写在前面：从《库表与服务》这篇中的‘服务’可以看出来，processEngine是核心关键 通过流程引擎创建出对应需要的服务。从使用RepositoryService部署流程 可以创建模型并转化成部署文件进行部署，部署完成启动后 可以使用RuntimeService查看运行状态的示例，接下来任务的流程可以使用TaskService进行任务的签收 办理 指派。最后 可以使用HistoryService查看所有以往的流程实例 任务等信息。

环境：springboot2 ，activi6，mysql 5.7

项目地址： 码云： <https://gitee.com/belonghuang/activiti-test> github： <https://github.com/Blankwhiter/activiti-test>

项目关键目录一览：

**一、springboot集成activiti**

**1 pom.xml jar包环境**

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.0.3.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.example</groupId>

<artifactId>activiti-test</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>activiti-test</name>

<description>Demo project for Spring Boot</description>

<properties>

<java.version>1.8</java.version>

</properties>

<dependencies>

<!--aop 拦截-->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-aop</artifactId>

<version>2.0.3.RELEASE</version>

<scope>compile</scope>

</dependency>

<!--web模块-->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!--spring-boot集成mybatis-->

<dependency>

<groupId>com.ruijc</groupId>

<artifactId>spring-boot-starter-mybatis</artifactId>

<version>3.2.2</version>

</dependency>

<!--lombok -->

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

<!--test 用例-->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<!--druid 连接池-->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>druid-spring-boot-starter</artifactId>

<version>1.1.10</version>

</dependency>

<!--mysql-->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.43</version>

<!--<version>8.0.15</version>-->

</dependency>

<!--activiti基础包-->

<dependency>

<groupId>org.activiti</groupId>

<artifactId>activiti-spring-boot-starter-basic</artifactId>

<version>6.0.0</version>

</dependency>

<dependency>

<groupId>org.activiti</groupId>

<artifactId>activiti-json-converter</artifactId>

<version>6.0.0</version>

</dependency>

<dependency>

<groupId>org.apache.xmlgraphics</groupId>

<artifactId>batik-transcoder</artifactId>

<version>1.7</version>

</dependency>

<dependency>

<groupId>org.apache.xmlgraphics</groupId>

<artifactId>batik-codec</artifactId>

<version>1.7</version>

</dependency>

<!--分页插件 4.1.5 版本以上修复 selectProvider不兼容问题-->

<dependency>

<groupId>com.github.pagehelper</groupId>

<artifactId>pagehelper</artifactId>

<version>4.1.5</version>

</dependency>

<!--common-->

<!--<dependency>-->

<!--<groupId>org.apache.commons</groupId>-->

<!--<artifactId>commons-lang3</artifactId>-->

<!--<version>3.0</version>-->

<!--</dependency>-->

<!--热部署-->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<optional>true</optional>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<!--设置外部打包-->

<configuration>

<includeSystemScope>true</includeSystemScope>

</configuration>

</plugin>

<!--mybatis generator begin-->

<plugin>

<!--Mybatis-generator插件,用于自动生成Mapper和POJO-->

<groupId>org.mybatis.generator</groupId>

<artifactId>mybatis-generator-maven-plugin</artifactId>

<version>1.3.5</version>

<configuration>

<!--配置文件的位置-->

<configurationFile>src/main/resources/generator/mybatis\_generator\_config.xml

</configurationFile>

<verbose>true</verbose>

<overwrite>true</overwrite>

</configuration>

<executions>

<execution>

<id>Generate MyBatis Artifacts</id>

<phase>deploy</phase>

<goals>

<goal>generate</goal>

</goals>

</execution>

</executions>

<dependencies>

<!--防止找不到驱动 报出如下异常：Exception getting JDBC Driver: com.mysql.jdbc.Driver-->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.43</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.mybatis/mybatis -->

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

<version>3.4.4</version>

</dependency>

<!-- mybatis-generator-core 反向生成java代码-->

<dependency>

<groupId>org.mybatis.generator</groupId>

<artifactId>mybatis-generator-core</artifactId>

<version>1.3.5</version>

</dependency>

</dependencies>

</plugin>

<!--mybatis generator end-->

</plugins>

</build>

</project>

**2.application.properties 配置数据库连接、 activiti、 静态资源路径等属性**

server.port=8083

#访问的静态资源

spring.mvc.view.prefix=classpath:/static/

#数据库连接

spring.datasource.type=com.alibaba.druid.pool.DruidDataSource

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/act?serverTimezone=Hongkong&useUnicode=true&characterEncoding=utf8&zeroDateTimeBehavior=convertToNull&useSSL=false

spring.datasource.username=root

spring.datasource.password=111111

# 下面为连接池的补充设置，应用到上面所有数据源中

# 初始化大小，最小，最大

spring.datasource.druid.initial-size=5

spring.datasource.druid.min-idle=5

spring.datasource.druid.max-active=20

# 配置获取连接等待超时的时间

spring.datasource.druid.max-wait=60000

# 配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒

spring.datasource.druid.time-between-eviction-runs-millis=60000

# 配置一个连接在池中最小生存的时间，单位是毫秒

spring.datasource.druid.min-evictable-idle-time-millis=300000

spring.datasource.druid.validation-query=SELECT 1 FROM DUAL

spring.datasource.druid.test-while-idle=true

spring.datasource.druid.test-on-borrow=false

spring.datasource.druid.test-on-return=false

# 打开PSCache，并且指定每个连接上PSCache的大小

spring.datasource.druid.pool-prepared-statements=true

spring.datasource.druid.max-pool-prepared-statement-per-connection-size=20

# 配置监控统计拦截的filters，去掉后监控界面sql无法统计，'wall'用于防火墙

spring.datasource.druid.filter.commons-log.connection-logger-name=stat,wall,log4j

spring.datasource.druid.filter.stat.log-slow-sql=true

spring.datasource.druid.filter.stat.slow-sql-millis=2000

# 通过connectProperties属性来打开mergeSql功能；慢SQL记录

spring.datasource.druid.connect-properties.=druid.stat.mergeSql=true;druid.stat.slowSqlMillis=5000

# 合并多个DruidDataSource的监控数据

spring.datasource.druid.use-global-data-source-stat=true

#druid 登陆配置

spring.datasource.druid.web-stat-filter.enabled=false

spring.datasource.druid.stat-view-servlet.login-username=admin

spring.datasource.druid.stat-view-servlet.login-password=admin

#mybatis 配置

mybatis.mapper-locations=classpath:mappers/\*Mapper.xml

mybatis.type-aliases-package=com.example.activititest.po

mybatis.check-config-location=true

mybatis.config-location=classpath:mybatis-config.xml

# activiti 自动部署验证设置:true-开启（默认）、false-关闭

spring.activiti.check-process-definitions=false

# activiti 当自动检查 并更新数据库接表不存在则进行创建

spring.activiti.database-schema-update=true

#监控地址端口

management.server.port=7000

#springboot2.0之后，在Http环境下将默认的endpoint只设置为info和health，要想开启其他的监控功能，需要手动配置

management.endpoints.web.exposure.include=\*

#请求连接前缀 默认是/actuator

management.endpoints.web.base-path=/actuator

**3.ActivitiTestApplication 启动类 需要去除org.activiti.spring.boot.SecurityAutoConfiguration**

import org.activiti.spring.boot.SecurityAutoConfiguration;

import org.mybatis.spring.annotation.MapperScan;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

/\*\*

\* 配置mybatis扫描包 去除exclude = SecurityAutoConfiguration.class ，

\* org.activiti.spring.boot.SecurityAutoConfiguration会导致

\* Invocation of init method failed; nested exception is java.lang.ArrayStoreException: sun.reflect.annotation.TypeNotPresentExceptionProxy

\*/

@SpringBootApplication(exclude = SecurityAutoConfiguration.class)

@MapperScan(basePackages = "com.example.activititest.dao")

public class ActivitiTestApplication {

public static void main(String[] args) {

SpringApplication.run(ActivitiTestApplication.class, args);

}

}

**4.将流程在线设计器放置再static目录下**

此步骤 请从源码中拷贝。 需要注意的是 1.activiti-test\src\main\resources\static\editor-app\app-cfg.js 配置请求的根路径 2.activiti-test\src\main\resources\static\editor-app\configuration\url-config.js 配置了 模型json，获得模具集合，保存模型

**二、接口演示**

**案例测试：**

1.流程图展示：  *注：此图是使用Idea中的actiBPM插件*

2.示例请假流程说明：员工发起流程，经有人事（group：affair）进行审核： 如审核不通过，则结束流程，否则审核通过则判断审核天数 ：如天数小于等于3天交予经理（group：manager）审批 而后结束流程，超过三天 则交予总经理（group：president）审核 而后结束流程。 *注：角色后面跟的group是对应每个角色的分组标识。用于查看待签任务时候，查看属于自己组别的可签收的任务*

**1.部署流程文件**

测试ActivitiDefineController下的deploy方法 经测试可以看到返回部署成功

**2.部署成功了，当然可以查看对应的部署列表了**

测试ActivitiDefineController下的list方法 经测试可以看到返回列表数据

**3.当然了这样可能不够直观 ，可以通过列表中的 部署id，查看流程图**

测试ActivitiDefineController下的viewProcessImage方法 经测试可以看到返回定义的流程图

**4.查看该这个流程图觉得没有问题了，我们就准备启动该流程定义，创建流程实例（即员工张三发起请假）。**

测试ActivitiDefineController下的startProcessInstanceById方法 经测试可以看到返回成功（这里方便测试将assignee-‘当前用户’，由前台传入，后面将不再赘述。 读者可将用户id做为assignee，此文为了直观使用名称作为assignee） *注：流程图人事不同意那条线 单词是错误的。供后面在线流程编辑器进行修改使用。*

**5.人事审核过程：人事角色赵四（group：affair）登陆，可查看待签任务**

测试ActivitTaskController下的claimList方法 经测试可以看到返回列表数据 *注：这里多说一句可以看到catalog这里是null，可以开始在流程部署文件中直接定义，这样流程定义等就可以按照catelog进行查找了。*

**6.人事赵四看到有可签收的任务，进行签收**

测试ActivitTaskController下的claim方法 经测试可以看到返回成功

*注：这是签收列表中就没有这一条记录，数据往待办任务走*

**7.赵四签收了任务，现在就可以在待办任务中 找到刚才签收的任务了**

测试ActivitTaskController下的todoList方法 经测试可以看到返回列表数据

**8.赵四找到了待办任务，进行审批**

测试ActivitTaskController下的complete方法 经测试可以看到返回成功（需要注意这里人事需要进行审批，审批通过才会有后面的流程即需要传参agree=‘yes’， agress=‘yes’， days=3） 注：这里是将 不同意那条线的判断参数错将agree写成</font color=red>agress 此处留下一个伏笔将会用到模型转换并使用流程设计器进行修改（此处暂不赘述），完成该任务，待办列表中就没有了该条记录，数据将往后一个流程经理审批进行签收，读者也可以将days大于3，数据将往总经理审批。

**9.经理审核流程：经理角色王五（group：manager）登录，可查看待签任务**

测试ActivitTaskController下的claimList方法 经测试可以看到返回列表数据

**10.王五签收了任务，而后从待办任务中 可查看该任务**

测试ActivitTaskController下的claim方法 经测试可以看到返回成功

**11.王五找到待办任务**

测试ActivitTaskController下的todoList方法 经测试可以看到返回列表数据

**12.王五完成任务，即结束了流程**

测试ActivitTaskController下的complete方法 经测试可以看到返回成功

**13.在上诉任务开始的时候，张三突然想看自己发起流程的进度情况了**

测试ActivitTaskController下的sentList方法 经测试可以看到返回列表数据 *注：可以看到流程开始时间 结束时间 流程全部时间等信息*

**14.或者某个人突然想查看自己参与过的办结流程记录**

测试ActivitTaskController下的finishList方法 经测试可以看到返回列表数据（这里以赵四为例）

**15.或者查看流程所审批的流程过程**

测试ActivitTaskController下的viewProgressPathImage方法 经测试可以看到返回流程轨迹图

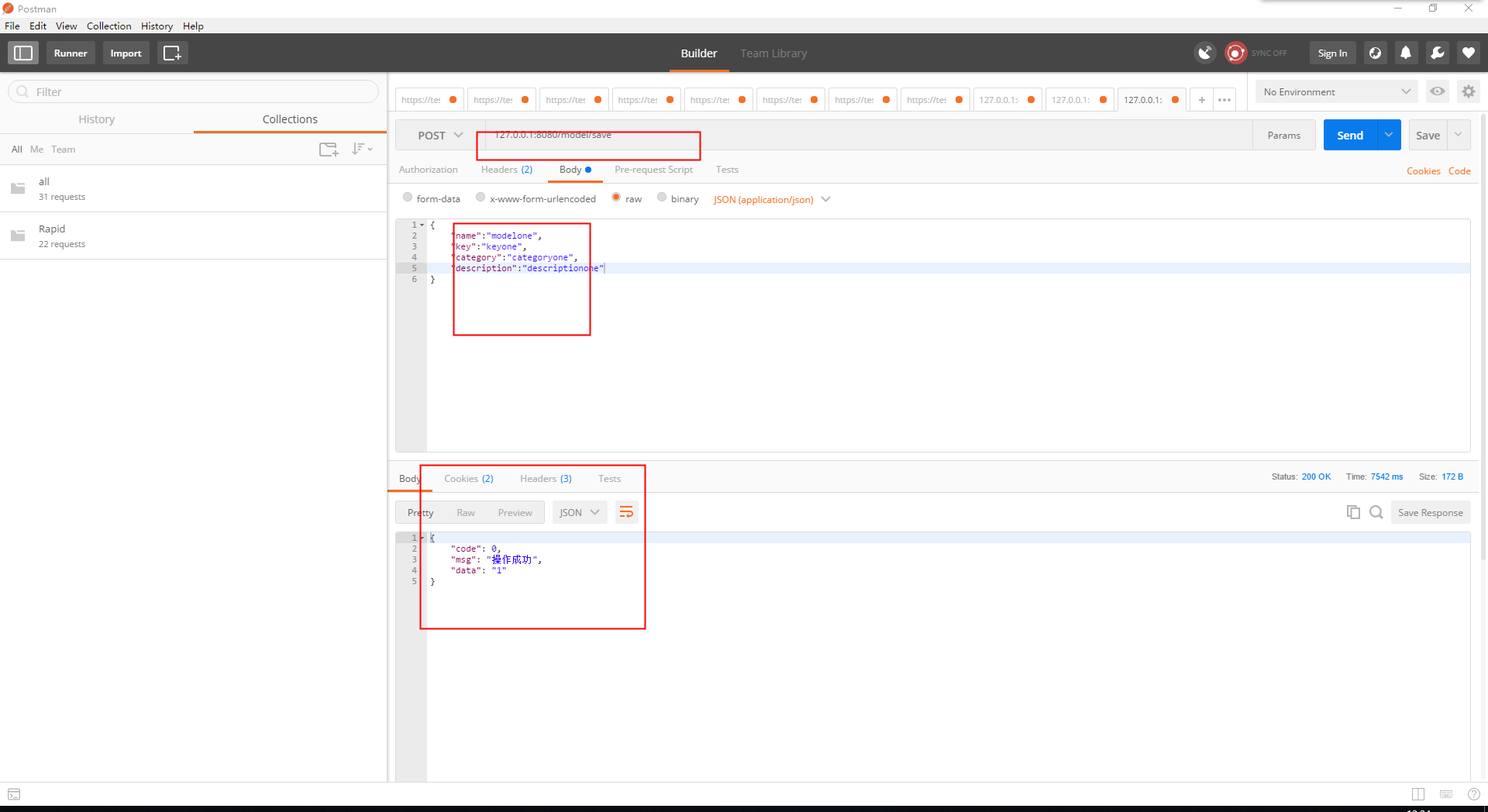
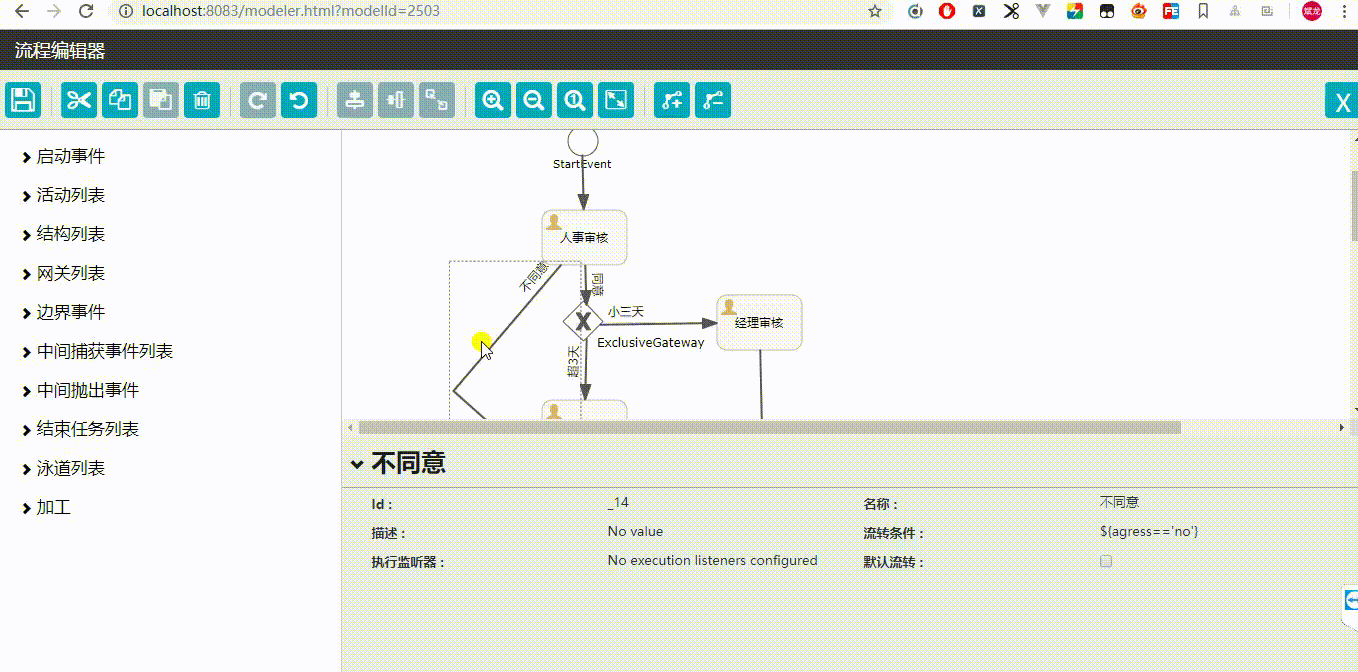
**16.在发现部署文件有问题时候，我们可以转换成模型，用在线流程编辑器修改。**

测试ActivitiDefineController下的convertToModel方法 经测试可以看到返回成功

**17.查看模型列表**

测试ActivitModelController下的list方法 经测试可以看到返回列表数据

**18.使用线流程编辑器 将id传入**

打开浏览器<http://localhost:8083/modeler.html?modelId=2503> （这里涉及到ActivitiVisualController的方法，请看源码此处不再赘述） [](https://camo.githubusercontent.com/fd795d4587fb18ebc44fc211f42dc82a1af9a1fa028c40946bcacb70b6e870ce/68747470733a2f2f696d672d626c6f672e6373646e696d672e636e2f32303139303831343230353531363932392e676966)*注：修改后保存，而后进行部署。*

**19.流程编辑保存后进行部署生效，迭代版本**

测试ActivitModelController下的deploy方法 经测试可以看到返回成功

**20.这时候再看流程定义，可以看到版本（version：2）迭代**

测试ActivitiDefineController下的list方法 经测试可以看到返回列表数据

可以用ActivitModelController中的save方法创建出模型，而后使用在线流程设计器画出流程部署，效果跟官网的样例是一样的。

表的介绍

Activiti的后台是有数据库的支持，所有的表都以ACT\_开头。 第二部分是表示表的用途的两个字母标识。 用途也和服务的API对应。

ACT\_RE\_\*: 'RE'表示repository。 这个前缀的表包含了流程定义和流程静态资源 （图片，规则，等等）。

ACT\_RU\_\*: 'RU'表示runtime。 这些运行时的表，包含流程实例，任务，变量，异步任务，等运行中的数据。 Activiti只在流程实例执行过程中保存这些数据， 在流程结束时就会删除这些记录。 这样运行时表可以一直很小速度很快。

ACT\_ID\_\*: 'ID'表示identity。 这些表包含身份信息，比如用户，组等等。

ACT\_HI\_\*: 'HI'表示history。 这些表包含历史数据，比如历史流程实例， 变量，任务等等。

ACT\_GE\_\*: 通用数据， 用于不同场景下，如存放资源文件。

资源库流程规则表

   1) act\_re\_deployment 部署信息表

   2) act\_re\_model  流程设计模型部署表

   3) act\_re\_procdef  流程定义数据表

运行时数据库表

   1) act\_ru\_execution运行时流程执行实例表

   2) act\_ru\_identitylink运行时流程人员表，主要存储任务节点与参与者的相关信息

   3) act\_ru\_task运行时任务节点表

   4) act\_ru\_variable运行时流程变量数据表

历史数据库表

   1) act\_hi\_actinst 历史节点表

   2) act\_hi\_attachment历史附件表

   3) act\_hi\_comment历史意见表

   4) act\_hi\_identitylink历史流程人员表

   5) act\_hi\_detail历史详情表，提供历史变量的查询

   6) act\_hi\_procinst历史流程实例表

   7) act\_hi\_taskinst历史任务实例表

   8) act\_hi\_varinst历史变量表

组织机构表

   1) act\_id\_group用户组信息表

   2) act\_id\_info用户扩展信息表

   3) act\_id\_membership用户与用户组对应信息表

   4) act\_id\_user用户信息表

这四张表很常见，基本的组织机构管理，关于用户认证方面建议还是自己开发一套，组件自带的功能太简单，使用中有很多需求难以满足

通用数据表

   1) act\_ge\_bytearray二进制数据表

   2) act\_ge\_property属性数据表存储整个流程引擎级别的数据,初始化表结构时，会默认插入三条记录

————————————————

版权声明：本文为CSDN博主「艾斯-李」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/las723/article/details/88417152/