目录

版权说明 1.1

第1章 项目介绍 1.2

- 1.1 项目特点 1.2.1
- 1.2 项目结构 1.2.2
- 1.3 Nacos环境搭建 1.2.3
- 1.4 开发环境搭建 1.2.4
- 1.5 Sentinel使用 1.2.5
- 1.6 SkyWalking使用 1.2.6
- 1.7 Seata使用 1.2.7
- 1.8 获取帮助 1.2.8

第2章 数据库支持 1.3

- 2.1 MySQL数据库支持 1.3.1
- 2.2 Oracle数据库支持 1.3.2
- 2.3 SQL Server数据库支持 1.3.3
- 2.4 PostgreSQL数据库支持 1.3.4

第3章 多数据源支持 1.4

- 3.1 多数据源配置 1.4.1
- 3.2 多数据源使用 1.4.2
- 3.3 源码讲解 1.4.3

第4章 基础知识讲解 1.5

- 4.1 Spring MVC使用 1.5.1
- 4.2 Swagger使用 1.5.2
- 4.3 Mybatis-plus使用 1.5.3
- 4.4 Hibernate Validator使用 1.5.4
- 4.5 Feign使用 1.5.5

第5章 工作流模块 1.6

- 5.1 工作流模块 1.6.1

第6章 生产环境部署 1.7

- 6.1 jar包部署 1.7.1
- 6.2 docker部署 1.7.2
- 6.3 Nginx配置 1.7.3

版权说明

> 本文档为付费文档，版权归人人开源（renren.io）所有，并保留一切权利，本文档及其描述的内容受有关法律的版权保护，对本文档以任何形式的非法复制、泄露或散布到网络提供下载，都将导致相应的法律责任。

免责声明

> 本文档仅提供阶段性信息，所含内容可根据项目的实际情况随时更新，以人人开源社区公告为准。如因文档使用不当造成的直接或间接损失，人人开源不承担任何责任。

第1章 项目介绍

人人微服务平台(renren-cloud)，是基于Spring Cloud打造的微服务平台，为企业提供一站式微服务解决方案！主要包括用户管理、角色管理、部门管理、菜单管理、参数管理、字典管理、登录日志、操作日志、异常日志、国际化支持、Redis缓存、数据权限、接口文档等功能。

1.1 项目特点

1.2 项目结构

1.3 Nacos环境搭建

1.4 开发环境搭建

1.5 Sentinel使用

1.6 SkyWalking使用

1.7 Seata使用

1.8 获取帮助

1.1 项目描述

- 基于最新的Spring Cloud Alibaba 2.1、Nacos、Spring Cloud Gateway、Spring Boot 2.1、Mybatis、Element 2.0+开发
- 代码风格优雅简洁、通俗易懂，且符合《阿里巴巴Java开发手册》规范要求，可作为企业代码规范
- 优秀的菜单功能权限，前端可灵活控制页面及按钮的展示，后端可对未授权的请求进行拦截
- 优秀的数据权限管理，只需增加相应注解，无需其他任何代码，即可实现数据过滤，达到数据权限目的
- 灵活的角色权限管理，新增角色时，角色权限只能是创建者权限的子集，可有效防止权限越权
- 灵活的日志管理，可查看登录日志、操作日志、异常日志，方便审计及BUG定位
- 灵活的国际化配置，目前已支持简体中文、繁体中文、English，如需增加新语言，只需增加新语言[i18n]文件即可
- 灵活的前端动态路由，新增页面无需修改路由文件，也可在页面动态新增tab标签
- 支持MySQL、Oracle、SQL Server、PostgreSQL等主流数据库
- 推荐使用阿里云服务器部署项目，免费领取阿里云优惠券，请点击【免费领取】

1.2 项目结构

```
renren-cloud
|
├─db     数据库初始化脚本
|  |
|  ├─mysql.sql        MySQL数据库
|  |
|  ├─oracle.sql       Oracle数据库
|  |
|  ├─sqlserver.sql    SQL Server数据库
|  |
|  └─postgresql.sql   PostgreSQL数据库
|
|
├─docker   创建docker镜像文件
|
|
├─docker-compose   启动docker服务
|
|
├─renren-admin    后台管理
|    |
|    ├─renren-admin-client     其他服务调用封装
|    |
|    └─renren-admin-server     后台管理服务
|
|
├─renren-auth      授权服务
|
|
├─renren-commons  代码生成器
|       |
|       ├─renren-commons-dynamic-datasource    多数据源
|       |
|       ├─renren-commons-mybatis               mybatis封装
|       |
|       └─renren-commons-tools                 工具包
|
|
├─renren-module  服务模块
|       |
|       ├─renren-activiti          工作流模块
|       |
|       ├─renren-api               api模块
|       |
|       ├─renren-job               定时任务模块
|       |
|       ├─renren-message           短信、邮件模块
|       |
```

```
|       └renren-oss                文件上传模块
|
|
├renren-gateway     Spring Cloud GateWay网关服务
|
|
├renren-monitor     服务监控
|
```

1.3 Nacos环境搭建

Nacos 致力于帮助您发现、配置和管理微服务。Nacos 提供了一组简单易用的特性集，帮助您快速实现动态服务发现、服务配置、服务元数据及流量管理。

1.3.1 搭建Nacos环境

- 从GitHub下载Nacos部署文件，下载地址：https://github.com/alibaba/nacos/releases ， 下载nacos-server-1.2.1.zip(Windows) 或 nacos-server-1.2.1.tar.gz(Linux)，解压后如下所示：

```
[root@renren nacos]# pwd
/work/nacos
[root@renren nacos]# ll
total 28
drwxr-xr-x. 4 root root   123 Dec 17 11:40 bin
drwxr-xr-x. 4 root root   192 Dec 17 11:45 conf
-rw-r--r--. 1 root root 17336 Dec 17 10:43 LICENSE
-rw-r--r--. 1 root root  1305 Dec 17 10:44 NOTICE
drwxr-xr-x. 2 root root    30 Dec 17 10:44 target
```

- 配置Nacos的MySQL数据源，需要在application.properties文件末尾，新增如下配置：

```
[root@renren config]# pwd
/work/nacos/config
[root@renren config]# vim application.properties
spring.datasource.platform=mysql
db.num=1
db.url.0=jdbc:mysql://localhost:3306/nacos?characterEncoding=utf8&connectTimeout=1000&socketT
imeout=3000&autoReconnect=true
db.user=renren
db.password=123456
```

- 执行Nacos的SQL脚本

  新建数据库nacos，运行【conf/nacos-mysql.sql】文件即可

- MySQL8.0支持

如果使用的是MySQL8.0版本，则需要做相应的修改，才能支持MySQL8.0，如下所示：

```
#在nacos根目录下，新建plugins/mysql目录
[root@renren nacos]# mkdir -p plugins/mysql
[root@renren nacos]# cd plugins/mysql
[root@renren mysql]# pwd
/work/nacos/plugins/mysql
#把MySQL8.0的驱动上传到新建的【plugins/mysql】目录里，如下
[root@renren mysql]# ll
total 2276
-rw-r--r--. 1 root root 2330539 Dec 17 11:38 mysql-connector-java-8.0.20.jar
```

```
#修改application.properties
[root@renren config]# vim /work/nacos/config/application.properties
spring.datasource.platform=mysql
db.num=1
db.url.0=jdbc:mysql://localhost:3306/nacos?characterEncoding=utf8&connectTimeout=1000&socketT
imeout=3000&autoReconnect=true&useUnicode=true&useSSL=false&serverTimezone=UTC
db.user=renren
db.password=123456
```

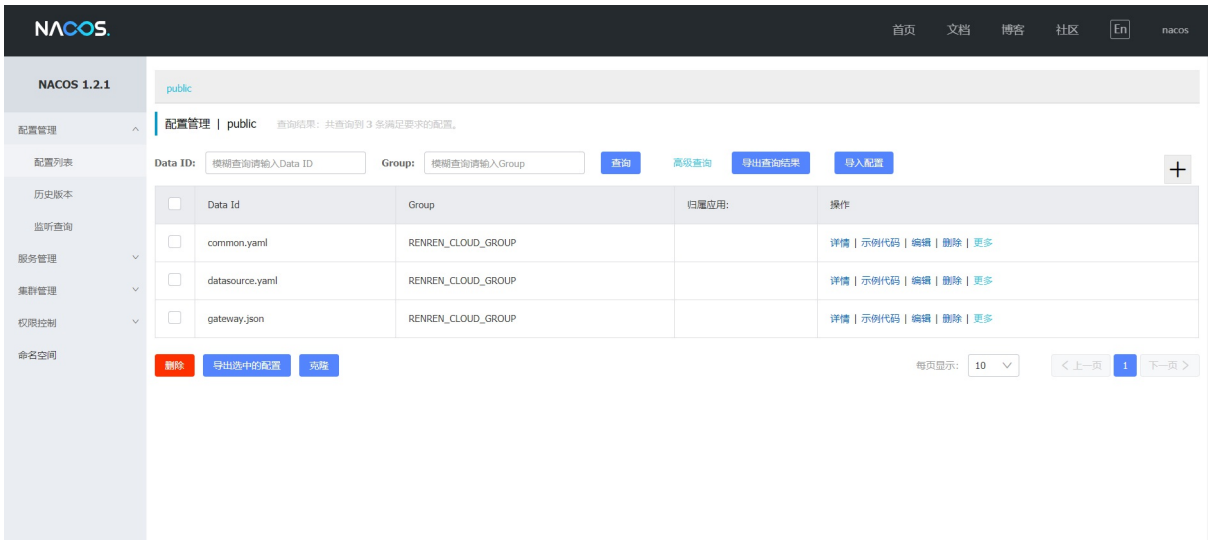
- 启动nacos服务：

```
Linux/Unix/Mac: 【sh startup.sh -m standalone】

Windows：双击【startup.cmd】即可
```

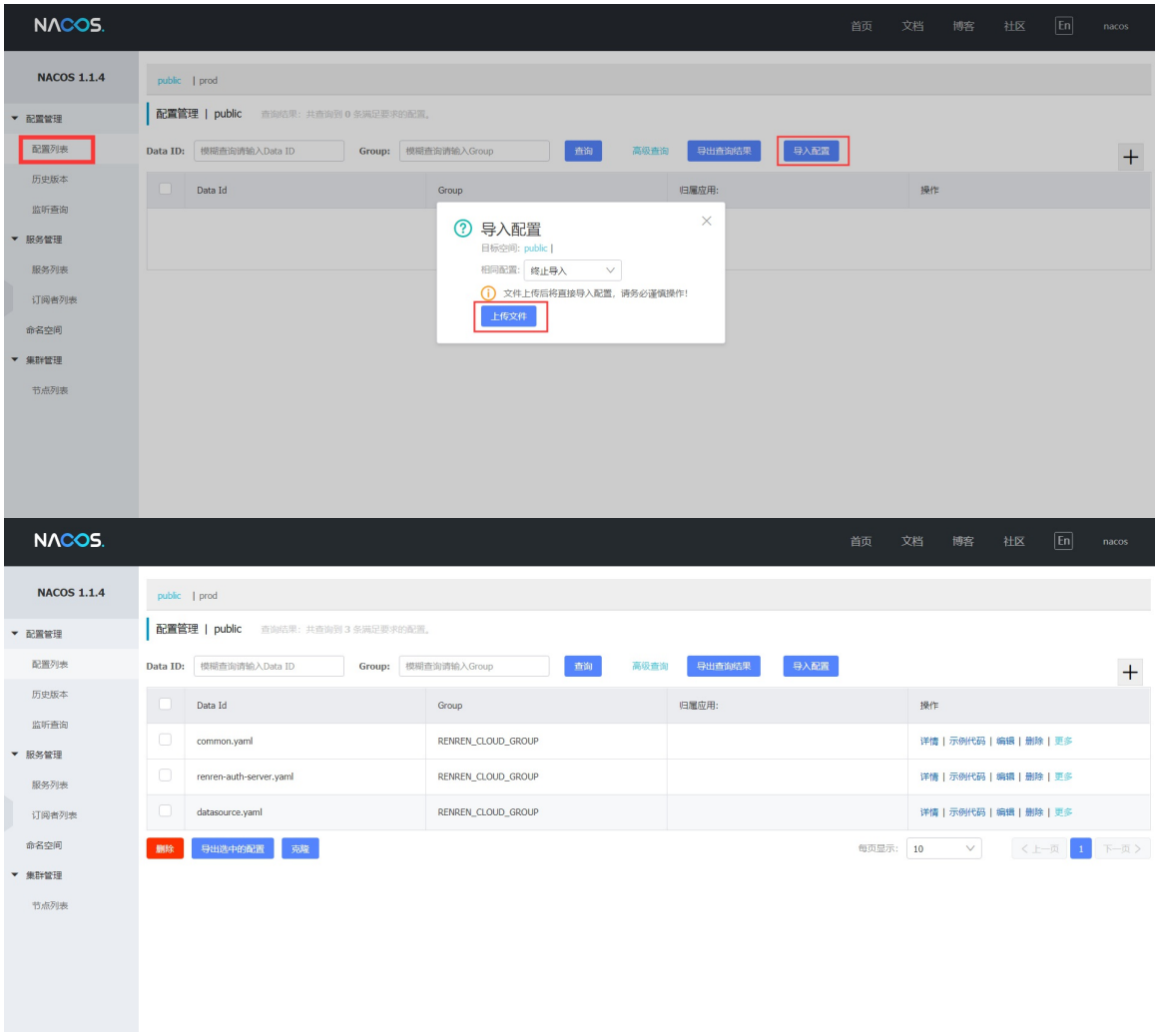
- 记得关闭Linux防火墙

```
[root@renren config]# systemctl stop firewalld
```

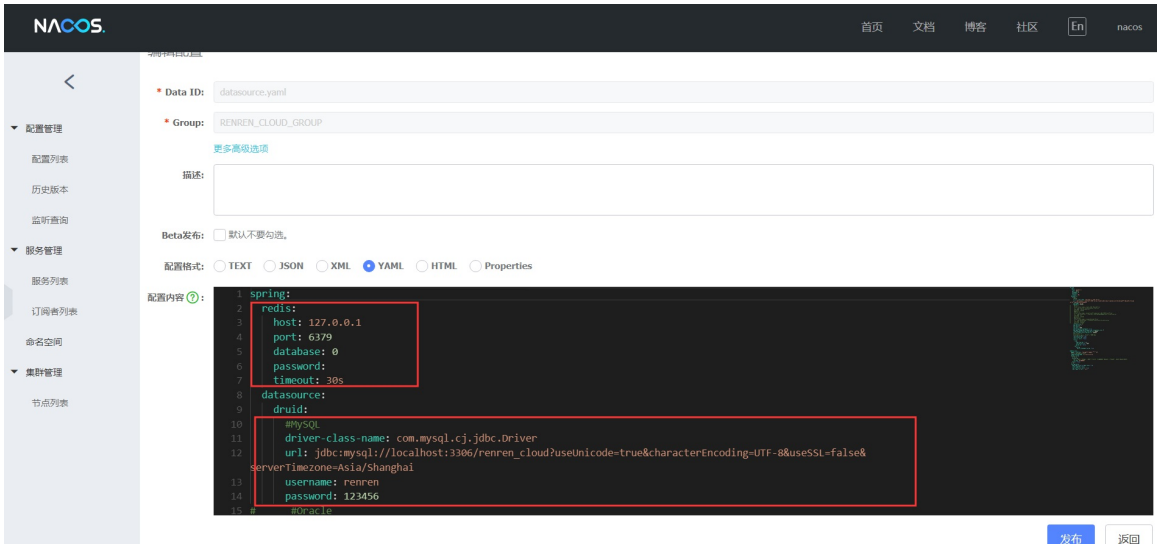
- 打开nacos管理界面(http://localhost:8848/nacos)，初始用户名nacos，密码nacos，登录之后，如下所示：



- 导入nacos配置文件，配置文件在项目里，文件名为：【~/doc/nacos/nacos_config.zip】，如下所示：



- 在nacos里，还需要修改datasource.yaml，如：redis、MySQL信息，如下所示：



- Nacos相关资料

官方文档：https://nacos.io/zh-cn/docs/quick-start.html

1.4 开发环境搭建

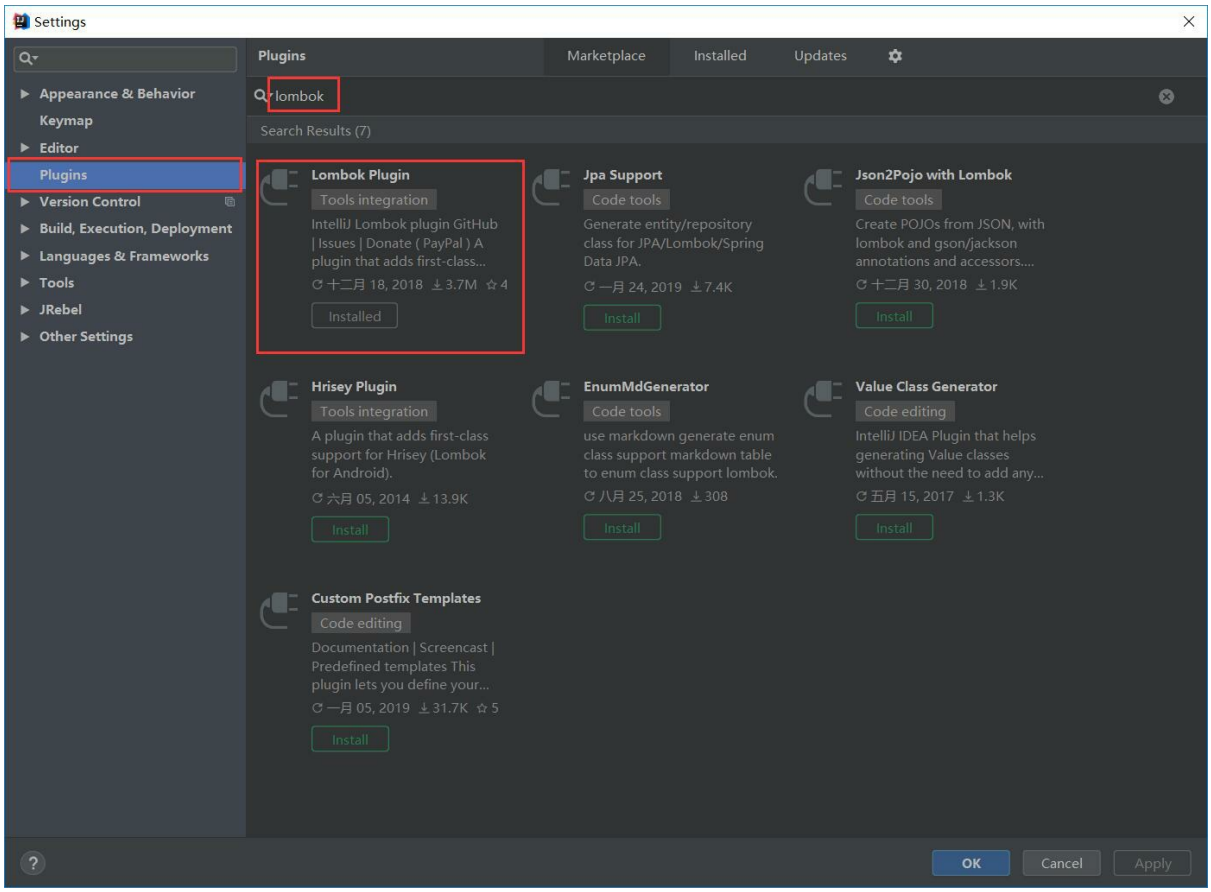
1.4.1 软件需求

- JDK 1.8+
- Maven 3.0+
- Redis 2.8+
- MySQL 8.0
- Oracle 11g+
- SQL Server 2012+
- PostgreSQL 9.4+
- Nacos 1.x

1.4.2 安装Lombok插件

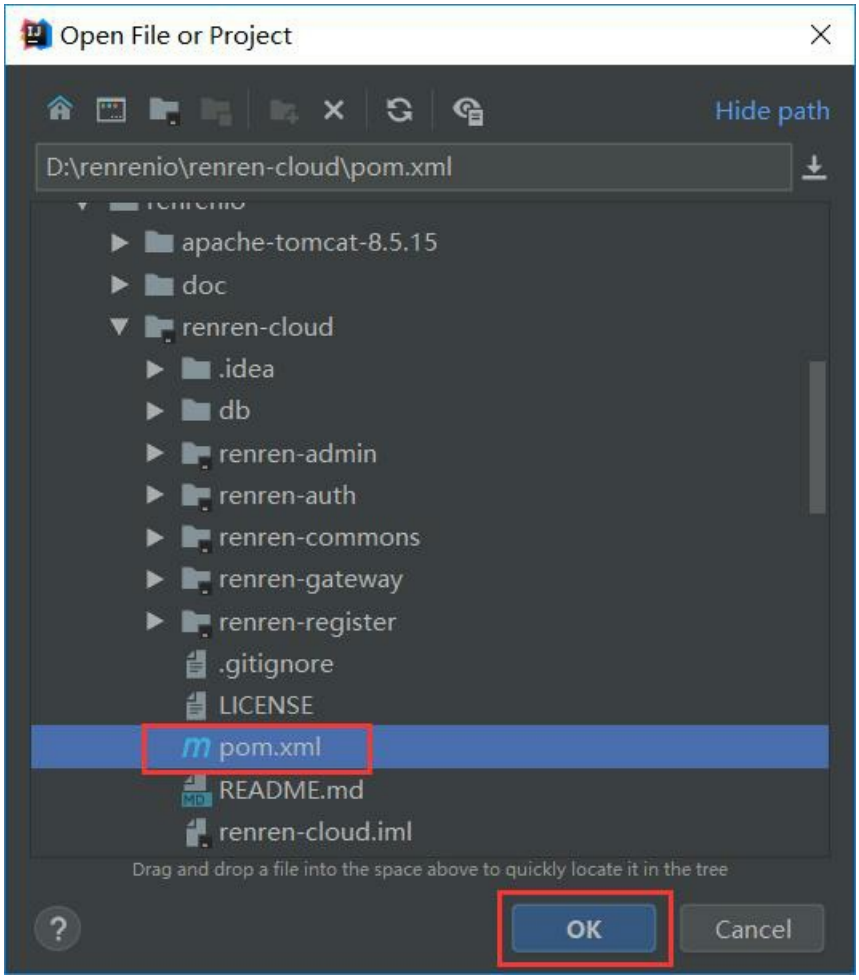
在项目中使用Lombok可以减少很多重复代码，如：getter、setter、toString等方法的编写

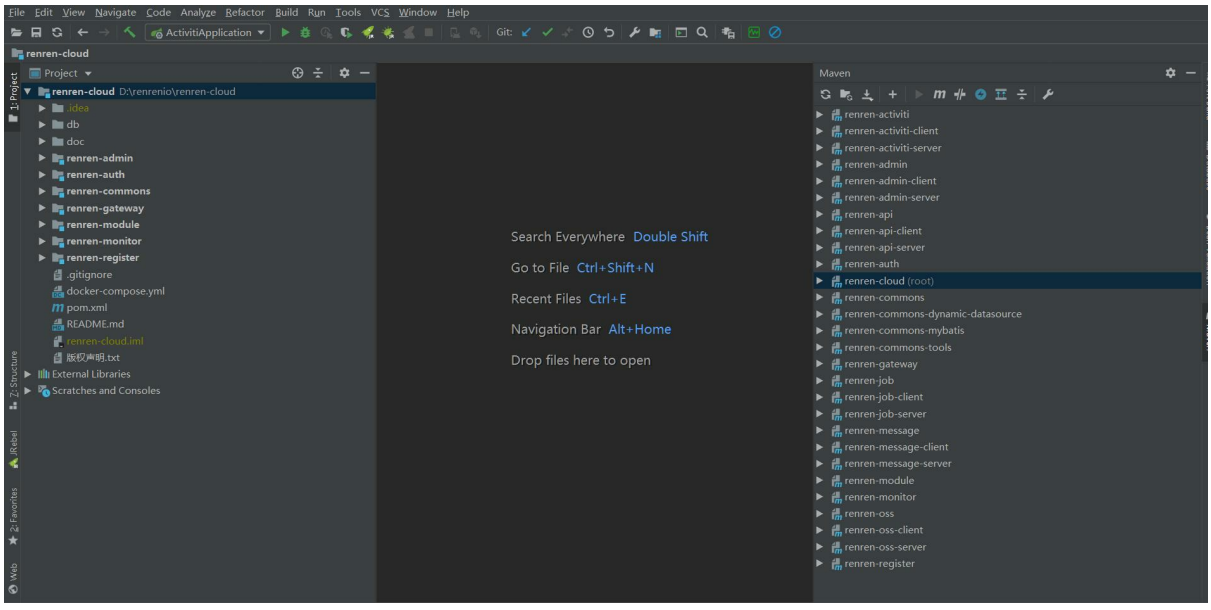
下图是idea安装Lombok方法，Eclipse类似，如下图所示：



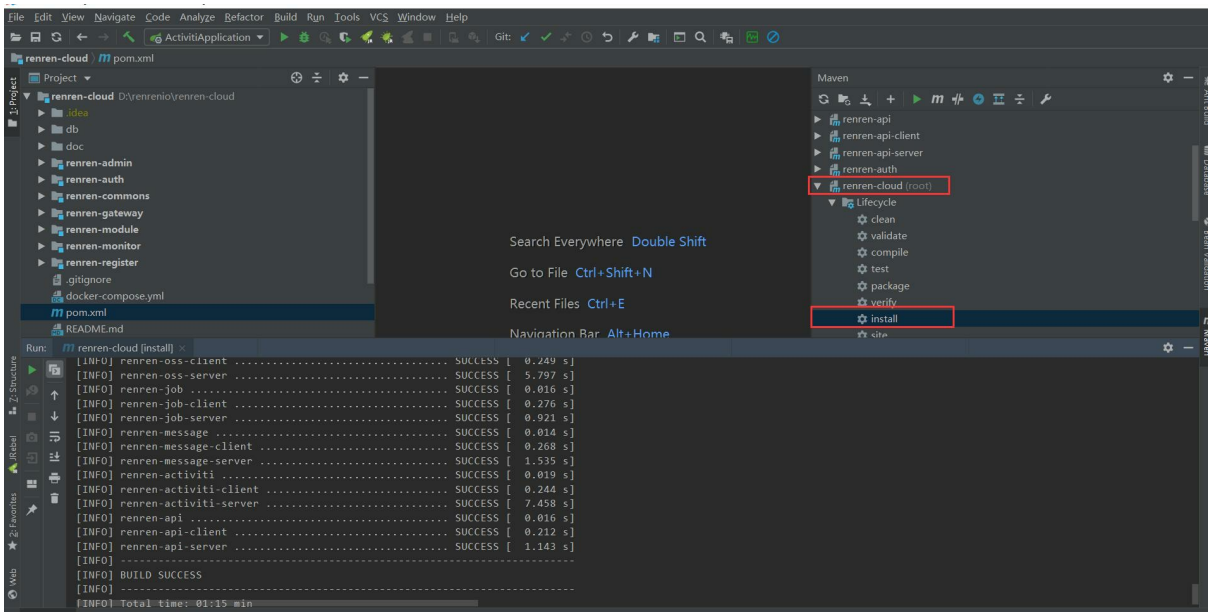
1.4.3 IDEA（Eclipse类似）运行项目

- IDEA打开项目， `File -> Open` 如下图：





- 在renren-cloud目录下，执行mvn install，如下图所示：



1.4.4 创建项目数据库

- 创建renren_cloud数据库，数据库编码为`UTF-8`
- 执行数据库脚本，如MySQL数据库，则执行`db/mysql.sql`文件，初始化数据

1.4.5 安装并启动Redis

> 如未使用过Redis的，可网上查询资料，并自行安装及启动Redis

1.4.6 配置hosts文件

> nacos需在hosts里配置，如下所示：

Windows：`C:\Windows\System32\drivers\etc\hosts`

Linux：`/etc/hosts`

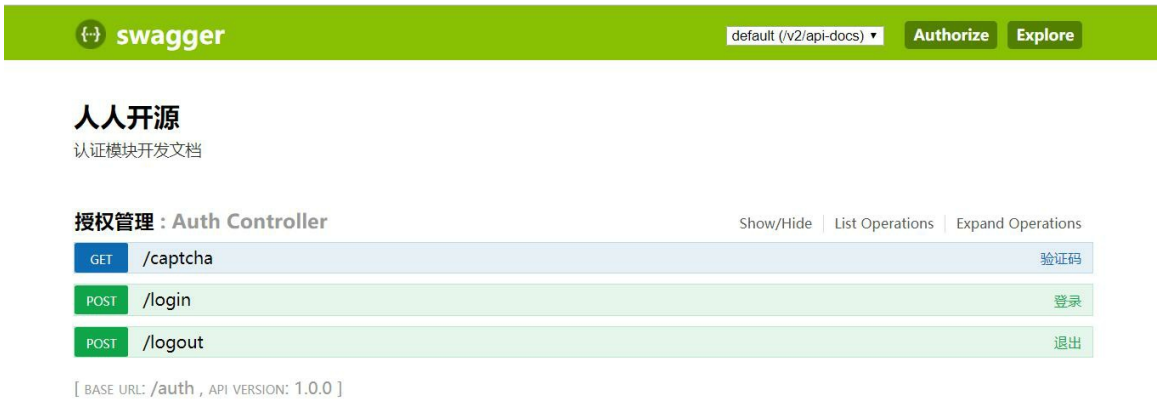
```
127.0.0.1 nacos-host
```

1.4.7 启动项目

> 需启动renren-auth、renren-admin-server、renren-gateway、renren-monitor，如下所示：

1) 启动renren-auth

- 运行`io.renren.AuthApplication.java`的`main`方法，则可启动renren-auth
- Swagger地址：http://localhost:8081/auth/doc.html



---

2) 启动renren-admin-server

- 运行 `io.renren.AdminApplication.java` 的 `main` 方法，则可启动renren-admin项目
- Swagger地址：http://localhost:8082/sys/doc.html

---

3) 启动renren-gateway

- 运行 `io.renren.GatewayApplication.java` 的 `main` 方法，则可启动GateWay网关
- Swagger地址：http://localhost:8080/doc.html

4) 启动renren-monitor

- 运行 `io.renren.MonitorApplication.java` 的 `main` 方法，则可启动服务监控

5) 配置前端接口地址(请参考前端开发文档)，并启动前端，如下所示：

6) 输入账号admin，密码admin，则可登陆系统，如下所示：

1.5 Sentinel使用

随着微服务的流行，服务和服务之间的稳定性变得越来越重要。Sentinel 以流量为切入点，从流量控制、熔断降级、系统负载保护等多个维度保护服务的稳定性。

Sentinel 具有以下特征:

- 丰富的应用场景：Sentinel 承接了阿里巴巴近 10 年的双十一大促流量的核心场景，例如秒杀（即突发流量控制在系统容量可以承受的范围）、消息削峰填谷、集群流量控制、实时熔断下游不可用应用等。
- 完备的实时监控：Sentinel 同时提供实时的监控功能。您可以在控制台中看到接入应用的单台机器秒级数据，甚至 500 台以下规模的集群的汇总运行情况。
- 广泛的开源生态：Sentinel 提供开箱即用的与其它开源框架/库的整合模块，例如与 Spring Cloud、Dubbo、gRPC 的整合。您只需要引入相应的依赖并进行简单的配置即可快速地接入 Sentinel。
- 完善的 **SPI** 扩展点：Sentinel 提供简单易用、完善的 SPI 扩展接口。您可以通过实现扩展接口来快速地定制逻辑。例如定制规则管理、适配动态数据源等。

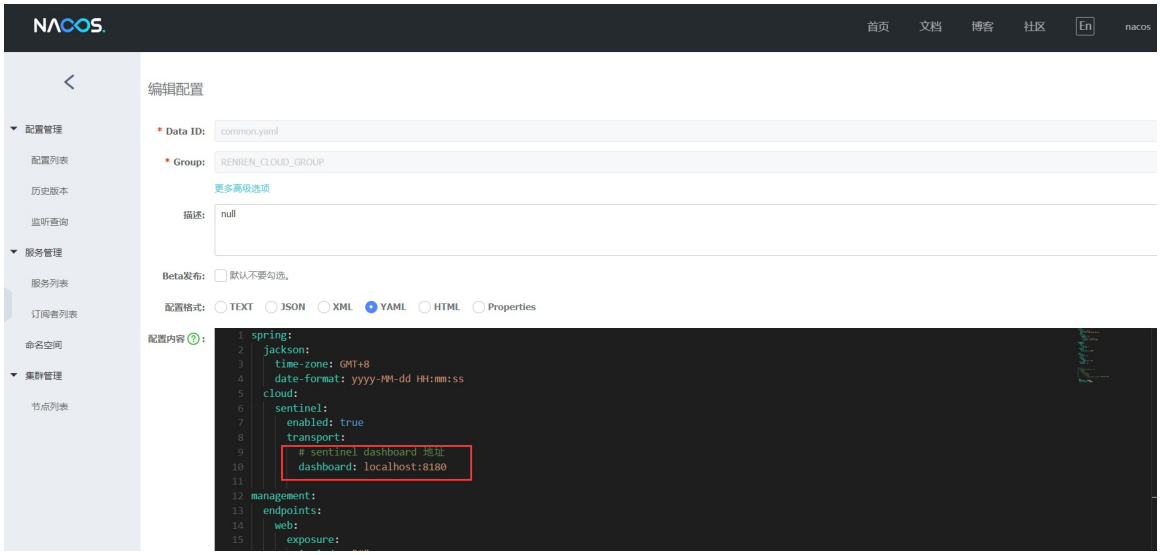
1.5.1 搭建Sentinel环境

- 从GitHub下载Sentinel部署文件，下载地址：https://github.com/alibaba/Sentinel/releases，下载[sentinel-dashboard-1.7.1.jar](https://github.com/alibaba/Sentinel/releases/download/1.7.1/sentinel-dashboard-1.7.1.jar)
- 启动 Sentinel 控制台，执行下面的命令

```
java -Dserver.port=8180 -Dcsp.sentinel.dashboard.server=localhost:8180 -Dproject.name=sentinel-dashboard -jar sentinel-dashboard-1.7.1.jar
```

- 访问Sentinel，地址：http://localhost:8180/ 账号密码：sentinel

- 修改nacos的common.yaml配置文件，指定Sentinel地址，如下

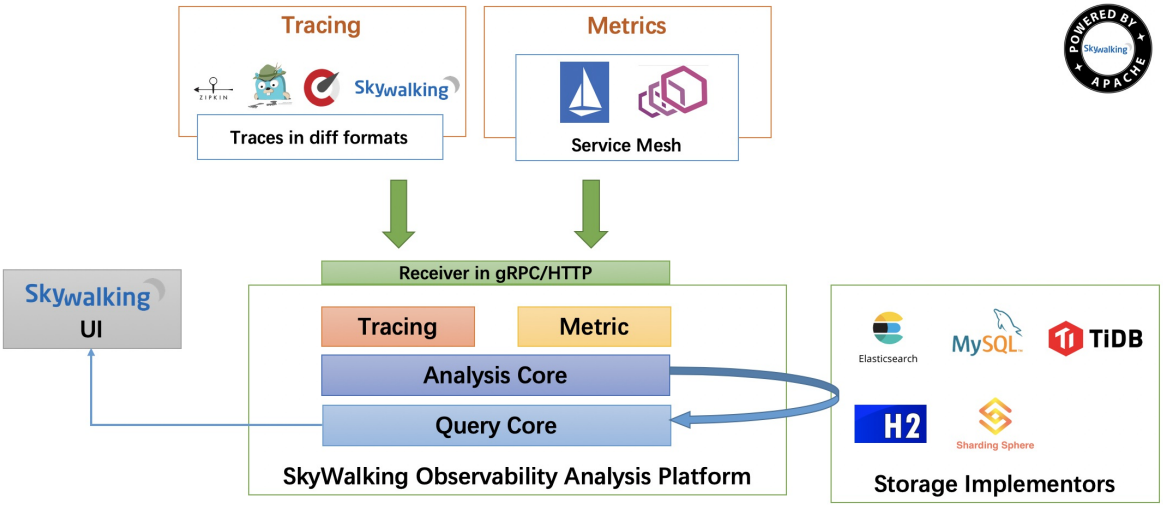


- 官方文档：https://github.com/alibaba/Sentinel/wiki/%E6%8E%A7%E5%88%B6%E5%8F%B0

1.6 SkyWalking使用

SkyWalking 是分布式系统的应用程序性能监视工具，提供分布式追踪、服务网格遥测分析、度量聚合和可视化一体化解决方案。

Skywalking的主要结构图如下所示:



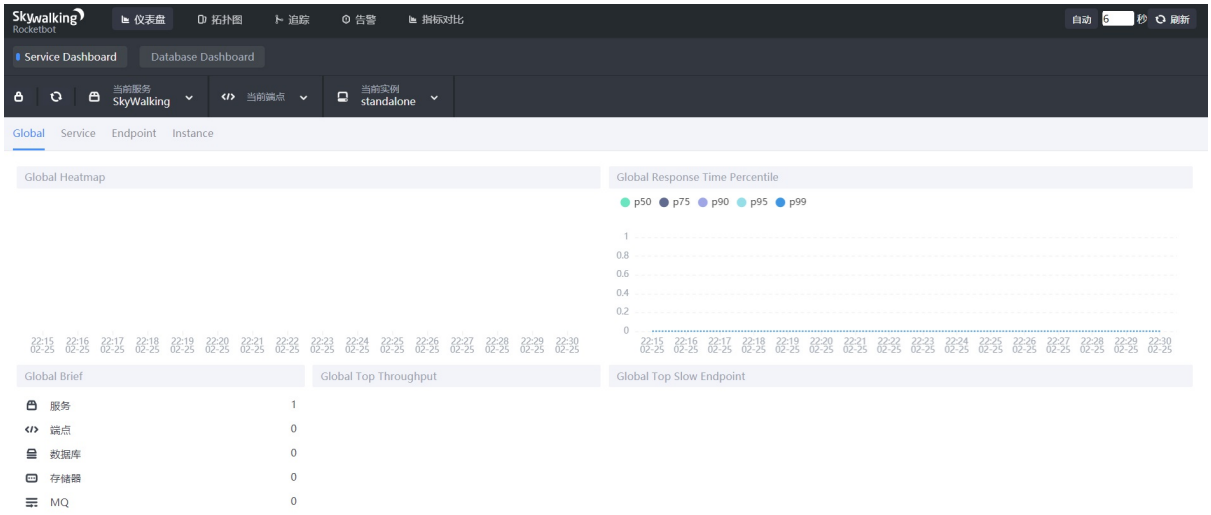
Skywalking主要由三大部分组成：agent、collector、webapp-ui。我们可以使用Skywalking agent探针无侵入地接入到Spring Cloud应用，然后agent探针将应用数据采集到collector收集器。collector中的数据存储在Elasticsearch、H2、MySQL、TiDB中（通过配置文件指定存储方式，默认是H2），通过webapp-ui界面，将这些数据展现给用户。

1.6.1 搭建SkyWalking环境

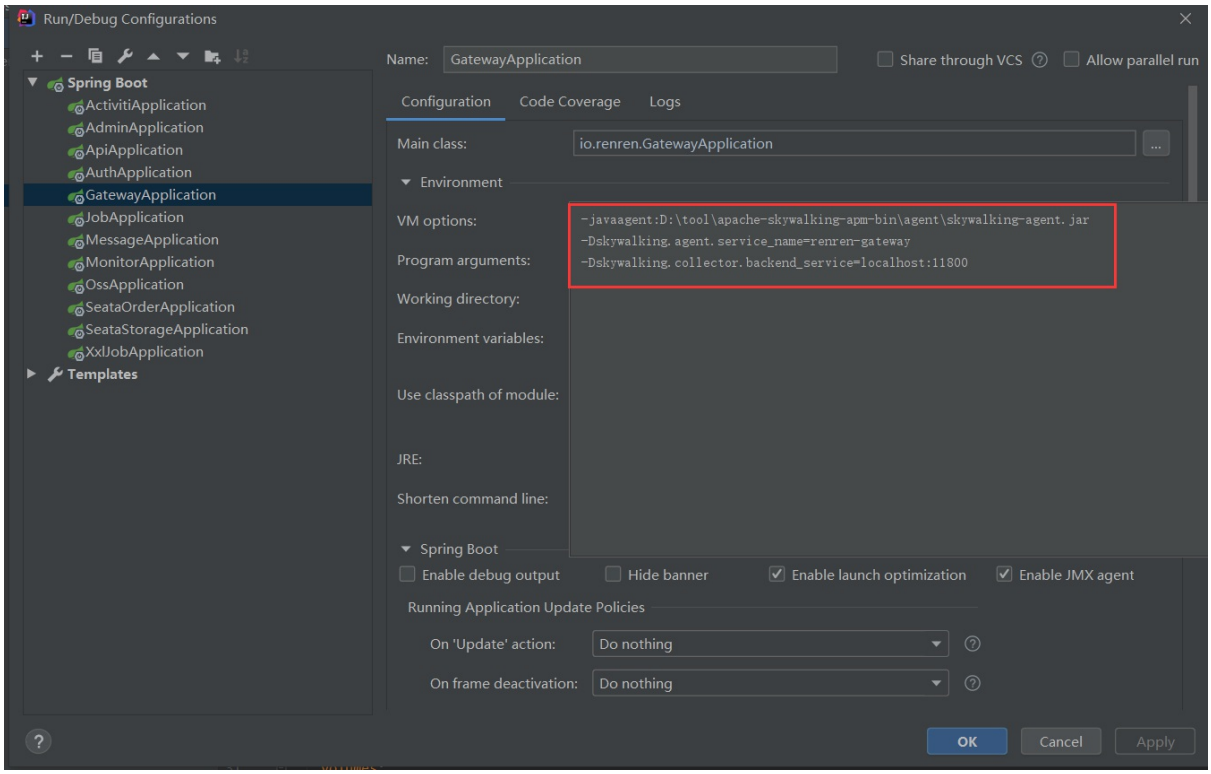
- 下载地址：https://www.apache.org/dyn/closer.cgi/skywalking/6.6.0/apache-skywalking-apm-6.6.0.zip
- webapp-ui默认的端口号是8080，修改成8480，以免造成端口冲突，修改文件：webapp/webapp.yml

```
server:
  port: 8480

collector:
  path: /graphql
  ribbon:
    ReadTimeout: 10000
    # Point to all backend's restHost:restPort, split by ,
    listOfServers: 127.0.0.1:12800
```

- 启动SkyWalking，只需运行【bin/startup.bat】或【bin/startup.sh】即可
- 访问SkyWalking，地址：http://localhost:8480，如下图所示：



- 然后在IDEA的启动配置中，添加Skywalking agent探针配置，以renren-gateway为例（其他服务也是一样的配置，如：renren-admin），进行如下配置：

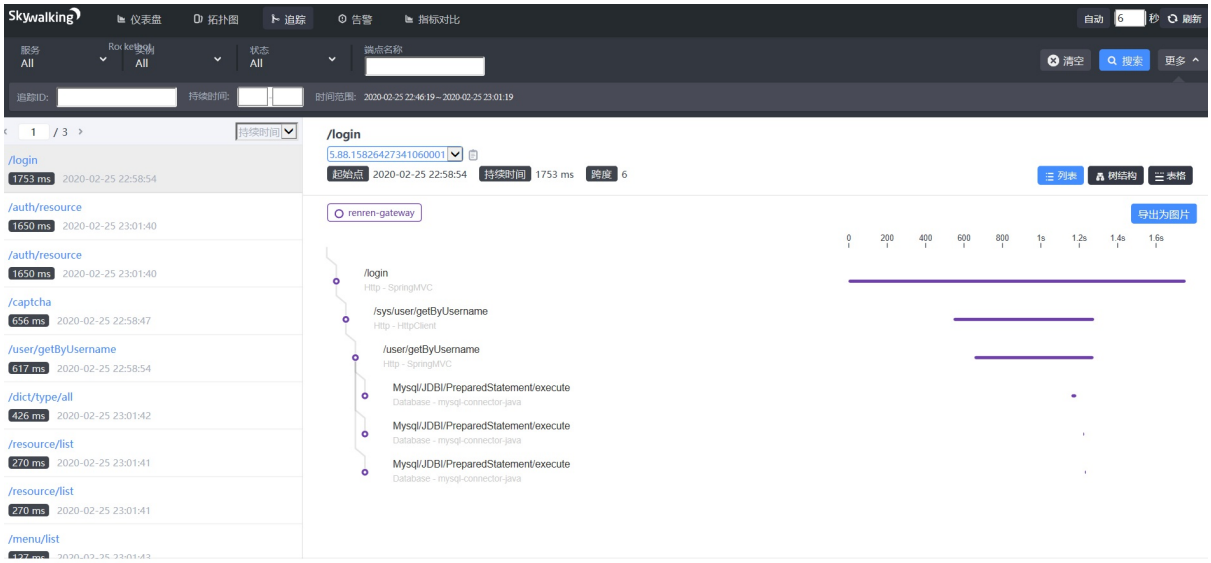


配置内容如下：

```
-javaagent:D:\tool\apache-skywalking-apm-bin\agent\skywalking-agent.jar
-Dskywalking.agent.service_name=renren-gateway
-Dskywalking.collector.backend_service=localhost:11800
```

其中：`D:\tool\apache-skywalking-apm-bin\agent\skywalking-agent.jar`为agent探针的路径，`localhost:11800`为skywalking部署的地址，如果部署在测试机器，则指定测试机器的ip即可。

- 启动微服务，登录后，就可以在skywalking里看链路追踪等信息，如下所示：


Skywalking
仪表盘
拓扑图
追踪
告警
指标对比
自动
6
秒
刷新
服务
All
实例
All
状态
All
端点名称
清空
搜索
更多
追踪ID:
持续时间:
时间范围: 2020-02-25 22:46:19 ~ 2020-02-25 23:01:19
1 / 3
持续时间
/login
1753 ms 2020-02-25 22:58:54
/auth/resource
1650 ms 2020-02-25 23:01:40
/auth/resource
1650 ms 2020-02-25 23:01:40
/captcha
656 ms 2020-02-25 22:58:47
/user/getByUsername
617 ms 2020-02-25 22:58:54
/dict/type/all
426 ms 2020-02-25 23:01:42
/resource/list
270 ms 2020-02-25 23:01:41
/resource/list
270 ms 2020-02-25 23:01:41
/menu/list
/login
5.88.15826427341060001
起始点 2020-02-25 22:58:54
持续时间 1753 ms
跨度 6
列表
树结构
表格
renren-gateway
导出为图片
0
200
400
600
800
1s
1.2s
1.4s
1.6s
/login
Http - SpringMVC
/sys/user/getByUsername
Http - HttpClient
/user/getByUsername
Http - SpringMVC
Mysql/JDBI/PreparedStatement/execute
Database - mysql-connector-java
Mysql/JDBI/PreparedStatement/execute
Database - mysql-connector-java
Mysql/JDBI/PreparedStatement/execute
Database - mysql-connector-java

1.7 Seata使用

Seata 是阿里巴巴一款开源的分布式事务解决方案，致力于提供高性能和简单易用的分布式事务服务。Seata 将为用户提供了 AT、TCC、SAGA 和 XA 事务模式，为用户打造一站式的分布式解决方案。

1.7.1 搭建Seata 环境

- 从GitHub下载Seata 部署文件，下载地址：https://github.com/seata/seata/releases， 下载[seata-server-1.2.0.zip](https://github.com/seata/seata/releases/download/v1.2.0/seata-server-1.2.0.zip)(Windows) 或 seata-server-1.2.0.tar.gz(Linux)
- 修改conf/file.conf配置文件，主要修改数据库信息，如下：

```
  store {
    ## store mode: file、db
    mode = "db"


    ## database store property
    db {
      ## the implement of javax.sql.DataSource, such as DruidDataSource(druid)/BasicDataSourc
e(dbcp) etc.
      datasource = "druid"
      ## mysql/oracle/postgresql/h2/oceanbase etc.
      dbType = "mysql"
      driverClassName = "com.mysql.jdbc.Driver"
      url = "jdbc:mysql://127.0.0.1:3306/seata?useUnicode=true&characterEncoding=UTF-8&useSSL
=false&serverTimezone=Asia/Shanghai"
      user = "renren"
      password = "123456"
      minConn = 5
      maxConn = 30
      globalTable = "global_table"
      branchTable = "branch_table"
      lockTable = "lock_table"
      queryLimit = 100
      maxWait = 5000
    }
  }
```

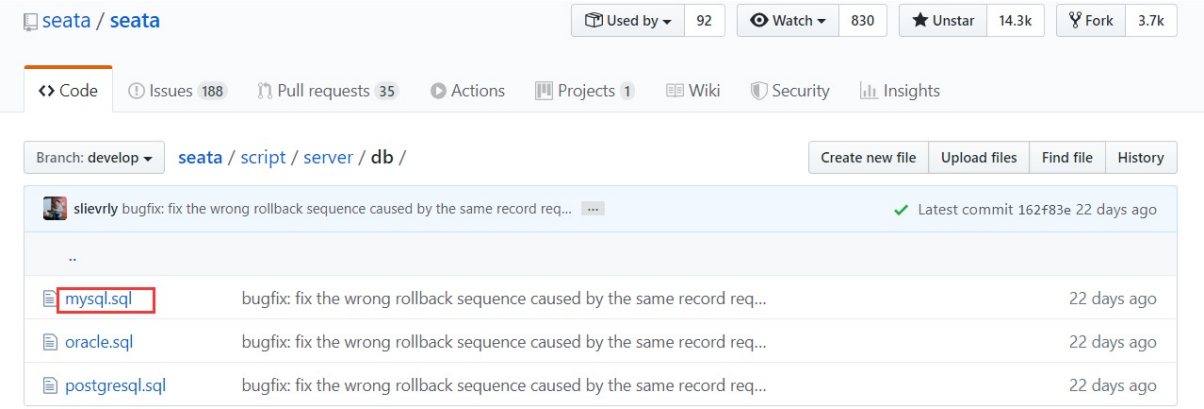
注意：需要把seata/lib/jdbc目录下的mysql驱动mysql-connector-java-8.0.19.jar，拷贝到seata/lib目录里面

- 修改conf/registry.conf，如下：

```
registry {
  # file 、nacos 、eureka、redis、zk、consul、etcd3、sofa
  type = "nacos"

  nacos {
```

```
    serverAddr = "localhost"
    namespace = ""
    cluster = "default"
  }

}

config {
  # file、nacos 、apollo、zk、consul、etcd3
  type = "file"

  file {
    name = "file.conf"
  }
}
```

- 创建seata数据库，并执行对应的SQL脚本，SQL脚本下载地址：https://github.com/seata/seata/blob/1.2.0/script/server/db



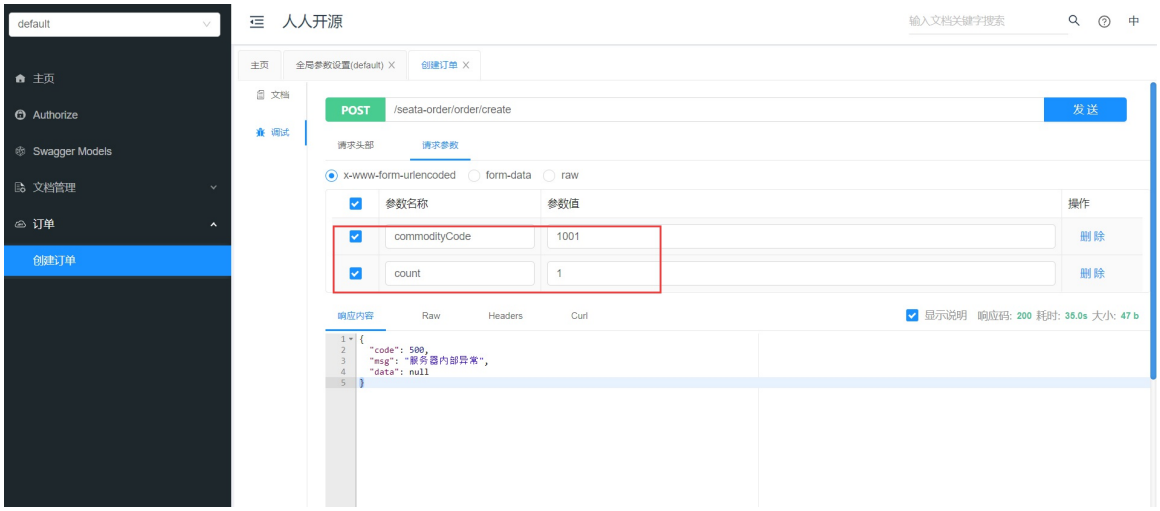
- 启动Seata服务

```
Usage: sh seata-server.sh(for linux and mac) or cmd seata-server.bat(for windows) [option
s]
  Options:
    --host, -h
      The host to bind.
      Default: 0.0.0.0
    --port, -p
      The port to listen.
      Default: 8091
    --storeMode, -m
      log store mode : file、db
      Default: file
    --help

e.g.

sh seata-server.sh
```

```
cmd seata-server.bat
```

- 测试分布式事务

  启动项目工程renren-seata-order、renren-seata-storage，则可以测试分布式事务，访问接口文档地址：http://localhost:8088/seata-order/doc.html



分布式事务测试代码如下：

```
/**
 * 订单
 *
 * @author Mark sunlightcs@gmail.com
 */
@Service
public class OrderServiceImpl extends CrudServiceImpl<OrderDao, OrderEntity, OrderDTO> implem
ents OrderService {
    @Autowired
    private StorageFeignClient storageFeignClient;

    /**
     * 创建订单、减库存，涉及到两个服务
     * @param commodityCode    商品编码
     * @param count            数量
     */
    @Override
    @GlobalTransactional
    @Transactional(rollbackFor = Exception.class)
    public void createOrder(String commodityCode, Integer count) {
        //商品金额
        BigDecimal commodityMoney = new BigDecimal(10);
        //订单金额
        BigDecimal orderMoney = new BigDecimal(count).multiply(commodityMoney);
        OrderEntity order = new OrderEntity()
                .setCommodityCode(commodityCode)
                .setCount(count)
                .setMoney(orderMoney);
```

```
        //减库存
        Result result = storageFeignClient.deduct(commodityCode, count);
        if(!result.success()){
            throw new RenException(result.getMsg());
        }

        //模拟异常
        int i = 1/0;

        //保存订单
        baseDao.insert(order);
    }

    @Override
    public QueryWrapper<OrderEntity> getWrapper(Map<String, Object> params){
        return null;
    }

}
```

1.4 获取帮助

- 官方社区：https://www.renren.io/community
- 如需寻求帮助、项目建议、技术讨论等，请移步到官方社区，我会在第一时间进行解答或回复

第2章 数据库支持

2.1 MySQL数据库支持

2.2 Oracle数据库支持

2.3 SQL Server数据库支持

2.4 PostgreSQL数据库支持

2.1 MySQL数据库支持

1) 修改数据库配置信息，配置信息需要在nacos里修改，如下所示：

```
driver-class-name: com.mysql.cj.jdbc.Driver
url: jdbc:mysql://localhost:3306/renren_cloud?useUnicode=true&characterEncoding=UTF-8&useSSL=
false&serverTimezone=Asia/Shanghai
username: renren
password: 123456
```

2) 执行db/mysql.sql，创建表及初始化数据，再启动项目即可

2.2 Oracle数据库支持

1) 修改数据库配置信息，配置信息需要在nacos里修改，如下所示：

```
#SQLServer
driver-class-name: com.microsoft.sqlserver.jdbc.SQLServerDriver
url: jdbc:sqlserver://localhost:1433;DatabaseName=renren_cloud
username: sa
password: 123456
```

2) 还需打开如下注释

```
validation-query: SELECT 1 FROM DUAL
```

2) 执行db/oracle.sql，创建表及初始化数据，再启动项目即可

2.3 SQL Server数据库支持

1) 修改数据库配置信息，配置信息需要在nacos里修改，如下所示：

```
driver-class-name: com.microsoft.sqlserver.jdbc.SQLServerDriver
url: jdbc:sqlserver://localhost:1433;DatabaseName=renren_cloud
username: sa
password: 123456
```

2) 执行db/sqlserver.sql，创建表及初始化数据，再启动项目即可

2.4 PostgreSQL数据库支持

1) 修改数据库配置信息，配置信息需要在nacos里修改，如下所示：

```
driver-class-name: org.postgresql.Driver
url: jdbc:postgresql://localhost:5432/renren_cloud
username: postgres
password: 123456
```

2) 执行db/postgresql.sql，创建表及初始化数据，再启动项目即可

第3章 多数据源支持

3.1 多数据源配置

3.2 多数据源使用

3.3 源码讲解

3.1 多数据源配置

> 多数据源的应用场景，主要针对跨多个数据库实例的情况，如果是同实例中的多个数据库，则没必要使用多数据源。

```
#下面演示单实例，多数据库的使用情况

select * from db.table;

#其中，db为数据库名，table为数据库表名
```

1) 在需要使用多数据源项目的pom.xml里，引入renren-commons-dynamic-datasource.jar，如下所示：

```
<dependency>
    <groupId>io.renren</groupId>
    <artifactId>renren-commons-dynamic-datasource</artifactId>
    <version>{version}</version>
</dependency>
```

2) 配置多数据源，配置信息需要在apollo里修改，如下所示

```
#多数据源的配置
dynamic:
  datasource:
    slave1:
      driver-class-name: com.microsoft.sqlserver.jdbc.SQLServerDriver
      url: jdbc:sqlserver://192.168.10.10:1433;DatabaseName=renren_cloud
      username: sa
      password: 123456
    slave2:
      driver-class-name: org.postgresql.Driver
      url: jdbc:postgresql://192.168.10.10:5432/renren_cloud
      username: postgres
      password: 123456
```

3.2 多数据源使用

> 多数据源的使用，只需在Service类、方法上添加@DataSource("")注解即可，比如在类上添加了@DataSource("userDB")注解，则表示该Service方法里的所有CURD，都会在 `userDB` 数据源里执行。

1) 多数据源注解使用规则

- 支持在Service类或方法上，添加多数据源的注解@DataSource
- 在Service类上添加了@DataSource注解，则该类下的所有方法，都会使用@DataSource标注的数据源
- 在Service类、方法上都添加了@DataSource注解，则方法上的注解会覆盖Service类上的注解

2) 编写DynamicDataSourceTestService.java，测试多数据源及事务

```
/**
 * Copyright 2018 人人开源 https://www.renren.io
 * <p>
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 * <p>
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 * <p>
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

package io.renren.service;

import io.renren.commons.dynamic.datasource.annotation.DataSource;
import io.renren.dao.SysUserDao;
import io.renren.entity.SysUserEntity;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

/**
 * 测试多数据源
 *
 * @author Mark sunlightcs@gmail.com
 * @since 1.0.0 2018-09-26
 */
@Service
//@DataSource("slave1")  多数据源全局配置
public class DynamicDataSourceTestService {
    @Autowired
    private SysUserDao sysUserDao;
```

```
    @Transactional
    public void updateUser(String id){
        SysUserEntity user = new SysUserEntity();
        user.setId(id);
        user.setMobile("13500000000");
        sysUserDao.updateById(user);
    }

    @DataSource("slave1")
    @Transactional
    public void updateUserBySlave1(String id){
        SysUserEntity user = new SysUserEntity();
        user.setId(id);
        user.setMobile("13500000001");
        sysUserDao.updateById(user);
    }

    @DataSource("slave2")
    @Transactional
    public void updateUserBySlave2(String id){
        SysUserEntity user = new SysUserEntity();
        user.setId(id);
        user.setMobile("13500000002");
        sysUserDao.updateById(user);

        //测试事务
        int i = 1/0;
    }
}
```

3) 运行测试类DynamicDataSourceTest.java，即可测试多数据源及事务是生效的

```
/**
 * Copyright 2018 人人开源 https://www.renren.io
 * <p>
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 * <p>
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 * <p>
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

package io.renren.service;
```

```java
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

/**
 * 多数据源测试
 *
 * @author Mark sunlightcs@gmail.com
 * @since 1.0.0 2018-09-26
 */
@RunWith(SpringRunner.class)
@SpringBootTest
public class DynamicDataSourceTest {
    @Autowired
    private DynamicDataSourceTestService dynamicDataSourceTestService;

    @Test
    public void test(){
        String id = "fdc8e752cc7f41828f3b1d5f7effc7dd";

        dynamicDataSourceTestService.updateUser(id);
        dynamicDataSourceTestService.updateUserBySlave1(id);
        dynamicDataSourceTestService.updateUserBySlave2(id);
    }
}
```

3) 其中，`@DataSource("slave1")`、`@DataSource("slave2")`里的`slave1`、`slave2`值，是在application-dev.xml里配置的，如下所示：

```yaml
dynamic:
  datasource:
    slave1:
      driver-class-name: com.microsoft.sqlserver.jdbc.SQLServerDriver
      url: jdbc:sqlserver://192.168.10.10:1433;DatabaseName=renren_cloud
      username: sa
      password: 123456
    slave2:
      driver-class-name: org.postgresql.Driver
      url: jdbc:postgresql://192.168.10.10:5432/renren_cloud
      username: postgres
      password: 123456
```

3.3 源码讲解

1) 定义多数据源注解类@DataSource，使用多数据源时，只需在Service方法上添加@DataSource注解即可

```
/**
 * Copyright 2018 人人开源 https://www.renren.io
 * <p>
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 * <p>
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 * <p>
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

package io.renren.commons.dynamic.datasource.annotation;

import java.lang.annotation.*;

/**
 * 多数据源注解
 *
 * @author Mark sunlightcs@gmail.com
 * @since 1.0.0
 */
@Target({ElementType.METHOD, ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface DataSource {
    String value() default "";
}
```

2) 定义读取多数据源配置文件的类，如下所示：

```
/**
 * Copyright 2018 人人开源 https://www.renren.io
 * <p>
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 * <p>
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 * <p>
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

package io.renren.commons.dynamic.datasource.properties;

/**
 * 多数据源属性
 *
 * @author Mark sunlightcs@gmail.com
 * @since 1.0.0
 */
public class DataSourceProperties {
    private String driverClassName;
    private String url;
    private String username;
    private String password;

    /**
     * Druid默认参数
     */
    private int initialSize = 2;
    private int maxActive = 10;
    private int minIdle = -1;
    private long maxWait = 60 * 1000L;
    private long timeBetweenEvictionRunsMillis = 60 * 1000L;
    private long minEvictableIdleTimeMillis = 1000L * 60L * 30L;
    private long maxEvictableIdleTimeMillis = 1000L * 60L * 60L * 7;
    private String validationQuery = "select 1";
    private int validationQueryTimeout = -1;
    private boolean testOnBorrow = false;
    private boolean testOnReturn = false;
    private boolean testWhileIdle = true;
    private boolean poolPreparedStatements = false;
    private int maxOpenPreparedStatements = -1;
    private boolean sharePreparedStatements = false;
    private String filters = "stat,wall";

    public String getDriverClassName() {
        return driverClassName;
    }

    public void setDriverClassName(String driverClassName) {
        this.driverClassName = driverClassName;
    }

    public String getUrl() {
        return url;
    }
```

```
public void setUrl(String url) {
    this.url = url;
}

public String getUsername() {
    return username;
}

public void setUsername(String username) {
    this.username = username;
}

public String getPassword() {
    return password;
}

public void setPassword(String password) {
    this.password = password;
}

public int getInitialSize() {
    return initialSize;
}

public void setInitialSize(int initialSize) {
    this.initialSize = initialSize;
}

public int getMaxActive() {
    return maxActive;
}

public void setMaxActive(int maxActive) {
    this.maxActive = maxActive;
}

public int getMinIdle() {
    return minIdle;
}

public void setMinIdle(int minIdle) {
    this.minIdle = minIdle;
}

public long getMaxWait() {
    return maxWait;
}

public void setMaxWait(long maxWait) {
    this.maxWait = maxWait;
}
```

```
public long getTimeBetweenEvictionRunsMillis() {
    return timeBetweenEvictionRunsMillis;
}

public void setTimeBetweenEvictionRunsMillis(long timeBetweenEvictionRunsMillis) {
    this.timeBetweenEvictionRunsMillis = timeBetweenEvictionRunsMillis;
}

public long getMinEvictableIdleTimeMillis() {
    return minEvictableIdleTimeMillis;
}

public void setMinEvictableIdleTimeMillis(long minEvictableIdleTimeMillis) {
    this.minEvictableIdleTimeMillis = minEvictableIdleTimeMillis;
}

public long getMaxEvictableIdleTimeMillis() {
    return maxEvictableIdleTimeMillis;
}

public void setMaxEvictableIdleTimeMillis(long maxEvictableIdleTimeMillis) {
    this.maxEvictableIdleTimeMillis = maxEvictableIdleTimeMillis;
}

public String getValidationQuery() {
    return validationQuery;
}

public void setValidationQuery(String validationQuery) {
    this.validationQuery = validationQuery;
}

public int getValidationQueryTimeout() {
    return validationQueryTimeout;
}

public void setValidationQueryTimeout(int validationQueryTimeout) {
    this.validationQueryTimeout = validationQueryTimeout;
}

public boolean isTestOnBorrow() {
    return testOnBorrow;
}

public void setTestOnBorrow(boolean testOnBorrow) {
    this.testOnBorrow = testOnBorrow;
}

public boolean isTestOnReturn() {
    return testOnReturn;
}
```

```
    public void setTestOnReturn(boolean testOnReturn) {
        this.testOnReturn = testOnReturn;
    }

    public boolean isTestWhileIdle() {
        return testWhileIdle;
    }

    public void setTestWhileIdle(boolean testWhileIdle) {
        this.testWhileIdle = testWhileIdle;
    }

    public boolean isPoolPreparedStatements() {
        return poolPreparedStatements;
    }

    public void setPoolPreparedStatements(boolean poolPreparedStatements) {
        this.poolPreparedStatements = poolPreparedStatements;
    }

    public int getMaxOpenPreparedStatements() {
        return maxOpenPreparedStatements;
    }

    public void setMaxOpenPreparedStatements(int maxOpenPreparedStatements) {
        this.maxOpenPreparedStatements = maxOpenPreparedStatements;
    }

    public boolean isSharePreparedStatements() {
        return sharePreparedStatements;
    }

    public void setSharePreparedStatements(boolean sharePreparedStatements) {
        this.sharePreparedStatements = sharePreparedStatements;
    }

    public String getFilters() {
        return filters;
    }

    public void setFilters(String filters) {
        this.filters = filters;
    }
}
```

```
/**
 * Copyright 2018 人人开源 https://www.renren.io
 * <p>
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
```

```
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 * <p>
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 * <p>
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

package io.renren.commons.dynamic.datasource.properties;

import org.springframework.boot.context.properties.ConfigurationProperties;

import java.util.LinkedHashMap;
import java.util.Map;

/**
 * 多数据源属性
 *
 * @author Mark sunlightcs@gmail.com
 * @since 1.0.0
 */
@ConfigurationProperties(prefix = "dynamic")
public class DynamicDataSourceProperties {
    private Map<String, DataSourceProperties> datasource = new LinkedHashMap<>();

    public Map<String, DataSourceProperties> getDatasource() {
        return datasource;
    }

    public void setDatasource(Map<String, DataSourceProperties> datasource) {
        this.datasource = datasource;
    }
}
```

3) 扩展Spring的AbstractRoutingDataSource抽象类， AbstractRoutingDataSource中的抽象方法determineCurrentLookupKey是实现多数据源的核心，并对该方法进行Override，如下所示：

```
/**
 * Copyright 2018 人人开源 https://www.renren.io
 * <p>
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 * <p>
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
```

```
 * GNU General Public License for more details.
 * <p>
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

package io.renren.commons.dynamic.datasource.config;

import org.springframework.jdbc.datasource.lookup.AbstractRoutingDataSource;

/**
 * 多数据源
 *
 * @author Mark sunlightcs@gmail.com
 * @since 1.0.0
 */
public class DynamicDataSource extends AbstractRoutingDataSource {
    private static final ThreadLocal<String> contextHolder = new ThreadLocal<>();

    @Override
    protected Object determineCurrentLookupKey() {
        return contextHolder.get();
    }

    public static void setDataSource(String dataSource) {
        contextHolder.set(dataSource);
    }

    public static void clearDataSource() {
        contextHolder.remove();
    }

}
```

4) 配置多数据源，如下所示：

```
/**
 * Copyright 2018 人人开源 https://www.renren.io
 * <p>
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 * <p>
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 * <p>
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */
```

```
package io.renren.commons.dynamic.datasource.config;

import com.alibaba.druid.pool.DruidDataSource;
import io.renren.commons.dynamic.datasource.properties.DataSourceProperties;
import io.renren.commons.dynamic.datasource.properties.DynamicDataSourceProperties;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.boot.context.properties.EnableConfigurationProperties;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.util.HashMap;
import java.util.Map;

/**
 * 配置多数据源
 *
 * @author Mark sunlightcs@gmail.com
 * @since 1.0.0
 */
@Configuration
@EnableConfigurationProperties(DynamicDataSourceProperties.class)
public class DynamicDataSourceConfig {
    @Autowired
    private DynamicDataSourceProperties properties;

    @Bean
    @ConfigurationProperties(prefix = "spring.datasource.druid")
    public DataSourceProperties dataSourceProperties() {
        return new DataSourceProperties();
    }

    //因为DynamicDataSource是继承与AbstractRoutingDataSource，而AbstractRoutingDataSource又是继承于AbstractDataSource，AbstractDataSource实现了统一的DataSource接口，所以DynamicDataSource也可以当做DataSource使用
    @Bean
    public DynamicDataSource dynamicDataSource(DataSourceProperties dataSourceProperties) {
        DynamicDataSource dynamicDataSource = new DynamicDataSource();
        dynamicDataSource.setTargetDataSources(getDynamicDataSource());

        //默认数据源
        DruidDataSource defaultDataSource = DynamicDataSourceFactory.buildDruidDataSource(dataSourceProperties);
        dynamicDataSource.setDefaultTargetDataSource(defaultDataSource);

        return dynamicDataSource;
    }

    private Map<Object, Object> getDynamicDataSource(){
        Map<Object, Object> targetDataSources = new HashMap<>();
        properties.getDatasource().forEach((k, v) -> {
```

```
            DruidDataSource druidDataSource = DynamicDataSourceFactory.buildDruidDataSource(v
);
            targetDataSources.put(k, druidDataSource);
        });

        return targetDataSources;
    }

}
```

```
/**
 * Copyright 2018 人人开源 https://www.renren.io
 * <p>
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 * <p>
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 * <p>
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

package io.renren.commons.dynamic.datasource.config;

import com.alibaba.druid.pool.DruidDataSource;
import io.renren.commons.dynamic.datasource.properties.DataSourceProperties;

import java.sql.SQLException;

/**
 * DruidDataSource
 *
 * @author Mark sunlightcs@gmail.com
 * @since 1.0.0
 */
public class DynamicDataSourceFactory {

    public static DruidDataSource buildDruidDataSource(DataSourceProperties properties) {
        DruidDataSource druidDataSource = new DruidDataSource();
        druidDataSource.setDriverClassName(properties.getDriverClassName());
        druidDataSource.setUrl(properties.getUrl());
        druidDataSource.setUsername(properties.getUsername());
        druidDataSource.setPassword(properties.getPassword());

        druidDataSource.setInitialSize(properties.getInitialSize());
        druidDataSource.setMaxActive(properties.getMaxActive());
```

```
        druidDataSource.setMinIdle(properties.getMinIdle());
        druidDataSource.setMaxWait(properties.getMaxWait());
        druidDataSource.setTimeBetweenEvictionRunsMillis(properties.getTimeBetweenEvictionRun
sMillis());
        druidDataSource.setMinEvictableIdleTimeMillis(properties.getMinEvictableIdleTimeMilli
s());
        druidDataSource.setMaxEvictableIdleTimeMillis(properties.getMaxEvictableIdleTimeMilli
s());
        druidDataSource.setValidationQuery(properties.getValidationQuery());
        druidDataSource.setValidationQueryTimeout(properties.getValidationQueryTimeout());
        druidDataSource.setTestOnBorrow(properties.isTestOnBorrow());
        druidDataSource.setTestOnReturn(properties.isTestOnReturn());
        druidDataSource.setPoolPreparedStatements(properties.isPoolPreparedStatements());
        druidDataSource.setMaxOpenPreparedStatements(properties.getMaxOpenPreparedStatements(
));
        druidDataSource.setSharePreparedStatements(properties.isSharePreparedStatements());

        try {
            druidDataSource.setFilters(properties.getFilters());
            druidDataSource.init();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return druidDataSource;
    }
}
```

5) @DataSource注解的切面处理类，动态切换的核心代码

```
/**
 * Copyright 2018 人人开源 https://www.renren.io
 * <p>
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 * <p>
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 * <p>
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

package io.renren.commons.dynamic.datasource.aspect;

import io.renren.commons.dynamic.datasource.annotation.DataSource;
import io.renren.commons.dynamic.datasource.config.DynamicDataSource;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
```

```java
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Pointcut;
import org.aspectj.lang.reflect.MethodSignature;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.core.Ordered;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;

import java.lang.reflect.Method;

/**
 * 多数据源，切面处理类
 *
 * @author Mark sunlightcs@gmail.com
 * @since 1.0.0
 */
@Aspect
@Component
@Order(Ordered.HIGHEST_PRECEDENCE)
public class DataSourceAspect  {
    protected Logger logger = LoggerFactory.getLogger(getClass());

    @Pointcut("@annotation(io.renren.commons.dynamic.datasource.annotation.DataSource) " +
            "|| @within(io.renren.commons.dynamic.datasource.annotation.DataSource)")
    public void dataSourcePointCut() {

    }

    @Around("dataSourcePointCut()")
    public Object around(ProceedingJoinPoint point) throws Throwable {
        MethodSignature signature = (MethodSignature) point.getSignature();
        Class targetClass = point.getTarget().getClass();
        Method method = signature.getMethod();

        DataSource targetDataSource = (DataSource)targetClass.getAnnotation(DataSource.class)
;
        DataSource methodDataSource = method.getAnnotation(DataSource.class);
        if(targetDataSource != null || methodDataSource != null){
            String value;
            if(methodDataSource != null){
                value = methodDataSource.value();
            }else {
                value = targetDataSource.value();
            }

            DynamicDataSource.setDataSource(value);
            logger.debug("set datasource is {}", value);
        }

        try {
            return point.proceed();
```

```
        } finally {
            DynamicDataSource.clearDataSource();
            logger.debug("clean datasource");
        }
    }
}
```

第**4**章 基础知识讲解

4.1 Spring MVC使用

4.2 Swagger使用

4.3 Mybatis-plus使用

4.4 Hibernate Validator使用

4.5 Feign使用

4.1 Spring MVC使用

> 对Spring MVC不太熟悉的，需要理解Spring MVC常用的注解，也方便日后排查问题，常用的注解如下所示：

4.1.1 @Controller注解

@Controller注解表明了一个类是作为控制器的角色而存在的。Spring不要求你去继承任何控制器基类，也不要求你去实现Servlet的那套API。当然，如果你需要的话也可以去使用任何与Servlet相关的特性。

```
@Controller
public class UserController {
    // ...
}
```

4.1.2 @RequestMapping注解

你可以使用@RequestMapping注解来将请求URL，如/user等，映射到整个类上或某个特定的处理器方法上。一般来说，类级别的注解负责将一个特定（或符合某种模式）的请求路径映射到一个控制器上，同时通过方法级别的注解来细化映射，即根据特定的HTTP请求方法（GET、POST方法等）、HTTP请求中是否携带特定参数等条件，将请求映射到匹配的方法上。

```
@Controller
public class UserController {

    @RequestMapping("/user")
    public String user() {
        return "user";
    }

}
```

以上代码没有指定请求必须是GET方法还是PUT/POST或其他方法，@RequestMapping注解默认会映射所有的HTTP请求方法。如果仅想接收某种请求方法，请在注解中指定之@RequestMapping(path = "/user", method = RequestMethod.GET)以缩小范围。

4.1.3 @PathVariable注解

在Spring MVC中你可以在方法参数上使用@PathVariable注解，将其与URI模板中的参数绑定起来，如下所示：

```
@RequestMapping(path="/user/{userId}", method=RequestMethod.GET)
public String userCenter(@PathVariable("userId") String userId, Model model) {

    UserDTO user = userService.get(userId);
    model.addAttribute("user", user);
```

```
    return "userCenter";

}
```

URI模板"/user/{userId}"指定了一个变量名为userId。当控制器处理这个请求的时候，userId的值就会被URI模板中对应部分的值所填充。比如说，如果请求的URI是/userId/1，此时变量userId的值就是1。

4.1.4 @GetMapping注解

@GetMapping是一个组合注解，是@RequestMapping(method = RequestMethod.GET)的缩写。该注解将HTTP GET映射到特定的处理方法上。可以使用@GetMapping("/user")来代替@RequestMapping(path="/user",method= RequestMethod.GET)。还有@PostMapping、@PutMapping、@DeleteMapping等同理。

4.1.5 @RequestBody注解

该注解用于读取Request请求的body部分数据，使用系统默认配置的HttpMessageConverter进行解析，然后把相应的数据绑定到要返回的对象上，再把HttpMessageConverter返回的对象数据绑定到Controller中方法的参数上。

```
@Controller
public class UserController {

    @GetMapping("/user")
    public String user(@RequestBody User user) {
        //...
        return "user";
    }

}
```

4.1.6 @ResponseBody注解

该注解用于将Controller的方法返回的对象，通过适当的HttpMessageConverter转换为指定格式后，写入到Response对象的body数据区。比如获取JSON数据，加上@ResponseBody后，会直接返回JSON数据，而不会被解析为视图。

```
@Controller
public class UserController {

    @ResponseBody
    @GetMapping("/user/{userId}")
    public User info(@PathVariable("userId") String userId) {
        UserDTO user = userService.get(userId);
        return user;
    }

}
```

4.1.7 @RestController注解

@RestController是一个组合注解，即@Controller + @ResponseBody的组合注解，请求完后，会返回JSON数据。

4.2 Swagger使用

> Swagger是一个根据Swagger注解，即可生成接口文档的服务。

4.2.1 搭建Swagger环境

- 在pom.xml文件中添加swagger相关依赖，如下所示：

```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>${springfox-version}</version>
</dependency>
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>${springfox-version}</version>
</dependency>
```

- 编写Swagger的Configuration配置文件，如下所示：

```
import io.swagger.annotations.ApiOperation;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import springfox.documentation.builders.ApiInfoBuilder;
import springfox.documentation.builders.PathSelectors;
import springfox.documentation.builders.RequestHandlerSelectors;
import springfox.documentation.service.ApiInfo;
import springfox.documentation.service.ApiKey;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;
import springfox.documentation.swagger2.annotations.EnableSwagger2;

import java.util.List;

import static com.google.common.collect.Lists.newArrayList;

@Configuration
@EnableSwagger2
public class SwaggerConfig implements WebMvcConfigurer {

    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler("swagger-ui.html").addResourceLocations("classpath:/META-
INF/resources/");
        registry.addResourceHandler("/webjars/**").addResourceLocations("classpath:/META-INF/
```

```
resources/webjars/");
    }

    @Bean
    public Docket createRestApi() {
        return new Docket(DocumentationType.SWAGGER_2)
            .apiInfo(apiInfo())
            .select()
            //加了ApiOperation注解的类，才生成接口文档
            .apis(RequestHandlerSelectors.withMethodAnnotation(ApiOperation.class))
            //io.renren.controller包下的类，才生成接口文档
            //.apis(RequestHandlerSelectors.basePackage("io.renren.controller"))
            .paths(PathSelectors.any())
            .build()
            .directModelSubstitute(java.util.Date.class, String.class);
    }

    private ApiInfo apiInfo() {
        return new ApiInfoBuilder()
            .title("人人开源")
            .description("人人开源接口文档")
            .termsOfServiceUrl("https://www.renren.io/community")
            .version("1.0.0")
            .build();
    }

}
```

4.2.2 Swagger常用注解

- @Api注解用在类上，说明该类的作用。可以标记一个Controller类做为swagger文档资源，如下所示：

```
@Api(tags="用户管理")
@RestController
public class UserController {

}
```

- @ApiOperation注解用在方法上，说明该方法的作用，如下所示：

```
@Api(tags="用户管理")
@RestController
public class UserController {

    @GetMapping("/user/list")
    @ApiOperation("列表")
    public List<UserDTO> list(){
        List<UserDTO> list = userService.list();
        return list;
    }
```

```
}
```

- @ApiParam注解用在方法参数上，如下所示：

```
@Api(tags="用户管理")
@RestController
public class UserController {

    @GetMapping("/user/list")
    @ApiOperation("列表")
    public List<UserDTO> list(@ApiParam(value= "用户名", required = true)  String username){
        List<UserDTO> list = userService.list();
        return list;
    }

}
```

- @ApiImplicitParams注解用在方法上，主要用于一组参数说明
- @ApiImplicitParam注解用在@ApiImplicitParams注解中，指定一个请求参数的信息，如下所示：

```
    @GetMapping("page")
    @ApiOperation("分页")
    @ApiImplicitParams({
        @ApiImplicitParam(name = "page", value = "当前页码，从1开始", paramType = "query", requ
ired = true, dataType="int") ,
        @ApiImplicitParam(name = "limit", value = "每页显示记录数", paramType = "query",requir
ed = true, dataType="int") ,
        @ApiImplicitParam(name = "order_field", value = "排序字段", paramType = "query", dataT
ype="String") ,
        @ApiImplicitParam(name = "order", value = "排序方式，可选值(asc、desc)", paramType = "q
uery", dataType="String") ,
        @ApiImplicitParam(name = "username", value = "用户名", paramType = "query", dataType="
String")
    })
    public Result<PageData<SysUserDTO>> page(@ApiIgnore @RequestParam Map<String, Object> par
ams){
        PageData<SysUserDTO> page = sysUserService.page(params);

        return new Result<PageData<SysUserDTO>>().ok(page);
    }
```

- @ApiIgnore注解，可用于类、方法或参数上，表示生成Swagger接口文档时，忽略类、方法或参数。

4.3 Mybatis-plus使用

- 在项目的pom.xml里引入依赖，如下所示：

```
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-boot-starter</artifactId>
    <version>${mybatisplus.version}</version>
</dependency>
```

- 在yml配置文件里，配置mybatis-plus，如下所示：

```
mybatis-plus:
  mapper-locations: classpath:/mapper/**/*.xml
  #实体扫描，多个package用逗号或者分号分隔
  typeAliasesPackage: io.renren.entity
  global-config:
    #数据库相关配置
    db-config:
      #主键类型  AUTO:"数据库ID自增", INPUT:"用户输入ID", ID_WORKER:"全局唯一ID (数字类型唯一ID)"
, UUID:"全局唯一ID UUID";
      id-type: UUID
      #字段策略 IGNORED:"忽略判断",NOT_NULL:"非 NULL 判断"),NOT_EMPTY:"非空判断"
      field-strategy: NOT_NULL
      #驼峰下划线转换
      column-underline: true
      db-type: mysql
    banner: false
  #原生配置
  configuration:
    map-underscore-to-camel-case: true
    cache-enabled: false
    call-setters-on-nulls: true
```

4.4 Hibernate Validator使用

> 官网文档：http://docs.jboss.org/hibernate/validator/6.0/reference/en-US/html_single/

- 定义工具类ValidatorUtils，且支持国际化，用于效验用户提交的数据，如下所示：

```
public class ValidatorUtils {
  private static Validator validator;

  static {
      validator = Validation.byDefaultProvider().configure().messageInterpolator(
          new ResourceBundleMessageInterpolator(new MessageSourceResourceBundleLocator(ge
tMessageSource())))
          .buildValidatorFactory().getValidator();
  }

  private static ResourceBundleMessageSource getMessageSource() {
      ResourceBundleMessageSource bundleMessageSource = new ResourceBundleMessageSource()
;
      bundleMessageSource.setDefaultEncoding("UTF-8");
      bundleMessageSource.setBasenames("i18n/validation", "i18n/validation_common");
      return bundleMessageSource;
  }

  /**
   * 校验对象
   * @param object        待校验对象
   * @param groups        待校验的组
   * @throws RenException  校验不通过，则报RenException异常
   */
  public static void validateEntity(Object object, Class<?>... groups)
          throws RenException {
      Set<ConstraintViolation<Object>> constraintViolations = validator.validate(object,
groups);
      if (!constraintViolations.isEmpty()) {
          ConstraintViolation<Object> constraint = constraintViolations.iterator().next()
;
          throw new RenException(constraint.getMessage());
      }
  }
}
```

- 使用方式如下所示：

```
public class SysUserDTO implements Serializable {
    private static final long serialVersionUID = 1L;

    @Null(message="{id.null}", groups = AddGroup.class)
    @NotBlank(message="{id.require}", groups = UpdateGroup.class)
```

```
    private String id;

    @NotBlank(message="{sysuser.username.require}", groups = DefaultGroup.class)
    private String username;

    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    @NotBlank(message="{sysuser.password.require}", groups = AddGroup.class)
    private String password;

    @NotBlank(message="{sysuser.realname.require}", groups = DefaultGroup.class)
    private String realName;

    private String headUrl;

    @Range(min=0, max=2, message = "{sysuser.gender.range}", groups = DefaultGroup.class)
    private Integer gender;

    @NotBlank(message="{sysuser.email.require}", groups = DefaultGroup.class)
    @Email(message="{sysuser.email.error}", groups = DefaultGroup.class)
    private String email;

    @NotBlank(message="{sysuser.mobile.require}", groups = DefaultGroup.class)
    private String mobile;

    @NotBlank(message="{sysuser.deptId.require}", groups = DefaultGroup.class)
    private String deptId;

    @Range(min=0, max=1, message = "{sysuser.superadmin.range}", groups = DefaultGroup.class)
    private Integer superAdmin;

    @Range(min=0, max=1, message = "{sysuser.status.range}", groups = DefaultGroup.class)
    private Integer status;

    private String remark;

    private Date createDate;

}
```

```
@PostMapping
public Result save(@RequestBody SysUserDTO dto){
    //效验数据
    ValidatorUtils.validateEntity(dto, AddGroup.class, DefaultGroup.class);

    sysUserService.save(dto);

    return new Result();
}

@PutMapping
public Result update(@RequestBody SysUserDTO dto){
```

```
    //效验数据
    ValidatorUtils.validateEntity(dto, UpdateGroup.class, DefaultGroup.class);

    sysUserService.update(dto);

    return new Result();
}
```

通过分析上面的代码，我们来理解Hibernate Validator校验框架的使用。 其中被DefaultGroup.class标识的属性，表示保存或修改用户时，都会效验； 而被AddGroup.class标识的属性，表示只在保存用户时，才会效验属性，也就是说，修改用户时，允许为空。

在需要效验数据的位置，添加代码ValidatorUtils.validateEntity(dto, UpdateGroup.class, DefaultGroup.class)即可，表示效验dto对象中，被UpdateGroup.class、DefaultGroup.class标识的字段，其他字段不效验。

- 效验提示支持国际化，如需新增其他语言，只需添加对应国际化文件即可，如下所示：

```
#validation_common_zh_CN.properties
sysuser.username.require=用户名不能为空
sysuser.password.require=密码不能为空
sysuser.realname.require=姓名不能为空
sysuser.gender.range=性别取值范围0~2
sysuser.email.require=邮箱不能为空
sysuser.email.error=邮箱格式不正确
sysuser.mobile.require=手机号不能为空
sysuser.deptId.require=部门不能为空
sysuser.superadmin.range=超级管理员取值范围0~1
sysuser.status.range=状态取值范围0~1
sysuser.captcha.require=验证码不能为空
sysuser.uuid.require=唯一标识不能为空
```

```
#validation_common_en_US.properties
sysuser.username.require=The username cannot be empty
sysuser.password.require=The password cannot be empty
sysuser.realname.require=The realname cannot be empty
sysuser.gender.range=Gender ranges from 0 to 2
sysuser.email.require=Mailbox cannot be empty
sysuser.email.error=Incorrect email format
sysuser.mobile.require=The phone number cannot be empty
sysuser.deptId.require=Departments cannot be empty
sysuser.superadmin.range=Super administrator values range from 0 to 1
sysuser.status.range=State ranges from 0 to 1
sysuser.captcha.require=The captcha cannot be empty
sysuser.uuid.require=The unique identifier cannot be empty
```

4.5 Feign使用

- 在项目的pom.xml里，添加Feign依赖，如下所示：

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
</dependency>
```

- 修改yml配置文件，如下所示：

```
feign:
  hystrix:
    enabled: true  #开启hystrix

hystrix:
  command:
    default:
      execution:
        isolation:
          thread:
            timeoutInMilliseconds: 60000 #缺省为1000ms

ribbon:
  ReadTimeout: 300000
  ConnectTimeout: 300000
```

- 项目启动类上，添加 `@EnableFeignClients` 注解，如下所示：

```
@SpringBootApplication
@EnableDiscoveryClient
@EnableFeignClients
public class AdminApplication {

    public static void main(String[] args) {
        SpringApplication.run(AdminApplication.class, args);
    }

}
```

- 完成上面操作，我们就可以使用Feign，如下所示：

```
/**
 * 用户接口
```

```
 *
 * @author Mark sunlightcs@gmail.com
 * @since 1.0.0
 */
@FeignClient(name = ServiceConstant.RENREN_ADMIN_SERVER, fallback = UserFeignClientFallback.c
lass)
public interface UserFeignClient {

    /**
     * 根据用户ID，获取用户信息
     */
    @GetMapping("sys/user/getById")
    Result<UserDetail> getById(@RequestParam("id") String id);

    /**
     * 根据用户名，获取用户信息
     * @param username  用户名
     */
    @GetMapping("sys/user/getByUsername")
    Result<UserDetail> getByUsername(@RequestParam("username") String username);

}

/**
 * 用户接口 Fallback
 *
 * @author Mark sunlightcs@gmail.com
 * @since 1.0.0
 */
@Component
public class UserFeignClientFallback implements UserFeignClient {

    @Override
    public Result<UserDetail> getById(String id) {
        return new Result<>();
    }

    @Override
    public Result<UserDetail> getByUsername(String username) {
        return new Result<>();
    }
}
```

第5章 工作流模块

> 工作流使用介绍。

5.1 工作流模块

5.1 工作流模块

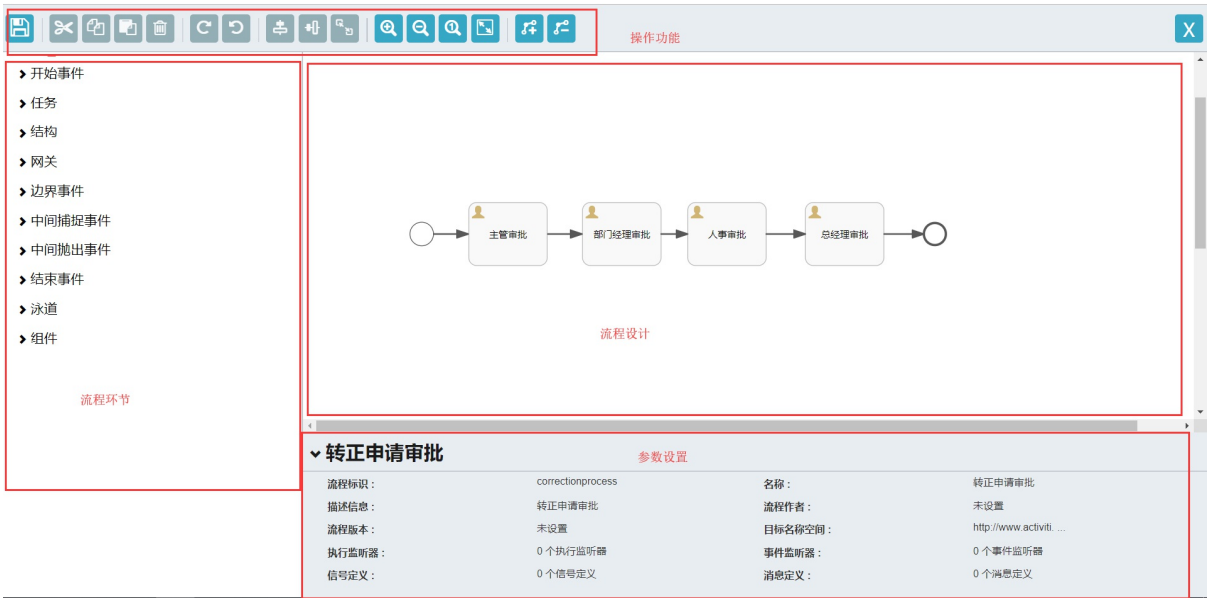
> activiti本身也提供了许多的API，但由于activiti的API比较碎，而且有些接口并不是很友好的满足中国流程审批。因此在activiti的基础上做了一层封装，使其能满足中国的审批流程。

功能介绍

activiti本身也提供了许多的API，但由于activiti的API比较碎，而且有些接口并不是很友好的满足中国流程审批。因此在activiti的基础上做了一层封装，使其能满足中国的审批流程。
使用activiti工作流引擎，大致可以分配5个部分。主要有：模型管理、流程定义管理、实例管理、任务管理。
模型管理：流程设计、部署流程版本、模型维护。
流程定义：管理系统中所有的流程。在模型管理中每部署一次流程，则新增一个版本的流程定义。新增版本主要为了避免和运行中的流程起冲突。
实例管理：管理所有运行中的流程。
任务管理：任务管理是可以细分的。包括：我的待办、我的已办、我的申请、我的委托以及待签收任务等。
系统集成activiti，并开发出以上功能的API接口。在使用中，直接调用这些API接口变可实现功能。具体功能API可参考swagger的API文档。
基于后台对activiti的API的封装。前端做了以下几个工作：1、开发模型管理、流程管理、运行中的流程、发起流程、我的待办、我的申请、已办任务以及代签收任务的功能；2、和业务结合的工作流相关功能也封装成组件，分别是：启动流程（ren-process-start）、处理任务（ren-process-start）、处理详情（ren-process-detail）以及综合组件（ren-process-multiple）。其中综合组件是综合了启动流程、处理任务以及处理详情的一个组件。业务和工作流结合的方式是普通表单，开发者设计业务表单，开发者定义业务表单后，结合现有activiti的封装可完成工作流相关的功能开发。具体请参考请假管理和转正申请两个示例。

流程设计器常用功能说明

在模型管理中，点击【新增】按钮。填写模型的基本信息并保存。保存信息成功后，点击【在线设计】进入流程设计界面。



流程设计页面分为：功能按钮区、流程环节区、流程设计区以及参数设置区。

功能按钮区从左到右分别是：保存、剪切、复制、黏贴、删除、重做、撤销、垂直对齐、水平对齐、相同大小、放大、缩小、缩放到实际大小、缩放到适应大小、给线条增加一个节点、给线条删除一个节点、关闭。

流程环节区：存放的是activiti流程引擎的环节列表信息。可拖放至流程设计区。

流程设计区：在这个区域设计流程，编制流程图。

参数设置区：流程参数设置、环节参数设置区域。

流程参数设置分为流程参数设置和环节参数设置，针对不同的环节参数又有不同的参数设置。下面细说几种常用的参数。

1、流程参数：

打开流程设计器时，参数设置区默认打开的是流程参数设置。常用的几个配置参数是流程标识、名称以及执行监听器这几个。流程标识须保证唯一，业务需要通过这个参数进行关联。执行监听器分为：启动、结束以及连线监听三种。分别对应流程启动时、流程结束时以及连接线连接的三个事件。

2、连线参数：

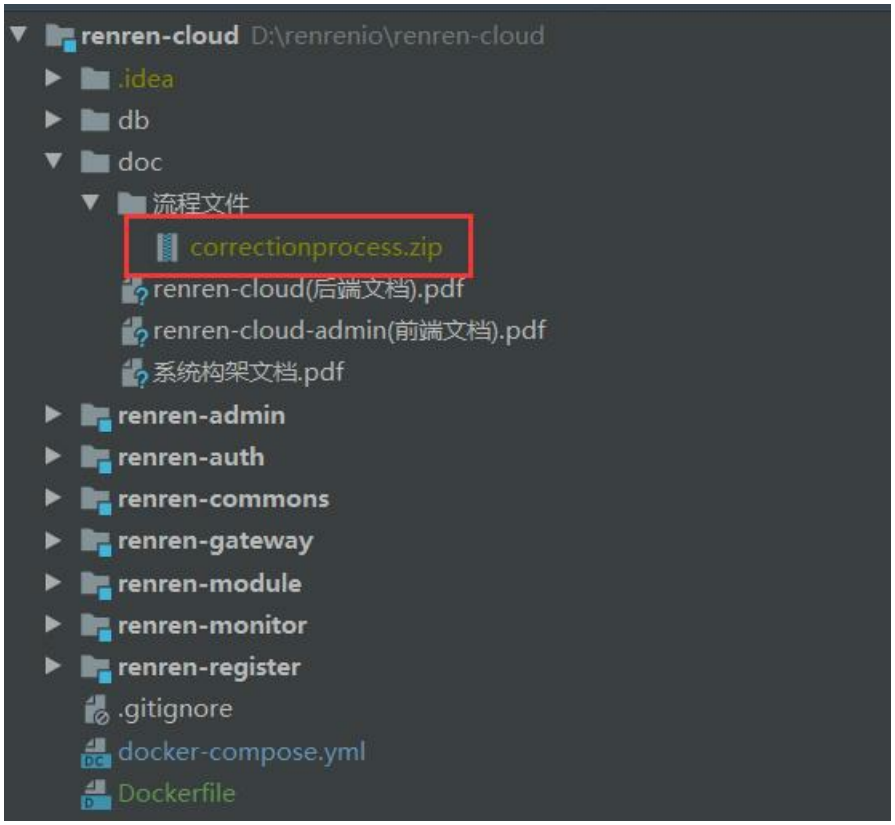
连线参数常用参数为流条件和执行监听器。流条件用于排他网关或包容网关时设置的条件参数。

3、用户任务参数：

用户任务主要设置执行监听器和分配用户两个功能。执行监听器为任务创建、结束以及连线监听。分配用户为分配受理人、候选人以及角色三种。

转正申请示例说明

在使用转正申请示例前，请先在模型管理导入流程配置信息。配置信息存放在工程的“doc/流程文件”目录下，请参考图下图：



流程定义的KEY是由开发人员自定义的，也必须在流程设计时定义好。在具体业务发起流程时需要使用流程KEY启动流程。因此，转正申请功能在使用前定义好流程定义KEY。示例中流程定义的KEY为：correctionprocess。

系统支持的业务和流程结合的方式为:

1、流程定义设计。系统提供两种方式流程设计：一、流程管理中“部署流程文件”；二、模型管理中在线设计流程，并部署；

2、业务表单设计。首先设计表结构，其次使用人人开源的代码生成工具生成增删改查代码。业务表单中可实例ID字段用于存放流程的实例ID；

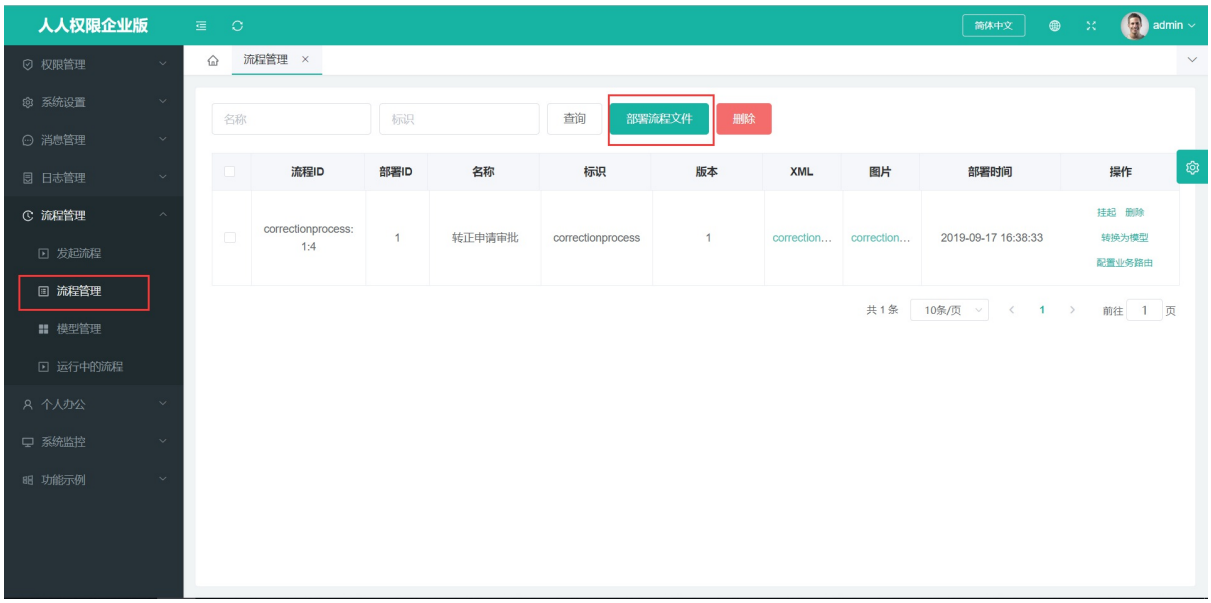
3、编写流程需要的业务表单页面；

4、流程管理中配置流程业务路由信息；

5、使用启动流程的组件启动流程；
6、使用查看流程详情组件查看流程处理情况。
下面就转正申请审核流程的开发过程进行详细说明。

一、流程定义设计

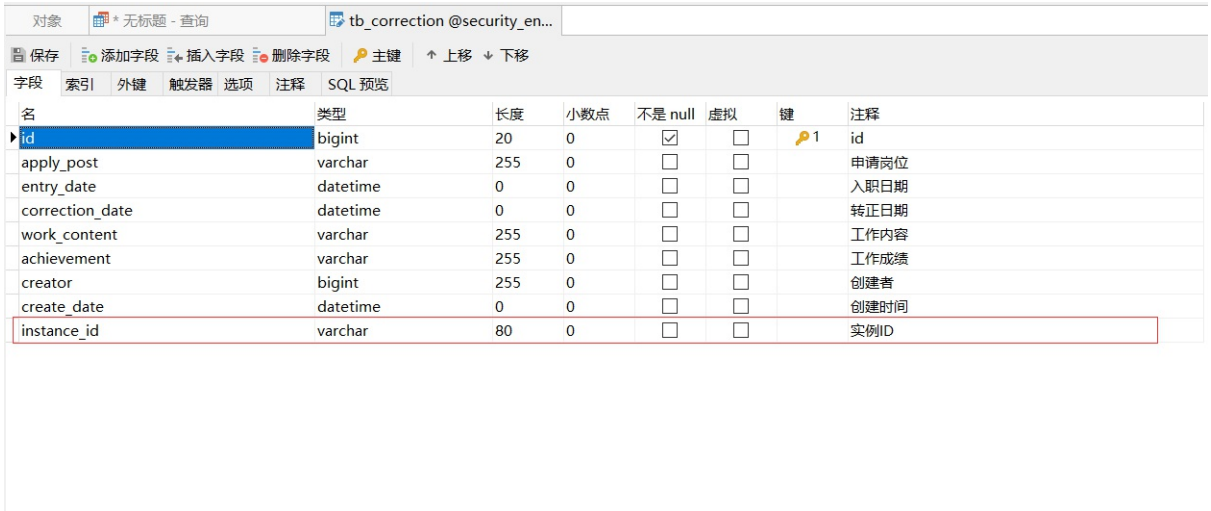
转正申请流程设计系统已经写成示例，并将流程文件打包成ZIP包存放在工程中。因此，使用转正申请流程DEMO前先在流程管理功能中导入流程定义信息。 如图：



将correctionprocess.zip包上传之后，流程设计工作就完成了。可以通过“转换为模型”功能将流程定义转成模型。转成模型后可以对流程进行再次设计，且发布成新版本的流程。

二、业务表单设计

1、表结构设计
如转正申请表结构：



名	类型	长度	小数点	不是 null	虚拟	键	注释
id	bigint	20	0	☑	☐	1	id
apply_post	varchar	255	0	☐	☐		申请岗位
entry_date	datetime	0	0	☐	☐		入职日期
correction_date	datetime	0	0	☐	☐		转正日期
work_content	varchar	255	0	☐	☐		工作内容
achievement	varchar	255	0	☐	☐		工作成绩
creator	bigint	255	0	☐	☐		创建者
create_date	datetime	0	0	☐	☐		创建时间
instance_id	varchar	80	0	☐	☐		实例ID

2、使用代码生成工具生成业务表单维护代码；

三、流程业务表单

由于流程过程处理时，需要显示业务表单信息。因此需要业务额外开发一个业务流程表单。如转正申请的流程业务表单为：correction-process.vue

```vue
<!--  流程业务表单  -->
<template>
  <el-card shadow="never" class="aui-card--fill">
    <el-form :model="dataForm" :rules="dataRule" ref="dataForm" @keyup.enter.native="dataFormSubmitHandle()" :label-width="$i18n.locale =
        <el-form-item :label="$t('correction.post')" prop="applyPost">
        <el-input v-model="dataForm.applyPost" :disabled="fieldDisabled" :placeholder="$t('correction.post')"></el-input>
      </el-form-item>
        <el-form-item label="入职日期" prop="entryDate">
        <el-date-picker v-model="dataForm.entryDate" :disabled="fieldDisabled" value-format="yyyy-MM-dd" :placeholder="$t('correction.e
      </el-form-item>
        <el-form-item :label="$t('correction.correctionDate')" prop="correctionDate">
        <el-date-picker v-model="dataForm.correctionDate" :disabled="fieldDisabled" value-format="yyyy-MM-dd" :placeholder="$t('correct
      </el-form-item>
        <el-form-item :label="$t('correction.workContent')" prop="workContent">
        <el-input v-model="dataForm.workContent" :disabled="fieldDisabled" :placeholder="$t('correction.workContent')"></el-input>
      </el-form-item>
        <el-form-item :label="$t('correction.achievement')" prop="achievement">
        <el-input v-model="dataForm.achievement" :disabled="fieldDisabled" :placeholder="$t('correction.achievement')"></el-input>
      </el-form-item>
      </el-form>
      <!--  流程综合组件  -->
      <ren-process-multiple v-if="processVisible" saveFormUrl="/act/demo/correction" dataFormName="dataForm" ref="renProcessMultiple" ></
  </el-card>
</template>

<script>
// 引入工作流公共方法
import processModule from '@/mixins/process-module'
export default {
  // 注入公共方法
  mixins: [processModule],
  data () {
    return {
      visible: false,
      // 表单属性是否可编辑
      fieldDisabled: false,
      dataForm: {
        id: '',
        applyPost: '',
        entryDate: '',
        correctionDate: '',
        workContent: '',
        achievement: '',
        creator: '',
        createDate: ''
      }
    }
  },
```

```
created () {
  // 将业务KEY赋值给表单
  this.dataForm.id = this.$route.params.businessKey
  this.init()
  // 流程回调
  var callbacks = {
    startProcessSuccessCallback: this.closeCurrentTab,
    startProcessErrorCallback: this.startProcessErrorCallback,
    taskHandleSuccessCallback: this.closeCurrentTab,
    taskHandleErrorCallback: this.taskHandleErrorCallback,
    formSaveSuccessCallback: null,
    formSaveErrorCallback: null
  }
  // 初始化综合组件
  this.initProcessMultiple(callbacks)
},
computed: {
  dataRule () {
    return {
      applyPost: [
        { required: true, message: this.$t('validate.required'), trigger: 'blur' }
      ],
      entryDate: [
        { required: true, message: this.$t('validate.required'), trigger: 'blur' }
      ],
      correctionDate: [
        { required: true, message: this.$t('validate.required'), trigger: 'blur' }
      ],
      workContent: [
        { required: true, message: this.$t('validate.required'), trigger: 'blur' }
      ],
      achievement: [
        { required: true, message: this.$t('validate.required'), trigger: 'blur' }
      ],
      createTime: [
        { required: true, message: this.$t('validate.required'), trigger: 'blur' }
      ]
    }
  }
},
methods: {
  init () {
    this.visible = true
    this.$nextTick(() => {
      this.$refs['dataForm'].resetFields()
      if (this.dataForm.id) {
        // 如业务KEY已存在，不允许编辑
        this.fieldDisabled = true
        this.getInfo()
      }
    })
  },
  // 获取信息
  getInfo () {
    this.$http.get(`/act/demo/correction/${this.dataForm.id}`).then(({ data: res }) => {
      if (res.code !== 0) {
        return this.$message.error(res.msg)
      }
      this.dataForm = {
        ...this.dataForm,
        ...res.data
      }
    }).catch(() => {})
  },
  // 启动流程出错回调
  startProcessErrorCallback (data) {
    console.log(data)
  },
  // 任务处理出错回调
  taskHandleErrorCallback (data) {
  }
}
}
</script>
```

初始化流程组件代码

代码说明：

1、表单属性中新增:disabled="fieldDisabled"

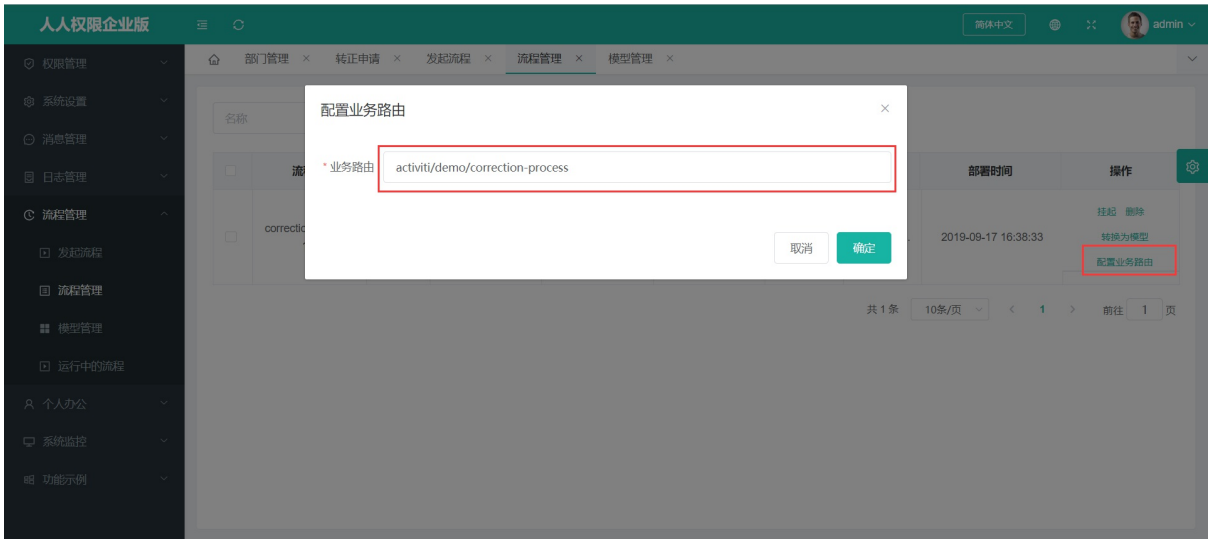
2、流程启动组件

```
<ren-process-multiple v-if="processVisible" saveFormUrl="/act/demo/correction" dataFormName="
dataForm" ref="renProcessMultiple" ></ren-process-multiple>
```

该组件为流程启动组件，参数说明如下：
saveFormUrl : 保存表单的URL（必须配置）；
updateInstanceIdUrl：更新流程实例ID的URL（可选），本实例中没有实例ID，因此没配；
dataFormName： 保存的form表单名称；
3、配置回调函数和初始化流程组件

四、配置业务路由

在流程管理功能配置业务路由信息，且配置到相应的流程中。路由地址规则为：路由文件以文件夹modules为根路径的相对路径。例如转正申请表单路由配置的路径为：activiti/demo/correction-process。
配置路由注意事项：每次修改流程定义设计时，新版本的流程定义需再次配置业务路由。



五、使用启动流程组件启动流程

转正申请流程使用启动流程组件启动流程的代码集成到correction-add-or-update.vue，下面针对启动流程相关代码进行标注：

correction-add-or-update.vue

src > views > modules > activiti > demo > correction-add-or-update.vue > { } "correction-add-or-update.vue" > template

```vue
<template>
  <el-dialog :visible.sync="visible" :title="!dataForm.id ? $t('add') : $t('update')" :close-on-click-modal="false" :close-on-press-escape="false">
    <el-form :model="dataForm" :rules="dataRule" ref="dataForm" @keyup.enter.native="dataFormSubmitHandle()" :label-width="$i18n.locale === 'en-US'
      <el-form-item :label="$t('correction.post')" prop="applyPost">
        <el-input v-model="dataForm.applyPost" :placeholder="$t('correction.post')"></el-input>
      </el-form-item>
      <el-row :gutter="40">
        <el-col :span="12">
          <el-form-item :label="$t('correction.entryDate')" prop="entryDate">
            <el-date-picker v-model="dataForm.entryDate" value-format="yyyy-MM-dd" :placeholder="$t('correction.entryDate')" style="width: 100%"></e
          </el-form-item>
        </el-col>
        <el-col :span="12">
          <el-form-item :label="$t('correction.correctionDate')" prop="correctionDate">
            <el-date-picker v-model="dataForm.correctionDate" value-format="yyyy-MM-dd" :placeholder="$t('correction.correctionDate')" style="width:
          </el-form-item>
        </el-col>
      </el-row>
      <el-form-item :label="$t('correction.workContent')" prop="workContent">
        <el-input type="textarea" v-model="dataForm.workContent" :placeholder="$t('correction.workContent')"></el-input>
      </el-form-item>
      <el-form-item :label="$t('correction.achievement')" prop="achievement">
        <el-input type="textarea" v-model="dataForm.achievement" :placeholder="$t('correction.achievement')"></el-input>
      </el-form-item>
    </el-form>
    <template slot="footer">
      <el-button @click="visible = false">{{ $t('cancel') }}</el-button>
      <!--  流程启动组件  -->
      <ren-process-start v-if="processVisible" updateInstanceIdUrl="/act/demo/correction/updateInstanceId" saveFormUrl="/act/demo/correction" dataFo
    </template>
  </el-dialog>
</template>

<script>
// 引入工作流公共方法
import processModule from '@/mixins/process-module'
export default {
  // 注入公共方法
  mixins: [processModule],
  data () {
    return {
      // 是否显示流程启动组件
      processVisible: true,
      visible: false,
      dataForm: {
        id: '',
        applyPost: '',
        entryDate: '',
        correctionDate: '',
        workContent: '',
        achievement: '',
        creator: '',
        createDate: ''
      }
    }
  },
  methods: {
    init () {
      this.visible = true
      this.$nextTick(() => {
        this.$refs['dataForm'].resetFields()
        if (this.dataForm.id) {
          this.getInfo()
        }
        // 将业务组件对象赋值给流程（回调时需要用到）
        this.$refs.renProcessStart.rootObj = this
        // 配置回调函数
        this.$refs.renProcessStart.callbacks = {
          startProcessSuccessCallback: this.closeCurrentDialog,
          startProcessErrorCallback: this.startProcessErrorCallback,
          formSaveSuccessCallback: null,
          formSaveErrorCallback: null
        }
        // 配置流程定义KEY
        this.$refs.renProcessStart.dataForm.processDefinitionKey = 'correctionprocess'
      })
    },
    // 获取信息
    getInfo () {
      this.$http.get(`/act/demo/correction/${this.dataForm.id}`).then(({ data: res }) => {
        if (res.code !== 0) {
          return this.$message.error(res.msg)
        }
        this.dataForm = {
          ...this.dataForm,
          ...res.data
        }
      }).catch(() => {})
    },
```

代码说明：

1、将保存按钮修改为启动流程组件，并配置保存表单和更新流程实例ID的地址。

```
<ren-process-start v-if="processVisible" updateInstanceIdUrl="/act/demo/correction/updateInst
anceId" saveFormUrl="/act/demo/correction" dataFormName="dataForm" ref="renProcessStart" ></r
en-process-start>
```

updateInstanceIdUrl：更新流程实例IDURL；
saveFormUrl：保存表单URL；
dataFormName：表单对象名称。中的ref内容。
2、启动流程组件初始。

六、配置查看流程详情组件

转正申请配置配置查看流程详情组件代码在correction.vue中。详细请参考：

correction.vue

src > views > modules > activiti > demo > correction.vue > {} "correction.vue" > script > data

```vue
<template>
  <el-card shadow="never" class="aui-card--fill">
    <div class="mod-demo__correction}">
      <el-form :inline="true" :model="dataForm" @keyup.enter.native="getDataList()">
        <el-form-item>
          <el-button v-if="$hasPermission('activiti:correction:all')" type="primary" @click="addOrUpdateHandle()">{{ $t('add') }}</el-button>
        </el-form-item>
        <el-form-item>
          <el-button v-if="$hasPermission('activiti:correction:all')" type="danger" @click="deleteHandle()">{{ $t('deleteBatch') }}</el-button>
        </el-form-item>
      </el-form>
      <el-table v-loading="dataListLoading" :data="dataList" border @selection-change="dataListSelectionChangeHandle" style="width: 100%;">
        <el-table-column type="selection" header-align="center" align="center" width="50"></el-table-column>
        <el-table-column prop="instanceId" :label="$t('running.id')" header-align="center" align="center"></el-table-column>
        <el-table-column prop="applyPost" :label="$t('correction.post')" header-align="center" align="center"></el-table-column>
        <el-table-column prop="entryDate" :label="$t('correction.entryDate')" header-align="center" align="center"></el-table-column>
        <el-table-column prop="correctionDate" :label="$t('correction.correctionDate')" header-align="center" align="center"></el-table-column>
        <el-table-column prop="workContent" :label="$t('correction.workContent')" header-align="center" align="center"></el-table-column>
        <el-table-column prop="achievement" :label="$t('correction.achievement')" header-align="center" align="center"></el-table-column>
        <el-table-column prop="createDate" :label="$t('createDate')" header-align="center" align="center"></el-table-column>
        <el-table-column :label="$t('handle')" fixed="right" header-align="center" align="center" width="150">
          <template slot-scope="scope">
            <el-button type="text" size="small" @click="showDetail(scope.row)">{{ $t('process.viewFlowImage') }}</el-button>
          </template>
        </el-table-column>
      </el-table>
      <el-pagination
        :current-page="page"
        :page-sizes="[10, 20, 50, 100]"
        :page-size="limit"
        :total="total"
        layout="total, sizes, prev, pager, next, jumper"
        @size-change="pageSizeChangeHandle"
        @current-change="pageCurrentChangeHandle">
      </el-pagination>
      <!-- 弹窗, 新增 / 修改 -->
      <add-or-update v-if="addOrUpdateVisible" ref="addOrUpdate" @refreshDataList="getDataList"></add-or-update>
    </div>
  </el-card>
</template>

<script>
import mixinViewModule from '@/mixins/view-module'
import AddOrUpdate from './correction-add-or-update'
// 引入工作流公共方法
import processModule from '@/mixins/process-module'
export default {
  // 注入公共方法
  mixins: [mixinViewModule, processModule],
  data () {
    return {
      mixinViewModuleOptions: {
        getDataListURL: '/act/demo/correction/page',
        getDataListIsPage: true,
        exportURL: '/act/demo/correction/export',
        deleteURL: '/act/demo/correction',
        deleteIsBatch: true
      },
      dataForm: {
        id: ''
      },
      // 流程定义KEY
      procDefKey: 'correctionprocess'
    }
  },
  components: {
    AddOrUpdate
  },
  methods: {
    // 查看流程处理详情
    showDetail (row) {
      if (!row.id) {
        return this.$message.error(this.$t('task.detailError'))
      }
      var params = {
        businessKey: row.id,
        procDefKey: this.procDefKey
      }
      this.getProcDefBizRouteAndProcessInstance(params, this.forwardDetail)
    }
  }
}
</script>
```

七、任务处理

第三，流程实例启动后，activiti会根据流程图生成相应的任务提供给用户处理，且流程审批就会根据流程图中的配置进行流转。业务在流程处理中，对于审批无需其他开发，只要调用任务处理相关的API即可完成流程审批。流程处理主要有以下几个操作：

一、完成。直接完成任务。API接口：“/act/task/complete”。

二、认领任务。该操作是流程设计时用户任务设置为角色或用户组，需要角色或用户组下的用户认领任务后，方可进行处理。API接口：“/act/task/claim”。

三、取消认领。与认领的操作相反。将任务放进任务池中。API接口：“/act/task/unclaim”。

四、任务驳回。将任务驳回至发起人进行审批。API接口：“/act/task/backToFristStep”。

五、任务委托。将分配给自己的任务委托给其他人。API接口：“/act/task/changeTaskAssignee”。

六、任务回退。将任务回退至上一节点。API接口：“/act/task/backPreviousTask”。

七、终止。终止流程。API接口：“/act/task/endProcess”。

接口的参数请参考swagger的API接口说明。

转正申请流程处理操作界面：

第四，流程处理跟踪。分为流程图和处理详情跟踪。流程图API：“/his/queryInstImage”；处理详情：“/his/getTaskHandleDetailInfo”。转正申请示例操作详情界面：

第6章 生产环境部署

> 部署项目前，需要准备JDK8、MySQL、Redis、Nacos环境，参考开发环境搭建。

6.1 jar包部署

6.2 docker部署

6.3 Nginx配置

6.1 jar包部署

> Spring Boot项目，推荐打成jar包的方式，部署到服务器上。

- Spring Boot内置了Tomcat，可配置Tomcat的端口号、初始化线程数、最大线程数、连接超时时长、https等等，如下所示：

```
server:
  tomcat:
    uri-encoding: UTF-8
    max-threads: 1000
    min-spare-threads: 30
  port: 8082
  connection-timeout: 5000ms
  servlet:
    context-path: /sys
    session:
      cookie:
        http-only: true
  ssl:
    key-store: classpath:.keystore
    key-store-type: JKS
    key-password: 123456
    key-alias: tomcat
```

- 当然，还可以指定jvm的内存大小，如下所示：

```
java -Xms4g -Xmx4g -Xmn1g -server -jar renren-admin-server.jar
```

- 在windows下部署，只需打开cmd窗口，输入如下命令：

```
java -jar renren-admin-server.jar --spring.profiles.active=prod
```

- 在Linux下部署，只需输入如下命令，即可在Linux后台运行：

```
nohup java -jar renren-admin-server.jar --spring.profiles.active=prod > renren.log &
```

- 在Linux环境下，我们一般可以创建shell脚本，用于重启项目，如下所示：

```
#创建启动的shell脚本
[root@renren renren-admin]# vim start.sh
#!/bin/sh

process=`ps -fe|grep "renren-admin-server.jar" |grep -ivE "grep|cron" |awk '{print $2}'`
if [ !$process ];
then
    echo "stop erp process $process ....."
```

```
    kill -9 $process
    sleep 1
fi

echo "start erp process....."
nohup java -Dspring.profiles.active=prod -jar renren-admin-server.jar --server.port=8082 --se
rver.servlet.context-path=/sys 2>&1 | cronolog log.%Y-%m-%d.out >> /dev/null &

echo "start erp success!"

#通过shell脚本启动项目
[root@renren renren-admin]# yum install -y cronolog
[root@renren renren-admin]# chomd +x start.sh
[root@renren renren-admin]# ./start.sh
```

6.2 docker部署

- 安装docker环境

```
#安装docker
[root@renren ~]# curl -fsSL https://get.docker.com | bash -s docker --mirror Aliyun

#启动docker
[root@renren ~]# service docker start

#查看docker版本信息
[root@renren ~]# docker version
Client:
 Version:           18.06.1-ce
 API version:       1.38
 Go version:        go1.10.3
 Git commit:        e68fc7a
 Built:             Tue Aug 21 17:23:03 2018
 OS/Arch:           linux/amd64
 Experimental:      false

Server:
 Engine:
  Version:          18.06.1-ce
  API version:      1.38 (minimum version 1.12)
  Go version:       go1.10.3
  Git commit:       e68fc7a
  Built:            Tue Aug 21 17:25:29 2018
  OS/Arch:          linux/amd64
  Experimental:     false
```

- docker命令自动补全

```
#安装补全工具
[root@renren ~]# yum install -y bash-completion

#安装完后，关闭当前命令窗口，重新打开，再输入docker 按2下tab键，则可出现所有docker命令
[root@renren ~]# docker
attach     config     create     exec       history    import     kill       logout     node
      port       push       rm         save       service    stats      system     trust
 version
build      container  diff       export     image      info       load       logs       pause
      ps         rename     rmi        search     stack      stop       tag        unpause
 volume
commit     cp         events     help       images     inspect    login      network    plugi
n     pull       restart    run        secret     start      swarm      top        update
 wait
[root@renren ~]# docker e
events  exec    export
```

- 安装JDK8环境

```
#下载JDK8
https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

#安装JDK8
[root@renren work]# rpm -ivh jdk-8u181-linux-x64.rpm
warning: jdk-8u181-linux-x64.rpm: Header V3 RSA/SHA256 Signature, key ID ec551f03: NOKEY
Preparing...                          ################################# [100%]
Updating / installing...
   1:jdk1.8-2000:1.8.0_181-fcs        ################################# [100%]
Unpacking JAR files...
    tools.jar...
    plugin.jar...
    javaws.jar...
    deploy.jar...
    rt.jar...
    jsse.jar...
    charsets.jar...
    localedata.jar...
[root@renren work]# java -version
java version "1.8.0_181"
Java(TM) SE Runtime Environment (build 1.8.0_181-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.181-b13, mixed mode)
```

- 安装maven环境

```
#下载Maven3+
http://maven.apache.org/download.cgi

#安装Maven3+
[root@renren work]# tar xf apache-maven-3.5.4-bin.tar.gz
[root@renren work]# cd apache-maven-3.5.4/
[root@renren apache-maven-3.5.4]# ll
total 48
drwxr-xr-x 2 root root   4096 Oct  3 23:46 bin
drwxr-xr-x 2 root root   4096 Oct  3 23:46 boot
drwxr-xr-x 3  501 games  4096 Jun 18 02:30 conf
drwxr-xr-x 4  501 games  4096 Oct  3 23:46 lib
-rw-r--r-- 1  501 games 20965 Jun 18 02:35 LICENSE
-rw-r--r-- 1  501 games   182 Jun 18 02:35 NOTICE
-rw-r--r-- 1  501 games  2530 Jun 18 02:30 README.txt
[root@renren apache-maven-3.5.4]# pwd
/work/apache-maven-3.5.4
```

配置环境变量

```
[root@renren ~]# vim /etc/profile
export MAVEN_HOME=/work/apache-maven-3.5.4
```

```
export PATH=$MAVEN_HOME/bin:$PATH

#环境变量生效
[root@renren ~]# source /etc/profile

#查看版本号
[root@renren ~]# mvn -v
Apache Maven 3.5.4 (1edded0938998edf8bf061f1ceb3cfdeccf443fe; 2018-06-18T02:33:14+08:00)
Maven home: /work/apache-maven-3.5.4
Java version: 1.8.0_181, vendor: Oracle Corporation, runtime: /usr/java/jdk1.8.0_181-amd64/jr
e
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "3.10.0-693.2.2.el7.x86_64", arch: "amd64", family: "unix"
```

- 通过Dockfile文件，构建docker镜像

```
#将下载的SkyWalking6.6，解压到docker目录下，如下：
[root@renren docker]# pwd
/work/docker
[root@renren docker]# ll
total 4
drwxr-xr-x. 8 root root 137 Dec 24 06:30 apache-skywalking-apm-bin
-rw-r--r--. 1 root root 509 Feb 26 15:32 Dockerfile

#构建docker镜像，进入项目的docker目录，执行如下命令即可
[root@renren docker]# docker build -t renren_io:1.0 .

#查看Docker镜像
[root@renren docker]# docker images
```

- 安装docker-compose，用来管理容器

```
#下载docker-compose
[root@renren ~]# curl -L https://github.com/docker/compose/releases/download/1.22.0/docker-co
mpose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   617    0   617    0     0    555      0 --:--:--  0:00:01 --:--:--   556
100 11.2M  100 11.2M    0     0  1661k      0  0:00:06  0:00:06 --:--:-- 2845k

#增加可执行权限
[root@renren ~]# chmod +x /usr/local/bin/docker-compose

#查看版本信息
[root@renren ~]# docker-compose version
docker-compose version 1.22.0, build f46880fe
docker-py version: 3.4.1
CPython version: 3.6.6
OpenSSL version: OpenSSL 1.1.0f  25 May 2017
```

如果下载不了，可以用迅雷将 `https://github.com/docker/compose/releases/download/1.16.1/docker-compose-Linux-x86_64` 下载到本地，再上传到服务器

- 编译打包项目，并把项目jar文件拷贝到【/data/renren-cloud】目录
- 通过docker-compose启动项目

1) 进入项目的docker-compose目录

2) 编辑common.env文件，改成nacos对应的信息，如下所示：

```
spring.profiles.active=dev
nacos-host=172.17.0.1
nacos-port=8848
nacos-namespace=xxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

3) 通过docker-compose命令，启动项目，如下所示：

```
#启动项目
[root@renren renren-cloud]# docker-compose up -d

#查看启动的容器
[root@renren renren-cloud]# docker ps

#查看启动的logs
[root@renren renren-cloud]# docker logs -f renren-admin-server
```

- docker-compose官方文档
  > https://docs.docker.com/compose/compose-file/compose-file-v2/

6.3 Nginx配置

> Nginx配置如下所示：

```
server {
    listen       80;
    server_name  localhost;

    location /renren-cloud {
        alias /data/renren-cloud-admin;
        index index.html;
    }

    location /renren-cloud-server/ {
        proxy_pass      http://localhost:8080/;
        client_max_body_size 1024m;
        proxy_set_header  X-Real-IP  $remote_addr;
        proxy_set_header  X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header  Host $host;
        proxy_redirect    off;
    }
}
```

- 重启Nginx之后，就可以通过【http://ip、域名/renren-cloud】访问