[Activiti学习——模型管理模块](http://blog.csdn.net/flygoa/article/details/51909659)

需求

1. 拆解咖啡兔项目，分离出模型管理模块
2. 实现**【查询】**功能
3. 实现**【新增】**功能
4. 实现**【删除】**功能
5. 实现**【编辑】**功能
6. 实现**【部署】**功能
7. 实现**【导出】**功能

前提

1. 项目已经集成ActivitiModeler
2. 可参考[Activiti学习——整合ActivitiModeler到项目中](http://blog.csdn.net/flygoa/article/details/51890434" \t "_blank)

实现

查询

1. 使用activiti提供的接口，创建Model查询
2. 指定排序字段和排序方式
3. 返回集合list
4. 页面展示

/\*\*

\* 查询 客户端分页

\* @return

\*/

@RequestMapping(value="/selectAll")

@ResponseBody

public String selectAll(){

List<Model> resultList = repositoryService.createModelQuery().orderByCreateTime().desc().list();

JSONObject resultJson = new JSONObject();

resultJson.put("data", resultList);

return resultJson.toString();

}

新增

1. 创建模型对象
2. 设置对象值
3. 存储模型对象（表act\_re\_model）
4. 存储模型对象基础数据（表act\_ge\_bytearray）
5. 跳转到ActivitiModeler，编辑流程图，存储流程图片和流程定义等（表act\_ge\_bytearray）

/\*\*

\* 新增

\* @return

\*/

@RequestMapping(value = "/create",method = RequestMethod.POST)

public void getEditor(

@RequestParam("description") String description,

@RequestParam("name") String name,

@RequestParam("key") String key,

HttpServletRequest request, HttpServletResponse response){

try {

ObjectMapper objectMapper = new ObjectMapper();

ObjectNode editorNode = objectMapper.createObjectNode();

editorNode.put("id", "canvas");

editorNode.put("resourceId", "canvas");

ObjectNode stencilSetNode = objectMapper.createObjectNode();

stencilSetNode.put("namespace", "http://b3mn.org/stencilset/bpmn2.0#");

editorNode.put("stencilset", stencilSetNode);

Model modelData = repositoryService.newModel();

ObjectNode modelObjectNode = objectMapper.createObjectNode();

modelObjectNode.put(ModelDataJsonConstants.MODEL\_NAME, name);

modelObjectNode.put(ModelDataJsonConstants.MODEL\_REVISION, 1);

description = StringUtils.defaultString(description);

modelObjectNode.put(ModelDataJsonConstants.MODEL\_DESCRIPTION, description);

modelData.setMetaInfo(modelObjectNode.toString());

modelData.setName(name);

modelData.setKey(StringUtils.defaultString(key));

repositoryService.saveModel(modelData);

repositoryService.addModelEditorSource(modelData.getId(), editorNode.toString().getBytes("utf-8"));

System.out.println("跳转页面");

response.sendRedirect(request.getContextPath() + "/service/editor?id=" + modelData.getId());

} catch (Exception e) {

System.out.println("创建模型失败");

}

}

删除

1. 根据modelId删除

/\*\*

\* 批量删除

\* @param ids

\* @param request

\* @return

\*/

@RequestMapping(value = "deleteByIds")

@ResponseBody

public String deleteByIds(String[] ids,HttpServletRequest request) {

JSONObject result = new JSONObject();

for(String id : ids){

repositoryService.deleteModel(id);

}

result.put("msg", "删除成功");

result.put("type", "success");

return result.toString();

}

编辑

1. 编辑和新增的跳转接口是一样的
2. 跳转到ActivitiModeler，带上modelId
3. 这里使用a标签直接跳转
4. ActivitiDemo5是项目名
5. row.id是modelId
6. target=”\_blank切换到新的标签页,同样适用于新增成功时，跳转新页面，写在form表单上

<a class="btn btn-success" href="/ActivitiDemo5/service/editor?id='+row.id+'" target="\_blank">编辑</a>

导出

1. 获取节点信息
2. 转换为xml数据
3. 写入流中输出

/\*\*

\* 导出model的xml文件

\*/

@RequestMapping(value = "export/{modelId}")

public void export(@PathVariable("modelId") String modelId, HttpServletResponse response) {

response.setCharacterEncoding("UTF-8");

response.setContentType("application/json; charset=utf-8");

try {

Model modelData = repositoryService.getModel(modelId);

BpmnJsonConverter jsonConverter = new BpmnJsonConverter();

//获取节点信息

byte[] arg0 = repositoryService.getModelEditorSource(modelData.getId());

JsonNode editorNode = new ObjectMapper().readTree(arg0);

//将节点信息转换为xml

BpmnModel bpmnModel = jsonConverter.convertToBpmnModel(editorNode);

BpmnXMLConverter xmlConverter = new BpmnXMLConverter();

byte[] bpmnBytes = xmlConverter.convertToXML(bpmnModel);

ByteArrayInputStream in = new ByteArrayInputStream(bpmnBytes);

IOUtils.copy(in, response.getOutputStream());

// String filename = bpmnModel.getMainProcess().getId() + ".bpmn20.xml";

String filename = modelData.getName() + ".bpmn20.xml";

response.setHeader("Content-Disposition", "attachment; filename=" + java.net.URLEncoder.encode(filename, "UTF-8"));

response.flushBuffer();

} catch (Exception e){

PrintWriter out = null;

try {

out = response.getWriter();

} catch (IOException e1) {

e1.printStackTrace();

}

out.write("未找到对应数据");

e.printStackTrace();

}

}

部署

1. 根据modelId获取模型信息
2. 转换为xml对象
3. 部署对象以模型对象的名称命名
4. 部署转换出来的xml对象
5. 部署方式有很多种，可以再扩展研究

/\*\*

\* 部署

\*/

@RequestMapping(value = "deploy",method=RequestMethod.POST)

@ResponseBody

public String deploy(@RequestParam("modelId") String modelId, HttpServletRequest request) {

JSONObject result = new JSONObject();

try {

Model modelData = repositoryService.getModel(modelId);

ObjectNode modelNode = (ObjectNode) new ObjectMapper().readTree(repositoryService.getModelEditorSource(modelData.getId()));

byte[] bpmnBytes = null;

BpmnModel model = new BpmnJsonConverter().convertToBpmnModel(modelNode);

bpmnBytes = new BpmnXMLConverter().convertToXML(model);

String processName = modelData.getName() + ".bpmn20.xml";

Deployment deployment = repositoryService.createDeployment().name(modelData.getName()).addString(processName, new String(bpmnBytes,"utf-8")).deploy();

result.put("msg", "部署成功");

result.put("type", "success");

} catch (Exception e) {

result.put("msg", "部署失败");

result.put("type", "error");

e.printStackTrace();

}

return result.toString();

}

总结

1. 模型这块，主要使用到了RepositoryService,引用手册的话

RepositoryService可能是使用Activiti引擎时最先接触的服务。

它提供了管理和控制发布包和流程定义的操作。

这里不涉及太多细节，流程定义是BPMN 2.0流程的java实现。

它包含了一个流程每个环节的结构和行为。

发布包是Activiti引擎的打包单位。

一个发布包可以包含多个BPMN 2.0 xml文件和其他资源。

开发者可以自由选择把任意资源包含到发布包中。

既可以把一个单独的BPMN 2.0 xml文件放到发布包里，也可以把整个流程和相关资源都放在一起。

（比如，'hr-processes'实例可以包含hr流程相关的任何资源）。

可以通过RepositoryService来部署这种发布包。

发布一个发布包，意味着把它上传到引擎中，所有流程都会在保存进数据库之前分析解析好。

从这点来说，系统知道这个发布包的存在，发布包中包含的流程就已经可以启动了。

除此之外，服务可以

查询引擎中的发布包和流程定义。

暂停或激活发布包，对应全部和特定流程定义。 暂停意味着它们不能再执行任何操作了，激活是对应的反向操作。

获得多种资源，像是包含在发布包里的文件， 或引擎自动生成的流程图。

获得流程定义的pojo版本， 可以用来通过java解析流程，而不必通过xml。

1. Activiti的查询提供了很多接口   
   查询全部

List<Model> resultList = repositoryService.createModelQuery().orderByCreateTime().desc().list();

分页查询

List<Model> resultList = repositoryService.createModelQuery().listPage(0, 10);

使用原生sql查询

List<Model> resultList = repositoryService.createNativeModelQuery().sql("sql").list();