${**item.resourceName**}**</a></td>  <td><a href="**${**ctx**}**/deployment/resources?id=**${**item.deploymentId**}**&name=**${**item.diagramResourceName**}**">**${**item.diagramResourceName**}**</a></td>  <td><a href="javascript:void(0)" onclick="deletes(**${**item.deploymentId**}**)">删除</a></td>  </tr>   </**c:forEach**>   </table>  </div>  </div>  <script src="**${**ctx**}**/static/js/bootstrap.min.js"></script> <script>   **function** deletes(id){    **var** index = confirm('是否删除');   **if**(index){    $.ajax({   url:'**${**ctx**}**/deployment/'+id,  type:'DELETE',  success: **function**(data){    data = JSON.parse(data);  **if**(data.success){   document.location.reload();   }**else**{   alert('删除失败');   }   }  });    }  }  </script> </body> </html> |

Model.jsp

|  |
| --- |
| <%@ **page** contentType="text/html;charset=UTF-8" language="java" %> <%@**include** file="../base/common.jsp"%> <html> <head>  <title>流程部署</title>  <script src="**${**ctx**}**/static/js/jquery-3.2.1.min.js"></script>  <link rel="stylesheet" href="**${**ctx**}**/static/css/bootstrap.min.css">  <script src="**${**ctx**}**/plugins/layer/layer.js"></script>   </head> <body>  <div style="width:1200px;margin:50px auto;">  <button class="btn btn-primary" data-toggle="modal" data-target="#myModal">新建模型</button>  <h2>流程信息列表：</h2>  <div class="table-responsive">  <table class="table">  <tr>  <th>模型ID</th>  <th>模型版本</th>  <th>模型名称</th>  <th>模型KEY</th>  <th>创建时间</th>  <th>xml资源文件</th>  <th>图片资源文件</th>  <th>操作</th>  </tr>  <**c:forEach** items="**${**dataList**}**" var="item">   <tr>  <td>**${**item.id**}**</td>  <td>**${**item.version**}**</td>  <td>**${**item.name**}**</td>  <td>**${**item.key**}**</td>  <td><**fmt:formatDate** value="**${**item.createTime**}**" pattern="yyyy-MM-dd"></**fmt:formatDate**></td>  <td><a href="**${**ctx**}**/deployment/model/resources?id=**${**item.id**}**&vid=**${**item.editorSourceValueId**}**">xml资源文件</a></td>  <td><a href="**${**ctx**}**/deployment/model/resources?id=**${**item.id**}**&evid=**${**item.editorSourceExtraValueId**}**">png图片</a></td>  <td>  <a href="javascript:void(0)" onclick="deletes('**${**item.id**}**')">删除</a>   <a href="javascript:void(0)" onclick="deploy('**${**item.id**}**')">部署</a>   <a href="**${**ctx**}**/plugins/procerss-editor/modeler.html?modelId=**${**item.id**}**">编辑</a>  </td>  </tr>  </**c:forEach**>   </table>  </div>  </div>  <!-- Modal -->  <div class="modal fade" id="myModal" tabindex="-1" role="dialog" aria-labelledby="myModalLabel">  <div class="modal-dialog" role="document">  <form action="**${**ctx**}**/deployment/model/management" method="post" name="form">  <div class="modal-content">  <div class="modal-header">  <button type="button" class="close" data-dismiss="modal" aria-label="Close"><span aria-hidden="true">&times;</span></button>  <h4 class="modal-title" id="myModalLabel">新建模型</h4>  </div>  <div class="modal-body">   <div class="form-group">  <label for="name">模型名称</label>  <input type="text" class="form-control" name="name" id="name" placeholder="模型名称" value="helloTest">  </div>  <div class="form-group">  <label for="key">key</label>  <input type="text" class="form-control" id="key" name="key" placeholder="key" value="111111111">  </div>  <div class="form-group">  <label for="desc">描述</label>  <input type="text" class="form-control" name="description" id="desc" placeholder="描述" value="模型的描述">  </div>   </div>  <div class="modal-footer">  <button type="button" class="btn btn-default" data-dismiss="modal">关闭</button>  <button type="submit" class="btn btn-primary">提交</button>  </div>  </div>  </form>  </div>  </div>   <script src="**${**ctx**}**/static/js/bootstrap.min.js"></script> <script>   **function** deploy(id){    **var** index = confirm('是否部署该model');   **if**(index){   $.ajax({   url:'**${**ctx**}**/deployment/model/deployment/'+id,  type:'PUT',  success: **function**(data){    data = JSON.parse(data);  **if**(data.success){   document.location.reload();   }**else**{   alert('部署失败');   }   }  });    }   }   **function** deletes(id){    **var** index = confirm('是否删除');   **if**(index){    $.ajax({   url:'**${**ctx**}**/deployment/model/'+id,  type:'DELETE',  success: **function**(data){    data = JSON.parse(data);  **if**(data.success){    document.location.reload();    }**else**{   layer.msg("部署失败");   }   }  });    }  }  </script> </body> </html> |