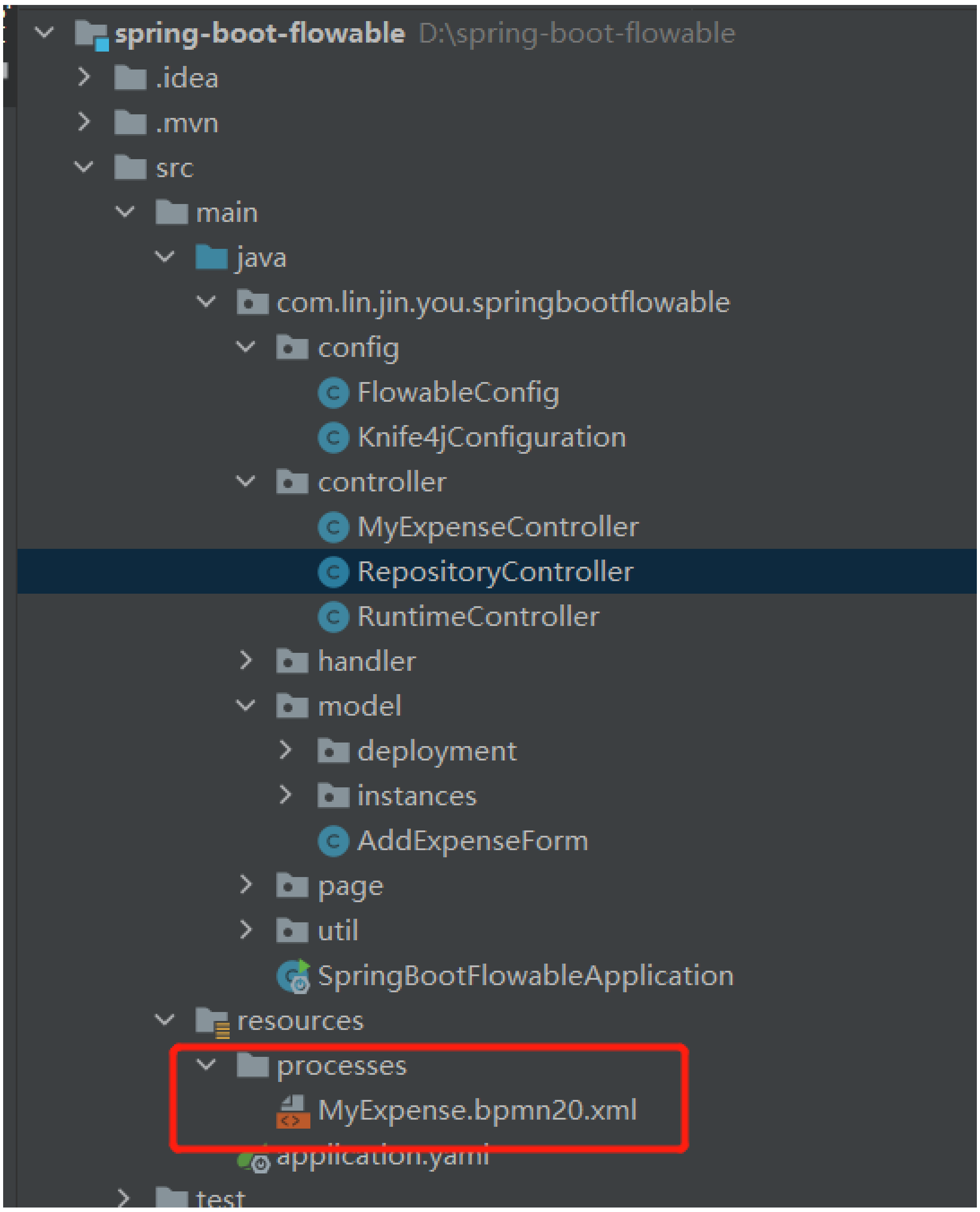
Flowable常见API和表结构解析

1、部署API

1.1 自动部署

在项目工程中新建processes目录,将bpmn.xml文件放到该目录底下。同一份bpmn.xml文件只会自动部署一次。



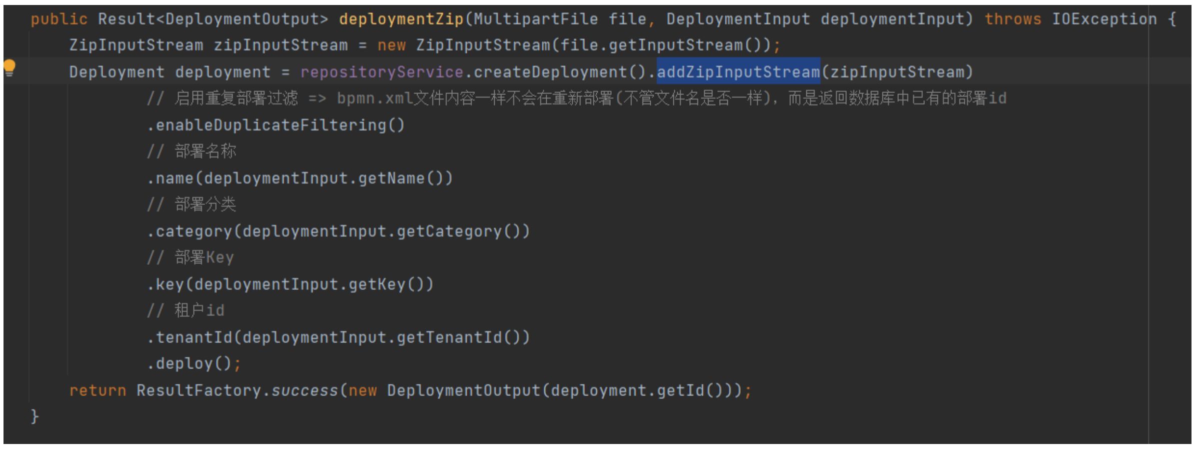
```
spring-boot-flowable D:\spring-boot-flowable
  .idea
  .mvn
  src
    main
      java
        com.lin.jin.you.springbootflowable
          config
            FlowableConfig
            Knife4jConfiguration
          controller
            MyExpenseController
            RepositoryController
            RuntimeController
          handler
          model
            deployment
            instances
            AddExpenseForm
          page
          util
          SpringBootFlowableApplication
      resources
        processes
          MyExpense.bpmn20.xml
        application.yaml
    test
```

1.2 api部署

1.2.1 部署xml文件

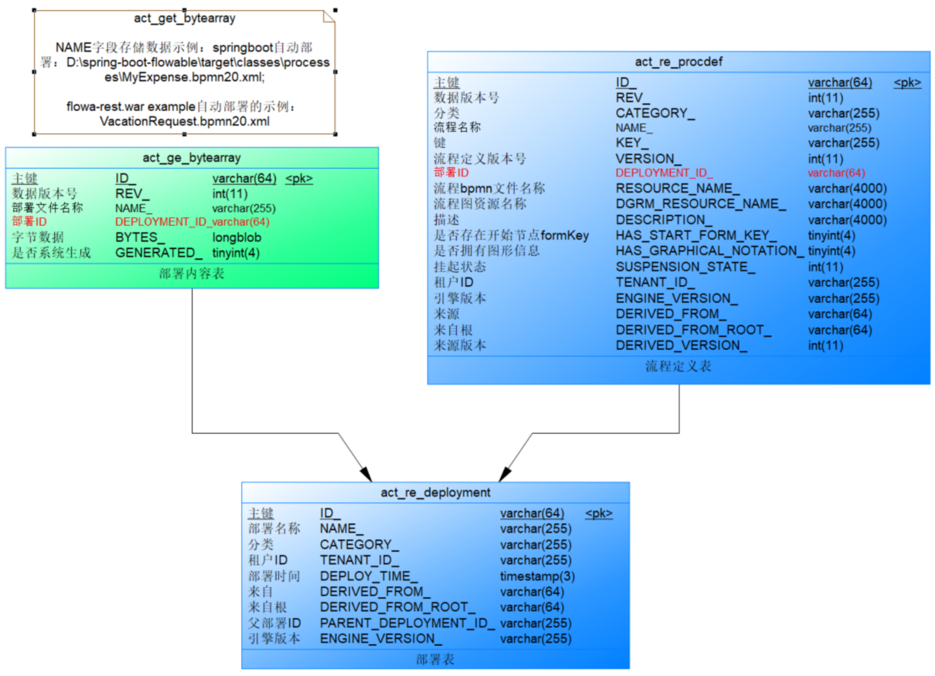
```java
public Result<DeploymentOutput> deployment(MultipartFile file, DeploymentInput deploymentInput) throws IOException {
    Deployment deployment = repositoryService.createDeployment().addInputStream(file.getOriginalFilename(), file.getInputStream())
            // 启用重复部署过滤 => bpmn.xml文件内容一样不会在重新部署，而是返回数据库中已有的部署id
            .enableDuplicateFiltering()
            // 部署名称
            .name(deploymentInput.getName())
            // 部署分类
            .category(deploymentInput.getCategory())
            // 部署Key
            .key(deploymentInput.getKey())
            // 租户id
            .tenantId(deploymentInput.getTenantId())
            .deploy();
    return ResultFactory.success(new DeploymentOutput(deployment.getId()));
```

1.2.2 部署zip包



```java
public Result<DeploymentOutput> deploymentZip(MultipartFile file, DeploymentInput deploymentInput) throws IOException {
    ZipInputStream zipInputStream = new ZipInputStream(file.getInputStream());
    Deployment deployment = repositoryService.createDeployment().addZipInputStream(zipInputStream)
            // 启用重复部署过滤 => bpmn.xml文件内容一样不会在重新部署(不管文件名是否一样)，而是返回数据库中已有的部署id
            .enableDuplicateFiltering()
            // 部署名称
            .name(deploymentInput.getName())
            // 部署分类
            .category(deploymentInput.getCategory())
            // 部署Key
            .key(deploymentInput.getKey())
            // 租户id
            .tenantId(deploymentInput.getTenantId())
            .deploy();
    return ResultFactory.success(new DeploymentOutput(deployment.getId()));
}
```

1.3 部署相关表结构



act_get_bytearray

NAME字段存储数据示例：springboot自动部署：D:\spring-boot-flowable\target\classes\processes\MyExpense.bpmn20.xml;

flowa-rest.war example自动部署的示例：VacationRequest.bpmn20.xml

act_ge_bytearray

主键	ID_	varchar(64)	<pk>
数据版本号	REV_	int(11)	
部署文件名称	NAME_	varchar(255)	
部署ID	DEPLOYMENT_ID_	varchar(64)	
字节数据	BYTES_	longblob	
是否系统生成	GENERATED_	tinyint(4)	

部署内容表

act_re_procdef

主键	ID_	varchar(64)	<pk>
数据版本号	REV_	int(11)	
分类	CATEGORY_	varchar(255)	
流程名称	NAME_	varchar(255)	
键	KEY_	varchar(255)	
流程定义版本号	VERSION_	int(11)	
部署ID	DEPLOYMENT_ID_	varchar(64)	
流程bpmn文件名称	RESOURCE_NAME_	varchar(4000)	
流程图资源名称	DGRM_RESOURCE_NAME_	varchar(4000)	
描述	DESCRIPTION_	varchar(4000)	
是否存在开始节点formKey	HAS_START_FORM_KEY_	tinyint(4)	
是否拥有图形信息	HAS_GRAPHICAL_NOTATION_	tinyint(4)	
挂起状态	SUSPENSION_STATE_	int(11)	
租户ID	TENANT_ID_	varchar(255)	
引擎版本	ENGINE_VERSION_	varchar(255)	
来源	DERIVED_FROM_	varchar(64)	
来自根	DERIVED_FROM_ROOT_	varchar(64)	
来源版本	DERIVED_VERSION_	int(11)	

流程定义表

act_re_deployment

主键	ID_	varchar(64)	<pk>
部署名称	NAME_	varchar(255)	
分类	CATEGORY_	varchar(255)	
租户ID	TENANT_ID_	varchar(255)	
部署时间	DEPLOY_TIME_	timestamp(3)	
来自	DERIVED_FROM_	varchar(64)	
来自根	DERIVED_FROM_ROOT_	varchar(64)	
父部署ID	PARENT_DEPLOYMENT_ID_	varchar(255)	
引擎版本	ENGINE_VERSION_	varchar(255)	

部署表

1.3.1 act_re_deployment

存储部署的基本信息，部署bpmn.xml文件后生成

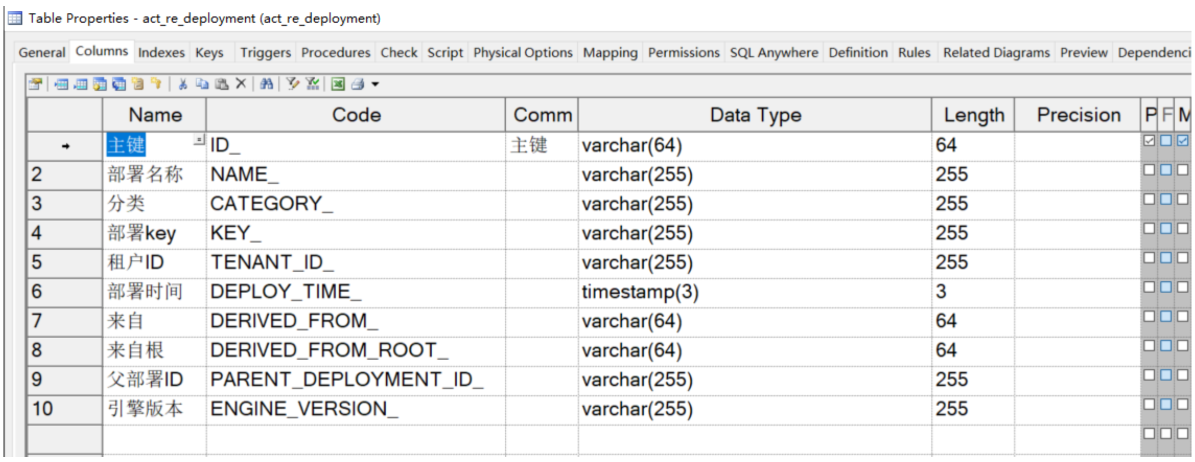
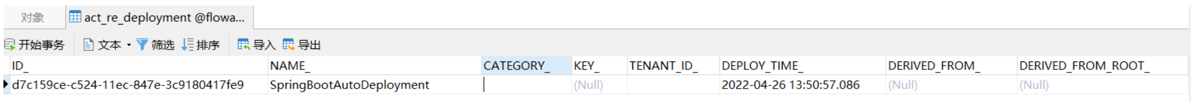


Table Properties - act_re_deployment (act_re_deployment)

	Name	Code	Comm	Data Type	Length	Precision	P	F	M
→	主键	ID_	主键	varchar(64)	64		✓	☐	✓
2	部署名称	NAME_		varchar(255)	255		☐	☐	☐
3	分类	CATEGORY_		varchar(255)	255		☐	☐	☐
4	部署key	KEY_		varchar(255)	255		☐	☐	☐
5	租户ID	TENANT_ID_		varchar(255)	255		☐	☐	☐
6	部署时间	DEPLOY_TIME_		timestamp(3)	3		☐	☐	☐
7	来自	DERIVED_FROM_		varchar(64)	64		☐	☐	☐
8	来自根	DERIVED_FROM_ROOT_		varchar(64)	64		☐	☐	☐
9	父部署ID	PARENT_DEPLOYMENT_ID_		varchar(255)	255		☐	☐	☐
10	引擎版本	ENGINE_VERSION_		varchar(255)	255		☐	☐	☐
							☐	☐	☐

- 自动部署生成的表数据



ID_	NAME_	CATEGORY_	KEY_	TENANT_ID_	DEPLOY_TIME_	DERIVED_FROM_	DERIVED_FROM_ROOT_
d7c159ce-c524-11ec-847e-3c9180417fe9	SpringBootAutoDeployment		(Null)		2022-04-26 13:50:57.086	(Null)	(Null)

1.3.2 act_re_procdef

存储流程定义信息，部署bpmn.xml文件后生成

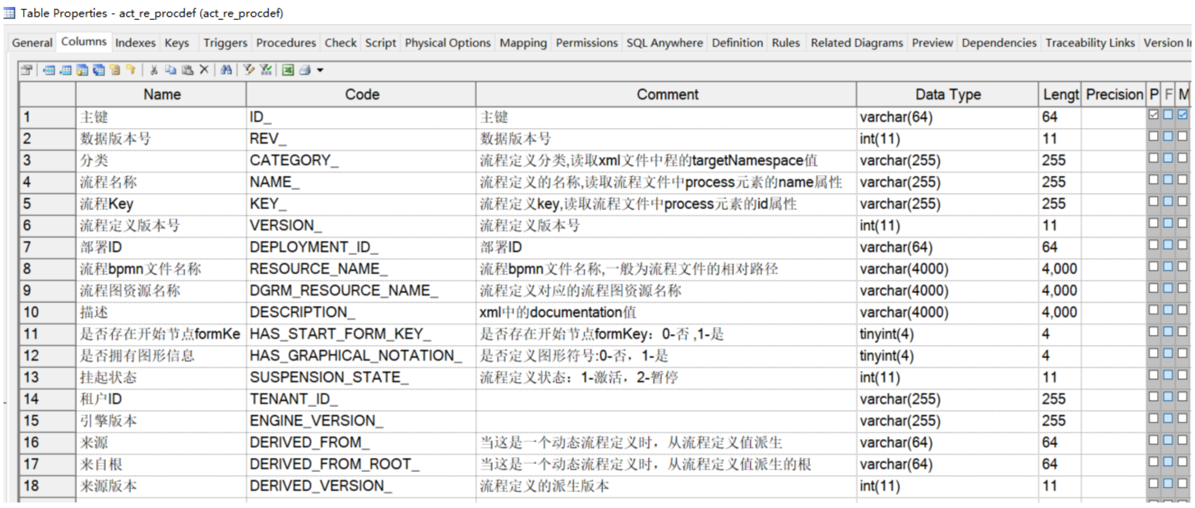


Table Properties - act_re_procdef (act_re_procdef)

General | Columns | Indexes | Keys | Triggers | Procedures | Check | Script | Physical Options | Mapping | Permissions | SQL Anywhere | Definition | Rules | Related Diagrams | Preview | Dependencies | Traceability Links | Version I

	Name	Code	Comment	Data Type	Lengt	Precision	P	F	M
1	主键	ID_	主键	varchar(64)	64		✓		✓
2	数据版本号	REV_	数据版本号	int(11)	11				
3	分类	CATEGORY_	流程定义分类,读取xml文件中程的targetNamespace值	varchar(255)	255				
4	流程名称	NAME_	流程定义的名称,读取流程文件中process元素的name属性	varchar(255)	255				
5	流程Key	KEY_	流程定义key,读取流程文件中process元素的id属性	varchar(255)	255				
6	流程定义版本号	VERSION_	流程定义版本号	int(11)	11				
7	部署ID	DEPLOYMENT_ID_	部署ID	varchar(64)	64				
8	流程bpmn文件名称	RESOURCE_NAME_	流程bpmn文件名称,一般为流程文件的相对路径	varchar(4000)	4,000				
9	流程图资源名称	DGRM_RESOURCE_NAME_	流程定义对应的流程图资源名称	varchar(4000)	4,000				
10	描述	DESCRIPTION_	xml中的documentation值	varchar(4000)	4,000				
11	是否存在开始节点formKe	HAS_START_FORM_KEY_	是否存在开始节点formKey：0-否 ,1-是	tinyint(4)	4				
12	是否拥有图形信息	HAS_GRAPHICAL_NOTATION_	是否定义图形符号:0-否，1-是	tinyint(4)	4				
13	挂起状态	SUSPENSION_STATE_	流程定义状态：1-激活，2-暂停	int(11)	11				
14	租户ID	TENANT_ID_		varchar(255)	255				
15	引擎版本	ENGINE_VERSION_		varchar(255)	255				
16	来源	DERIVED_FROM_	当这是一个动态流程定义时，从流程定义值派生	varchar(64)	64				
17	来自根	DERIVED_FROM_ROOT_	当这是一个动态流程定义时，从流程定义值派生的根	varchar(64)	64				
18	来源版本	DERIVED_VERSION_	流程定义的派生版本	int(11)	11				

1.3.3 act_ge_bytearray

部署文件内容信息，部署bpmn.xml文件后生成

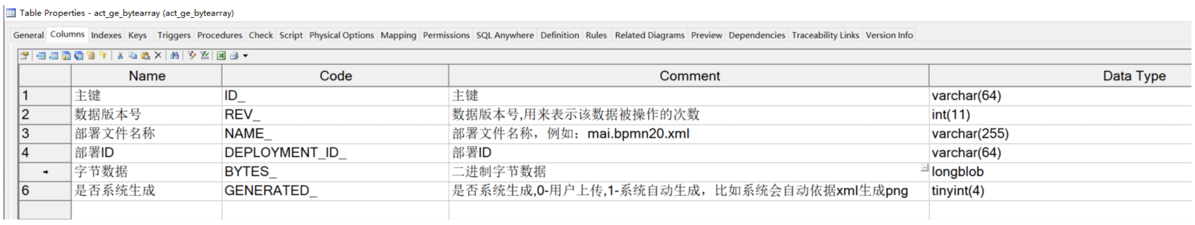


Table Properties - act_ge_bytearray (act_ge_bytearray)

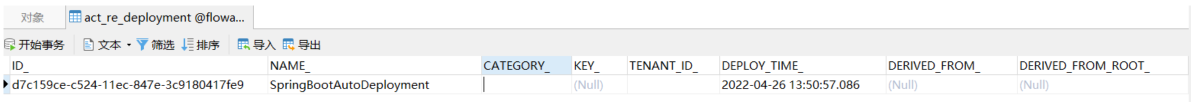
General | Columns | Indexes | Keys | Triggers | Procedures | Check | Script | Physical Options | Mapping | Permissions | SQL Anywhere | Definition | Rules | Related Diagrams | Preview | Dependencies | Traceability Links | Version Info

	Name	Code	Comment	Data Type
1	主键	ID_	主键	varchar(64)
2	数据版本号	REV_	数据版本号,用来表示该数据被操作的次数	int(11)
3	部署文件名称	NAME_	部署文件名称，例如：mai.bpmn20.xml	varchar(255)
4	部署ID	DEPLOYMENT_ID_	部署ID	varchar(64)
→	字节数据	BYTES_	二进制字节数据	longblob
6	是否系统生成	GENERATED_	是否系统生成,0-用户上传,1-系统自动生成，比如系统会自动依据xml生成png	tinyint(4)

1.4 部署多次

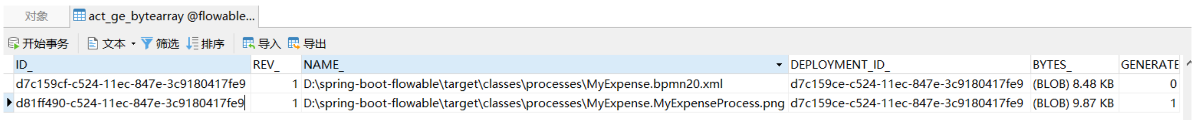
同一份xml文件，使用自动部署一次，再使用api部署一次,流程定义表中的会出现version为1和2的两条记录,并且suspension_state状态为1-激活状态。

- 自动部署 -只会部署一次


对象 | act_re_deployment @flowa...

开始事务 | 文本 | 筛选 | 排序 | 导入 | 导出

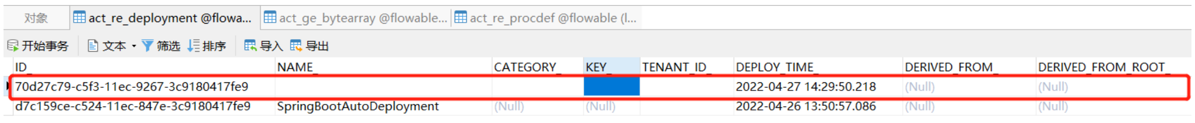
ID_	NAME_	CATEGORY_	KEY_	TENANT_ID_	DEPLOY_TIME_	DERIVED_FROM_	DERIVED_FROM_ROOT_
d7c159ce-c524-11ec-847e-3c9180417fe9	SpringBootAutoDeployment		(Null)		2022-04-26 13:50:57.086	(Null)	(Null)


对象 | act_ge_bytearray @flowable...

开始事务 | 文本 | 筛选 | 排序 | 导入 | 导出

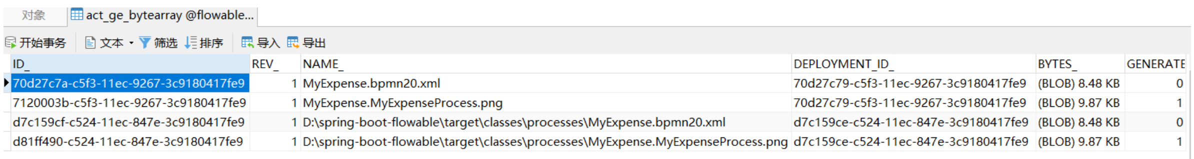
ID_	REV_	NAME_	DEPLOYMENT_ID_	BYTES_	GENERATE
d7c159cf-c524-11ec-847e-3c9180417fe9	1	D:\spring-boot-flowable\target\classes\processes\MyExpense.bpmn20.xml	d7c159ce-c524-11ec-847e-3c9180417fe9	(BLOB) 8.48 KB	0
d81ff490-c524-11ec-847e-3c9180417fe9	1	D:\spring-boot-flowable\target\classes\processes\MyExpense.MyExpenseProcess.png	d7c159ce-c524-11ec-847e-3c9180417fe9	(BLOB) 9.87 KB	1

- api部署


对象 | act_re_deployment @flowa... | act_ge_bytearray @flowable... | act_re_procdef @flowable (l...

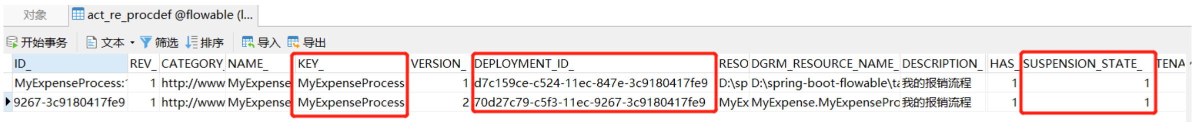
开始事务 | 文本 | 筛选 | 排序 | 导入 | 导出

ID_	NAME_	CATEGORY_	KEY_	TENANT_ID_	DEPLOY_TIME_	DERIVED_FROM_	DERIVED_FROM_ROOT_
70d27c79-c5f3-11ec-9267-3c9180417fe9					2022-04-27 14:29:50.218	(Null)	(Null)
d7c159ce-c524-11ec-847e-3c9180417fe9	SpringBootAutoDeployment	(Null)	(Null)		2022-04-26 13:50:57.086	(Null)	(Null)


对象 | act_ge_bytearray @flowable...

开始事务 | 文本 | 筛选 | 排序 | 导入 | 导出

ID_	REV_	NAME_	DEPLOYMENT_ID_	BYTES_	GENERATE
70d27c7a-c5f3-11ec-9267-3c9180417fe9	1	MyExpense.bpmn20.xml	70d27c79-c5f3-11ec-9267-3c9180417fe9	(BLOB) 8.48 KB	0
7120003b-c5f3-11ec-9267-3c9180417fe9	1	MyExpense.MyExpenseProcess.png	70d27c79-c5f3-11ec-9267-3c9180417fe9	(BLOB) 9.87 KB	1
d7c159cf-c524-11ec-847e-3c9180417fe9	1	D:\spring-boot-flowable\target\classes\processes\MyExpense.bpmn20.xml	d7c159ce-c524-11ec-847e-3c9180417fe9	(BLOB) 8.48 KB	0
d81ff490-c524-11ec-847e-3c9180417fe9	1	D:\spring-boot-flowable\target\classes\processes\MyExpense.MyExpenseProcess.png	d7c159ce-c524-11ec-847e-3c9180417fe9	(BLOB) 9.87 KB	1

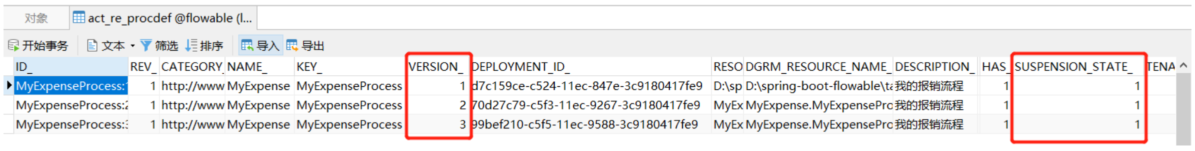

对象 | act_re_procdef @flowable (l...

开始事务 | 文本 | 筛选 | 排序 | 导入 | 导出

ID_	REV_	CATEGORY	NAME_	KEY_	VERSION_	DEPLOYMENT_ID_	RESO	DGRM_RESOURCE_NAME_	DESCRIPTION_	HAS_	SUSPENSION_STATE_	TENA
MyExpenseProcess:'	1	http://www	MyExpense	MyExpenseProcess	1	d7c159ce-c524-11ec-847e-3c9180417fe9	D:\sp	D:\spring-boot-flowable\t	我的报销流程	1	1	
9267-3c9180417fe9	1	http://www	MyExpense	MyExpenseProcess	2	70d27c79-c5f3-11ec-9267-3c9180417fe9	MyEx	MyExpense.MyExpensePrc	我的报销流程	1	1	

- api部署-不启用重复部署过滤-多次部署-会生成多条记录

```
        Deployment deployment =
repositoryService.createDeployment().addInputStream(file.getOriginalFilename
(), file.getInputStream())
                // 启用重复部署过滤 => bpmn.xml文件内容一样不会在重新部署，而是返回数
据库中已有的部署id
//                  .enableDuplicateFiltering()
                // 部署名称
                .name(deploymentInput.getName())
                // 部署分类
                .category(deploymentInput.getCategory())
                // 部署Key
                .key(deploymentInput.getKey())
                // 租户id
                .tenantId(deploymentInput.getTenantId())
                .deploy();
```



对象 act_re_procdef @flowable (l...

开始事务 文本 筛选 排序 导入 导出

ID_	REV_	CATEGORY	NAME_	KEY_	VERSION_	DEPLOYMENT_ID_	RESO	DGRM_RESOURCE_NAME_	DESCRIPTION_	HAS_	SUSPENSION_STATE_	TENA
MyExpenseProcess:1	1	http://www	MyExpense	MyExpenseProcess	1	d7c159ce-c524-11ec-847e-3c9180417fe9	D:\sp	D:\spring-boot-flowable\ta	我的报销流程	1	1	
MyExpenseProcess:2	1	http://www	MyExpense	MyExpenseProcess	2	70d27c79-c5f3-11ec-9267-3c9180417fe9	MyEx	MyExpense.MyExpensePro	我的报销流程	1	1	
MyExpenseProcess:3	1	http://www	MyExpense	MyExpenseProcess	3	99bef210-c5f5-11ec-9588-3c9180417fe9	MyEx	MyExpense.MyExpensePro	我的报销流程	1	1	

1.5删除部署

删除部署，会同时删除act_re_deployment 、act_re_procdef 、act_ge_bytearray 三张表的信息。

```
public Result<String> deleteDeployment(@PathVariable String deploymentId) {
    //  正在运行中的流程是否会受影响? => 如果存在正在运行的流程实例、历史流程实例、job会抛出
RuntimeException
    try {
        repositoryService.deleteDeployment(deploymentId);
    } catch (FlowableObjectNotFoundException e1) {
        return ResultFactory.failed("deploymentId为: " + deploymentId + "的部署信息
不存在", null);
    } catch (RuntimeException e2) {
        return ResultFactory.failed("deploymentId为: " + deploymentId + "的部署，存
在正在运行的流程实例、历史流程实例、工作信息之一,不允许删除", null);
    }
    return ResultFactory.success(deploymentId);
}
```

1.6 获取部署资源

```
public Result<List<String>> getDeploymentResourceNames(@PathVariable String
deploymentId) {
    List<String> resourceNames =
repositoryService.getDeploymentResourceNames(deploymentId);
    return ResultFactory.success(resourceNames);
}
```

```
public void getDeploymentResourceNames(@PathVariable String deploymentId,
@NotNull String resourceName, HttpServletResponse response) throws IOException {
    InputStream inputStream =
repositoryService.getResourceAsStream(deploymentId, resourceName);
    // inline:默认值,表示它可以显示在网页内; attachment表示可下载 ; fileName是下载文件名
字
    response.setHeader("Content-Disposition", "attachment;fileName=" +
URLEncoder.encode(resourceName, "UTF-8"));
    DownloadUtil.fastCopyStream(inputStream, response.getOutputStream());
}
```

2、流程定义API

2.1 流程定义查询

```
public Result<PageResult<ProcessDefinitionOutput>>
getProcessDefinitions(PageParam pageParam, ProcessDefinitionConditionInput
input) {
    ProcessDefinitionQuery processDefinitionQuery =
repositoryService.createProcessDefinitionQuery();
    listByCondition(input, processDefinitionQuery);
    List<ProcessDefinition> processDefinitions =
processDefinitionQuery.listPage((pageParam.getPageNum() - 1) *
pageParam.getPageSize(), pageParam.getPageNum() * pageParam.getPageSize());
    List<ProcessDefinitionOutput> list =
processDefinitions.stream().map(this::buildProcessDefinitionOutput).collect(Coll
ectors.toList());
    PageResult<ProcessDefinitionOutput> pageResult = new
PageResult(pageParam.getPageSize(), pageParam.getPageNum(),
processDefinitionQuery.count(), list);
    return ResultFactory.success(pageResult);
}
```

```
public Result<ProcessDefinitionOutput> getProcessDefinition(@PathVariable String
processDefinitionId) {
    ProcessDefinition processDefinition =
repositoryService.getProcessDefinition(processDefinitionId);
    return
ResultFactory.success(buildProcessDefinitionOutput(processDefinition));
}
```

2.2 暂停/激活流程定义

```java
public Result<Boolean> suspendProcessDefinition(@RequestBody
SuspendProcessDefinitionInput input) {
    try {

 repositoryService.suspendProcessDefinitionById(input.getProcessDefinitionId(),
input.isSuspendProcessInstances(), input.getSuspensionDate());
    } catch (FlowableObjectNotFoundException e1) {
        return ResultFactory.failed("流程id为： " + input.getProcessDefinitionId()
+ "的流程定义不存在", null);
    } catch (FlowableException e2) {
        return ResultFactory.failed("流程id为： " + input.getProcessDefinitionId()
+ "的流程已经是挂起状态", null);
    }
    return ResultFactory.success(Boolean.TRUE);
}
```

```java
public Result<Boolean> suspendProcessDefinitionByKey(@RequestBody
SuspendProcessDefinitionKeyInput input) {
    try {

 repositoryService.suspendProcessDefinitionByKey(input.getProcessDefinitionKey()
, input.isSuspendProcessInstances(),
input.getSuspensionDate(),input.getTenantId());
    } catch (FlowableObjectNotFoundException e1) {
        return ResultFactory.failed("流程Key为： " +
input.getProcessDefinitionKey() + "的流程定义不存在", null);
    } catch (FlowableException e2) {
        return ResultFactory.failed("流程Key为： " +
input.getProcessDefinitionKey() + "的流程已经是挂起状态", null);
    }
    return ResultFactory.success(Boolean.TRUE);
}
```

```java
public Result<Boolean> activateProcessDefinition(@RequestBody
ActivateProcessDefinitionInput input) {
    try {

 repositoryService.activateProcessDefinitionById(input.getProcessDefinitionId(),
input.isActivateProcessInstances(), input.getActivateDate());
    } catch (FlowableObjectNotFoundException e1) {
        return ResultFactory.failed("流程id为： " + input.getProcessDefinitionId()
+ "的流程定义不存在", null);
    } catch (FlowableException e2) {
        return ResultFactory.failed("流程id为： " + input.getProcessDefinitionId()
+ "的流程已经是挂起状态", null);
    }
    return ResultFactory.success(Boolean.TRUE);
}
```

2.3 查询流程定义状态

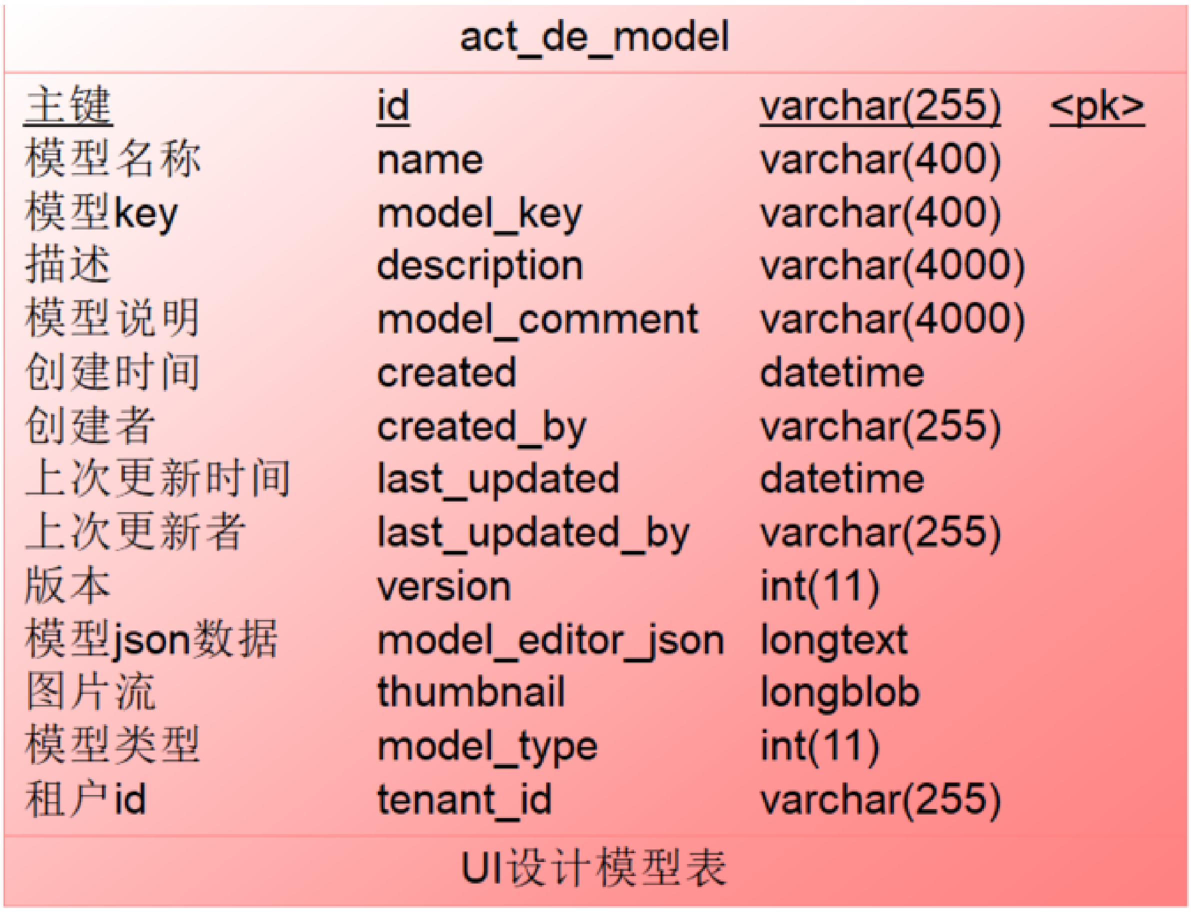
```
public Result<Boolean> isProcessDefinitionSuspended(@PathVariable String
processDefinitionId) {
    boolean suspended =
repositoryService.isProcessDefinitionSuspended(processDefinitionId);
    return ResultFactory.success(suspended);
}
```

3、流程模型API

3.1 Flowable UI Modeler

使用 Flowable UI Modeler设计器设计并保存流程模型，数据保存在act_de_model表中。

- act_de_model表结构



act_de_model

主键	id	varchar(255)	<pk>
模型名称	name	varchar(400)	
模型key	model_key	varchar(400)	
描述	description	varchar(4000)	
模型说明	model_comment	varchar(4000)	
创建时间	created	datetime	
创建者	created_by	varchar(255)	
上次更新时间	last_updated	datetime	
上次更新者	last_updated_by	varchar(255)	
版本	version	int(11)	
模型json数据	model_editor_json	longtext	
图片流	thumbnail	longblob	
模型类型	model_type	int(11)	
租户id	tenant_id	varchar(255)	

UI设计模型表

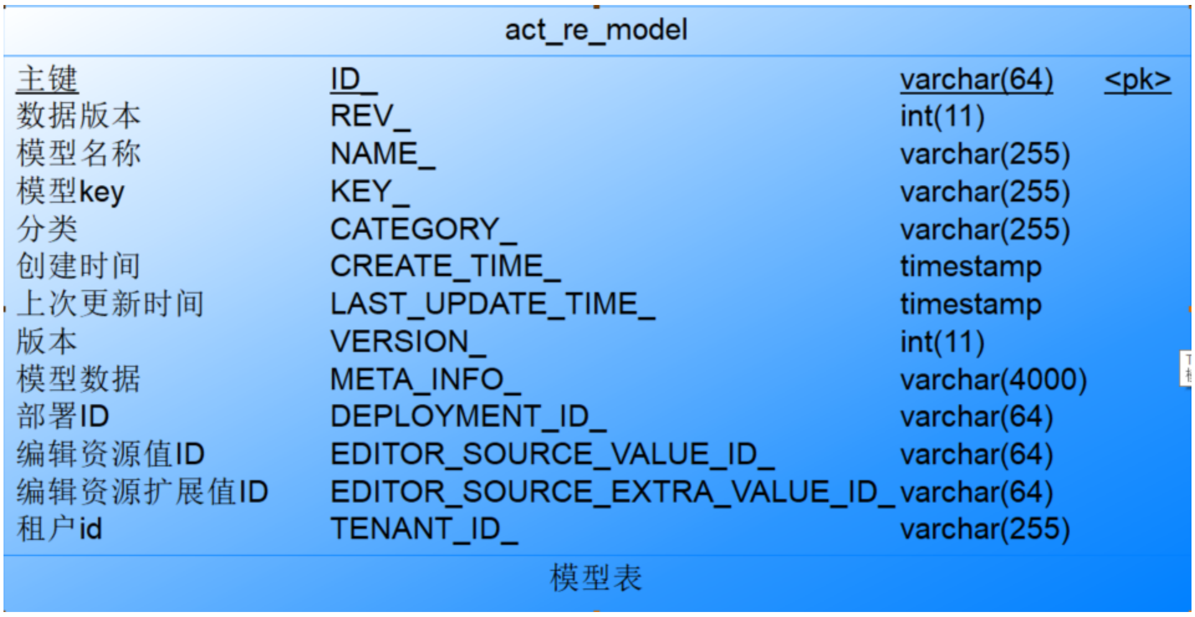
- 实体类

```
package org.flowable.ui.modeler.domain;

import java.util.Date;

public class AbstractModel {
    public static final int MODEL_TYPE_BPMN = 0;
    public static final int MODEL_TYPE_FORM = 2;
    public static final int MODEL_TYPE_APP = 3;
    public static final int MODEL_TYPE_DECISION_TABLE = 4;
    public static final int MODEL_TYPE_CMMN = 5;
    public static final int MODEL_TYPE_DECISION_SERVICE = 6;
    protected String id;
    protected String name;
    protected String key;
    protected String description;
    protected Date created = new Date();
    protected Date lastUpdated;
    private String createdBy;
    private String lastUpdatedBy;
    protected int version;
    protected String modelEditorJson;
    protected String comment;
    protected Integer modelType;
    protected String tenantId;
    ...
}
```

3.2 Model REST API

使用 Model REST API 其模型数据存储在表act_re_model中。

- act_re_model表结构



act_re_model

主键	ID_	varchar(64)	<pk>
数据版本	REV_	int(11)	
模型名称	NAME_	varchar(255)	
模型key	KEY_	varchar(255)	
分类	CATEGORY_	varchar(255)	
创建时间	CREATE_TIME_	timestamp	
上次更新时间	LAST_UPDATE_TIME_	timestamp	
版本	VERSION_	int(11)	
模型数据	META_INFO_	varchar(4000)	
部署ID	DEPLOYMENT_ID_	varchar(64)	
编辑资源值ID	EDITOR_SOURCE_VALUE_ID_	varchar(64)	
编辑资源扩展值ID	EDITOR_SOURCE_EXTRA_VALUE_ID_	varchar(64)	
租户id	TENANT_ID_	varchar(255)	

模型表

- 实体类

```
package org.flowable.engine.impl.persistence.entity;

import java.io.Serializable;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;

import org.flowable.engine.ProcessEngineConfiguration;

/**
 * @author Tijs Rademakers
```

```java
 * @author Joram Barrez
 */
public class ModelEntityImpl extends AbstractBpmnEngineEntity implements
ModelEntity, Serializable {

    private static final long serialVersionUID = 1L;

    protected String name;
    protected String key;
    protected String category;
    protected Date createTime;
    protected Date lastUpdateTime;
    protected Integer version = 1;
    protected String metaInfo;
    protected String deploymentId;
    protected String editorSourceValueId;
    protected String editorSourceExtraValueId;
    protected String tenantId = ProcessEngineConfiguration.NO_TENANT_ID;
    ...
}
```

3.3 查询流程模型列表

```java
public Result getModels(PageParam pageParam, ModelCondition condition) {
    ModelQuery modelQuery = repositoryService.createModelQuery();
    listModelByCondition(modelQuery, condition);
    PageResult<ProcessDefinitionOutput> pageResult = new
PageResult(pageParam.getPageSize(), pageParam.getPageNum(),modelQuery.count(),
modelQuery.list());
    return ResultFactory.success(pageResult);
}
```

3.4 查询单个流程模型

```java
public Result getModelById(@PathVariable String modelId) {
    Model model = repositoryService.getModel(modelId);
    return ResultFactory.success(model);
}
```

3.5 新增流程模型

```java
public Result createModels(@RequestBody ModelInput modelInput) {
    ModelEntityImpl model = new ModelEntityImpl();
    model.setCreateTime(new Date());
    buildModelEntity(modelInput, model);
    repositoryService.saveModel(model);
    return ResultFactory.success(null);
}
```

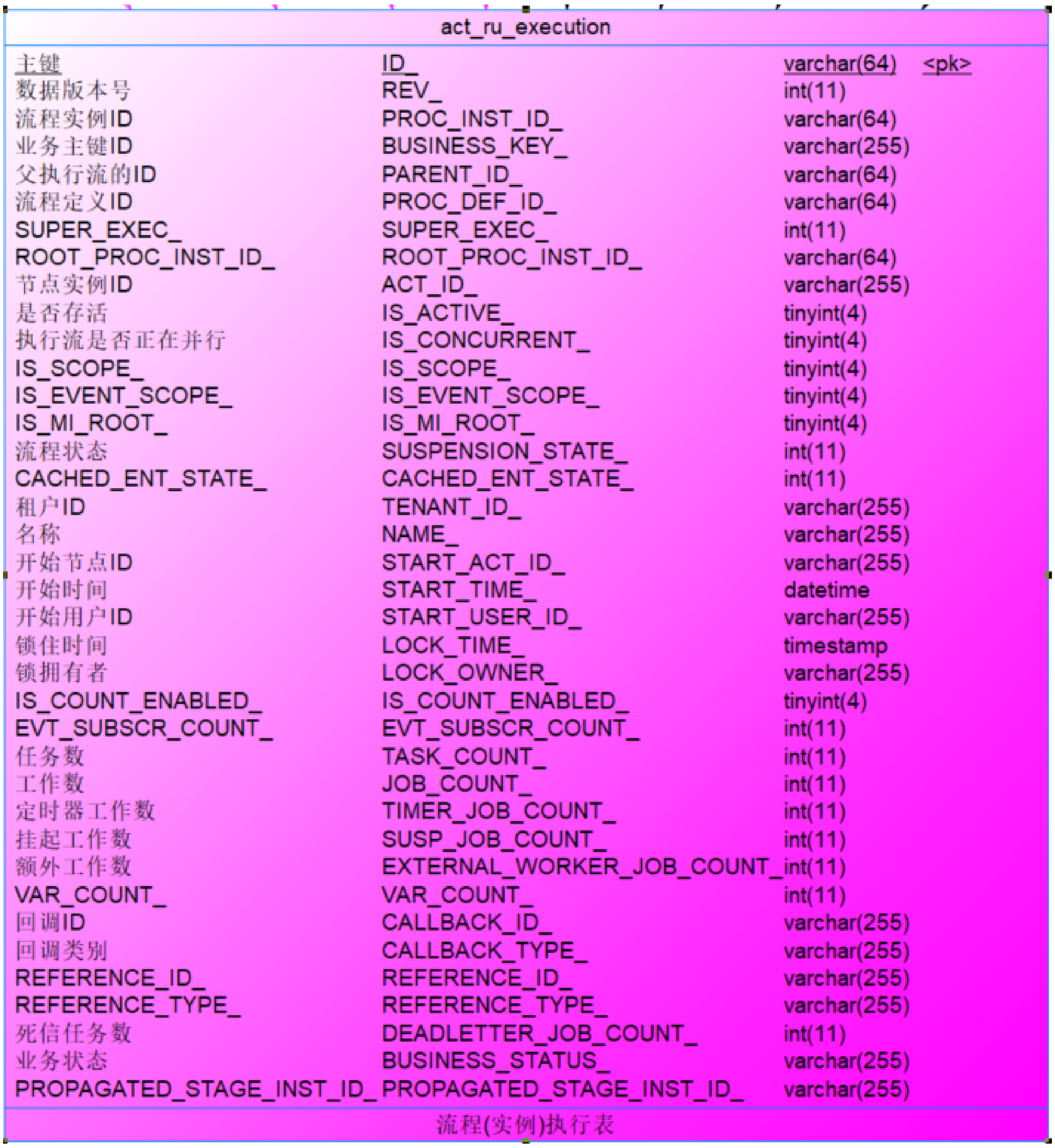
3.6 更新流程模型

```java
public Result updateModels(@PathVariable String modelId,@RequestBody ModelInput
modelInput) {
    ModelEntityImpl model = new ModelEntityImpl();
    model.setId(modelId);
    model.setLastUpdateTime(new Date());
    buildModelEntity(modelInput, model);
    repositoryService.saveModel(model);
    return ResultFactory.success(null);
}
```

3.7 删除流程模型

```java
public Result deleteModel(@PathVariable String modelId){
    repositoryService.deleteModel(modelId);
    return ResultFactory.success(null);
}
```

4、流程实例API



act_ru_execution

主键	ID_	varchar(64)	<pk>
数据版本号	REV_	int(11)	
流程实例ID	PROC_INST_ID_	varchar(64)	
业务主键ID	BUSINESS_KEY_	varchar(255)	
父执行流的ID	PARENT_ID_	varchar(64)	
流程定义ID	PROC_DEF_ID_	varchar(64)	
SUPER_EXEC_	SUPER_EXEC_	int(11)	
ROOT_PROC_INST_ID_	ROOT_PROC_INST_ID_	varchar(64)	
节点实例ID	ACT_ID_	varchar(255)	
是否存活	IS_ACTIVE_	tinyint(4)	
执行流是否正在并行	IS_CONCURRENT_	tinyint(4)	
IS_SCOPE_	IS_SCOPE_	tinyint(4)	
IS_EVENT_SCOPE_	IS_EVENT_SCOPE_	tinyint(4)	
IS_MI_ROOT_	IS_MI_ROOT_	tinyint(4)	
流程状态	SUSPENSION_STATE_	int(11)	
CACHED_ENT_STATE_	CACHED_ENT_STATE_	int(11)	
租户ID	TENANT_ID_	varchar(255)	
名称	NAME_	varchar(255)	
开始节点ID	START_ACT_ID_	varchar(255)	
开始时间	START_TIME_	datetime	
开始用户ID	START_USER_ID_	varchar(255)	
锁住时间	LOCK_TIME_	timestamp	
锁拥有者	LOCK_OWNER_	varchar(255)	
IS_COUNT_ENABLED_	IS_COUNT_ENABLED_	tinyint(4)	
EVT_SUBSCR_COUNT_	EVT_SUBSCR_COUNT_	int(11)	
任务数	TASK_COUNT_	int(11)	
工作数	JOB_COUNT_	int(11)	
定时器工作数	TIMER_JOB_COUNT_	int(11)	
挂起工作数	SUSP_JOB_COUNT_	int(11)	
额外工作数	EXTERNAL_WORKER_JOB_COUNT_	int(11)	
VAR_COUNT_	VAR_COUNT_	int(11)	
回调ID	CALLBACK_ID_	varchar(255)	
回调类别	CALLBACK_TYPE_	varchar(255)	
REFERENCE_ID_	REFERENCE_ID_	varchar(255)	
REFERENCE_TYPE_	REFERENCE_TYPE_	varchar(255)	
死信任务数	DEADLETTER_JOB_COUNT_	int(11)	
业务状态	BUSINESS_STATUS_	varchar(255)	
PROPAGATED_STAGE_INST_ID_	PROPAGATED_STAGE_INST_ID_	varchar(255)	

流程(实例)执行表

4.1 启动流程实例

```
public Result<ProcessInstanceOutput> startProcessInstance(@RequestBody
StartProcessInstanceInput input) {
    if (StringUtils.hasText(input.getProcessDefinitionId()) ||
StringUtils.hasText(input.getProcessDefinitionKey())
            || StringUtils.hasText(input.getMessageName())) {
        ProcessInstanceBuilder processInstanceBuilder =
runtimeService.createProcessInstanceBuilder();
        try {
            buildProcessInstanceBuilderParams(input, processInstanceBuilder);
            ProcessInstance processInstance = processInstanceBuilder.start();
            return
ResultFactory.success(buildProcessInstanceOutput(processInstance));
        } catch (FlowableObjectNotFoundException e) {
            return ResultFactory.failed("流程定义不存在", null);
        }
    } else {
        String message = "通过processDefinitionId或者processDefinitionKey或者
messageName启动一个实例，三者必填其一";
```

```
            return ResultFactory.failed(message, null);
        }
}
```

4.2 删除流程实例

```
public Result<Boolean> deleteProcessInstance(@PathVariable String
processInstanceId, String deleteReason) {
    try {
        runtimeService.deleteProcessInstance(processInstanceId, deleteReason);
    } catch (FlowableObjectNotFoundException e) {
        return ResultFactory.failed("流程实例id: " + processInstanceId + "的流程实例
不存在", null);
    }
    return ResultFactory.success(Boolean.TRUE);
}
```

4.3 查询流程实例

```
public Result getProcessInstances(PageParam pageParam, ProcessInstanceCondition
condition) {
    ProcessInstanceQuery processInstanceQuery =
runtimeService.createProcessInstanceQuery();
    listByCondition(condition, processInstanceQuery);
    List<ProcessInstance> processInstances =
processInstanceQuery.listPage((pageParam.getPageNum() - 1) *
pageParam.getPageSize(), pageParam.getPageNum() * pageParam.getPageSize());
    List<ProcessInstanceOutput> processInstanceOutputs =
processInstances.stream().map(this::buildProcessInstanceOutput).collect(Collecto
rs.toList());
    PageResult pageResult = new PageResult(pageParam.getPageSize(),
pageParam.getPageNum(), processInstanceQuery.count(), processInstanceOutputs);
    return ResultFactory.success(pageResult);
}
```

5、官方文档地址

flowable快速入门