

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 开发环境搭建	1.2.3
1.4 数据交互	1.2.4
1.5 获取帮助	1.2.5
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
第5章 项目实战	1.6
5.1 需求说明	1.6.1
5.2 代码生成器	1.6.2
第6章 后端源码分析	1.7
6.1 功能模块移除	1.7.1
6.2 前后端分离	1.7.2
6.3 功能权限设计	1.7.3
6.4 数据权限设计	1.7.4
6.5 数据权限使用	1.7.5
6.6 XSS脚本过滤	1.7.6
6.7 Redis缓存	1.7.7
6.8 异常处理机制	1.7.8
6.9 操作日志	1.7.9
6.10 定时任务模块	1.7.10
6.11 API模块	1.7.11

6.12 工作流模块	1.7.12
第7章 生产环境部署	1.8
7.1 jar包部署	1.8.1
7.2 docker部署	1.8.2
7.3 跨域配置	1.8.3

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

人人权限系统是一套轻量级的权限系统，主要包括用户管理、角色管理、部门管理、菜单管理、定时任务、参数管理、字典管理、文件上传、登录日志、操作日志、异常日志、文章管理、APP模块等功能。其中，还拥有多数据源、数据权限、国际化支持、Redis缓存动态开启与关闭、统一异常处理等技术特点。

1.1 项目特点

1.2 项目结构

1.3 开发环境搭建

1.4 数据交互

1.5 获取帮助

1.1 项目描述

- 基于最新的SpringBoot 2.0、MyBatis、Shiro、Element 2.0+框架，开发的一套权限系统，极低门槛，拿来即用。设计之初，就非常注重安全性，为企业系统保驾护航，让一切都变得如此简单
- 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
- 完善的 xss 防范及脚本过滤，彻底杜绝 xss 攻击，且基于白名单的富文本XSS过滤
- 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

项目一共分为五个模块，如下所示：

```
security-enterprise
├── renren-admin    后台管理
│   ├── 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
│   └── renren-dynamic-datasource 多数据源
├── renren-common  工具包
├── renren-api     API服务
└── renren-generator 代码生成器
```

- renren-common为公共模块，其他模块以jar包的形式引入进去，主要提供些工具类，以及renren-admin、renren-api模块公共的entity、mapper、dao、service服务，防止一个功能重复多次编写代码。
- renren-dynamic-datasource为多数据源模块，其他模块以jar包的形式引入进去。
- renren-admin为后台模块，也是系统的核心，用来开发后台管理系统，可以打包成jar，部署到服务器上运行，或者打包成war，放到Tomcat8.5+容器里运行。
- renren-api为接口模块，主要是简化APP开发，如：为微信小程序、IOS、Android提供接口，拥有一套单独的用户体系，没有与renren-admin用户表共用，因为renren-admin用户表里存放的是企业内部人员账号，具有后台管理员权限，可以登录后台管理系统，而renren-api用户表里存放的是我们的真实用户，不具备登录后台管理系统的权限。renren-api主要是实现了用户注册、登录、接口权限认证、获取登录用户等功能，为APP接口的安全调用，提供一套优雅的解决方案，从而简化APP接口开发。
- renren-generator为代码生成器模块，只需创建好表结构，就可以生成新增、修改、删除、查询、导出等操作的代码，包括entity、mapper、dao、service、controller、vue等所有代码，项目开发神器。支持MySQL、Oracle、SQL Server、PostgreSQL数据库。

1.3 开发环境搭建

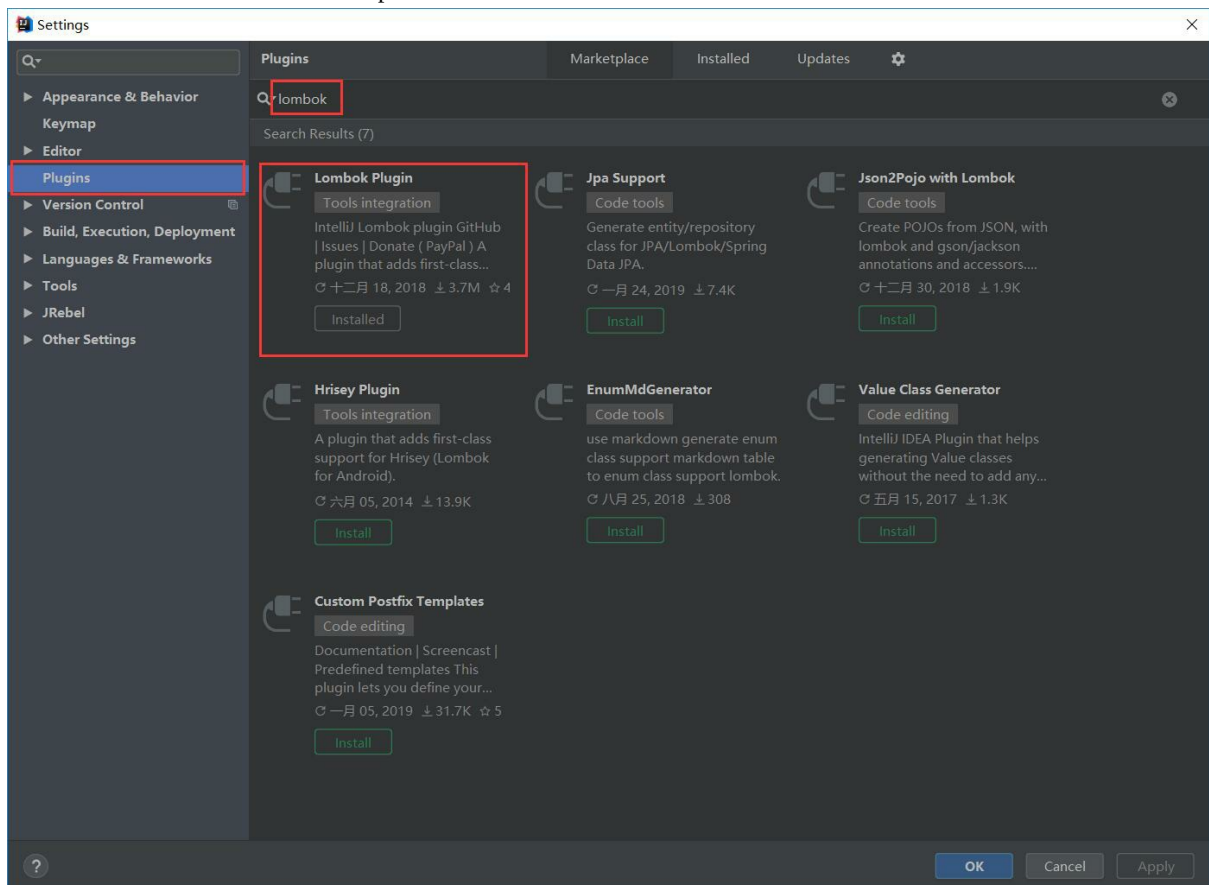
1.3.1 软件需求

- JDK 1.8+
- Maven 3.0+
- MySQL 8.0
- Oracle 11g+
- SQL Server 2012+
- PostgreSQL 9.4+

1.3.2 安装Lombok插件

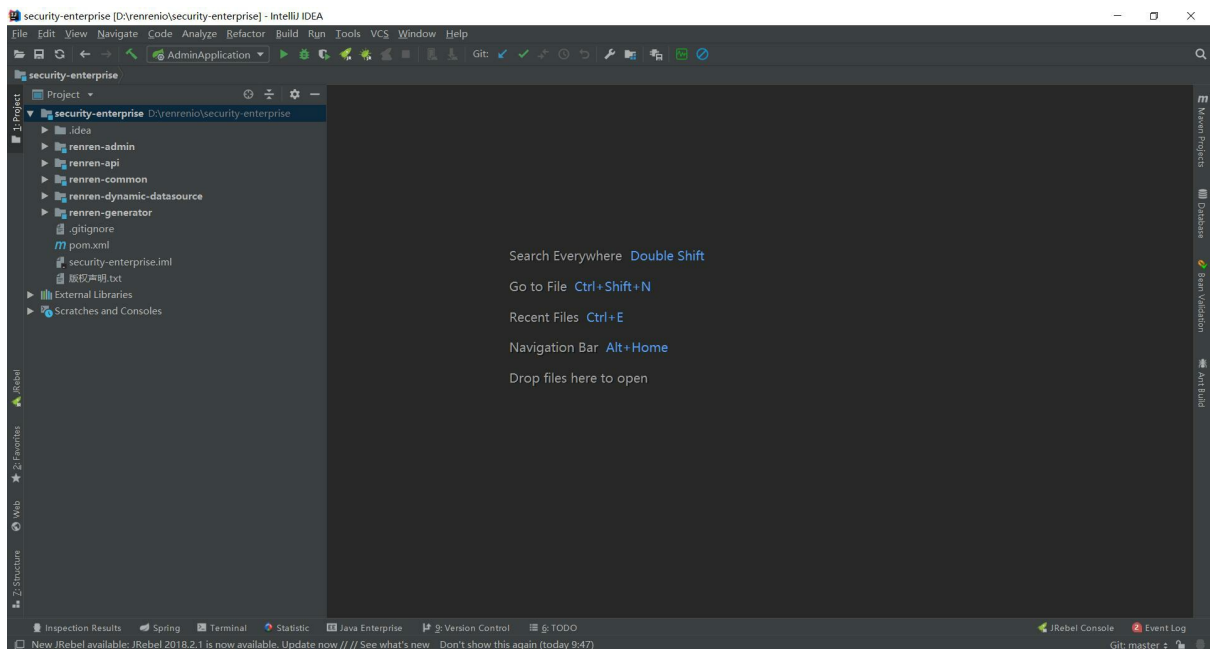
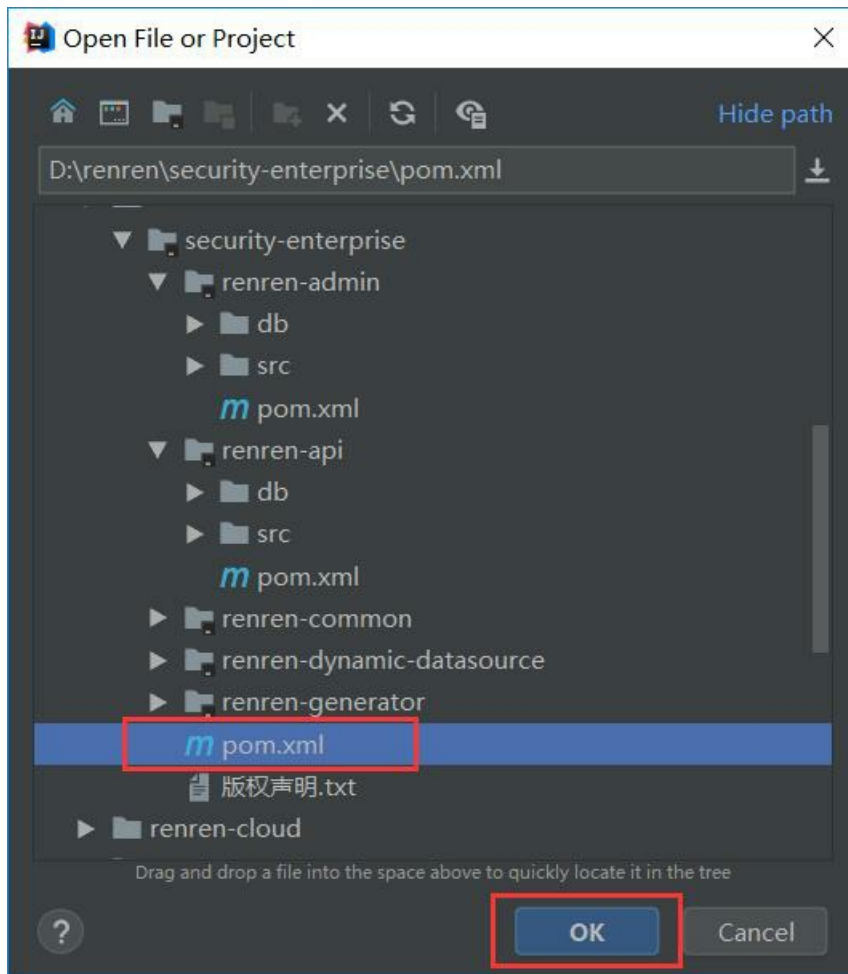
在项目中使用Lombok可以减少很多重复代码，如：getter、setter、toString等方法的编写

下图是idea安装Lombok方法，eclipse类似，如下图所示：

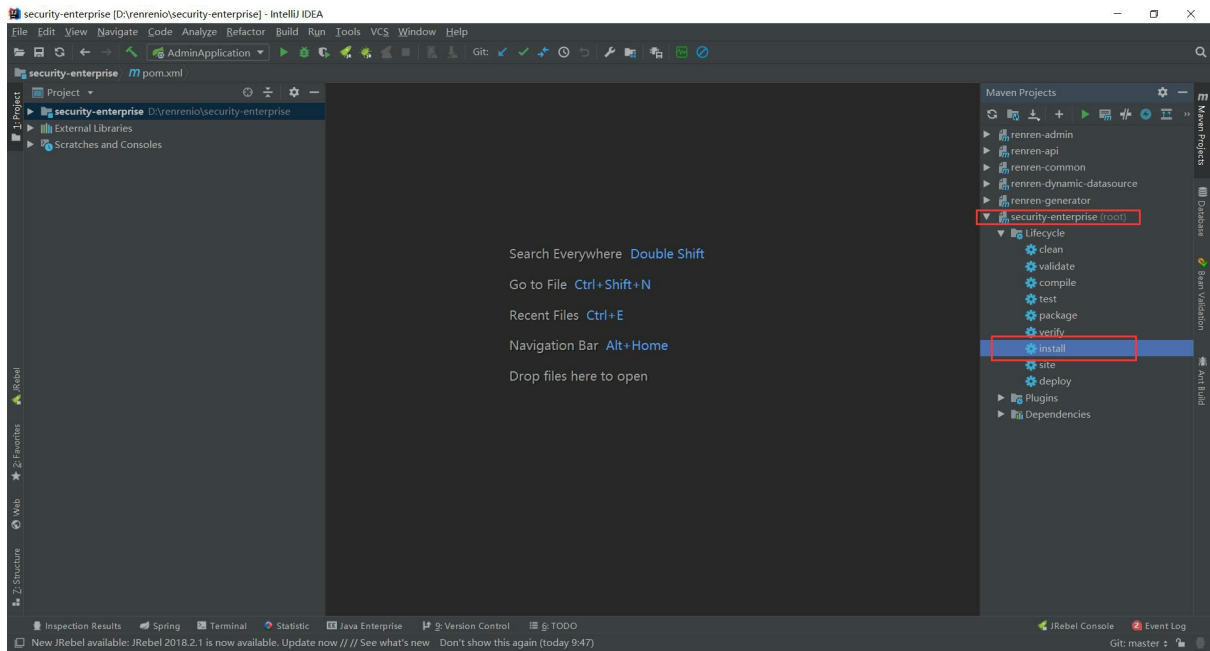


1.3.3 IDEA开发工具

- IDEA打开项目， `File -> Open` 如下图：

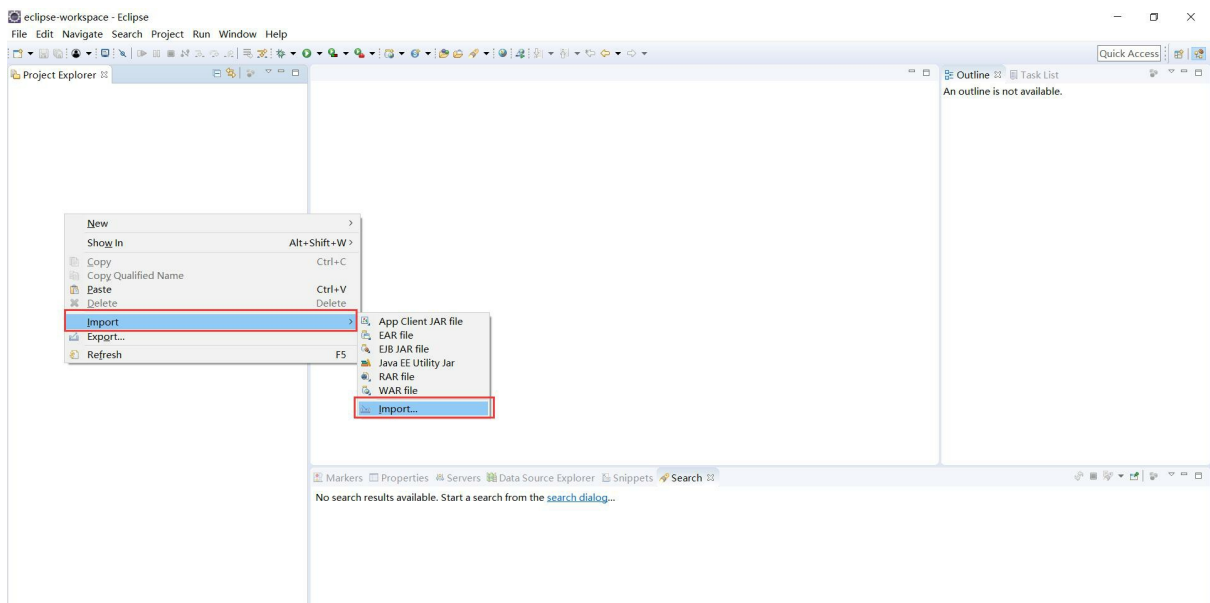


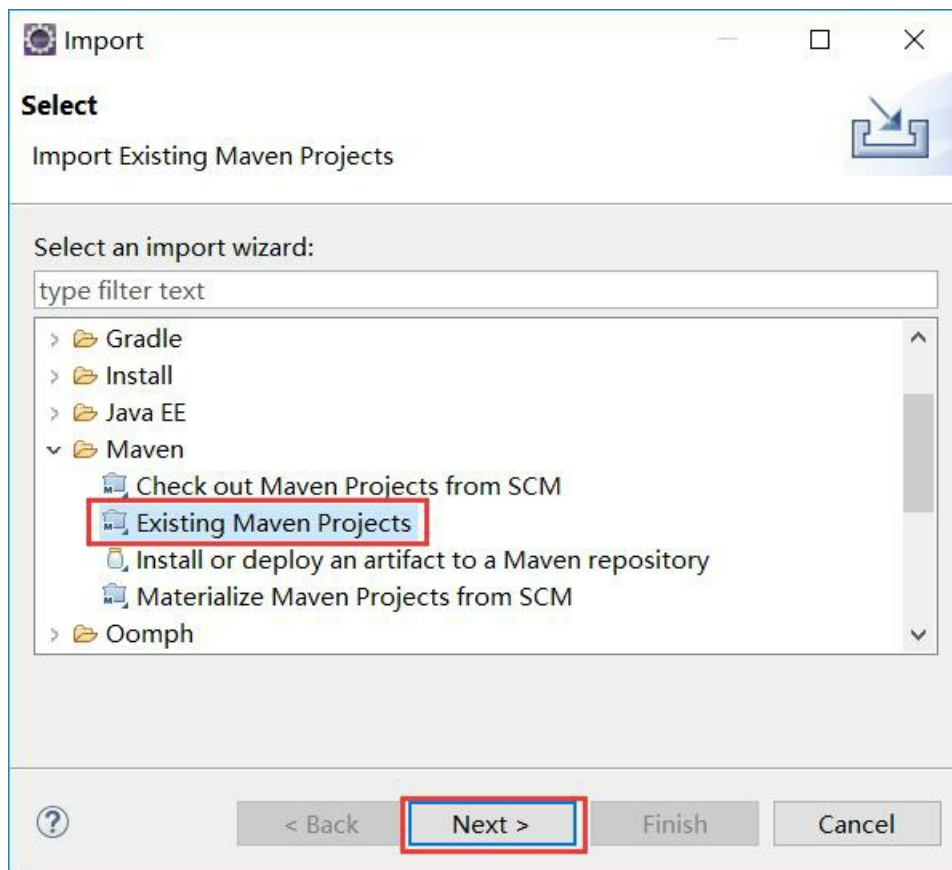
- 在security-enterprise目录下，执行mvn install，如下图所示：

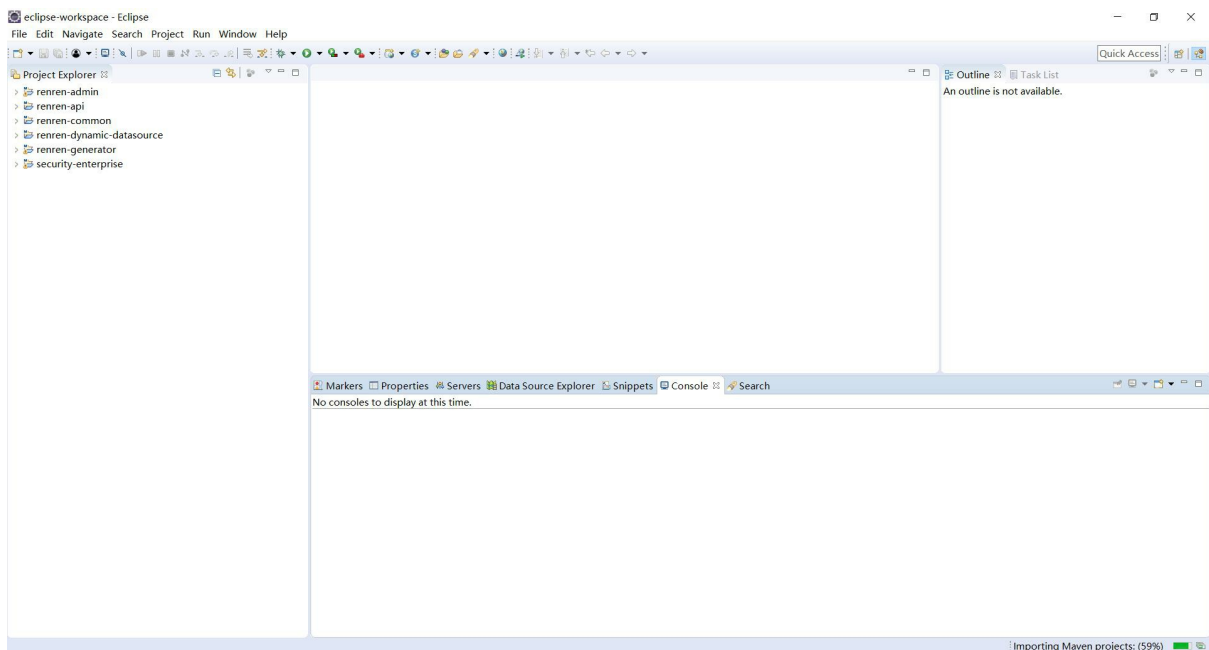
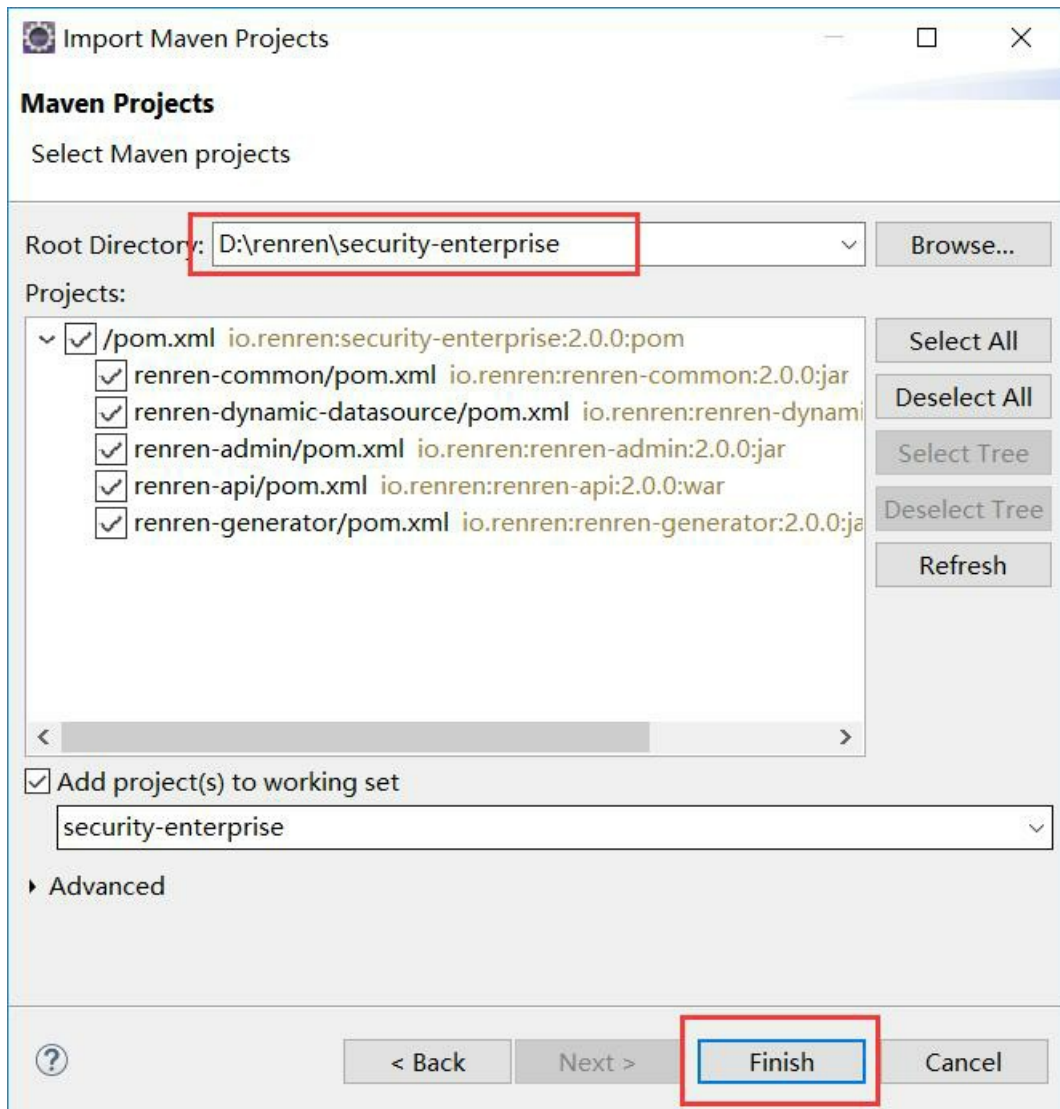


1.3.4 Eclipse开发工具

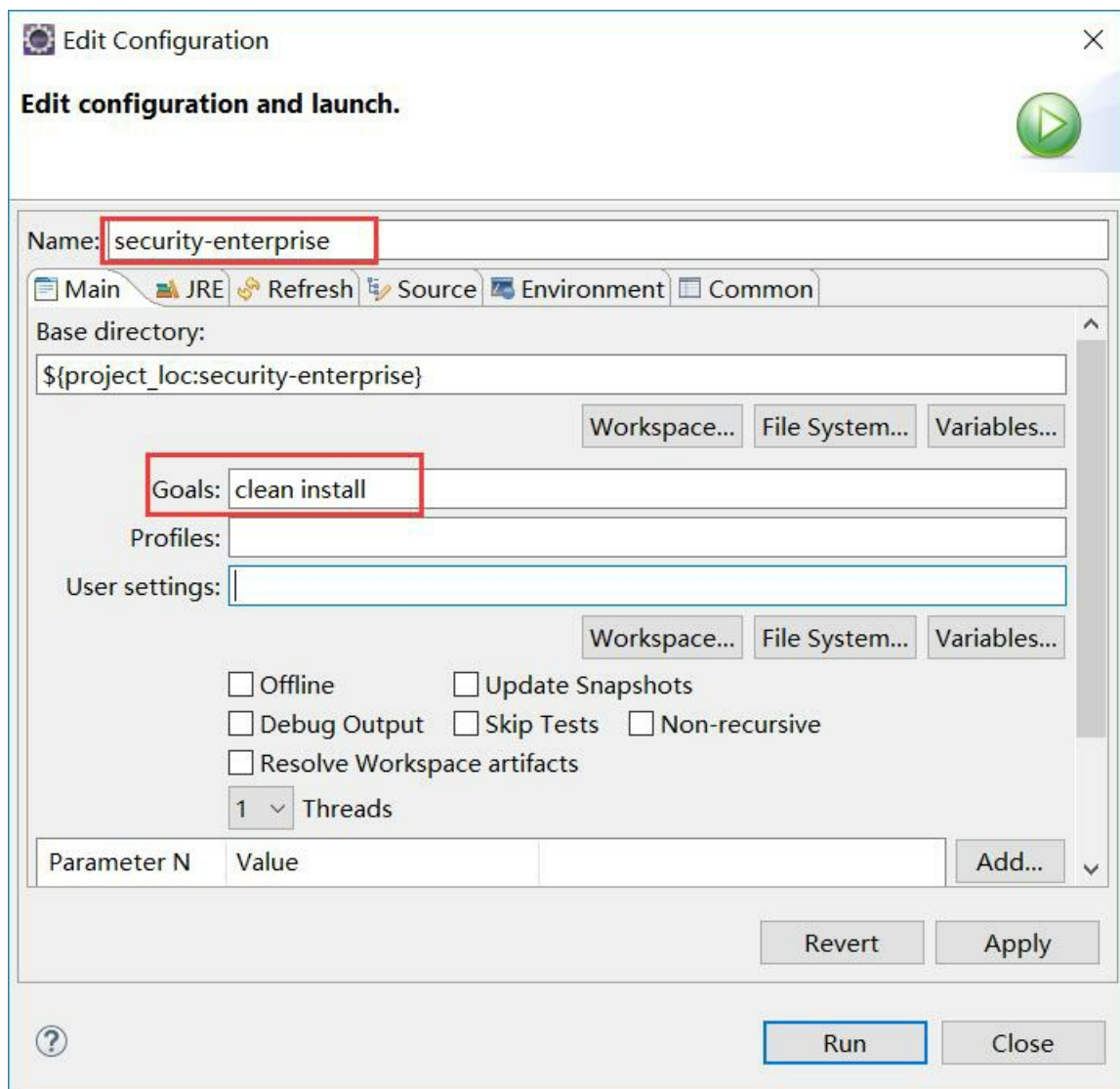
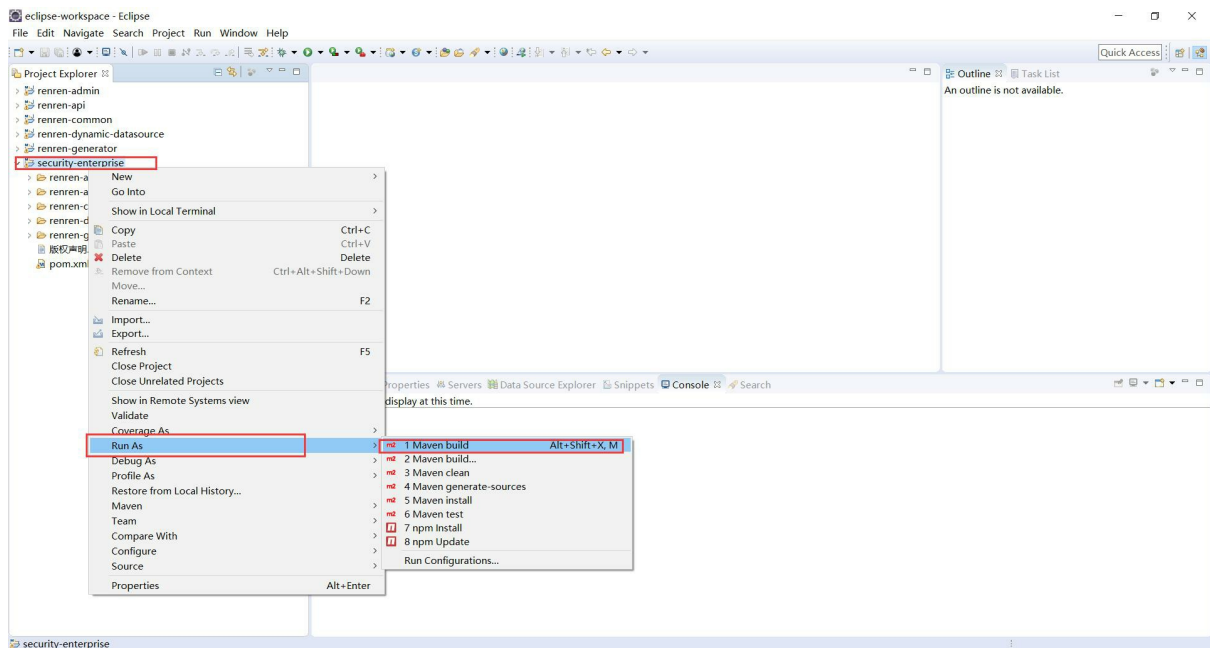
- Eclipse导入项目，如下图：

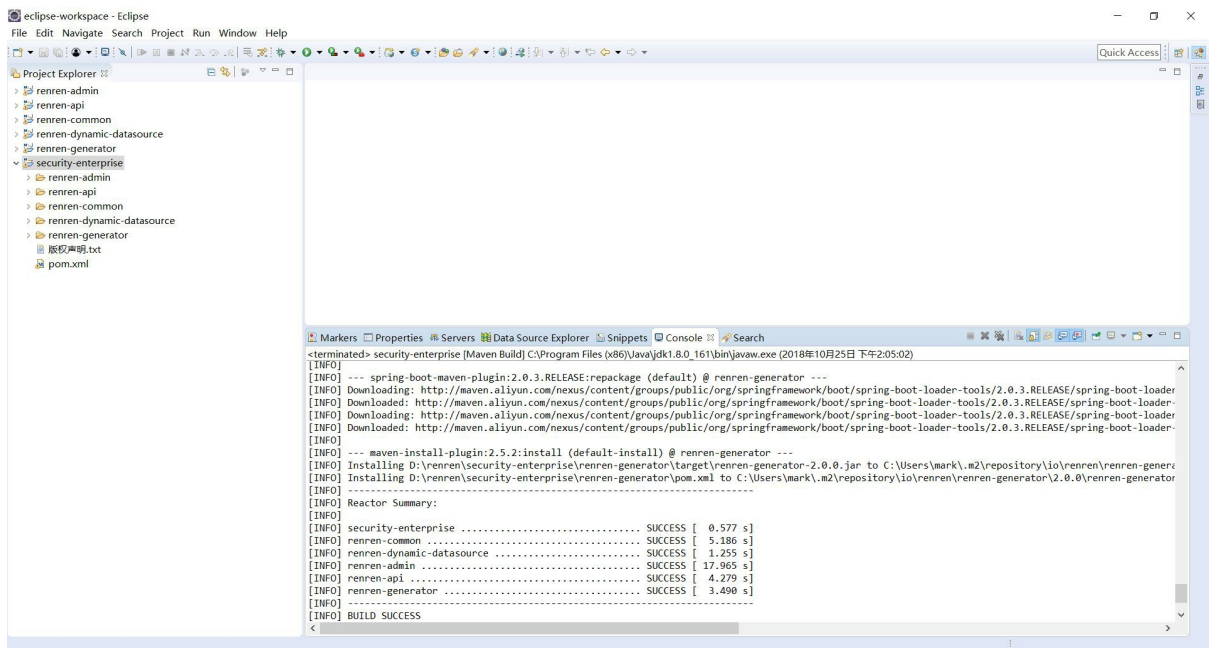






- 在security-enterprise目录下，执行mvn install，如下图所示：



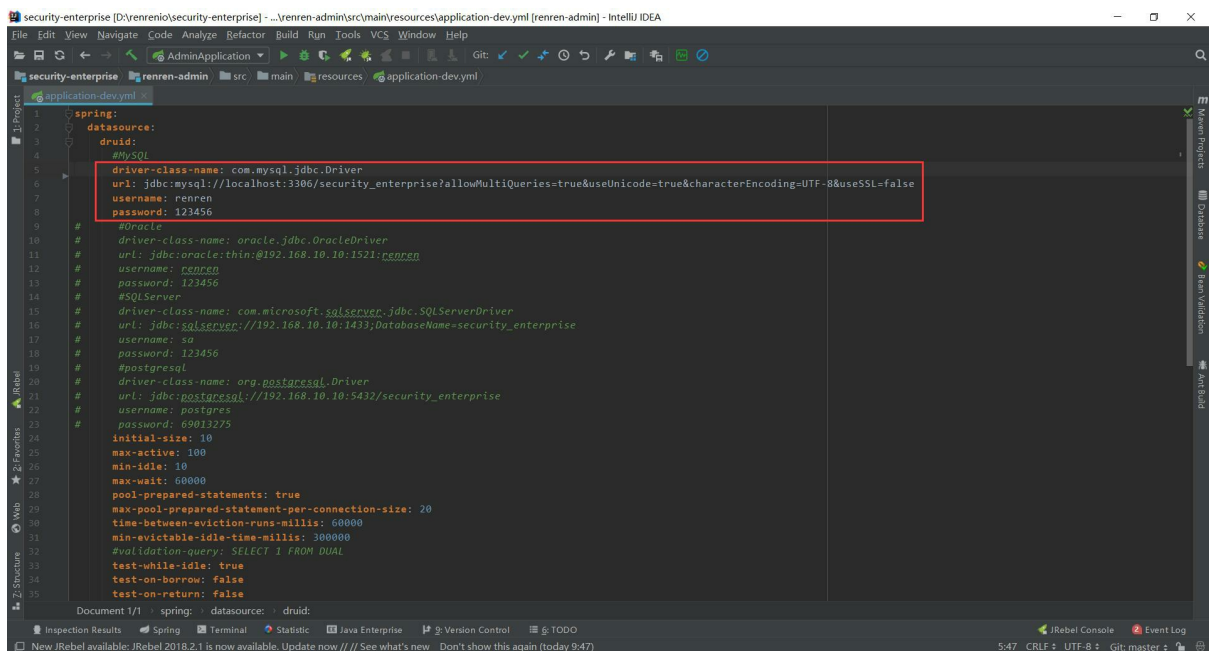


1.3.5 创建数据库

- 创建security_enterprise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

1.3.6 修改配置文件

修改renren-admin配置文件 application-dev.yml，如下图所示：



1.3.7 启动项目

- 1) 启动renren-admin

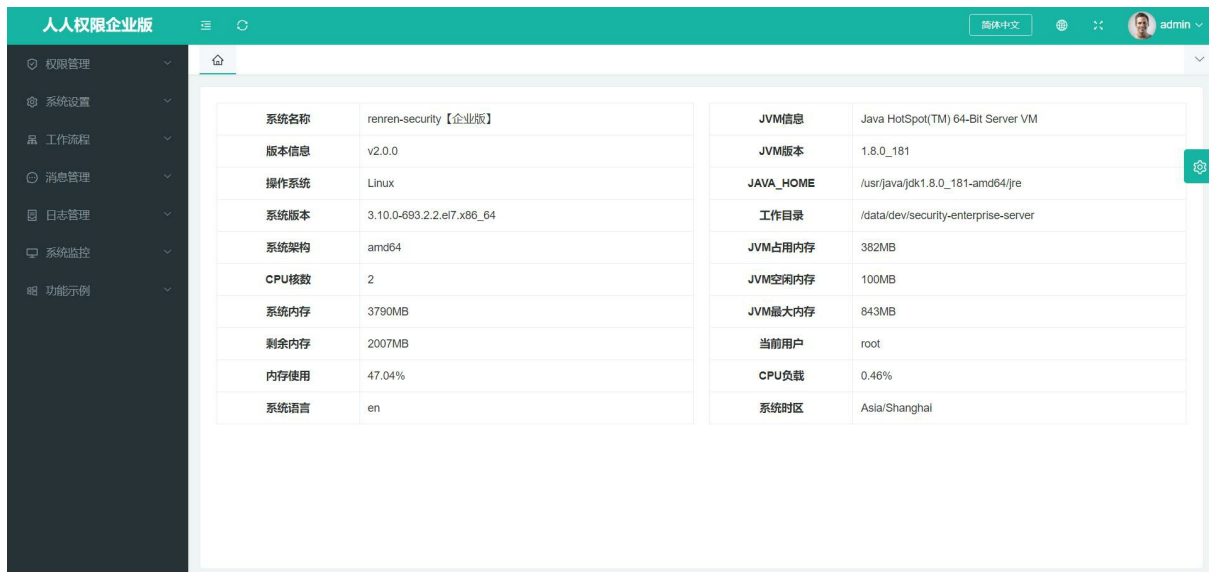
- 运行 `io.renren.AdminApplication.java` 的 `main` 方法，则可启动renren-admin项目
- 接口文档地址: <http://localhost:8080/renren-admin/doc.html>



- 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：



- 输入账号admin，密码admin，则可登陆系统，如下所示：



2) 启动renren-api项目

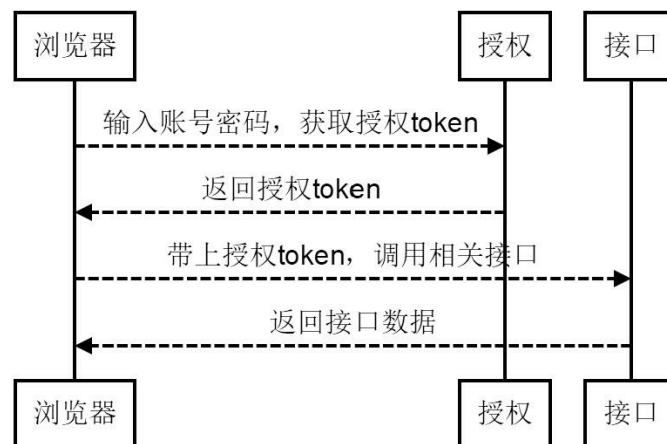
- Eclipse、IDEA运行ApiApplication.java，则可启动项目【renren-api】
- 接口文档地址：<http://localhost:8081/renren-api/doc.html>

3) 启动renren-generator项目

- Eclipse、IDEA运行GeneratorApplication.java，则可启动项目【renren-generator】
- 项目访问路径：<http://localhost:8082/renren-generator>

1.4 数据交互

- 一般情况下，web项目都是通过session进行认证，每次请求数据时，都会把jsessionId放在cookie中，以便与服务端保持会话
- 本项目是前后端分离的，通过token进行认证（登录时，生成唯一的token凭证），每次请求数据时，都会把token放在header中，服务端解析token，并确定用户身份及用户权限，数据通过json交互
- 数据交互流程，如下所示：



1.5 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，开发环境的配置文件在application-dev.yml，如下所示：

```
spring:
  datasource:
    druid:
      driver-class-name: com.mysql.jdbc.Driver
      url: jdbc:mysql://localhost:3306/security_enterprise?allowMultiQueries=true&useUnicode=
true&characterEncoding=UTF-8&useSSL=false
      username: root
      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，开发环境的配置文件在application-dev.yml，如下所示：

```
spring:
  datasource:
    druid:
      driver-class-name: oracle.jdbc.OracleDriver
      url: jdbc:oracle:thin:@192.168.10.10:1521:renren
      username: security_enterprise
      password: 123456
```

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，开发环境的配置文件在application-dev.yml，如下所示：

```
spring:
  datasource:
    druid:
      driver-class-name: com.microsoft.sqlserver.jdbc.SQLServerDriver
      url: jdbc:sqlserver://192.168.10.10:1433;DatabaseName=security_enterprise
      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，开发环境的配置文件在application-dev.yml，如下所示：

```
spring:
  datasource:
    druid:
      driver-class-name: org.postgresql.Driver
      url: jdbc:postgresql://192.168.10.10:5432/security_enterprise
      username: renren
      password: 123456
```

2) 修改quartz配置信息，quartz配置文件 ScheduleConfig.java，打开注释，如下所示：

```
//PostgreSQL数据库，需要打开此注释
prop.put("org.quartz.jobStore.driverDelegateClass", "org.quartz.impl.jdbcjobstore.PostgreSQLDelegator");
```

3) 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-dynamic-datasource.jar，如下所示：

```
<dependency>
  <groupId>io.renren</groupId>
  <artifactId>renren-dynamic-datasource</artifactId>
  <version>2.0.0</version>
</dependency>
```

2) 配置多数据源，如果是开发环境，则修改 application-dev.xml，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=security_enterprise
username: sa
password: 123456

slave2:

driver-class-name: org.postgresql.Driver
url: jdbc:postgresql://192.168.10.10:5432/security_enterprise
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

```
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
 */
@Service
//@DataSource("slave1") 多数据源全局配置
public class DynamicDataSourceTestService {

    @Autowired
    private SysUserDao sysUserDao;

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

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

```

    @DataSource("slave2")
    @Transactional
    public void updateUserBySlave2(Long id){
        SysUserEntity user = new SysUserEntity();
        user.setUserId(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

```

package io.renren.service;

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
 */
@RunWith(SpringRunner.class)
@SpringBootTest
public class DynamicDataSourceTest {

    @Autowired
    private DynamicDataSourceTestService dynamicDataSourceTestService;

    @Test
    public void test(){
        Long id = 1L;

        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：

```

dynamic:
  datasource:
    slave1:

```

```
driver-class-name: com.microsoft.sqlserver.jdbc.SQLServerDriver
url: jdbc:sqlserver://localhost:1433;DatabaseName=security_enterprise
username: sa
password: 123456
slave2:
driver-class-name: org.postgresql.Driver
url: jdbc:postgresql://localhost:5432/security_enterprise
username: renren
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

```
import java.lang.annotation.*;

/**
 * 多数据源注解
 *
 * @author Mark sunlightcs@gmail.com
 */
@Target({ElementType.METHOD, ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
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：

```
/**
 * 多数据源属性
 *
 * @author Mark sunlightcs@gmail.com
 */
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;
    }
}

```

```

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
 */
@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：

```

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

/**
 * 多数据源
 *
 * @author Mark sunlightcs@gmail.com
 */
public class DynamicDataSource extends AbstractRoutingDataSource {

    @Override
    protected Object determineCurrentLookupKey() {
        return DynamicContextHolder.peek();
    }
}

```

```
}
```

4) 多数据源上下文

```
/**
 * 多数据源上下文
 *
 * @author Mark sunlightcs@gmail.com
 */
public class DynamicContextHolder {
    @SuppressWarnings("unchecked")
    private static final ThreadLocal<Deque<String>> CONTEXT_HOLDER = new ThreadLocal() {
        @Override
        protected Object initialValue() {
            return new ArrayDeque();
        }
    };

    /**
     * 获得当前线程数据源
     *
     * @return 数据源名称
     */
    public static String peek() {
        return CONTEXT_HOLDER.get().peek();
    }

    /**
     * 设置当前线程数据源
     *
     * @param dataSource 数据源名称
     */
    public static void push(String dataSource) {
        CONTEXT_HOLDER.get().push(dataSource);
    }

    /**
     * 清空当前线程数据源
     */
    public static void poll() {
        Deque<String> deque = CONTEXT_HOLDER.get();
        deque.poll();
        if (deque.isEmpty()) {
            CONTEXT_HOLDER.remove();
        }
    }
}
```

5) 配置多数据源，如下所示：

```

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
 */
@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;
    }
}

```

```

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

import java.sql.SQLException;

/**
 * DruidDataSource
 *
 * @author Mark sunlightcs@gmail.com
 */
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

```

import io.renren.commons.dynamic.datasource.annotation.DataSource;
import io.renren.commons.dynamic.datasource.config.DynamicContextHolder;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
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
 */
@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();
        }

        DynamicContextHolder.push(value);
        logger.debug("set datasource is {}", value);
    }

    try {
        return point.proceed();
    } finally {
        DynamicContextHolder.poll();
        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

4.1 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/user/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 {

    @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", required = true, dataType="int") ,
    @ApiImplicitParam(name = "limit", value = "每页显示记录数", paramType = "query", required = true, dataType="int") ,
    @ApiImplicitParam(name = "order_field", value = "排序字段", paramType = "query", dataType="String") ,
    @ApiImplicitParam(name = "order", value = "排序方式，可选值(asc、desc)", paramType = "query", dataType="String") ,
    @ApiImplicitParam(name = "username", value = "用户名", paramType = "query", dataType="String")
})
public Result<PageData<SysUserDTO>> page(@ApiIgnore @RequestParam Map<String, Object> params){
    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.modules.*.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: AUTO
      #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
      logic-delete-value: -1
      logic-not-delete-value: 0
      banner: false
  #原生配置
  configuration:
    map-underscore-to-camel-case: true
    cache-enabled: false
    call-setters-on-nulls: true
    jdbc-type-for-null: 'null'
```

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，用于校验用户提交的数据，如下所示：

```
public class ValidatorUtils {

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

    /**
     * 校验对象
     * @param object      待校验对象
     * @param groups      待校验的组
     */
    public static void validateEntity(Object object, Class<?>... groups)
        throws RenException {
        Locale.setDefault(LocaleContextHolder.getLocale());
        Validator validator = Validation.byDefaultProvider().configure().messageInterpolator(
            new ResourceBundleMessageInterpolator(new MessageSourceResourceBundleLocator(
                getMessageSource())))
            .buildValidatorFactory().getValidator();

        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

```

第5章 项目实战

5.1 需求说明

5.2 代码生成器

5.1 需求说明

我们来完成一个商品的列表、添加、修改、删除功能，熟悉如何快速开发自己的业务功能模块。

- 我们先建一个商品表tb_goods，表结构如下所示：

```
CREATE TABLE tb_goods (  
  id                bigint NOT NULL COMMENT '商品ID',  
  name              varchar(50) COMMENT '商品名',  
  intro             varchar(500) COMMENT '介绍',  
  price             int COMMENT '价格',  
  num               int COMMENT '数量',  
  creator           bigint COMMENT '创建者',  
  create_date       datetime COMMENT '创建时间',  
  PRIMARY KEY (id)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='商品管理';
```

5.2 代码生成器

- 使用代码生成器前，我们先来看下代码生成器的配置，看看那些是可配置的，打开renren-generator模块的配置文件generator.properties，如下所示：

```
#代码生成器，配置信息

mainPath=io.renren
#包名
package=io.renren.modules
moduleName=demo
#作者
author=Mark
#Email
email=sunlightcs@gmail.com
#表前缀(类名不会包含表前缀)
tablePrefix=tb_

#类型转换，配置信息
tinyint=Integer
smallint=Integer
mediumint=Integer
int=Integer
integer=Integer
bigint=Long
float=Float
double=Double
decimal=BigDecimal
bit=Boolean

char=String
varchar=String
tinytext=String
text=String
mediumtext=String
longtext=String

date=Date
datetime=Date
timestamp=Date

NUMBER=Integer
INT=Integer
INTEGER=Integer
BINARY_INTEGER=Integer
LONG=String
FLOAT=Float
BINARY_FLOAT=Float
```

```

DOUBLE=Double
BINARY_DOUBLE=Double
DECIMAL=BigDecimal
CHAR=String
VARCHAR=String
VARCHAR2=String
NVARCHAR=String
NVARCHAR2=String
CLOB=String
BLOB=String
DATE=Date
DATETIME=Date
TIMESTAMP=Date
TIMESTAMP(6)=Date

int8=Long
int4=Integer
int2=Integer
numeric=BigDecimal

```

上面的配置文件，可以配置包名、作者信息、表前缀、模块名称、类型转换等信息。其中，类型转换是指，MySQL中的类型与JavaBean中的类型，是怎么一个对应关系。如果有缺少的类型，可自行在generator.properties文件中补充。

- 再看看renren-generator模块的application.yml配置文件，我们只要修改数据库名、账号、密码，就可以了。其中，数据库名是指待生成的表，所在的数据库。

```

# Tomcat
server:
  port: 8082
  servlet:
    context-path: /renren-generator

spring:
  datasource:
    type: com.alibaba.druid.pool.DruidDataSource
    #MySQL配置
    driverClassName: com.mysql.jdbc.Driver
    url: jdbc:mysql://localhost:3306/security_enterprise?useUnicode=true&characterEncoding=UTF-8&useSSL=false
    username: renren
    password: 123456
    #oracle配置
    # driverClassName: oracle.jdbc.OracleDriver
    # url: jdbc:oracle:thin:@192.168.10.10:1521:renren
    # username: renren
    # password: 123456
    #SQLServer配置
    # driverClassName: com.microsoft.sqlserver.jdbc.SQLServerDriver
    # url: jdbc:sqlserver://192.168.10.10:1433;DatabaseName=security_enterprise
    # username: sa

```

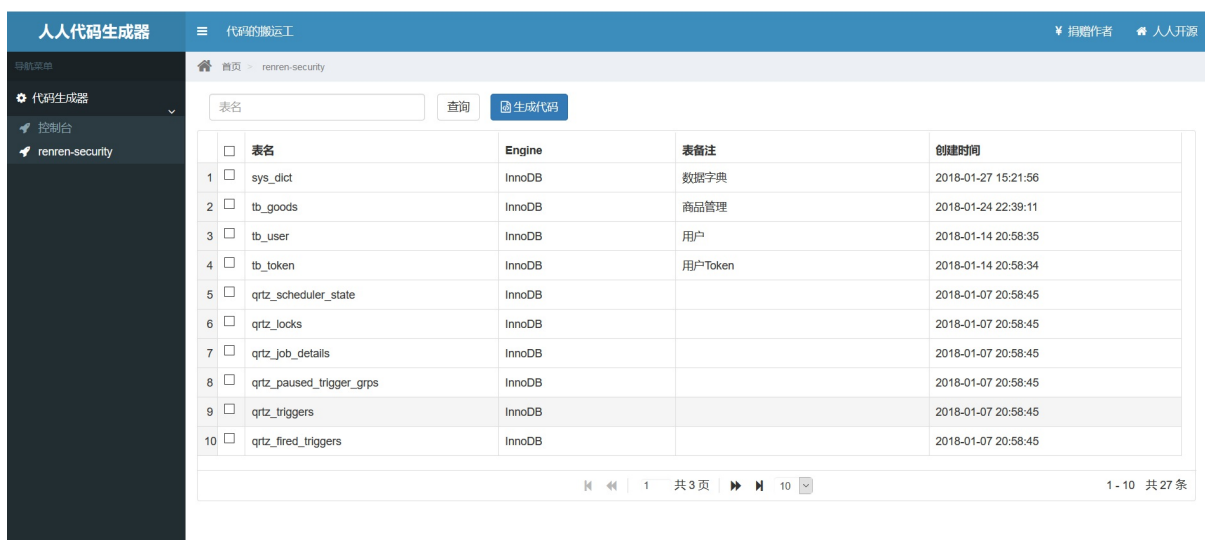
```
# password: 123456
#PostgreSQL配置
# driverClassName: org.postgresql.Driver
# url: jdbc:postgresql://192.168.10.10:5432/security_enterprise
# username: renren
# password: 123456
jackson:
  time-zone: GMT+8
  date-format: yyyy-MM-dd HH:mm:ss
resources:
  static-locations: classpath:/static/,classpath:/views/

mybatis:
  mapperLocations: classpath:mapper/**/*.xml

pagehelper:
  reasonable: true
  supportMethodsArguments: true
  params: count=countSql

#指定数据库, 可选值有【mysql、oracle、sqlserver、postgresql】
renren:
  database: mysql
```

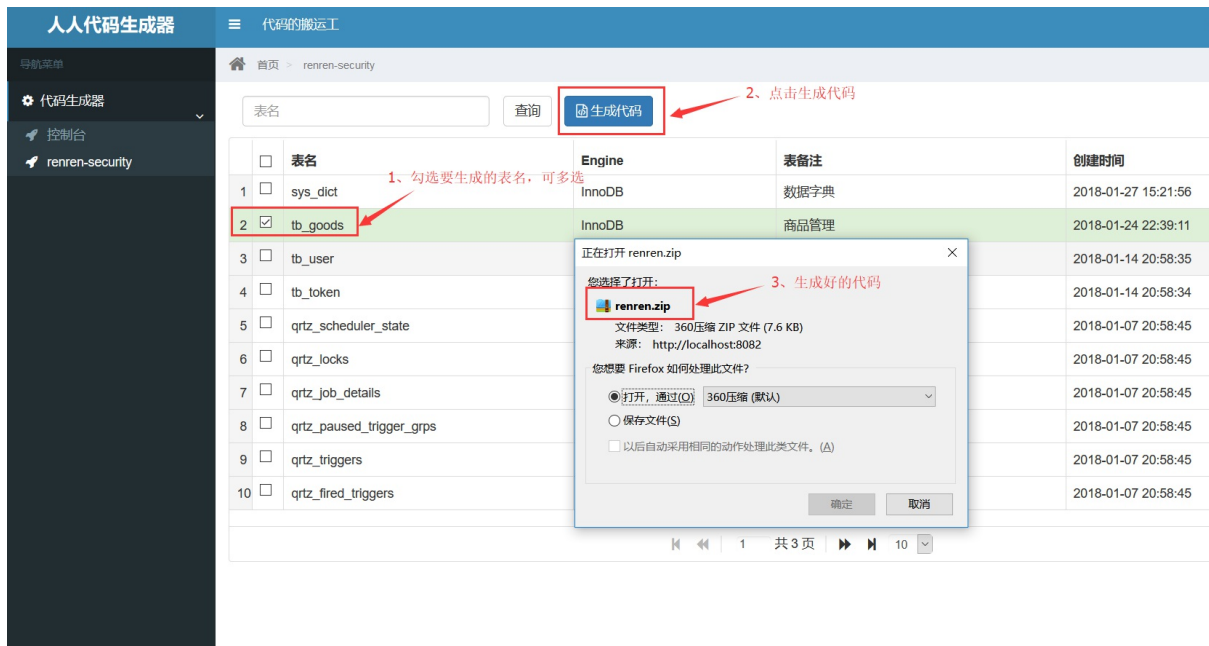
- 在数据库security_enterprise中, 执行建表语句, 创建tb_goods表, 再启动renren-generator项目, 运行GeneratorApplication.java的main方法即可
- 项目访问路径: <http://localhost:8082/renren-generator>
- 在浏览器里输入项目地址, 如下所示:



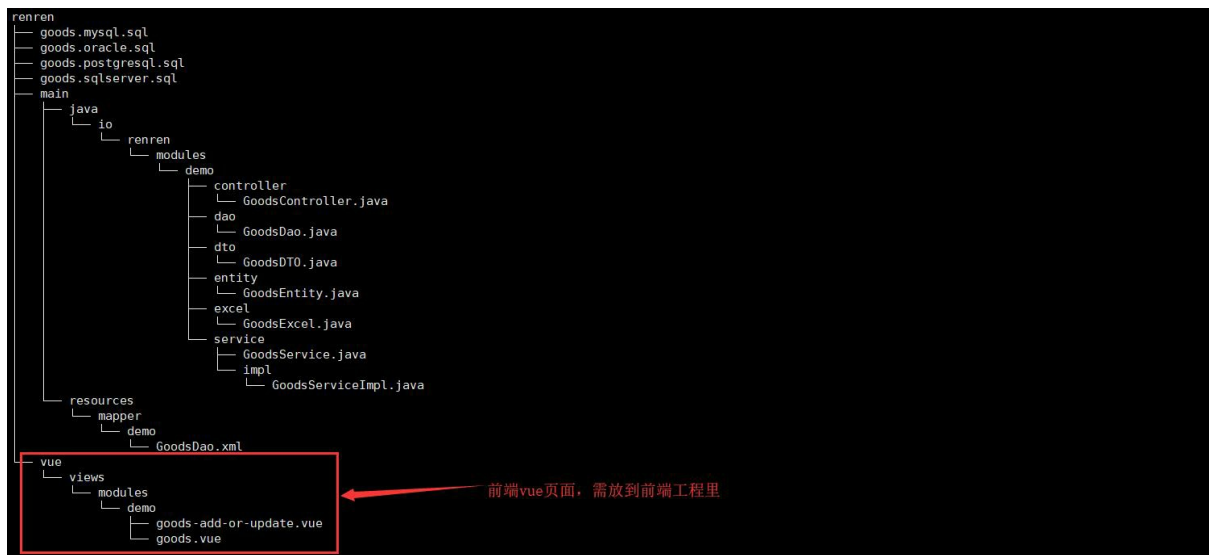
The screenshot shows the 'Renren-Generator' web application. On the left is a sidebar with navigation links. The main area displays a table of database tables. The table has the following columns: 'Table Name', 'Engine', 'Table Comment', and 'Creation Time'. The table 'tb_goods' is selected, and the 'Generate Code' button is highlighted.

	表名	Engine	表备注	创建时间
1	<input type="checkbox"/> sys_dict	InnoDB	数据字典	2018-01-27 15:21:56
2	<input checked="" type="checkbox"/> tb_goods	InnoDB	商品管理	2018-01-24 22:39:11
3	<input type="checkbox"/> tb_user	InnoDB	用户	2018-01-14 20:58:35
4	<input type="checkbox"/> tb_token	InnoDB	用户Token	2018-01-14 20:58:34
5	<input type="checkbox"/> qrtz_scheduler_state	InnoDB		2018-01-07 20:58:45
6	<input type="checkbox"/> qrtz_locks	InnoDB		2018-01-07 20:58:45
7	<input type="checkbox"/> qrtz_job_details	InnoDB		2018-01-07 20:58:45
8	<input type="checkbox"/> qrtz_paused_trigger_grps	InnoDB		2018-01-07 20:58:45
9	<input type="checkbox"/> qrtz_triggers	InnoDB		2018-01-07 20:58:45
10	<input type="checkbox"/> qrtz_fired_triggers	InnoDB		2018-01-07 20:58:45

- 我们只需勾选tb_goods, 点击【生成代码】按钮, 则可生成相应代码, 如下所示:



- 我们来看下生成的代码结构，如下所示：



- 生成好代码后，我们只需在数据库security_enterprise中，执行goods.mysql.sql语句(MySQL数据库)，这个SQL是生成菜单的，SQL语句如下所示：

```
-- 菜单初始SQL
INSERT INTO sys_menu(id, pid, url, permissions, type, icon, sort, creator, create_date, updater, update_date)VALUES (1098209998393167873, 1067246875800000002, 'demo/goods', NULL, 0, 'icon-desktop', 0, 1067246875800000001, now(), 1067246875800000001, now());
INSERT INTO sys_menu(id, pid, url, permissions, type, icon, sort, creator, create_date, updater, update_date) VALUES (1098209998393167874, 1098209998393167873, NULL, 'demo:goods:page,demo:goods:info', 1, NULL, 0, 1067246875800000001, now(), 1067246875800000001, now());
INSERT INTO sys_menu(id, pid, url, permissions, type, icon, sort, creator, create_date, updater, update_date) VALUES (1098209998393167875, 1098209998393167873, NULL, 'demo:goods:save', 1, NULL, 1, 1067246875800000001, now(), 1067246875800000001, now());
INSERT INTO sys_menu(id, pid, url, permissions, type, icon, sort, creator, create_date, updater, update_date) VALUES (1098209998393167876, 1098209998393167873, NULL, 'demo:goods:update',
```

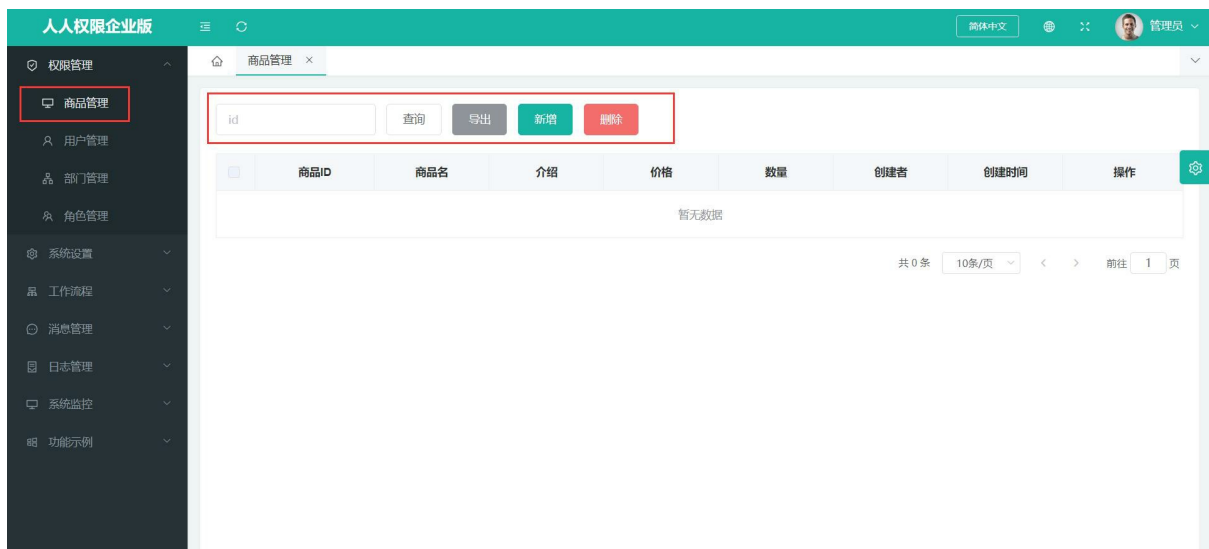
```

1, NULL, 2, 1067246875800000001, now(), 1067246875800000001, now());
INSERT INTO sys_menu(id, pid, url, permissions, type, icon, sort, creator, create_date, update_date) VALUES (1098209998393167877, 1098209998393167873, NULL, 'demo:goods:delete', 1, NULL, 3, 1067246875800000001, now(), 1067246875800000001, now());
INSERT INTO sys_menu(id, pid, url, permissions, type, icon, sort, creator, create_date, update_date) VALUES (1098209998393167878, 1098209998393167873, NULL, 'demo:goods:export', 1, NULL, 4, 1067246875800000001, now(), 1067246875800000001, now());

-- 菜单国际化初始SQL
INSERT INTO sys_language(table_name, table_id, field_name, field_value, language) VALUES ('sys_menu', 1098209998393167873, 'name', '商品管理', 'en-US');
INSERT INTO sys_language(table_name, table_id, field_name, field_value, language) VALUES ('sys_menu', 1098209998393167873, 'name', '商品管理', 'zh-CN');
INSERT INTO sys_language(table_name, table_id, field_name, field_value, language) VALUES ('sys_menu', 1098209998393167873, 'name', '商品管理', 'zh-TW');
INSERT INTO sys_language(table_name, table_id, field_name, field_value, language) VALUES ('sys_menu', 1098209998393167874, 'name', 'View', 'en-US');
INSERT INTO sys_language(table_name, table_id, field_name, field_value, language) VALUES ('sys_menu', 1098209998393167874, 'name', '查看', 'zh-CN');
INSERT INTO sys_language(table_name, table_id, field_name, field_value, language) VALUES ('sys_menu', 1098209998393167874, 'name', '查看', 'zh-TW');
INSERT INTO sys_language(table_name, table_id, field_name, field_value, language) VALUES ('sys_menu', 1098209998393167875, 'name', 'Add', 'en-US');
INSERT INTO sys_language(table_name, table_id, field_name, field_value, language) VALUES ('sys_menu', 1098209998393167875, 'name', '新增', 'zh-CN');
INSERT INTO sys_language(table_name, table_id, field_name, field_value, language) VALUES ('sys_menu', 1098209998393167875, 'name', '新增', 'zh-TW');
INSERT INTO sys_language(table_name, table_id, field_name, field_value, language) VALUES ('sys_menu', 1098209998393167876, 'name', 'Edit', 'en-US');
INSERT INTO sys_language(table_name, table_id, field_name, field_value, language) VALUES ('sys_menu', 1098209998393167876, 'name', '修改', 'zh-CN');
INSERT INTO sys_language(table_name, table_id, field_name, field_value, language) VALUES ('sys_menu', 1098209998393167876, 'name', '修改', 'zh-TW');
INSERT INTO sys_language(table_name, table_id, field_name, field_value, language) VALUES ('sys_menu', 1098209998393167877, 'name', 'Delete', 'en-US');
INSERT INTO sys_language(table_name, table_id, field_name, field_value, language) VALUES ('sys_menu', 1098209998393167877, 'name', '删除', 'zh-CN');
INSERT INTO sys_language(table_name, table_id, field_name, field_value, language) VALUES ('sys_menu', 1098209998393167877, 'name', '删除', 'zh-TW');
INSERT INTO sys_language(table_name, table_id, field_name, field_value, language) VALUES ('sys_menu', 1098209998393167878, 'name', 'Export', 'en-US');
INSERT INTO sys_language(table_name, table_id, field_name, field_value, language) VALUES ('sys_menu', 1098209998393167878, 'name', '导出', 'zh-CN');
INSERT INTO sys_language(table_name, table_id, field_name, field_value, language) VALUES ('sys_menu', 1098209998393167878, 'name', '导出', 'zh-TW');

```

- 接下来，再把刚生成的后端代码，添加到项目renren-admin里，前端vue代码，添加到前端项目里，在启动renren-admin项目，如下所示：



- 现在，我们就可以新增、修改、删除等操作

新增

* 商品名

* 介绍

* 价格

* 数量

商品管理

id 查询 导出 新增 删除

<input type="checkbox"/>	商品ID	商品名	介绍	价格	数量	创建者	创建时间	操作
<input type="checkbox"/>	1098220647387029506	电饭煲	美的 (Midea) 电饭煲电饭锅3LH加热迷你电饭煲智能预约触摸操控精钢厚釜内胆	399	100	1067246875800000001	2019-02-20 21:59:41	修改 删除

共 1 条 10条/页 1 前往 1 页

- 还可以导出Excel表格，如下所示

2019-02-20 (2).xls [兼容模式] - Excel														
文件 开始 插入 页面布局 公式 数据 审阅 视图 帮助 操作说明搜索														
<div> <div>剪贴板</div> <div> 剪贴 复制 格式刷 </div> </div> <div> 字体 自动换行 文本 条件格式 套用 单元格式 </div> <div> 数字 样式 </div>														
C2 美的 (Midea) 电饭煲电饭锅3LH加热迷你电饭煲智能预约触摸操控精钢厚釜内胆														
	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	商品ID	商品名	介绍	价格	数量	创建者	创建时间							
2	109822064	电饭煲	美的 (Midea)	399	100	106724687580000000	Wed Feb 20							
3														
4														
5														
6														
7														
8														
9														
10														
11														
12														
13														
14														
15														
16														
17														

第6章 后端源码分析

6.1 功能模块移除

6.2 前后端分离

6.3 功能权限设计

6.4 数据权限设计

6.5 数据权限使用

6.6 XSS脚本过滤

6.7 Redis缓存

6.8 异常处理机制

6.9 操作日志

6.10 定时任务模块

6.11 API模块

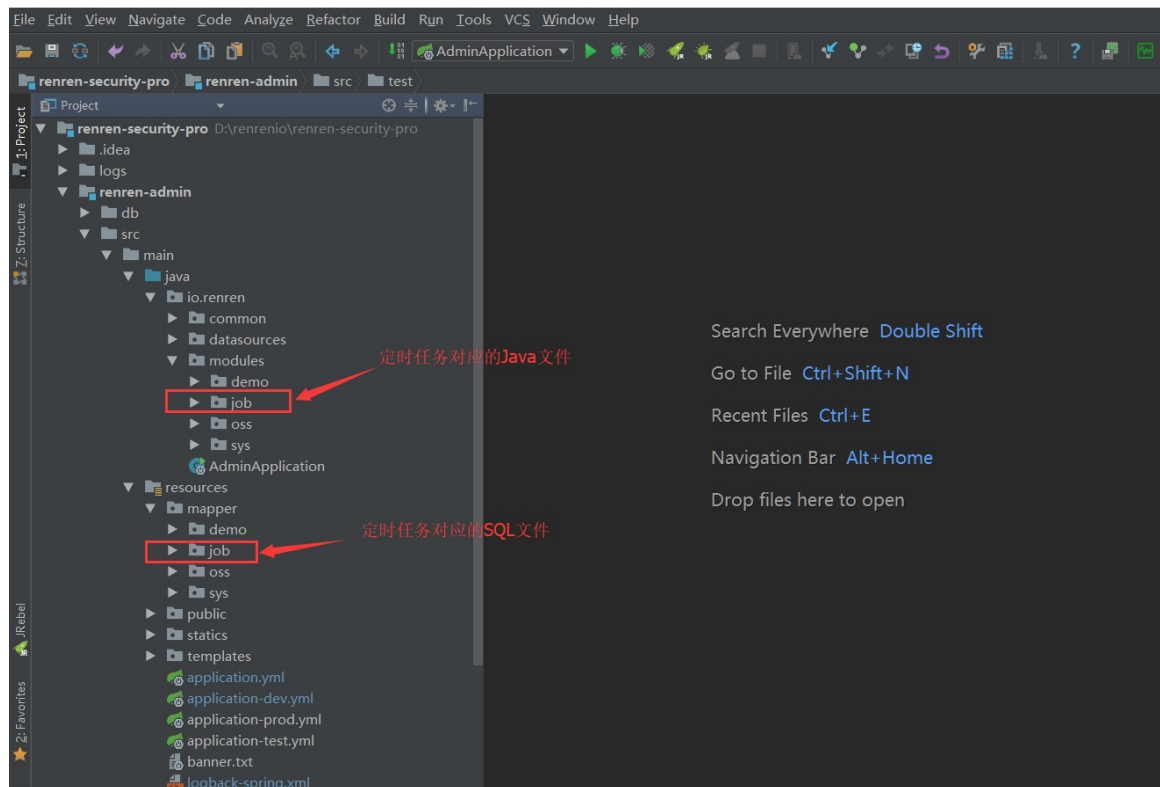
6.12 workflow模块

6.1 功能模块移除

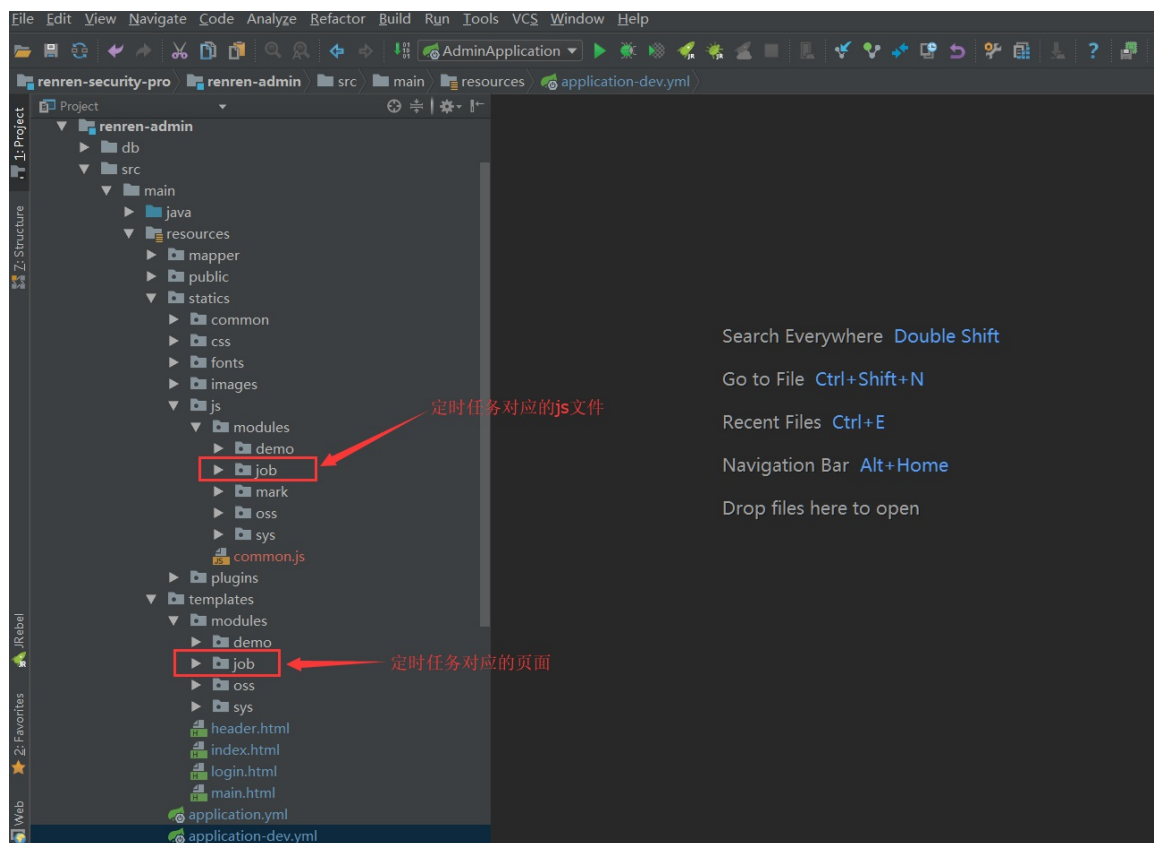
项目已包含用户管理、角色管理、部门管理、菜单管理、定时任务、参数管理、字典管理、文件上传、系统日志、文章管理、APP模块等功能，有些功能点，可能不需要，怎么才能移除掉不要的功能点呢？

项目结构是按模块划分的，功能点代码都在modules目录下面，所以，想要移除某个功能，只需删除modules目录对应的代码即可；比如，定时任务在我们的系统里，是不需要的，我们想移除掉，只需进行如下操作

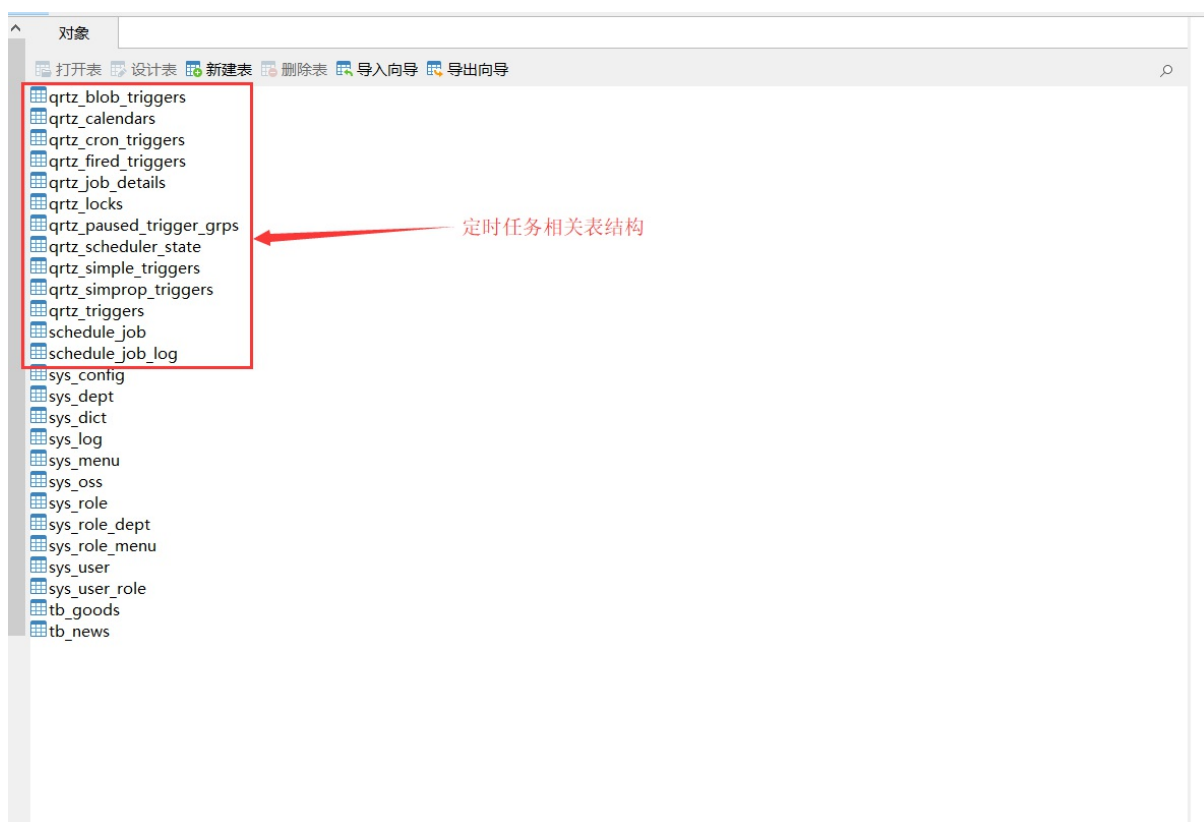
- 步骤1，删除定时任务的Java文件及SQL文件，如下图



- 步骤2，删除定时任务的js文件及页面，如下图



- 步骤3, 删除对应的表结构, 如下图



- 步骤4, 删除对应的菜单, 如下图

人人权限企业版

简体(中文)

管理员

权限管理

系统设置

菜单管理

参数管理

字典管理

定时任务

文件上传

工作流程

消息管理

日志管理

系统监控

功能示例

菜单管理

定时任务		菜单	3	job/schedule		修改删除
查看		按钮	0		sys:schedule:pag...	修改删除
新增		按钮	1		sys:schedule:save	修改删除
修改		按钮	2		sys:schedule:update	修改删除
删除		按钮	3		sys:schedule:delete	修改删除
暂停		按钮	4		sys:schedule:pause	修改删除
恢复		按钮	5		sys:schedule:resu...	修改删除
立即执行		按钮	6		sys:schedule:run	修改删除
日志列表		按钮	7		sys:schedule:log	修改删除
文件上传		菜单	4	oss/oss	sys:oss:all	修改删除
工作流程		菜单	2			修改删除

6.2 前后端分离

要实现前后端分离，需要考虑以下2个问题：

1. 项目不再基于session了，如何知道访问者是谁？
 2. 如何确认访问者的权限？
- 前后端分离，一般都是通过token实现，本项目也是一样；用户登录时，生成token及token过期时间，token与用户是一一对应关系，调用接口的时候，把token放到header或请求参数中，服务端就知道是谁在调用接口，登录如下所示：

```
/**
 * 验证码
 */
@GetMapping("captcha")
@ApiOperation(value = "验证码", produces="application/octet-stream")
@ApiImplicitParam(paramType = "query", dataType="string", name = "uuid", required = true)
public void captcha(HttpServletResponse response, String uuid)throws IOException {
    //uuid不能为空
    AssertUtils.isBlank(uuid, ErrorCode.IDENTIFIER_NOT_NULL);

    //生成图片验证码
    BufferedImage image = captchaService.create(uuid);

    response.setHeader("Cache-Control", "no-store, no-cache");
    response.setContentType("image/jpeg");
    ServletOutputStream out = response.getOutputStream();
    ImageIO.write(image, "jpg", out);
    out.close();
}

@PostMapping("login")
@ApiOperation(value = "登录")
public Result login(HttpServletRequest request, @RequestBody LoginDTO login) {
    //效验数据
    ValidatorUtils.validateEntity(login);

    //验证码是否正确
    boolean flag = captchaService.validate(login.getUuid(), login.getCaptcha());
    if(!flag){
        return new Result().error(ErrorCode.CAPTCHA_ERROR);
    }

    //用户信息
    SysUserDTO user = sysUserService.getByUsername(login.getUsername());

    SysLogLoginEntity log = new SysLogLoginEntity();
    log.setOperation(LoginOperationEnum.LOGIN.value());
}
```

```

log.setCreateDate(new Date());
log.setIp(IpUtils.getIpAddr(request));
log.setUserAgent(request.getHeader(HttpHeaders.USER_AGENT));
log.setIp(IpUtils.getIpAddr(request));

//用户不存在
if(user == null){
    log.setStatus(LoginStatusEnum.FAIL.value());
    log.setCreatorName(login.getUsername());
    sysLogLoginService.save(log);

    throw new RuntimeException(ErrorCode.ACCOUNT_PASSWORD_ERROR);
}

//密码错误
if(!PasswordUtils.matches(login.getPassword(), user.getPassword())){
    log.setStatus(LoginStatusEnum.FAIL.value());
    log.setCreator(user.getId());
    log.setCreatorName(user.getUsername());
    sysLogLoginService.save(log);

    throw new RuntimeException(ErrorCode.ACCOUNT_PASSWORD_ERROR);
}

//账号停用
if(user.getStatus() == UserStatusEnum.DISABLE.value()){
    log.setStatus(LoginStatusEnum.LOCK.value());
    log.setCreator(user.getId());
    log.setCreatorName(user.getUsername());
    sysLogLoginService.save(log);

    throw new RuntimeException(ErrorCode.ACCOUNT_DISABLE);
}

//登录成功
log.setStatus(LoginStatusEnum.SUCCESS.value());
log.setCreator(user.getId());
log.setCreatorName(user.getUsername());
sysLogLoginService.save(log);

return sysUserTokenService.createToken(user.getId());
}

```

- 调用接口时，接受传过来的token后，如何保证token有效及用户权限呢？其实，shiro提供了AuthenticatingFilter抽象类，继承AuthenticatingFilter抽象类即可。

步骤1，所有请求全部拒绝访问

```

@Override
protected boolean isAccessAllowed(ServletRequest request, ServletResponse response, Object mappedValue) {
    return false;
}

```

```
}
```

步骤2，拒绝访问的请求，会调用onAccessDenied方法，onAccessDenied方法先获取token，再调用executeLogin方法

```
@Override
protected boolean onAccessDenied(ServletRequest request, ServletResponse response) throws Exception {
    //获取请求token，如果token不存在，直接返回401
    String token = getRequestToken((HttpServletRequest) request);
    if(StringUtils.isBlank(token)){
        HttpServletResponse httpResponse = (HttpServletResponse) response;
        String json = new Gson().toJson(R.error(HttpStatus.SC_UNAUTHORIZED, "invalid token"));

        httpResponse.getWriter().print(json);

        return false;
    }

    return executeLogin(request, response);
}

/**
 * 获取请求的token
 */
private String getRequestToken(HttpServletRequest httpRequest){
    //从header中获取token
    String token = httpRequest.getHeader("token");

    //如果header中不存在token，则从参数中获取token
    if(StringUtils.isBlank(token)){
        token = httpRequest.getParameter("token");
    }

    return token;
}
```

步骤3，阅读AuthenticatingFilter抽象类中executeLogin方法，我们发现调用了 `subject.login(token)`，这是shiro的登录方法，且需要token参数，我们自定义OAuth2Token类，只要实现AuthenticationToken接口，就可以了

```
//AuthenticatingFilter类中的方法
protected boolean executeLogin(ServletRequest request, ServletResponse response) throws Exception {
    AuthenticationToken token = createToken(request, response);
    if (token == null) {
        String msg = "createToken method implementation returned null. A valid non-null AuthenticationToken " +
            "must be created in order to execute a login attempt.";
        throw new IllegalStateException(msg);
    }
}
```

```

    }
    try {
        Subject subject = getSubject(request, response);
        subject.login(token);
        return onLoginSuccess(token, subject, request, response);
    } catch (AuthenticationException e) {
        return onLoginFailure(token, e, request, response);
    }
}

//OAuth2Filter类中的方法，继承了AuthenticatingFilter类
@Override
protected AuthenticationToken createToken(ServletRequest request, ServletResponse response) throws Exception {
    //获取请求token
    String token = getRequestToken((HttpServletRequest) request);

    if(StringUtils.isBlank(token)){
        return null;
    }

    return new OAuth2Token(token);
}

//subject.login(token)中的token对象，需要实现AuthenticationToken接口
public class OAuth2Token implements AuthenticationToken {
    private String token;

    public OAuth2Token(String token){
        this.token = token;
    }

    @Override
    public String getPrincipal() {
        return token;
    }

    @Override
    public Object getCredentials() {
        return token;
    }
}

```

步骤4，定义OAuth2Realm类，并继承AuthorizingRealm抽象类，调用 `subject.login(token)` 时，则会调用 `doGetAuthenticationInfo` 方法，进行登录

```

@Override
@Override
protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) throws AuthenticationException {
    String accessToken = (String) token.getPrincipal();

```

```

        //根据accessToken, 查询用户信息
        SysUserTokenEntity tokenEntity = shiroService.getByToken(accessToken);
        //token失效
        if(tokenEntity == null || tokenEntity.getExpireDate().getTime() < System.currentTimeMillis()){
            throw new IncorrectCredentialsException(MessageUtils.getMessage(ErrorCode.TOKEN_INVALID));
        }

        //查询用户信息
        SysUserEntity userEntity = shiroService.getUser(tokenEntity.getUserId());

        //转换成UserDetail对象
        UserDetail userDetail = ConvertUtils.sourceToTarget(userEntity, UserDetail.class);

        //获取用户对应的部门数据权限
        List<Long> deptIdList = shiroService.getDataScopeList(userDetail.getId());
        userDetail.setDeptIdList(deptIdList);

        //账号锁定
        if(userDetail.getStatus() == 0){
            throw new LockedAccountException(MessageUtils.getMessage(ErrorCode.ACCOUNT_LOCK));
        }

        SimpleAuthenticationInfo info = new SimpleAuthenticationInfo(userDetail, accessToken, getName());
        return info;
    }

```

步骤5，登录失败后，则调用onLoginFailure，进行失败处理，整个流程结束

```

@Override
protected boolean onLoginFailure(AuthenticationToken token, AuthenticationException e, ServletRequest request, ServletResponse response) {
    HttpServletResponse httpResponse = (HttpServletResponse) response;
    httpResponse.setContentType("application/json;charset=utf-8");
    httpResponse.setHeader("Access-Control-Allow-Credentials", "true");
    httpResponse.setHeader("Access-Control-Allow-Origin", HttpContextUtils.getOrigin());
    try {
        //处理登录失败的异常
        Throwable throwable = e.getCause() == null ? e : e.getCause();
        Result r = new Result().error(HttpStatus.SC_UNAUTHORIZED, throwable.getMessage());

        String json = new Gson().toJson(r);
        httpResponse.getWriter().print(json);
    } catch (IOException e1) {
    }
}

```

```
        return false;
    }
```

步骤6，登录成功后，则调用doGetAuthorizationInfo方法，查询用户的权限，再调用具体的接口，整个流程结束

```
@Override
protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {
    UserDetails user = (UserDetails)principals.getPrimaryPrincipal();

    //用户权限列表
    Set<String> permsSet = shiroService.getUserPermissions(user);

    SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();
    info.setStringPermissions(permsSet);
    return info;
}
```

6.3 功能权限设计

权限相关的表结构，如下图所示

用户管理	部门管理	菜单管理																																																																																																																								
<table><tr><td>id</td><td>varchar(32)</td><td><pk></td></tr><tr><td>用户名</td><td>varchar(50)</td><td><ak1></td></tr><tr><td>密码</td><td>varchar(100)</td><td></td></tr><tr><td>姓名</td><td>varchar(50)</td><td></td></tr><tr><td>头像</td><td>varchar(200)</td><td></td></tr><tr><td>性别</td><td>tinyint(4)</td><td></td></tr><tr><td>邮箱</td><td>varchar(100)</td><td></td></tr><tr><td>手机号</td><td>varchar(20)</td><td></td></tr><tr><td>部门ID</td><td>varchar(32)</td><td></td></tr><tr><td>超级管理员</td><td>tinyint</td><td></td></tr><tr><td>状态</td><td>tinyint(4)</td><td></td></tr><tr><td>备注</td><td>varchar(200)</td><td></td></tr><tr><td>删除标识</td><td>tinyint(4)</td><td><ak2></td></tr><tr><td>创建者</td><td>varchar(32)</td><td></td></tr><tr><td>创建时间</td><td>datetime</td><td><ak3></td></tr><tr><td>更新者</td><td>varchar(32)</td><td></td></tr><tr><td>更新时间</td><td>datetime</td><td></td></tr></table>	id	varchar(32)	<pk>	用户名	varchar(50)	<ak1>	密码	varchar(100)		姓名	varchar(50)		头像	varchar(200)		性别	tinyint(4)		邮箱	varchar(100)		手机号	varchar(20)		部门ID	varchar(32)		超级管理员	tinyint		状态	tinyint(4)		备注	varchar(200)		删除标识	tinyint(4)	<ak2>	创建者	varchar(32)		创建时间	datetime	<ak3>	更新者	varchar(32)		更新时间	datetime		<table><tr><td>id</td><td>varchar(32)</td><td><pk></td></tr><tr><td>上级ID</td><td>varchar(32)</td><td><ak></td></tr><tr><td>所有上级ID</td><td>varchar(500)</td><td></td></tr><tr><td>部门名称</td><td>varchar(50)</td><td></td></tr><tr><td>排序</td><td>int</td><td></td></tr><tr><td>删除标识</td><td>tinyint(4)</td><td><ak></td></tr><tr><td>创建者</td><td>varchar(32)</td><td></td></tr><tr><td>创建时间</td><td>datetime</td><td><ak></td></tr><tr><td>更新者</td><td>varchar(32)</td><td></td></tr><tr><td>更新时间</td><td>datetime</td><td></td></tr></table>	id	varchar(32)	<pk>	上级ID	varchar(32)	<ak>	所有上级ID	varchar(500)		部门名称	varchar(50)		排序	int		删除标识	tinyint(4)	<ak>	创建者	varchar(32)		创建时间	datetime	<ak>	更新者	varchar(32)		更新时间	datetime		<table><tr><td>id</td><td>varchar(32)</td><td><pk></td></tr><tr><td>上级ID</td><td>varchar(32)</td><td><ak1></td></tr><tr><td>菜单名称</td><td>varchar(50)</td><td></td></tr><tr><td>菜单URL</td><td>varchar(200)</td><td></td></tr><tr><td>类型</td><td>tinyint</td><td></td></tr><tr><td>菜单图标</td><td>varchar(50)</td><td></td></tr><tr><td>权限标识</td><td>varchar(32)</td><td></td></tr><tr><td>排序</td><td>int(11)</td><td></td></tr><tr><td>删除标识</td><td>tinyint(4)</td><td><ak2></td></tr><tr><td>创建者</td><td>varchar(32)</td><td></td></tr><tr><td>创建时间</td><td>datetime</td><td><ak3></td></tr><tr><td>更新者</td><td>varchar(32)</td><td></td></tr><tr><td>更新时间</td><td>datetime</td><td></td></tr></table>	id	varchar(32)	<pk>	上级ID	varchar(32)	<ak1>	菜单名称	varchar(50)		菜单URL	varchar(200)		类型	tinyint		菜单图标	varchar(50)		权限标识	varchar(32)		排序	int(11)		删除标识	tinyint(4)	<ak2>	创建者	varchar(32)		创建时间	datetime	<ak3>	更新者	varchar(32)		更新时间	datetime	
id	varchar(32)	<pk>																																																																																																																								
用户名	varchar(50)	<ak1>																																																																																																																								
密码	varchar(100)																																																																																																																									
姓名	varchar(50)																																																																																																																									
头像	varchar(200)																																																																																																																									
性别	tinyint(4)																																																																																																																									
邮箱	varchar(100)																																																																																																																									
手机号	varchar(20)																																																																																																																									
部门ID	varchar(32)																																																																																																																									
超级管理员	tinyint																																																																																																																									
状态	tinyint(4)																																																																																																																									
备注	varchar(200)																																																																																																																									
删除标识	tinyint(4)	<ak2>																																																																																																																								
创建者	varchar(32)																																																																																																																									
创建时间	datetime	<ak3>																																																																																																																								
更新者	varchar(32)																																																																																																																									
更新时间	datetime																																																																																																																									
id	varchar(32)	<pk>																																																																																																																								
上级ID	varchar(32)	<ak>																																																																																																																								
所有上级ID	varchar(500)																																																																																																																									
部门名称	varchar(50)																																																																																																																									
排序	int																																																																																																																									
删除标识	tinyint(4)	<ak>																																																																																																																								
创建者	varchar(32)																																																																																																																									
创建时间	datetime	<ak>																																																																																																																								
更新者	varchar(32)																																																																																																																									
更新时间	datetime																																																																																																																									
id	varchar(32)	<pk>																																																																																																																								
上级ID	varchar(32)	<ak1>																																																																																																																								
菜单名称	varchar(50)																																																																																																																									
菜单URL	varchar(200)																																																																																																																									
类型	tinyint																																																																																																																									
菜单图标	varchar(50)																																																																																																																									
权限标识	varchar(32)																																																																																																																									
排序	int(11)																																																																																																																									
删除标识	tinyint(4)	<ak2>																																																																																																																								
创建者	varchar(32)																																																																																																																									
创建时间	datetime	<ak3>																																																																																																																								
更新者	varchar(32)																																																																																																																									
更新时间	datetime																																																																																																																									
角色管理	角色用户关系	角色菜单关系																																																																																																																								
<table><tr><td>id</td><td>varchar(32)</td><td><pk></td></tr><tr><td>角色名称</td><td>varchar(32)</td><td></td></tr><tr><td>备注</td><td>varchar(100)</td><td></td></tr><tr><td>删除标识</td><td>tinyint(4)</td><td><ak2></td></tr><tr><td>部门ID</td><td>varchar(32)</td><td><ak1></td></tr><tr><td>创建者</td><td>varchar(32)</td><td></td></tr><tr><td>创建时间</td><td>datetime</td><td><ak3></td></tr><tr><td>更新者</td><td>varchar(32)</td><td></td></tr><tr><td>更新时间</td><td>datetime</td><td></td></tr></table>	id	varchar(32)	<pk>	角色名称	varchar(32)		备注	varchar(100)		删除标识	tinyint(4)	<ak2>	部门ID	varchar(32)	<ak1>	创建者	varchar(32)		创建时间	datetime	<ak3>	更新者	varchar(32)		更新时间	datetime		<table><tr><td>id</td><td>varchar(32)</td><td><pk></td></tr><tr><td>角色ID</td><td>varchar(32)</td><td><ak1></td></tr><tr><td>用户ID</td><td>varchar(32)</td><td><ak2></td></tr><tr><td>创建者</td><td>varchar(32)</td><td></td></tr><tr><td>创建时间</td><td>datetime</td><td></td></tr></table>	id	varchar(32)	<pk>	角色ID	varchar(32)	<ak1>	用户ID	varchar(32)	<ak2>	创建者	varchar(32)		创建时间	datetime		<table><tr><td>id</td><td>varchar(32)</td><td><pk></td></tr><tr><td>角色ID</td><td>varchar(32)</td><td><ak1></td></tr><tr><td>菜单ID</td><td>varchar(32)</td><td><ak2></td></tr><tr><td>创建者</td><td>varchar(32)</td><td></td></tr><tr><td>创建时间</td><td>datetime</td><td></td></tr></table>	id	varchar(32)	<pk>	角色ID	varchar(32)	<ak1>	菜单ID	varchar(32)	<ak2>	创建者	varchar(32)		创建时间	datetime																																																																
id	varchar(32)	<pk>																																																																																																																								
角色名称	varchar(32)																																																																																																																									
备注	varchar(100)																																																																																																																									
删除标识	tinyint(4)	<ak2>																																																																																																																								
部门ID	varchar(32)	<ak1>																																																																																																																								
创建者	varchar(32)																																																																																																																									
创建时间	datetime	<ak3>																																																																																																																								
更新者	varchar(32)																																																																																																																									
更新时间	datetime																																																																																																																									
id	varchar(32)	<pk>																																																																																																																								
角色ID	varchar(32)	<ak1>																																																																																																																								
用户ID	varchar(32)	<ak2>																																																																																																																								
创建者	varchar(32)																																																																																																																									
创建时间	datetime																																																																																																																									
id	varchar(32)	<pk>																																																																																																																								
角色ID	varchar(32)	<ak1>																																																																																																																								
菜单ID	varchar(32)	<ak2>																																																																																																																								
创建者	varchar(32)																																																																																																																									
创建时间	datetime																																																																																																																									
角色数据权限																																																																																																																										
<table><tr><td>id</td><td>varchar(32)</td><td><pk></td></tr><tr><td>角色ID</td><td>varchar(32)</td><td><ak></td></tr><tr><td>部门ID</td><td>varchar(32)</td><td></td></tr><tr><td>创建者</td><td>varchar(32)</td><td></td></tr><tr><td>创建时间</td><td>datetime</td><td></td></tr></table>	id	varchar(32)	<pk>	角色ID	varchar(32)	<ak>	部门ID	varchar(32)		创建者	varchar(32)		创建时间	datetime																																																																																																												
id	varchar(32)	<pk>																																																																																																																								
角色ID	varchar(32)	<ak>																																																																																																																								
部门ID	varchar(32)																																																																																																																									
创建者	varchar(32)																																																																																																																									
创建时间	datetime																																																																																																																									

- 1) [用户管理]表，保存用户相关数据，通过[角色用户关系]表，与[角色管理]表关联；[菜单管理]表通过[角色菜单关系]表，与[角色管理]表关联
- 2) [菜单管理]表，保存菜单相关数据，并在[权限标识]字段里，保存了shiro的权限标识，也就是说，拥有此菜单，就拥有[权限标识]字段里的所有权限，比如，某用户拥有的菜单权限标识 `sys:user:info`，就可以访问下面的方法

```
@GetMapping("{id}")
@RequiresPermissions("sys:user:info")
public Result<SysUserDTO> get(@PathVariable("id") Long id){
    SysUserDTO data = sysUserService.get(id);

    //用户角色列表
    List<Long> roleIdList = sysRoleUserService.getRoleIdList(id);
    data.setRoleIdList(roleIdList);
}
```

```

        return new Result<SysUserDTO>().ok(data);
    }

```

3) [角色数据权限]表，保存角色部门相关数据，数据权限就是根据部门进行过滤的，下一小节具体讲解

4) 在shiro配置代码里，配置为 `anon` 的，表示不经过shiro处理，配置为 `oauth2` 的，表示要经过shiro处理，这样就保证了，没有权限的请求，拒绝访问

```

@Configuration
public class ShiroConfig {

    @Bean
    public DefaultWebSessionManager sessionManager(){
        DefaultWebSessionManager sessionManager = new DefaultWebSessionManager();
        sessionManager.setSessionValidationSchedulerEnabled(false);
        sessionManager.setSessionIdUrlRewritingEnabled(false);

        return sessionManager;
    }

    @Bean("securityManager")
    public SecurityManager securityManager(Oauth2Realm oauth2Realm, SessionManager sessionManager) {
        DefaultWebSecurityManager securityManager = new DefaultWebSecurityManager();
        securityManager.setRealm(oauth2Realm);
        securityManager.setSessionManager(sessionManager);
        securityManager.setRememberMeManager(null);
        return securityManager;
    }

    @Bean("shiroFilter")
    public ShiroFilterFactoryBean shiroFilter(SecurityManager securityManager) {
        ShiroFilterFactoryBean shiroFilter = new ShiroFilterFactoryBean();
        shiroFilter.setSecurityManager(securityManager);

        //oauth过滤
        Map<String, Filter> filters = new HashMap<>();
        filters.put("oauth2", new Oauth2Filter());
        shiroFilter.setFilters(filters);

        Map<String, String> filterMap = new LinkedHashMap<>();
        filterMap.put("/webjars/**", "anon");
        filterMap.put("/druid/**", "anon");
        filterMap.put("/login", "anon");
        filterMap.put("/swagger/**", "anon");
        filterMap.put("/v2/api-docs", "anon");
        filterMap.put("/swagger-ui.html", "anon");
        filterMap.put("/swagger-resources/**", "anon");
        filterMap.put("/service/**", "anon");
        filterMap.put("/editor-app/**", "anon");
        filterMap.put("/diagram-viewer/**", "anon");
    }
}

```

```

        filterMap.put("/modeler.html", "anon");
        filterMap.put("/captcha", "anon");
        filterMap.put("/favicon.ico", "anon");
        filterMap.put("/**", "oauth2");
        shiroFilter.setFilterChainDefinitionMap(filterMap);

        return shiroFilter;
    }

    @Bean("lifecycleBeanPostProcessor")
    public LifecycleBeanPostProcessor lifecycleBeanPostProcessor() {
        return new LifecycleBeanPostProcessor();
    }

    @Bean
    public AuthorizationAttributeSourceAdvisor authorizationAttributeSourceAdvisor(SecurityManager securityManager) {
        AuthorizationAttributeSourceAdvisor advisor = new AuthorizationAttributeSourceAdvisor(
        );
        advisor.setSecurityManager(securityManager);
        return advisor;
    }
}

```

5) 上面的配置，我们可以看出来，只有访问下面这些路径，就不用登录，也就是不会经过shiro处理，其他访问路径，都会经过shiro处理，没有对应的权限，都会拒绝访问。

```

Map<String, String> filterMap = new LinkedHashMap<>();
filterMap.put("/webjars/**", "anon");
filterMap.put("/druid/**", "anon");
filterMap.put("/login", "anon");
filterMap.put("/swagger/**", "anon");
filterMap.put("/v2/api-docs", "anon");
filterMap.put("/swagger-ui.html", "anon");
filterMap.put("/swagger-resources/**", "anon");
filterMap.put("/service/**", "anon");
filterMap.put("/editor-app/**", "anon");
filterMap.put("/diagram-viewer/**", "anon");
filterMap.put("/modeler.html", "anon");
filterMap.put("/captcha", "anon");
filterMap.put("/favicon.ico", "anon");

```

6.4 数据权限设计

本系统采用注解的方式，实现了数据权限的功能，代码更加灵活简洁。在需要数据权限的service方法上，添加 `@DataFilter` 注解，就可以达到数据过滤的功能，也就是我们常说的数据权限。该实现方式，适应绝大多数企业后台管理系统，对数据权限的要求。

- 数据权限是通过 `@DataFilter` 注解实现的，代码如下所示

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface DataFilter {

    /**
     * 表的别名
     */
    String tableAlias() default "";

    /**
     * 查询条件前缀，可选值有：[where、and]
     */
    String prefix() default "";

    /**
     * 用户ID
     */
    String userId() default "creator";

    /**
     * 部门ID
     */
    String deptId() default "dept_id";

}
```

上面的注解类，定义了tableAlias、prefix、userId、deptId及每个方法的作用，这个仅仅是定义，如需赋予功能，还需要编写具体的实现代码。

- 上面定义好了注解，我们再来看下具体的实现，也就是 `@DataFilter` 注解的切面实现类 `DataFilterAspect`，如下所示：

```
@Aspect
@Component
public class DataFilterAspect {

    @Pointcut("@annotation(io.renren.common.annotation.DataFilter)")
    public void dataFilterCut() {

    }

}
```

```

@Before("dataFilterCut()")
public void dataFilter(JoinPoint point) {
    Object params = point.getArgs()[0];
    if(params != null && params instanceof Map){
        UserDetails user = SecurityUser.getUser();

        //如果不是超级管理员，则进行数据过滤
        if(user.getSuperAdmin() == SuperAdminEnum.NO.value()) {
            Map map = (Map)params;
            String sqlFilter = getSqlFilter(user, point);
            map.put(Constant.SQL_FILTER, new DataScope(sqlFilter));
        }

        return ;
    }

    throw new RenException(ErrorCode.DATA_SCOPE_PARAMS_ERROR);
}

/**
 * 获取数据过滤的SQL
 */
private String getSqlFilter(UserDetails user, JoinPoint point){
    MethodSignature signature = (MethodSignature) point.getSignature();
    DataFilter dataFilter = signature.getMethod().getAnnotation(DataFilter.class);
    //获取表的别名
    String tableAlias = dataFilter.tableAlias();
    if(StringUtils.isNotBlank(tableAlias)){
        tableAlias += ".";
    }

    StringBuilder sqlFilter = new StringBuilder();

    //查询条件前缀
    String prefix = dataFilter.prefix();
    if(StringUtils.isNotBlank(prefix)){
        sqlFilter.append(" ").append(prefix);
    }

    sqlFilter.append(" (");

    //部门ID列表
    List<Long> deptIdList = user.getDeptIdList();
    if(CollectionUtil.isNotEmpty(deptIdList)){
        sqlFilter.append(tableAlias).append(dataFilter.deptId());

        sqlFilter.append(" in(").append(StringUtils.join(deptIdList, ",")).append(")");
    }

    //查询本人数据
    if(CollectionUtil.isNotEmpty(deptIdList)){
        sqlFilter.append(" or ");
    }
}

```

```

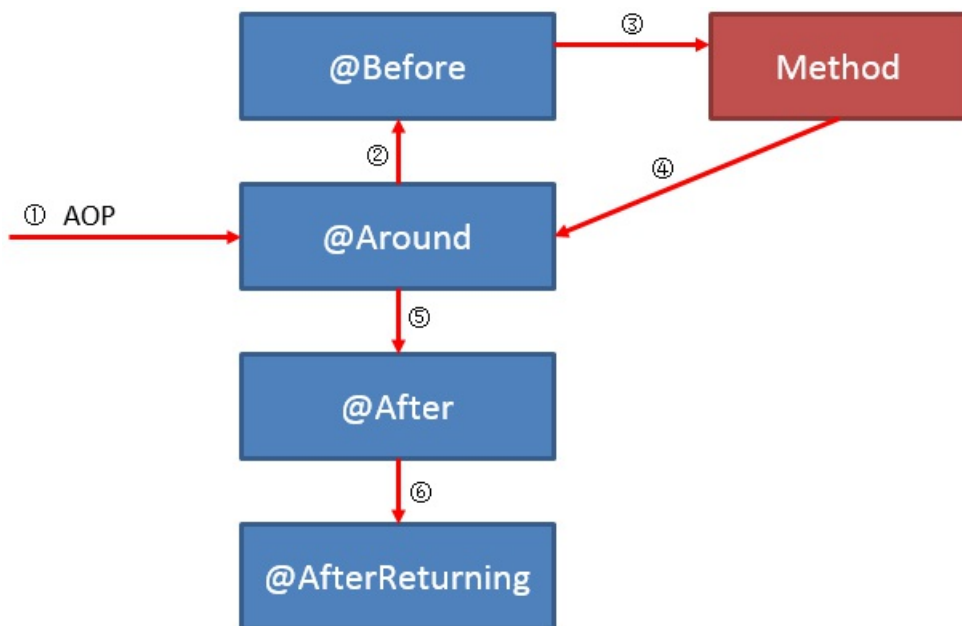
    }
    sqlFilter.append(tableAlias).append(dataFilter.userId()).append("=").append(user.getId());

    sqlFilter.append(")");

    return sqlFilter.toString();
}
}

```

- 上面的实现类中，定义了一个切入点，只要方法上加了 `@DataFilter` 注解，执行该方法之前，会进入 `dataFilter()` 方法，因为使用了Spring AOP的 `@Before`，Spring AOP将按照如下图的顺序执行



- 接下来，我们来具体分析里面的代码，下面的这个方法规定了，第一个参数如果不是Map类型，就直接抛出异常，提示不能使用 `@DataFilter` 注解，如要使用，请使用Map类型的参数，且不能为NULL。参数没问题了，就会判断是不是超级管理员，如果是超级管理员，则直接返回，不再进行下一步操作，也就是说，超级管理员拥有最高权限，能查看所有数据。如果不是超级管理员，则进行数据过滤。

```

@Before("dataFilterCut()")
public void dataFilter(JoinPoint point) {
    Object params = point.getArgs()[0];
    if(params != null && params instanceof Map){
        UserDetails user = SecurityUser.getUser();

        //如果不是超级管理员，则进行数据过滤
        if(user.getSuperAdmin() == SuperAdminEnum.NO.value()) {
            Map map = (Map)params;
            String sqlFilter = getSqlFilter(user, point);
            map.put(Constant.SQL_FILTER, new DataScope(sqlFilter));
        }

        return ;
    }
}

```

```
throw new RenException(ErrorCode.DATA_SCOPE_PARAMS_ERROR);
}
```

- 上面的代码中，可以看到有这么一行 `map.put(Constant.SQL_FILTER, new DataScope(sqlFilter))`，这行的意思，就是把过滤的SQL条件，追加到map参数里，map参数对应的key为 `Constant.SQL_FILTER`，下面是常量定义的代码：

```
public interface Constant {

    /**
     * 数据权限过滤
     */
    String SQL_FILTER = "sqlFilter";

}

/**
 * 超级管理员枚举
 *
 * @author Mark sunlightcs@gmail.com
 */
public enum SuperAdminEnum {
    YES(1),
    NO(0);

    private int value;

    SuperAdminEnum(int value) {
        this.value = value;
    }

    public int value() {
        return this.value;
    }
}
```

- 接下来，看看具体是怎么生成过滤的SQL条件，代码片段如下

```
private String getSqlFilter(UserDetail user, JoinPoint point){
    MethodSignature signature = (MethodSignature) point.getSignature();
    DataFilter dataFilter = signature.getMethod().getAnnotation(DataFilter.class);
    //获取表的别名
    String tableAlias = dataFilter.tableAlias();
    if(StringUtils.isNotBlank(tableAlias)){
        tableAlias += ".";
    }

    StringBuilder sqlFilter = new StringBuilder();
```

```

//查询条件前缀
String prefix = dataFilter.prefix();
if(StringUtils.isNotBlank(prefix)){
    sqlFilter.append(" ").append(prefix);
}

sqlFilter.append(" (");

//部门ID列表
List<Long> deptIdList = user.getDeptIdList();
if(CollUtil.isNotEmpty(deptIdList)){
    sqlFilter.append(tableAlias).append(dataFilter.deptId());

    sqlFilter.append(" in(").append(StringUtils.join(deptIdList, ",")).append(")");
}

//查询本人数据
if(CollUtil.isNotEmpty(deptIdList)){
    sqlFilter.append(" or ");
}
sqlFilter.append(tableAlias).append(dataFilter.userId()).append("=").append(user.getId());

sqlFilter.append(")");

return sqlFilter.toString();
}

```

1) 先判断有没有表的别名，如果有别名，就在别名后追加 `.` ，多表关联查询及过滤数据时，会使用到别名，如下所示：

```

select * from tb_sales_report t1, tb_product t2 where t1.product_id=t2.id

#下面就是过滤的SQL条件，假设sql_filter对应的SQL条件为: t1.dept_id in(1,2)
and ${sql_filter}

```

2) 接下来，就是获取用户的部门ID列表，及用户角色对应的部门

```

//部门ID列表
List<Long> deptIdList = user.getDeptIdList();
if(CollUtil.isNotEmpty(deptIdList)){
    sqlFilter.append(tableAlias).append(dataFilter.deptId());
    //把数据过滤条件组装成dept_id in(1,2,3)
    sqlFilter.append(" in(").append(StringUtils.join(deptIdList, ",")).append(")");
}

```

3) 如下代码片段，就是把数据过滤条件组装成dept_id in(1,2,3)

```

StringBuilder sqlFilter = new StringBuilder();
sqlFilter.append(" (");

```

```

if(deptIdList.size() > 0){
    sqlFilter.append(tableAlias).append(dataFilter.deptId()).append(" in(").append(StringUtil
s.join(deptIdList, ",")).append(")");
}

```

4) 如下代码片段，表示没有任何部门数据权限，也能查看自己的数据

```

//查询本人数据
if(CollUtil.isEmpty(deptIdList)){
    sqlFilter.append(" or ");
}
sqlFilter.append(tableAlias).append(dataFilter.userId()).append("=").append(user.getId());

```

5) 上面的代码，就生成了数据过滤的SQL片段，我们只需在方法上，添加注解 `@DataFilter`，就可以了，如下所示：

```

@DataFilter(prefix = "and")
public PageData<NewsDTO> page(Map<String, Object> params) {
    paramsToLike(params, "title");

    //分页
    IPage<NewsEntity> page = getPage(params, Constant.CREATE_DATE, false);

    //查询
    List<NewsEntity> list = baseDao.getList(params);

    return getPageData(list, page.getTotal(), NewsDTO.class);
}

```

6) 下面演示使用mybatis-plus怎么进行数据过滤

```

/**
 * mybatis-plus数据权限演示
 */
@Override
@DataFilter
public PageData<NewsDTO> page(Map<String, Object> params) {
    IPage<NewsEntity> page = baseDao.selectPage(
        getPage(params, Constant.CREATE_DATE, false),
        getWrapper(params)
    );
    return getPageData(page, NewsDTO.class);
}

private QueryWrapper<NewsEntity> getWrapper(Map<String, Object> params){
    String title = (String)params.get("title");
    String startDate = (String)params.get("startDate");
    String endDate = (String)params.get("endDate");
}

```

```

        QueryWrapper<NewsEntity> wrapper = new QueryWrapper<>();
        wrapper.like(StringUtils.isNotBlank(title), "title", title);
        wrapper.ge(StringUtils.isNotBlank(startDate), "pub_date", startDate);
        wrapper.le(StringUtils.isNotBlank(endDate), "pub_date", endDate);

        //数据过滤
        wrapper.apply(params.get(Constant.SQL_FILTER) != null, params.get(Constant.SQL_FILTER)
            .toString());

        return wrapper;
    }

```

7) 上面的代码，先判断Map里的 `Constant.SQL_FILTER` 是否为null，不为空则追加进去，这样就可以实现数据过滤了，也就是我们常说的数据权限

```

wrapper.apply(params.get(Constant.SQL_FILTER) != null, params.get(Constant.SQL_FILTER).toString());

```

6.5 数据权限使用

如销售系统，销售经理，可以查看本部门所有销售情况，销售组长，可以查看本小组成员的销售情况，而普通销售，则只能查看自己的销售情况，要实现这样的数据权限，下面就是具体的实现步骤。

- 先创建好销售报表，如下所示：

```
CREATE TABLE tb_sales_report (
  id bigint NOT NULL COMMENT 'id',
  product_id bigint(20) DEFAULT NULL COMMENT '产品ID',
  product_name varchar(50) DEFAULT NULL COMMENT '产品名称',
  sale_id bigint(20) DEFAULT NULL COMMENT '销售ID',
  sale_name varchar(20) DEFAULT NULL COMMENT '销售',
  dept_id bigint(20) DEFAULT NULL COMMENT '销售人员所属部门ID',
  dept_name varchar(20) DEFAULT NULL COMMENT '部门',
  creator bigint COMMENT '创建者',
  create_date datetime COMMENT '创建时间',
  primary key (id)
) ENGINE=InnoDB CHARSET=utf8 COMMENT='销售报表';
```

- 数据权限是通过dept_id、user_id进行数据过滤的，所以销售表里，需要有这2个字段，不然数据权限就无法实现，当然，这2个字段名是可以修改的，下面就是把user_id修改成了sale_id

```
@DataFilter(userId = "sale_id")
```

- 下面就是实现数据权限具体的代码，如下所示：

```
@DataFilter(userId = "sale_id")
public PageData<SalesReportDTO> page(Map<String, Object> params) {
    IPage<SalesReportEntity> page = baseDao.selectPage(
        getPage(params, Constant.CREATE_DATE, false),
        getWrapper(params)
    );

    return getPageData(page, SalesReportDTO.class);
}

private QueryWrapper<SalesReportDTO> getWrapper(Map<String, Object> params){
    QueryWrapper<SalesReportEntity> wrapper = new QueryWrapper<>();

    //数据过滤
    wrapper.apply(params.get(Constant.SQL_FILTER) != null, params.get(Constant.SQL_FILTER).toString());

    return wrapper;
}
```

- 因为mybatis-plus不支持多表关联，如需多表关联，实现数据权限，则只能通过原生的MyBatis实现，且无需添加数据过滤条件，会自动追加过滤条件到SQL后面，如下所示：

```
<select id="queryList" resultType="io.renren.modules.demo.entity.SalesReportEntity">
    select * from tb_sales_report t1, tb_product t2 where t1.product_id = t2.id
</select>
```

- 完成以上代码，代码部分就完成了，接下来就只要配置角色的数据权限了，销售经理所属角色，拥有部门及小组的数据权限，如下：

后台首页 角色管理 × 刷新 常用操作

新增 ×

角色名称 销售经理角色

所属部门 销售部

备注 销售经理

功能权限 ☐ 系统管理 ☒ 功能示例 ☐ 文章管理 ☒ 销售报表

数据权限 ☐ 人人开源集团 ☐ 长沙分公司 ☐ 上海分公司 ☒ 销售部 ☒ 销售一组 ☒ 销售二组

确定

- 销售一组组长所属角色，拥有销售一组的数据权限，如下：

后台首页 角色管理 × 刷新 常用操作

新增 ×

角色名称 销售一组组长

所属部门 销售一组

备注 销售一组

功能权限 ☐ 系统管理 ☒ 功能示例 ☐ 文章管理 ☒ 销售报表

数据权限 ☐ 人人开源集团 ☐ 长沙分公司 ☐ 上海分公司 ☐ 销售部 ☒ 销售一组 ☐ 销售二组

确定

- 销售二组组长所属角色，拥有销售二组的数据权限，如下：

后台首页
角色管理
刷新
常用操作

新增

角色名称

销售二组组长

所属部门

销售二组

备注

销售二组

功能权限

☐ 系统管理
☒ 功能示例

☐ 文章管理
☒ 销售报表

数据权限

☐ 人人开源集团
☐ 长沙分公司
☐ 上海分公司
☐ 技术部
☐ 销售部

☐ 销售一组
☒ 销售二组

确定

- 普通销售所属角色，只能参看本人数据权限，如下：

后台首页
角色管理
刷新
常用操作

新增

角色名称

普通销售角色

所属部门

销售一组

备注

普通销售

功能权限

☐ 系统管理
☒ 功能示例

☐ 文章管理
☒ 销售报表

数据权限

☐ 人人开源集团
☐ 长沙分公司
☐ 上海分公司
☐ 技术部
☐ 销售部

☐ 销售一组
☐ 销售二组

确定

- 添加些测试数据，我们来看看数据权限，是否符合我们的需求

1) 我们用销售经理登录系统，可以查看所有数据，如下所示：

后台首页

销售报表 ×

刷新

常用操作

请输入关键字

查询

新增

批量删除

导出

<input type="checkbox"/>		产品ID	产品名称	销售	部门	创建时间	操作
<input type="checkbox"/>	1	1	MacBook	销售经理	销售部	2018-04-02 03:59:12	<div>编辑删除</div>
<input type="checkbox"/>	2	1	MacBook	销售一组组长	销售一组	2018-04-02 04:40:27	<div>编辑删除</div>
<input type="checkbox"/>	3	1	MacBook	销售二组组长	销售二组	2018-04-02 04:42:19	<div>编辑删除</div>
<input type="checkbox"/>	4	1	MacBook	普通销售(销售一组)	销售一组	2018-04-02 04:43:13	<div>编辑删除</div>

< 1 >

到第 1 页

确定

共 4 条

20 条/页

2) 我们用销售一组组长登录系统，可以查看销售一组数据，如下所示：

后台首页

销售报表 ×

刷新

常用操作

请输入关键字

查询

新增

批量删除

导出

<input type="checkbox"/>		产品ID	产品名称	销售	部门	创建时间	操作
<input type="checkbox"/>	2	1	MacBook	销售一组组长	销售一组	2018-04-02 04:40:27	<div>编辑删除</div>
<input type="checkbox"/>	4	1	MacBook	普通销售(销售一组)	销售一组	2018-04-02 04:43:13	<div>编辑删除</div>

< 1 >

到第 1 页

确定

共 2 条

20 条/页

3) 我们用普通销售(销售一组)登录系统, 只能查看本人数据, 如下所示:

后台首页 销售报表 ×

刷新 常用操作

请输入关键字

查询 新增 批量删除 导出

		产品ID	产品名称	销售	部门	创建时间	操作
<input type="checkbox"/>	4	1	MacBook	普通销售(销售一组)	销售一组	2018-04-02 04:43:13	<div>编辑 删除</div>

< 1 >

到第 1 页 确定

共 1 条 20 条/页

4) 我们用销售二组组长登录系统, 可以查看销售二组数据, 如下所示:

后台首页 销售报表 ×

刷新 常用操作

请输入关键字

查询 新增 批量删除 导出

		产品ID	产品名称	销售	部门	创建时间	操作
<input type="checkbox"/>	3	1	MacBook	销售二组组长	销售二组	2018-04-02 04:42:19	<div>编辑 删除</div>

< 1 >

到第 1 页 确定

共 1 条 20 条/页

- 现在我们就把数据权限功能演示完了。其实, 通过现有的系统, 要实现数据权限, 是一件挺简单的事情。

6.7 Redis缓存

缓存大家都很熟悉，但能否灵活运用，就不一定了。一般设计缓存架构时，我们需要考虑如下几个问题：

1. 查询数据的时候，尽量减少DB查询，DB主要负责写数据
2. 尽量不使用 `LEFT JOIN` 等关联查询，缓存命中率不高，还浪费内存
3. 多使用单表查询，缓存命中率最高
4. 数据库 `insert`、`update`、`delete` 时，同步更新缓存数据
5. 合理运用Redis数据结构，也许有质的飞跃
6. 对于访问量不大的项目，使用缓存只会增加项目的复杂度

本系统采用Redis作为缓存，并可配置是否开启redis缓存，主要还是通过Spring AOP实现的，配置如下所示：

```
redis:
  database: 0
  host: localhost
  port: 6379
  password:      # 密码（默认为空）
  timeout: 6000ms # 连接超时时长（毫秒）
  jedis:
    pool:
      max-active: 1000 # 连接池最大连接数（使用负值表示没有限制）
      max-wait: -1ms   # 连接池最大阻塞等待时间（使用负值表示没有限制）
      max-idle: 10     # 连接池中的最大空闲连接
      min-idle: 5      # 连接池中的最小空闲连接

renren:
  redis:
    open: false #是否开启redis缓存 true开启 false关闭
```

本项目中，使用Redis服务的代码，如下所示：

```
@Service
public class SysParamsServiceImpl extends BaseServiceImpl<SysParamsDao, SysParamsEntity> implements SysParamsService {
    @Autowired
    private SysParamsRedis sysParamsRedis;

    @Override
    public PageData<SysParamsDTO> page(Map<String, Object> params) {
        IPage<SysParamsEntity> page = baseDao.selectPage(
            getPage(params, Constant.CREATE_DATE, false),
            getWrapper(params)
        );

        return getPageData(page, SysParamsDTO.class);
    }
}
```

```

    }

    @Override
    public List<SysParamsDTO> list(Map<String, Object> params) {
        List<SysParamsEntity> entityList = baseDao.selectList(getWrapper(params));

        return ConvertUtils.sourceToTarget(entityList, SysParamsDTO.class);
    }

    private QueryWrapper<SysParamsEntity> getWrapper(Map<String, Object> params){
        String paramCode = (String) params.get("paramCode");

        QueryWrapper<SysParamsEntity> wrapper = new QueryWrapper<>();
        wrapper.eq("param_type", 1);
        wrapper.like(StringUtils.isNotBlank(paramCode), "param_code", paramCode);

        return wrapper;
    }

    @Override
    public SysParamsDTO get(Long id) {
        SysParamsEntity entity = baseDao.selectById(id);

        return ConvertUtils.sourceToTarget(entity, SysParamsDTO.class);
    }

    @Override
    @Transactional(rollbackFor = Exception.class)
    public void save(SysParamsDTO dto) {
        SysParamsEntity entity = ConvertUtils.sourceToTarget(dto, SysParamsEntity.class);
        insert(entity);

        sysParamsRedis.set(entity.getParamCode(), entity.getParamValue());
    }

    @Override
    @Transactional(rollbackFor = Exception.class)
    public void update(SysParamsDTO dto) {
        SysParamsEntity entity = ConvertUtils.sourceToTarget(dto, SysParamsEntity.class);
        updateById(entity);

        sysParamsRedis.set(entity.getParamCode(), entity.getParamValue());
    }

    @Override
    @Transactional(rollbackFor = Exception.class)
    public void delete(Long[] ids) {
        //删除Redis数据
        List<String> paramCodeList = baseDao.getParamCodeList(ids);
        String[] paramCodes = paramCodeList.toArray(new String[paramCodeList.size()]);
        sysParamsRedis.delete(paramCodes);
    }

```

```

        //逻辑删除
        deleteBatchIds(Arrays.asList(ids));
    }

    @Override
    public String getValue(String paramCode) {
        String paramValue = sysParamsRedis.get(paramCode);
        if(paramValue == null){
            paramValue = baseDao.getValueByCode(paramCode);

            sysParamsRedis.set(paramCode, paramValue);
        }
        return paramValue;
    }

    @Override
    public <T> T getValueObject(String paramCode, Class<T> clazz) {
        String paramValue = getValue(paramCode);
        if(StringUtils.isNotBlank(paramValue)){
            return JSON.parseObject(paramValue, clazz);
        }

        try {
            return clazz.newInstance();
        } catch (Exception e) {
            throw new RenException(ErrorCode.PARAMS_GET_ERROR);
        }
    }

    @Override
    @Transactional(rollbackFor = Exception.class)
    public int updateValueByCode(String paramCode, String paramValue) {
        return baseDao.updateValueByCode(paramCode, paramValue);
    }
}

-----

@Component
public class SysParamsRedis {
    @Autowired
    private RedisUtils redisUtils;

    public void delete(Object[] paramCodes) {
        String key = RedisKeys.getSysParamsKey();
        redisUtils.hDel(key, paramCodes);
    }

    public void set(String paramCode, String paramValue){

```

```

        if(paramValue == null){
            return ;
        }
        String key = RedisKeys.getSysParamsKey();
        redisUtils.hSet(key, paramCode, paramValue);
    }

    public String get(String paramCode){
        String key = RedisKeys.getSysParamsKey();
        return (String)redisUtils.hGet(key, paramCode);
    }
}

-----

@Component
public class RedisUtils {
    @Autowired
    private RedisTemplate<String, Object> redisTemplate;

    /** 默认过期时长为24小时，单位：秒 */
    public final static long DEFAULT_EXPIRE = 60 * 60 * 24L;
    /** 过期时长为1小时，单位：秒 */
    public final static long HOUR_ONE_EXPIRE = 60 * 60 * 1L;
    /** 过期时长为6小时，单位：秒 */
    public final static long HOUR_SIX_EXPIRE = 60 * 60 * 6L;
    /** 不设置过期时长 */
    public final static long NOT_EXPIRE = -1L;

    public void set(String key, Object value, long expire){
        redisTemplate.opsForValue().set(key, value);
        if(expire != NOT_EXPIRE){
            expire(key, expire);
        }
    }

    public void set(String key, Object value){
        set(key, value, DEFAULT_EXPIRE);
    }

    public Object get(String key, long expire) {
        Object value = redisTemplate.opsForValue().get(key);
        if(expire != NOT_EXPIRE){
            expire(key, expire);
        }
        return value;
    }

    public Object get(String key) {

```

```

        return get(key, NOT_EXPIRE);
    }

    public void delete(String key) {
        redisTemplate.delete(key);
    }

    public void delete(Collection<String> keys) {
        redisTemplate.delete(keys);
    }

    public Object hGet(String key, String field) {
        return redisTemplate.opsForHash().get(key, field);
    }

    public Map<String, Object> hGetAll(String key){
        HashOperations<String, String, Object> hashOperations = redisTemplate.opsForHash();
        return hashOperations.entries(key);
    }

    public void hMSet(String key, Map<String, Object> map){
        hMSet(key, map, DEFAULT_EXPIRE);
    }

    public void hMSet(String key, Map<String, Object> map, long expire){
        redisTemplate.opsForHash().putAll(key, map);

        if(expire != NOT_EXPIRE){
            expire(key, expire);
        }
    }

    public void hSet(String key, String field, Object value) {
        hSet(key, field, value, DEFAULT_EXPIRE);
    }

    public void hSet(String key, String field, Object value, long expire) {
        redisTemplate.opsForHash().put(key, field, value);

        if(expire != NOT_EXPIRE){
            expire(key, expire);
        }
    }

    public void expire(String key, long expire){
        redisTemplate.expire(key, expire, TimeUnit.SECONDS);
    }

    public void hDel(String key, Object... fields){
        redisTemplate.opsForHash().delete(key, fields);
    }

```

```

    public void leftPush(String key, Object value){
        leftPush(key, value, DEFAULT_EXPIRE);
    }

    public void leftPush(String key, Object value, long expire){
        redisTemplate.opsForList().leftPush(key, value);

        if(expire != NOT_EXPIRE){
            expire(key, expire);
        }
    }

    public Object rightPop(String key){
        return redisTemplate.opsForList().rightPop(key);
    }
}

```

大家可能会有疑问，认为这个项目必须要配置Redis缓存，不然会报错，因为有操作Redis的代码，其实不然，通过Spring AOP，我们可以控制，是否真的使用Redis，代码如下：

```

@Aspect
@Component
public class RedisAspect {
    private Logger logger = LoggerFactory.getLogger(getClass());
    /**
     * 是否开启redis缓存 true开启 false关闭
     */
    @Value("${renren.redis.open: false}")
    private boolean open;

    @Around("execution(* io.renren.common.redis.RedisUtils.*(..))")
    public Object around(ProceedingJoinPoint point) throws Throwable {
        Object result = null;
        if(open){
            try{
                result = point.proceed();
            }catch (Exception e){
                logger.error("redis error", e);
                throw new RenException(ErrorCode.REDIS_ERROR);
            }
        }
        return result;
    }
}

```

6.8 异常处理机制

本项目通过RenException异常类，抛出自定义异常，RenException继承RuntimeException，不能继承Exception，如果继承Exception，则Spring事务不会回滚。

RenException代码如下所示：

```
public class RenException extends RuntimeException {
    private static final long serialVersionUID = 1L;

    private int code;
    private String msg;

    public RenException(int code) {
        this.code = code;
        this.msg = MessageUtils.getMessage(code);
    }

    public RenException(int code, String... params) {
        this.code = code;
        this.msg = MessageUtils.getMessage(code, params);
    }

    public RenException(int code, Throwable e) {
        super(e);
        this.code = code;
        this.msg = MessageUtils.getMessage(code);
    }

    public RenException(int code, Throwable e, String... params) {
        super(e);
        this.code = code;
        this.msg = MessageUtils.getMessage(code, params);
    }

    public RenException(String msg) {
        super(msg);
        this.code = ErrorCode.INTERNAL_SERVER_ERROR;
        this.msg = msg;
    }

    public RenException(String msg, Throwable e) {
        super(msg, e);
        this.code = ErrorCode.INTERNAL_SERVER_ERROR;
        this.msg = msg;
    }

    public String getMsg() {
        return msg;
    }
}
```

```

    }

    public void setMsg(String msg) {
        this.msg = msg;
    }

    public int getCode() {
        return code;
    }

    public void setCode(int code) {
        this.code = code;
    }
}

```

如何处理抛出的异常呢，我们定义了RenExceptionHandler类，并加上注解@RestControllerAdvice，就可以处理所有抛出的异常，并返回JSON数据。@RestControllerAdvice是由@ControllerAdvice、@ResponseBody注解组合而来的，可以查找@ControllerAdvice相关的资料，理解@ControllerAdvice注解的使用。

RenExceptionHandler代码如下所示：

```

@RestControllerAdvice
public class RenExceptionHandler {
    private static final Logger logger = LoggerFactory.getLogger(RenExceptionHandler.class);

    @Autowired
    private SysLogErrorService sysLogErrorService;

    /**
     * 处理自定义异常
     */
    @ExceptionHandler(RenException.class)
    public Result handleRenException(RenException ex){
        Result result = new Result();
        result.error(ex.getCode(), ex.getMsg());

        return result;
    }

    @ExceptionHandler(DuplicateKeyException.class)
    public Result handleDuplicateKeyException(DuplicateKeyException ex){
        Result result = new Result();
        result.error(ErrorCode.DB_RECORD_EXISTS);

        return result;
    }

    @ExceptionHandler(Exception.class)
    public Result handleException(Exception ex){
        logger.error(ex.getMessage(), ex);
    }
}

```

```

        saveLog(ex);

        return new Result().error();
    }

    /**
     * 保存异常日志
     */
    private void saveLog(Exception ex){
        SysLogErrorEntity log = new SysLogErrorEntity();

        //请求相关信息
        HttpServletRequest request = HttpContextUtils.getHttpServletRequest();
        log.setIp(IpUtils.getIpAddr(request));
        log.setUserAgent(request.getHeader(HttpHeaders.USER_AGENT));
        log.setRequestUri(request.getRequestURI());
        log.setRequestMethod(request.getMethod());
        Map<String, String> params = HttpContextUtils.getParameterMap(request);
        if(MapUtil.isEmpty(params)){
            log.setRequestParams(JSON.toJSONString(params));
        }

        //异常信息
        log.setErrorInfo(ExceptionUtils.getErrorStackTrace(ex));

        //保存
        sysLogErrorService.save(log);
    }
}

```

6.9 操作日志

系统日志是通过Spring AOP实现的，我们自定义了注解 `@LogOperation`，且只能在方法上使用，如下所示：

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface LogOperation {

    String value() default "";
}
```

下面是自定义注解 `@LogOperation` 的使用方式，如下所示：

```
@RestController
@RequestMapping("/sys/user")
public class SysUserController {

    @PostMapping
    @LogOperation("保存")
    @RequiresPermissions("sys:user:save")
    public Result save(@RequestBody SysUserDTO dto){
        //效验数据
        ValidatorUtils.validateEntity(dto, AddGroup.class, DefaultGroup.class);

        sysUserService.save(dto);

        return new Result();
    }
}
```

我们可以发现，只需要在保存日志的请求方法上，加上 `@LogOperation` 注解，就可以把日志保存到数据库里了。

具体是在哪里把数据保存到数据库里的呢，我们定义了 `LogOperationAspect` 处理类，就是来干这事的，如下所示：

```
/**
 * 系统日志，切面处理类
 */
@Aspect
@Component
public class LogOperationAspect {

    @Autowired
    private SysLogOperationService sysLogOperationService;

    @Pointcut("@annotation(io.renren.common.annotation.LogOperation)")
```

```

public void logPointCut() {

}

@Around("logPointCut()")
public Object around(ProceedingJoinPoint point) throws Throwable {
    long beginTime = System.currentTimeMillis();
    try {
        //执行方法
        Object result = point.proceed();

        //执行时长(毫秒)
        long time = System.currentTimeMillis() - beginTime;
        //保存日志
        saveLog(point, time, OperationStatusEnum.SUCCESS.value());

        return result;
    } catch (Exception e) {
        //执行时长(毫秒)
        long time = System.currentTimeMillis() - beginTime;
        //保存日志
        saveLog(point, time, OperationStatusEnum.FAIL.value());

        throw e;
    }
}

private void saveLog(ProceedingJoinPoint joinPoint, long time, Integer status) {
    MethodSignature signature = (MethodSignature) joinPoint.getSignature();
    Method method = signature.getMethod();

    SysLogOperationEntity log = new SysLogOperationEntity();
    LogOperation annotation = method.getAnnotation(LogOperation.class);
    if(annotation != null){
        //注解上的描述
        log.setOperation(annotation.value());
    }

    //登录用户信息
    UserDetail user = SecurityUser.getUser();
    if(user != null){
        log.setCreatorName(user.getUsername());
    }

    log.setStatus(status);
    log.setRequestTime((int)time);

    //请求相关信息
    HttpServletRequest request = HttpContextUtils.getHttpServletRequest();
    log.setIp(IpUtils.getIpAddr(request));
    log.setUserAgent(request.getHeader(HttpHeaders.USER_AGENT));
    log.setRequestUri(request.getRequestURI());
}

```

```
log.setRequestMethod(request.getMethod());

//请求参数
Object[] args = joinPoint.getArgs();
try{
    String params = JSON.toJSONString(args[0]);
    log.setRequestParams(params);
}catch (Exception e){

}

//保存到DB
sysLogOperationService.save(log);
}
}
```

`LogOperationAspect` 类定义了一个切入点，请求 `@LogOperation` 注解的方法时，会进入 `around` 方法，把系统日志保存到数据库中。

6.10 定时任务模块

本系统使用开源框架Quartz，实现的定时任务，已实现分布式定时任务，可部署多台服务器，不重复执行，以及动态增加、修改、删除、暂停、恢复、立即执行定时任务。Quartz自带了各数据库的SQL脚本，如果想更改成其他数据库，可参考Quartz相应的SQL脚本。

6.10.1 新增定时任务

新增一个定时任务，其实很简单，只要定义一个普通的Spring Bean即可，如下所示：

```
@Component("testTask")
public class TestTask implements ITask {
    private Logger logger = LoggerFactory.getLogger(getClass());

    @Override
    public void run(String params){
        logger.debug("TestTask定时任务正在执行，参数为: {}", params);
    }
}
```

如何让Quartz，定时执行testTask里的方法呢？只需要在管理后台，新增一个定时任务即可，如下图所示：

The screenshot displays the '定时任务' (Scheduled Task) management interface. On the left is a sidebar menu with options like '权限管理', '系统设置', '菜单管理', '参数管理', '字典管理', '定时任务', '文件上传', '工作流程', '消息管理', '日志管理', '系统监控', and '功能示例'. The main area is divided into two sections. The top section is the '新增' (Add) form for a task named 'testTask'. It includes fields for 'bean名称' (testTask), '参数' (renren), 'cron表达式' (0 0/30 * * * ?), and '备注' (有参测试, 多个参数使用json). The bottom section is a table listing the tasks. The table has columns: 'bean名称', '参数', 'cron表达式', '备注', '状态', and '操作'. The table contains one entry for 'testTask' with status '正常'. The '操作' column has buttons for '修改', '暂停', '恢复', '执行', and '删除'.

bean名称	参数	cron表达式	备注	状态	操作
testTask	renren	0 0/30 * * * ?	有参测试, 多个参数使用json	正常	修改 暂停 恢复 执行 删除

刚才配置的定时任务，每隔30分钟，就会调用TestTask的run方法了，是不是很简单啊。

6.10.2 源码分析

Quartz提供了相关的API，我们可以调用API，对Quartz进行增加、修改、删除、暂停、恢复、立即执行等。本系统中，ScheduleUtils 类就是对Quartz API进行的封装，代码如下所示：

```
public class ScheduleUtils {
    private final static String JOB_NAME = "TASK_";
    /**
     * 任务调度参数key
     */
    public static final String JOB_PARAM_KEY = "JOB_PARAM_KEY";

    /**
     * 获取触发器key
     */
    public static TriggerKey getTriggerKey(Long jobId) {
        return TriggerKey.triggerKey(JOB_NAME + jobId);
    }

    /**
     * 获取jobKey
     */
    public static JobKey getJobKey(Long jobId) {
        return JobKey.jobKey(JOB_NAME + jobId);
    }

    /**
     * 获取表达式触发器
     */
    public static CronTrigger getCronTrigger(Scheduler scheduler, Long jobId) {
        try {
            return (CronTrigger) scheduler.getTrigger(getTriggerKey(jobId));
        } catch (SchedulerException e) {
            throw new RenException(ErrorCode.JOB_ERROR, e);
        }
    }

    /**
     * 创建定时任务
     */
    public static void createScheduleJob(Scheduler scheduler, ScheduleJobEntity scheduleJob)
    {
        try {
            //构建job信息
            JobDetail jobDetail = JobBuilder.newJob(ScheduleJob.class).withIdentity(getJobKey(
            (scheduleJob.getId()))).build();

            //表达式调度构建器
            CronScheduleBuilder scheduleBuilder = CronScheduleBuilder.cronSchedule(scheduleJob
```

```

b.getCronExpression()
    .withMisfireHandlingInstructionDoNothing();

    //按新的cronExpression表达式构建一个新的trigger
    CronTrigger trigger = TriggerBuilder.newTrigger().withIdentity(getTriggerKey(scheduleJob.getId())).withSchedule(scheduleBuilder).build();

    //放入参数，运行时的方法可以获取
    jobDetail.getJobDataMap().put(JOB_PARAM_KEY, scheduleJob);

    scheduler.scheduleJob(jobDetail, trigger);

    //暂停任务
    if(scheduleJob.getStatus() == Constant.ScheduleStatus.PAUSE.getValue()){
        pauseJob(scheduler, scheduleJob.getId());
    }
} catch (SchedulerException e) {
    throw new RenException(ErrorCode.JOB_ERROR, e);
}
}

/**
 * 更新定时任务
 */
public static void updateScheduleJob(Scheduler scheduler, ScheduleJobEntity scheduleJob)
{
    try {
        TriggerKey triggerKey = getTriggerKey(scheduleJob.getId());

        //表达式调度构建器
        CronScheduleBuilder scheduleBuilder = CronScheduleBuilder.cronSchedule(scheduleJob
b.getCronExpression()
    .withMisfireHandlingInstructionDoNothing();

        CronTrigger trigger = getCronTrigger(scheduler, scheduleJob.getId());

        //按新的cronExpression表达式重新构建trigger
        trigger = trigger.getTriggerBuilder().withIdentity(triggerKey).withSchedule(scheduleBuilder).build();

        //参数
        trigger.getJobDataMap().put(JOB_PARAM_KEY, scheduleJob);

        scheduler.rescheduleJob(triggerKey, trigger);

        //暂停任务
        if(scheduleJob.getStatus() == Constant.ScheduleStatus.PAUSE.getValue()){
            pauseJob(scheduler, scheduleJob.getId());
        }
    } catch (SchedulerException e) {
        throw new RenException(ErrorCode.JOB_ERROR, e);
    }
}

```

```

    }
}

/**
 * 立即执行任务
 */
public static void run(Scheduler scheduler, ScheduleJobEntity scheduleJob) {
    try {
        //参数
        JobDataMap dataMap = new JobDataMap();
        dataMap.put(JOB_PARAM_KEY, scheduleJob);

        scheduler.triggerJob(getJobKey(scheduleJob.getId()), dataMap);
    } catch (SchedulerException e) {
        throw new RenException(ErrorCode.JOB_ERROR, e);
    }
}

/**
 * 暂停任务
 */
public static void pauseJob(Scheduler scheduler, Long jobId) {
    try {
        scheduler.pauseJob(getJobKey(jobId));
    } catch (SchedulerException e) {
        throw new RenException(ErrorCode.JOB_ERROR, e);
    }
}

/**
 * 恢复任务
 */
public static void resumeJob(Scheduler scheduler, Long jobId) {
    try {
        scheduler.resumeJob(getJobKey(jobId));
    } catch (SchedulerException e) {
        throw new RenException(ErrorCode.JOB_ERROR, e);
    }
}

/**
 * 删除定时任务
 */
public static void deleteScheduleJob(Scheduler scheduler, Long jobId) {
    try {
        scheduler.deleteJob(getJobKey(jobId));
    } catch (SchedulerException e) {
        throw new RenException(ErrorCode.JOB_ERROR, e);
    }
}
}

```

以下是几个核心的方法：

- **createScheduleJob【创建定时任务】**：在管理后台新增任务时，会调用该方法，把任务添加到Quartz中，再根据cron表达式，定时执行任务。
- **updateScheduleJob【更新定时任务】**：修改任务时，调用该方法，修改Quartz中的任务信息。
- **run【立即执行定时任务】**：马上执行一次该任务，只执行一次。
- **pauseJob【暂停定时任务】**：这个不是暂停正在执行的任务，而是以后不再执行这个定时任务了。正在执行的任务，还是照常执行完。
- **resumeJob【恢复定时任务】**：这个是针对pauseJob来的，如果任务暂停了，以后都不会再执行，要想再执行，则需要调用resumeJob，使定时任务恢复执行。
- **deleteScheduleJob【删除定时任务】**：删除定时任务

其中，`createScheduleJob`、`updateScheduleJob` 在启动项目的时候，也会调用，把数据库里，新增或修改的任务，更新到Quartz中，如下所示：

```
/**
 * 初始化定时任务数据
 *
 * @author Mark sunlightcs@gmail.com
 */
@Component
public class JobCommandLineRunner implements CommandLineRunner {
    @Autowired
    private Scheduler scheduler;
    @Autowired
    private ScheduleJobDao scheduleJobDao;

    @Override
    public void run(String... args) {
        List<ScheduleJobEntity> scheduleJobList = scheduleJobDao.selectList(null);
        for(ScheduleJobEntity scheduleJob : scheduleJobList){
            CronTrigger cronTrigger = ScheduleUtils.getCronTrigger(scheduler, scheduleJob.getId());
            //如果不存在，则创建
            if(cronTrigger == null) {
                ScheduleUtils.createScheduleJob(scheduler, scheduleJob);
            }else {
                ScheduleUtils.updateScheduleJob(scheduler, scheduleJob);
            }
        }
    }
}
```

大家是不是还有疑问呢，怎么就能定时执行，刚才在管理后台新增的任务testTask呢？下面我们再来分析下 `createScheduleJob` 方法，创建定时任务的时候，要调用该方法，代码如下所示：

```
//构建一个新的定时任务，JobBuilder.newJob()只能接受Job类型的参数
//把ScheduleJob.class作为参数传进去，ScheduleJob继承QuartzJobBean，而QuartzJobBean实现了Job接口
JobDetail jobDetail = JobBuilder.newJob(ScheduleJob.class).withIdentity(getJobKey(scheduleJob.getId())).build();
```

```

//构建cron，定时任务的周期
CronScheduleBuilder scheduleBuilder = CronScheduleBuilder.cronSchedule(scheduleJob.getCronExpression())
    .withMisfireHandlingInstructionDoNothing();

//根据cron，构建一个CronTrigger
CronTrigger trigger = TriggerBuilder.newTrigger().withIdentity(getTriggerKey(scheduleJob.getId()))
    .withSchedule(scheduleBuilder).build();

//放入参数，运行时的方法可以获取
jobDetail.getJobDataMap().put(ScheduleJobEntity.JOB_PARAM_KEY, new Gson().toJson(scheduleJob));

//把任务添加到Quartz中
scheduler.scheduleJob(jobDetail, trigger);

```

把任务添加到 Quartz 后，等cron定义的时间周期到了，就会执行 ScheduleJob 类的 executeInternal 方法，ScheduleJob 代码如下所示：

```

public class ScheduleJob extends QuartzJobBean {
    private Logger logger = LoggerFactory.getLogger(getClass());

    @Override
    protected void executeInternal(JobExecutionContext context) {
        ScheduleJobEntity scheduleJob = (ScheduleJobEntity) context.getMergedJobDataMap().get(ScheduleUtils.JOB_PARAM_KEY);

        //数据库保存执行记录
        ScheduleJobLogEntity log = new ScheduleJobLogEntity();
        log.setJobId(scheduleJob.getId());
        log.setBeanName(scheduleJob.getBeanName());
        log.setParams(scheduleJob.getParams());
        log.setCreateDate(new Date());

        //任务开始时间
        long startTime = System.currentTimeMillis();

        try {
            //执行任务
            logger.info("任务准备执行，任务ID: {}", scheduleJob.getId());
            Object target = SpringContextUtils.getBean(scheduleJob.getBeanName());
            Method method = target.getClass().getDeclaredMethod("run", String.class);
            method.invoke(target, scheduleJob.getParams());

            //任务执行总时长
            long times = System.currentTimeMillis() - startTime;
            log.setTimes((int)times);
            //任务状态
            log.setStatus(Constant.SUCCESS);

            logger.info("任务执行完毕，任务ID: {} 总共耗时: {} 毫秒", scheduleJob.getId(), times);

```

```

s);
    } catch (Exception e) {
        logger.error("任务执行失败, 任务ID: {}", scheduleJob.getId(), e);

        //任务执行总时长
        long times = System.currentTimeMillis() - startTime;
        log.setTimes((int)times);

        //任务状态
        log.setStatus(Constant.FAIL);
        log.setError(ExceptionUtils.getErrorStackTrace(e));
    }finally {
        //获取spring bean
        ScheduleJobLogService scheduleJobLogService = SpringContextUtils.getBean(Schedule
JobLogService.class);
        scheduleJobLogService.insert(log);
    }
}
}
}

```

6.11 API模块

APP模块，主要是简化APP开发，如：为微信小程序、IOS、Android提供接口，拥有一套单独的用户体系，没有与renren-admin用户表共用，因为renren-admin用户表里存放的是企业内部人员账号，具有后台管理员权限，可以登录后台管理系统，而renren-api用户表里存放的是我们的真实用户，不具备登录后台管理系统的权限。renren-api主要是实现了用户注册、登录、接口权限认证、获取登录用户等功能，为APP接口的安全调用，提供一套优雅的解决方案，从而简化APP接口开发。

6.11.1 API的使用

API的设计思路：用户通过APP，输入手机号、密码登录后，系统会生成与登录用户一一对应的token，用户调用需要登录的接口时，只需把token传过来，服务端就知道是谁在访问接口，token如果过期，则拒绝访问，从而保证系统的安全性。

使用很简单，看看下面的例子，就会使用了。仔细观察，我们会发现，有2个自定义的注解。其中，@LoginUser注解是获取当前登录用户的信息，有哪些信息，下面会分析的。@Login注解则是需要用户认证，没有登录的用户，不能访问该接口。

```
import io.renren.annotation.Login;
import io.renren.annotation.LoginUser;

@RestController
@RequestMapping("/api")
@Api(tags="测试接口")
public class ApiTestController {

    @Login
    @GetMapping("userInfo")
    @ApiOperation(value="获取用户信息", response=UserEntity.class)
    public Result<UserEntity> userInfo(@ApiIgnore @LoginUser UserEntity user){
        return new Result<UserEntity>().ok(user);
    }

    @Login
    @GetMapping("userId")
    @ApiOperation("获取用户ID")
    public Result<Long> userInfo(@ApiIgnore @RequestAttribute("userId") Long userId){
        return new Result<Long>().ok(userId);
    }

    @GetMapping("notToken")
    @ApiOperation("忽略Token验证测试")
    public Result<String> notToken(){
        return new Result<String>().ok("无需token也能访问。。。");
    }
}
```

6.11.2 源码分析

- 我们先来看看，API用户登录的时候，都干了那些事情，如下所示：

```
@RestController
@RequestMapping("/api")
@Api(tags="登录接口")
public class ApiLoginController {
    @Autowired
    private UserService userService;
    @Autowired
    private TokenService tokenService;

    @PostMapping("login")
    @ApiOperation("登录")
    public Result<Map<String, Object>> login(@RequestBody LoginDTO dto){
        //表单校验
        ValidatorUtils.validateEntity(dto);

        //用户登录
        Map<String, Object> map = userService.login(dto);

        return new Result().ok(map);
    }

    @Login
    @PostMapping("logout")
    @ApiOperation("退出")
    public Result logout(@ApiIgnore @RequestAttribute("userId") Long userId){
        tokenService.expireToken(userId);
        return new Result();
    }
}

-----

/**
 * jwt工具类
 */
@ConfigurationProperties(prefix = "renren.jwt")
@Component
public class JwtUtils {
    private Logger logger = LoggerFactory.getLogger(getClass());

    private String secret;
    private long expire;
    private String header;
```

```

/**
 * 生成jwt token
 */
public String generateToken(long userId) {
    Date nowDate = new Date();
    //过期时间
    Date expireDate = new Date(nowDate.getTime() + expire * 1000);

    return Jwts.builder()
        .setHeaderParam("typ", "JWT")
        .setSubject(userId+"")
        .setIssuedAt(nowDate)
        .setExpiration(expireDate)
        .signWith(SignatureAlgorithm.HS512, secret)
        .compact();
}

public Claims getClaimByToken(String token) {
    try {
        return Jwts.parser()
            .setSigningKey(secret)
            .parseClaimsJws(token)
            .getBody();
    } catch (Exception e){
        logger.debug("validate is token error ", e);
        return null;
    }
}

/**
 * token是否过期
 * @return true: 过期
 */
public boolean isTokenExpired(Date expiration) {
    return expiration.before(new Date());
}

public String getSecret() {
    return secret;
}

public void setSecret(String secret) {
    this.secret = secret;
}

public long getExpire() {
    return expire;
}

public void setExpire(long expire) {
    this.expire = expire;
}

```

```

    }

    public String getHeader() {
        return header;
    }

    public void setHeader(String header) {
        this.header = header;
    }
}

```

我们从上面的代码，可以看到，用户每次登录的时候，都会生成一个唯一的token，这个token是通过jwt生成的。

- APP模块的核心配置，如下所示：

```

import io.renren.modules.api.interceptor.AuthorizationInterceptor;
import io.renren.modules.api.resolver.LoginUserHandlerMethodArgumentResolver;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.method.support.HandlerMethodArgumentResolver;
import org.springframework.web.servlet.config.annotation.InterceptorRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;

@Configuration
public class WebMvcConfig extends WebMvcConfigurerAdapter {
    @Autowired
    private AuthorizationInterceptor authorizationInterceptor;
    @Autowired
    private LoginUserHandlerMethodArgumentResolver loginUserHandlerMethodArgumentResolver;

    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(authorizationInterceptor).addPathPatterns("/api/**");
    }

    @Override
    public void addArgumentResolvers(List<HandlerMethodArgumentResolver> argumentResolvers) {
        argumentResolvers.add(loginUserHandlerMethodArgumentResolver);
    }
}

```

我们可以看到，配置了个Interceptor，用来拦截 /api 开头的请求，拦截后，会到 AuthorizationInterceptor类preHandle方法处理。只有以 /api 开头的请求，API模块认证才会起作用，如果要以 /mobile 开头，则需要修改此处。还配置了argumentResolver，别忽略了啊，下面会讲解。

- 分析AuthorizationInterceptor类，我们可以发现，拦截 /api 开头的请求后，都干了些什么，如下所示：

```

import io.renren.annotation.Login;
import io.renren.common.exception.ErrorCode;
import io.renren.common.exception.RenException;

```

```

import io.renren.entity.TokenEntity;
import io.renren.service.TokenService;
import org.apache.commons.lang3.StringUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.web.method.HandlerMethod;
import org.springframework.web.servlet.handler.HandlerInterceptorAdapter;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * 权限(Token)验证
 */
@Component
public class AuthorizationInterceptor extends HandlerInterceptorAdapter {
    @Autowired
    private TokenService tokenService;

    public static final String USER_KEY = "userId";

    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object
handler) throws Exception {
        Login annotation;
        if(handler instanceof HandlerMethod) {
            annotation = ((HandlerMethod) handler).getMethodAnnotation(Login.class);
        }else{
            return true;
        }

        if(annotation == null){
            return true;
        }

        //从header中获取token
        String token = request.getHeader("token");
        //如果header中不存在token，则从参数中获取token
        if(StringUtils.isBlank(token)){
            token = request.getParameter("token");
        }

        //token为空
        if(StringUtils.isBlank(token)){
            throw new RenException(ErrorCode.TOKEN_NOT_EMPTY);
        }

        //查询token信息
        TokenEntity tokenEntity = tokenService.getByToken(token);
        if(tokenEntity == null || tokenEntity.getExpireDate().getTime() < System.currentTimeMillis()){
            throw new RenException(ErrorCode.TOKEN_INVALID);
        }
    }
}

```

```

    }

    //设置userId到request里，后续根据userId，获取用户信息
    request.setAttribute(USER_KEY, tokenEntity.getUserId());

    return true;
}
}

```

我们可以发现，进入 `/api` 请求的接口之前，会判断请求的接口，是否加了`@Login`注解(需要token认证)，如果没有`@Login`注解，则不验证token，可以直接访问接口。如果有`@Login`注解，则需要验证token的正确性，并把userId放到request的USER_KEY里，后续会用到。

- 此时，`@Login`注解的作用，相信大家都明白了。再看看下面的代码，加了`@LoginUser`注解后，`user`对象里，就变成当前登录用户的信息，这是什么时候设置进去的呢？

```

/**
 * 获取用户信息
 */
@login
@GetMapping("userInfo")
public Result<UserEntity> userInfo(@ApiIgnore @LoginUser UserEntity user){
    return new Result<UserEntity>().ok(user);
}

```

- 设置`user`对象进去，其实是在`LoginUserHandlerMethodArgumentResolver`里干的，`LoginUserHandlerMethodArgumentResolver`是我们自定义的参数转换器，只要实现`HandlerMethodArgumentResolver`接口即可，代码如下所示：

```

import io.renren.modules.api.annotation.LoginUser;
import io.renren.modules.api.entity.UserEntity;
import io.renren.modules.api.interceptor.AuthorizationInterceptor;
import io.renren.modules.api.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.MethodParameter;
import org.springframework.stereotype.Component;
import org.springframework.web.bind.support.WebDataBinderFactory;
import org.springframework.web.context.request.NativeWebRequest;
import org.springframework.web.context.request.RequestAttributes;
import org.springframework.web.method.support.HandlerMethodArgumentResolver;
import org.springframework.web.method.support.ModelAndViewContainer;

@Component
public class LoginUserHandlerMethodArgumentResolver implements HandlerMethodArgumentResolver {

    @Autowired
    private UserService userService;

    @Override
    public boolean supportsParameter(MethodParameter parameter) {

```

```

        //如果方法的参数是UserEntity，且参数前面有@loginUser注解，则进入resolveArgument方法，进行处理
        return parameter.getParameterType().isAssignableFrom(UserEntity.class) && parameter.hasParameterAnnotation(LoginUser.class);
    }

    @Override
    public Object resolveArgument(MethodParameter parameter, ModelAndViewContainer container,
                                NativeWebRequest request, WebDataBinderFactory factory) throws Exception {
        //获取用户ID，之前设置进去的，还有印象吧
        Object object = request.getAttribute(AuthorizationInterceptor.USER_KEY, RequestAttributes.SCOPE_REQUEST);
        if(object == null){
            return null;
        }

        //通过userId，获取用户信息
        UserEntity user = userService.queryObject((Long)object);

        //把当前用户信息，设置到UserEntity参数的user对象里
        return user;
    }
}

```

6.11.3 Swagger生成接口文档

本项目，支持Swagger注解的方式，生成接口文档，简化接口文档的维护工作

- 如要支持Swagger，需要引入对应的Jar包，我们使用的版本是2.7.0，如下所示：

```

<properties>
    <swagger.version>2.7.0</swagger.version>
</properties>

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

```

- 怎么生成Swagger接口文档，很简单，只要方法加了 `@ApiOperation` 注解，就会自动生成接口文档，如下所示，其中， `response=UserEntity.class` 是返回的数据对象

```

@Login
@GetMapping("userInfo")
@ApiOperation(value="获取用户信息", response=UserEntity.class)
public Result<UserEntity> userInfo(@ApiIgnore @LoginUser UserEntity user){
    return new Result<UserEntity>().ok(user);
}

```

- 我们再来看下Swagger的具体配置，其中配置了 `securitySchemes(security())`，目的就是要把登录的token，放到header头里，以免不能调用，需要认证的接口，如下所示：

```

@Configuration
@EnableSwagger2
public class SwaggerConfig {
    @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))
            //包下的类，才生成接口文档
            //.apis(RequestHandlerSelectors.basePackage("io.renren.controller"))
            .paths(PathSelectors.any())
            .build()
            .securitySchemes(security());
    }

    private ApiInfo apiInfo() {
        return new ApiInfoBuilder()
            .title("人人开源")
            .description("renren-api模块接口文档")
            .termsOfServiceUrl("https://www.renren.io")
            .version("1.0")
            .build();
    }

    private List<ApiKey> security() {
        return new ArrayList<
            new ApiKey("token", "token", "header")
        >;
    }
}

```

- 我们再来看下，生成的接口文档，以及如何通过界面，调用接口的，如下所示：

 swagger

default (/v2/api-docs) **Authorize** Explore

人人开源

renren-api模块接口文档

注册接口 : Api Register Controller

Show/Hide | List Operations | Expand Operations

POST

/api/register

注册

测试接口 : Api Test Controller

Show/Hide | List Operations | Expand Operations

GET

/api/notToken

忽略Token验证测试

GET

/api/userId

获取用户ID

GET

/api/userInfo

获取用户信息

登录接口 : Api Login Controller

Show/Hide | List Operations | Expand Operations

POST

/api/login

登录

POST

/api/logout

退出

[BASE URL: /renren-api , API VERSION: 1.0]

步骤1、注册账号

步骤3、输入获取的token

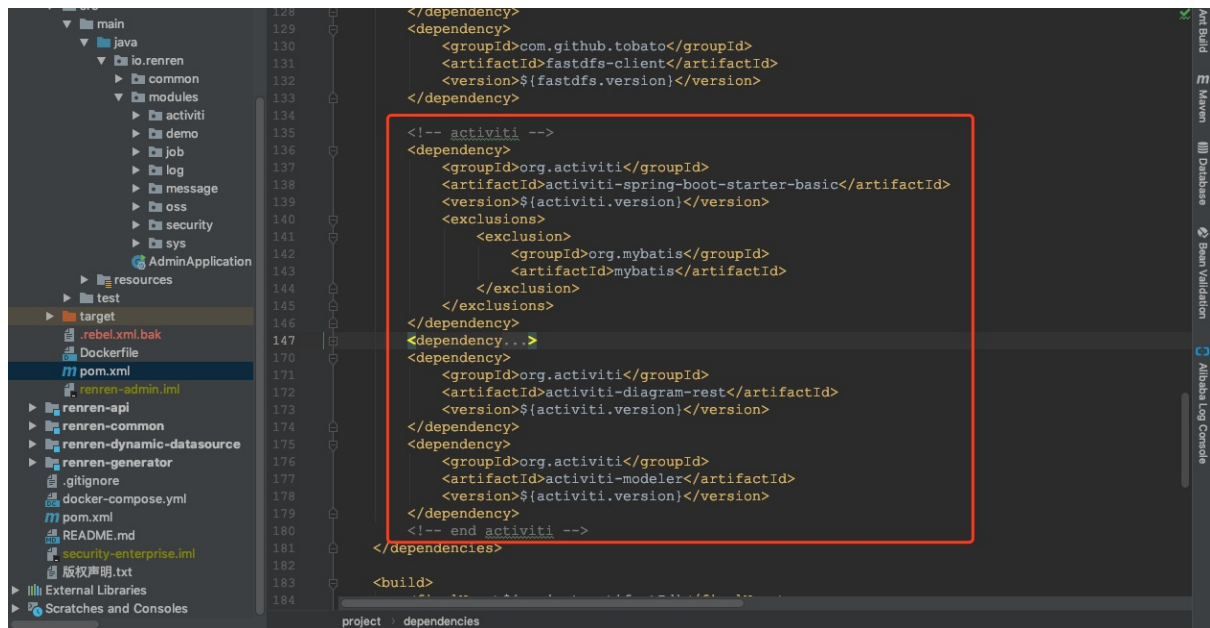
步骤4、调用需要权限访问的接口

步骤2、登录系统，获取token

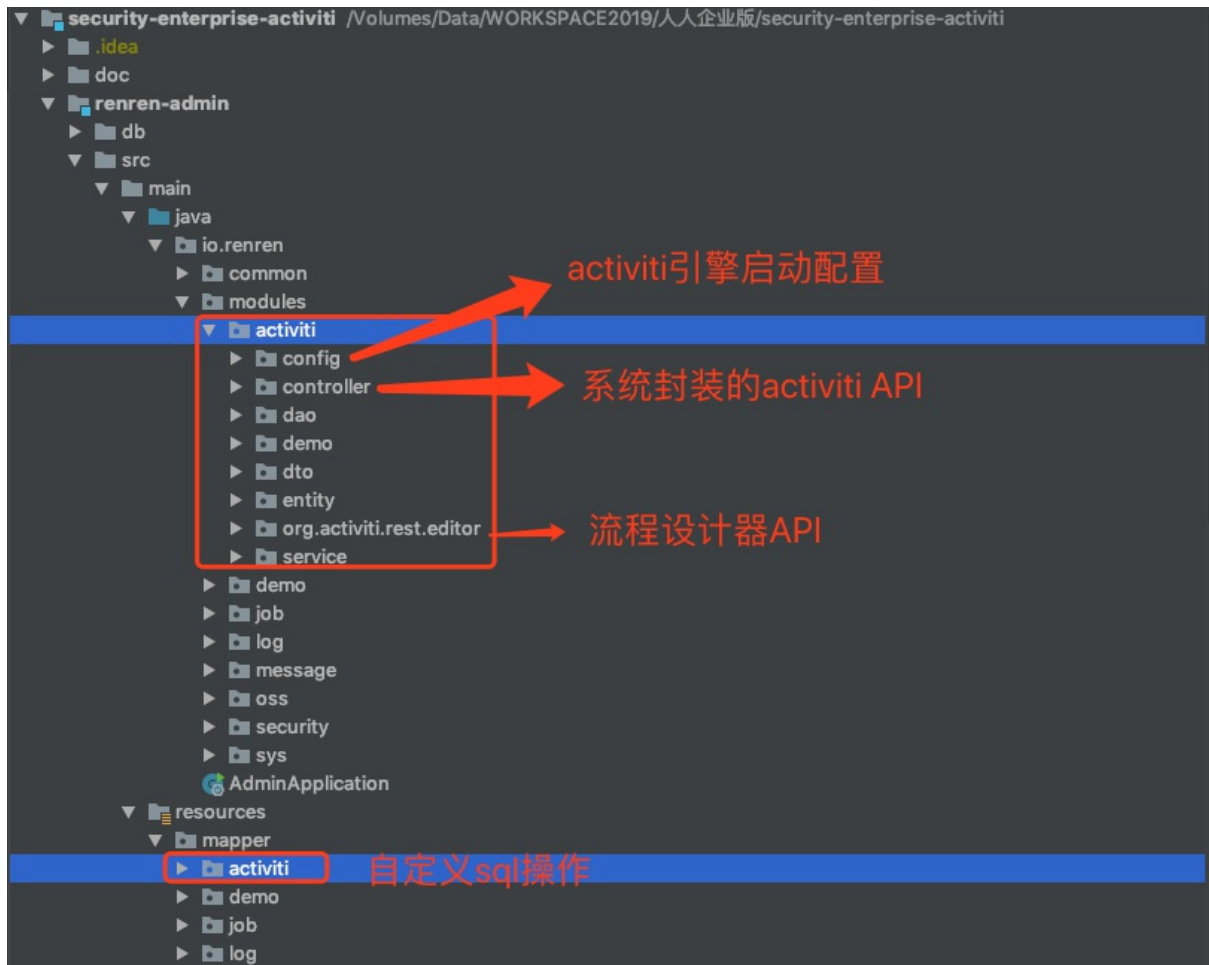
6.12 工作流模块

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。

后端代码结构



pom.xml引入jar包



代码结构

集成activiti引擎的所有的代码都在io.renren.modules.activiti.*下。

io.renren.modules.activiti.config : activiti的springboot启动配置;

io.renren.modules.activiti.controller: 系统实现的流程相关功能的API;

io.renren.modules.activiti.dao: 数据库操作;

io.renren.modules.activiti.demo: 工作流的示例;

io.renren.modules.activiti.dto: dto;

io.renren.modules.activiti.entity: entity;

io.renren.modules.activiti.org.activiti.rest.editor: 流程设计器;

io.renren.modules.activiti.service: activiti功能实现

前端代码结构



上图为前端工作流相关功能目录。1、公共组件：流程详情、流程处理、流程启动以及综合前面三种组件的公共组件，具体用法可参考请假管理和转正申请示例。2、流程相关公共方法，这里面封装了和流程相关的公共方法，在使用的地方引入即可。3、工作流功能：包括请假管理示例、转正申请示例、模型管理、流程定义管理、实例管理、我的申请、我的待办、已办任务、待签收任务、发起流程、流程管理、模型管理以及运行中的流程等功能。

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。

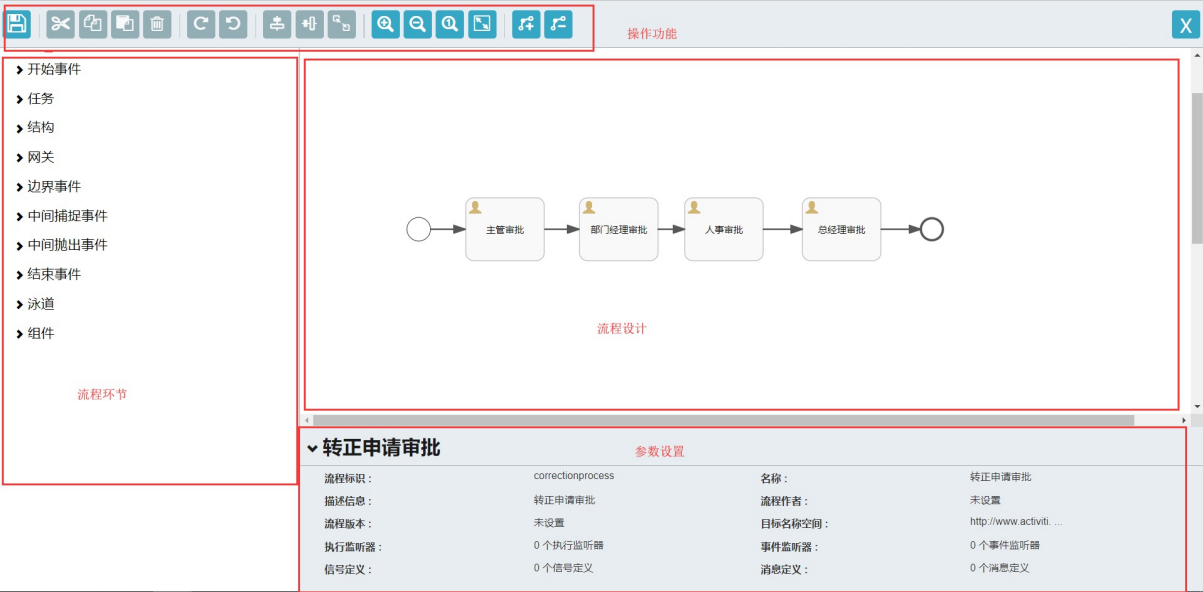
任务管理：任务管理是可以细分的。包括：我的待办、我的已办、我的申请、我的委托以及待签收任务等。系统集成activiti，并开发出以上功能的API接口。在使用中，直接调用这些API接口变可实现功能。具体功能API可参考swagger的API文档。

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：

▼ 转正申请审批			
流程标识：	correctionprocess	名称：	转正申请审批
描述信息：	转正申请审批	流程作者：	未设置
流程版本：	未设置	目标名称空间：	http://www.activiti...
执行监听器：	0 个执行监听器	事件监听器：	0 个事件监听器
信号定义：	0 个信号定义	消息定义：	0 个消息定义

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：

▼ 转正申请审批	
主键 (ID) :	sid-426CA66B-6239-4A ...
描述信息 :	未设置
执行监听器 :	未设置执行监听器
名称 :	未设置
流条件 :	No condition
默认流 :	<input type="checkbox"/>

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。

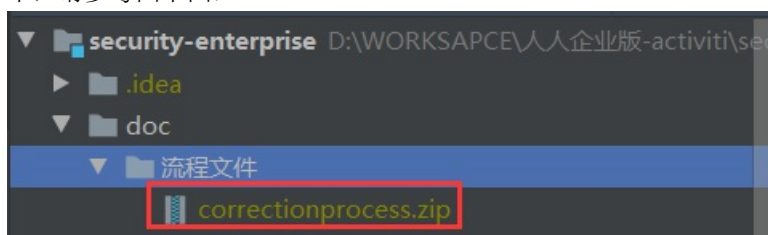
3、用户任务参数：

▼ 主管审批	
主键 (ID) :	sid-84AA3028-84C4-45 ...
描述信息 :	未设置
互斥任务 :	<input checked="" type="checkbox"/>
多实例类型 :	None
集合 (多实例) :	未设置
完成条件 (多实例) :	未设置
分配用户 :	用户 1067246875800000001
到期时间 :	未设置
名称 :	主管审批
异步 :	<input type="checkbox"/>
执行监听器 :	0 个执行监听器
基数 (多实例) :	未设置
元素变量 (多实例) :	未设置
是否补偿 :	<input type="checkbox"/>
表单编号 :	未设置
优先级 :	未设置

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。

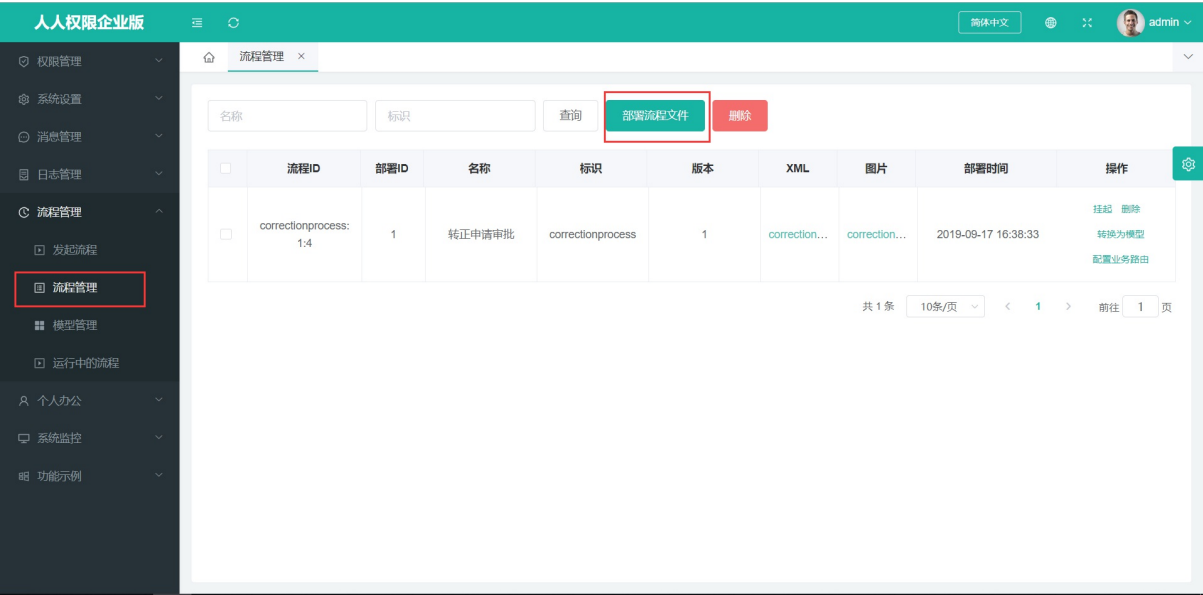
系统支持的业务和流程结合的方式为：

- 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：

对象		* 无标题 - 查询		tb_correction@security_en...	
保存		添加字段		插入字段	
删除字段		主键		上移	
下移		SQL 预览			
字段	索引	外键	触发器	选项	注释
名	类型	长度	小数点	不是 null	虚拟
id	bigint	20	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>
apply_post	varchar	255	0	<input type="checkbox"/>	<input type="checkbox"/>
entry_date	datetime	0	0	<input type="checkbox"/>	<input type="checkbox"/>
correction_date	datetime	0	0	<input type="checkbox"/>	<input type="checkbox"/>
work_content	varchar	255	0	<input type="checkbox"/>	<input type="checkbox"/>
achievement	varchar	255	0	<input type="checkbox"/>	<input type="checkbox"/>
creator	bigint	255	0	<input type="checkbox"/>	<input type="checkbox"/>
create_date	datetime	0	0	<input type="checkbox"/>	<input type="checkbox"/>
instance_id	varchar	80	0	<input type="checkbox"/>	<input type="checkbox"/>

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

```
<!-- 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="$i18n.locale === 'zh' ? '120px' : '100px'">
      <el-form-item label="$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.entryDate')">
      </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('correction.correctionDate')">
      </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" ></ren-process-multiple>
  </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: ''
        }
      }
    }
  },
</script>
```

```

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；

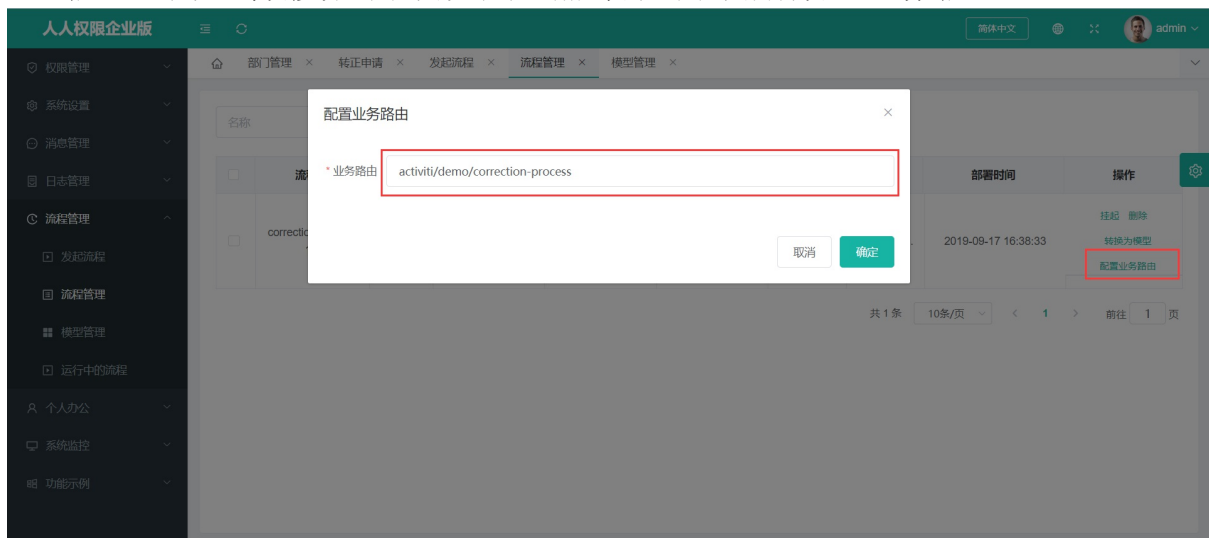
dataFormName：保存的form表单名称；

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
1 <template>
2 <el-dialog :visible.sync="visible" :title="!dataForm.id ? $t('add') : $t('update')" :close-on-click-modal="false" :close-on-press-escape="false">
3 <el-form :model="dataForm" :rules="dataRule" ref="dataForm" @keyup.enter.native="dataFormSubmitHandle()" :label-width="$i18n.locale === 'en-US'
4 <el-form-item :label="$t('correction.post')" prop="applyPost">
5 <el-input v-model="dataForm.applyPost" :placeholder="$t('correction.post')"/></el-input>
6 </el-form-item>
7 <el-row :gutter="40">
8 <el-col :span="12">
9 <el-form-item :label="$t('correction.entryDate')" prop="entryDate">
10 <el-date-picker v-model="dataForm.entryDate" value-format="yyyy-MM-dd" :placeholder="$t('correction.entryDate')" style="width: 100%"/></el-form-item>
11 </el-col>
12 <el-col :span="12">
13 <el-form-item :label="$t('correction.correctionDate')" prop="correctionDate">
14 <el-date-picker v-model="dataForm.correctionDate" value-format="yyyy-MM-dd" :placeholder="$t('correction.correctionDate')" style="width: 100%"/></el-form-item>
15 </el-col>
16 </el-row>
17 <el-form-item :label="$t('correction.workContent')" prop="workContent">
18 <el-input type="textarea" v-model="dataForm.workContent" :placeholder="$t('correction.workContent')"/></el-input>
19 </el-form-item>
20 <el-form-item :label="$t('correction.achievement')" prop="achievement">
21 <el-input type="textarea" v-model="dataForm.achievement" :placeholder="$t('correction.achievement')"/></el-input>
22 </el-form-item>
23 </el-form>
24 <template slot="footer">
25 <el-button @click="visible = false">{{ $t('cancel') }}</el-button>
26 <!-- 流程启动组件 -->
27 <ren-process-start v-if="processVisible" updateInstanceIdUrl="/act/demo/correction/updateInstanceId" saveFormUrl="/act/demo/correction" dataForm="dataForm"/>
28 </template>
29 </el-dialog>
30 </template>
31
32
33
34 <script>
35 // 引入工作流公共方法
36 import processModule from '@mixins/process-module'
37 export default {
38 // 注入公共方法
39 mixins: [processModule],
40 data () {
41 return {
42 // 是否显示流程启动组件
43 processVisible: true,
44 visible: false,
45 dataForm: {
46 id: '',
47 applyPost: '',
48 entryDate: '',
49 correctionDate: '',
50 workContent: '',
51 achievement: '',
52 creator: '',
53 createDate: ''
54 }
55 },
56
57 methods: {
58 init () {
59 this.visible = true
60 this.$nextTick(() => {
61 this.$refs['dataForm'].resetFields()
62 if (this.dataForm.id) {
63 this.getInfo()
64 }
65 })
66 // 将业务组件对象赋值给流程（回调时需要用到）
67 this.$refs.renProcessStart.rootObj = this
68 // 配置回调函数
69 this.$refs.renProcessStart.callbacks = {
70 startProcessSuccessCallback: this.closeCurrentDialog,
71 startProcessErrorCallback: this.startProcessErrorCallback,
72 formSaveSuccessCallback: null,
73 formSaveErrorCallback: null
74 }
75 // 配置流程定义KEY
76 this.$refs.renProcessStart.dataForm.processDefinitionKey = 'correctionprocess'
77 })
78 },
79 // 获取信息
80 getInfo () {
81 this.$http.get(`/act/demo/correction/${this.dataForm.id}`).then(({ data: res }) => {
82 if (res.code !== 0) {
83 return this.$message.error(res.msg)
84 }
85 this.dataForm = {
86 ...this.dataForm,
87 ...res.data
88 }
89 }).catch(() => {})
90 },
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113 }
```

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/updateInstanceId" saveFormUrl="/act/demo/correction" dataFormName="dataForm" ref="renProcessStart" ></ren-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

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

```
correction.vue X
src > views > modules > activiti > demo > correction.vue > {} "correction.vue" > script > data
1 <template>
2   <el-card shadow="never" class="avi-card--fill">
3     <div class="mod-demo_correction">
4       <el-form :inline="true" :model="dataForm" @keyup.enter.native="getDataList">
5         <el-form-item>
6           <el-button v-if="$hasPermission('activiti:correction:all')" type="primary" @click="addOrUpdateHandle()">{{ $t('add') }}</el-button>
7         </el-form-item>
8         <el-form-item>
9           <el-button v-if="$hasPermission('activiti:correction:all')" type="danger" @click="deleteHandle()">{{ $t('deleteBatch') }}</el-button>
10        </el-form-item>
11      </el-form>
12      <el-table v-loading="dataListLoading" :data="dataList" border @selection-change="dataListSelectionChangeHandle" style="width: 100%;">
13        <el-table-column type="selection" header-align="center" align="center" width="50"></el-table-column>
14        <el-table-column prop="instanceId" :label="$t('running.id')" header-align="center" align="center"></el-table-column>
15        <el-table-column prop="applyPost" :label="$t('correction.post')" header-align="center" align="center"></el-table-column>
16        <el-table-column prop="entryDate" :label="$t('correction.entryDate')" header-align="center" align="center"></el-table-column>
17        <el-table-column prop="correctionDate" :label="$t('correction.correctionDate')" header-align="center" align="center"></el-table-column>
18        <el-table-column prop="workContent" :label="$t('correction.workContent')" header-align="center" align="center"></el-table-column>
19        <el-table-column prop="achievement" :label="$t('correction.achievement')" header-align="center" align="center"></el-table-column>
20        <el-table-column prop="createDate" :label="$t('createDate')" header-align="center" align="center"></el-table-column>
21        <el-table-column :label="$t('handle')" fixed="right" header-align="center" align="center" width="150">
22          <template slot-scope="scope">
23            <el-button type="text" size="small" @click="showDetail(scope.row)">{{ $t('process.viewFlowImage') }}</el-button>
24          </template>
25        </el-table-column>
26      </el-table>
27      <el-pagination
28        :current-page="page"
29        :page-sizes="[10, 20, 50, 100]"
30        :page-size="limit"
31        :total="total"
32        layout="total, sizes, prev, pager, next, jumper"
33        @size-change="pageSizeChangeHandle"
34        @current-change="pageCurrentChangeHandle">
35      </el-pagination>
36      <!-- 弹窗, 新增 / 修改 -->
37      <add-or-update v-if="addOrUpdateVisible" ref="addOrUpdate" @refreshDataList="getDataList"></add-or-update>
38    </div>
39  </el-card>
40 </template>
41
42 <script>
43 import mixinViewModule from '@mixins/view-module'
44 import AddOrUpdate from './correction-add-or-update'
45 // 引入工作流公共方法
46 import processModule from '@mixins/process-module'
47 export default {
48   // 注入公共方法
49   mixins: [mixinViewModule, processModule],
50   data () {
51     return {
52       mixinViewModuleOptions: {
53         getDataListURL: '/act/demo/correction/page',
54         getDataListIsPage: true,
55         exportURL: '/act/demo/correction/export',
56         deleteURL: '/act/demo/correction',
57         deleteIsBatch: true
58       },
59       dataForm: {
60         id: ''
61       },
62       // 流程定义KEY
63       procDefKey: 'correctionprocess'
64     }
65   },
66   components: {
67     AddOrUpdate
68   },
69   methods: {
70     // 查看流程处理详情
71     showDetail (row) {
72       if (!row.id) {
73         return this.$message.error(this.$t('task.detailError'))
74       }
75       var params = {
76         businessKey: row.id,
77         procDefKey: this.procDefKey
78       }
79       this.getProcDefBizRouteAndProcessInstance(params, this.forwardDetail)
80     }
81   }
82 }
83 </script>
84
```

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/backToFirstStep”。
- 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：

用户管理 × 流程管理 × 我的待办 × 我的待办 - 部门经理审批 ×

* 申请岗位 测试

* 入职日期 2019-09-12

* 转正日期 2019-09-13

* 工作内容 测试

* 工作成绩 测试

通过

驳回

回退

委托

终止

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：

流程图



流转详情

执行环节	受理人	开始时间	结束时间	审核意见	任务历时(秒)
主管审批	1067246875800000001	2019-09-12 11:09:33	2019-09-12 14:07:10	22	10657

第7章 生产环境部署

部署项目前，需要准备JDK8、Maven、MySQL5.5+环境，参考开发环境搭建。

7.1 jar包部署

7.2 docker部署

7.3 跨域配置

7.1 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: 8080
  connection-timeout: 5000ms
  servlet:
    context-path: /renren-admin
    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.jar
```

- 在windows下部署，只需打开cmd窗口，输入如下命令：

```
java -jar renren-admin.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.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.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.jar --server.port=8080 --server.servlet.context-path=/renren-admin 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]# chmod +x start.sh
[root@renren renren-admin]# ./start.sh
```

7.2 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 按下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maven插件，构建docker镜像

```

#安装到本地仓库
[root@renren security-enterprise]# mvn install
[INFO] -----
[INFO] Reactor Summary:
[INFO]
[INFO] security-enterprise 2.0.0 ..... SUCCESS [ 0.385 s]
[INFO] renren-common ..... SUCCESS [ 1.569 s]
[INFO] renren-dynamic-datasource ..... SUCCESS [ 0.124 s]
[INFO] renren-admin ..... SUCCESS [ 4.179 s]
[INFO] renren-api ..... SUCCESS [ 0.610 s]
[INFO] renren-generator 2.0.0 ..... SUCCESS [ 58.582 s]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----

#打包并构建renren-admin镜像
[root@renren security-enterprise]# cd renren-admin
[root@renren renren-admin]# mvn clean package docker:build
#省略打包log...
[INFO] Building image renren/renren-admin
Step 1/6 : FROM java:8

---> d23bdf5b1b1b
Step 2/6 : EXPOSE 8080

---> Using cache
---> 47ffbf317b63
Step 3/6 : VOLUME /tmp

---> Using cache
---> 10c446ac3052
Step 4/6 : ADD renren-admin.jar /app.jar

---> 22f3909f6b79
Step 5/6 : RUN bash -c 'touch /app.jar'

```

```

---> Running in 5d374d281bd0
Removing intermediate container 5d374d281bd0
---> 054dc5b868b8
Step 6/6 : ENTRYPOINT ["java","-jar","/app.jar"]

---> Running in 20ffcbde1141
Removing intermediate container 20ffcbde1141
---> 6a5e0e02b2c6
ProgressMessage{id=null, status=null, stream=null, error=null, progress=null, progressDetail=
null}
Successfully built 6a5e0e02b2c6
Successfully tagged renren/renren-admin:latest
[INFO] Built renren/renren-admin
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----

#打包并构建renren-api镜像
[root@renren renren-admin]# cd ../renren-api
[root@renren renren-api]# mvn clean package docker:build
[INFO] Building image renren/renren-api
Step 1/6 : FROM java:8

---> d23bdf5b1b1b
Step 2/6 : EXPOSE 8081

---> Using cache
---> 141e6a4a62e1
Step 3/6 : VOLUME /tmp

---> Using cache
---> 249fbdf0d5a1
Step 4/6 : ADD renren-api.jar /app.jar

---> a775d2c75da1
Step 5/6 : RUN bash -c 'touch /app.jar'

---> Running in 53f3d695371f
Removing intermediate container 53f3d695371f
---> 909bedac70f3
Step 6/6 : ENTRYPOINT ["java","-jar","/app.jar"]

---> Running in 3eb224847362
Removing intermediate container 3eb224847362
---> 94a31a7785a0
ProgressMessage{id=null, status=null, stream=null, error=null, progress=null, progressDetail=
null}
Successfully built 94a31a7785a0
Successfully tagged renren/renren-api:latest
[INFO] Built renren/renren-api

```

```
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
```

#查看Docker镜像

```
[root@renren renren-api]# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED
renren/renren-api	latest	94a31a7785a0	3 minutes ago
renren/renren-admin	latest	6a5e0e02b2c6	6 minutes ago

- 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-compose-`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
100	11.2M	100	11.2M	0	0	1661k	0

#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/1.22.0/docker-compose-Linux-x86_64 下载到本地，再上传到服务器

- 通过docker-compose，启动项目，如下所示：

#启动项目

```
[root@renren security-enterprise]# docker-compose up -d
Creating network "security-enterprise_default" with the default driver
Creating security-enterprise_renren-api_1 ... done
Creating security-enterprise_renren-admin_1 ... done
```

#查看启动的容器

```
[root@renren security-enterprise]# docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED
bca4f661cb44	renren/renren-admin	"java -jar /app.jar"	About a minute ag

```
o   Up About a minute   0.0.0.0:8080->8080/tcp   security-enterprise_renren-admin_1
e902edfe59db           renren/renren-api           "java -jar /app.jar"       About a minute ag
o   Up About a minute   0.0.0.0:8081->8081/tcp   security-enterprise_renren-api_1
```

#查看启动的logs

```
[root@renren security-enterprise]# docker logs -f security-enterprise_renren-admin_1
```

#停掉并删除，docker-compose管理的容器

```
[root@renren security-enterprise]# docker-compose down
Stopping security-enterprise_renren-api_1   ... done
Stopping security-enterprise_renren-admin_1 ... done
Removing security-enterprise_renren-api_1   ... done
Removing security-enterprise_renren-admin_1 ... done
Removing network security-enterprise_default
```

- docker-compose官方文档

<https://docs.docker.com/compose/compose-file/compose-file-v2/>

7.3 跨域配置

跨域一般通过CORS解决，通过Nginx配置即可，CORS需要浏览器和服务器同时支持。目前，主流浏览器都支持该功能，Nginx配置如下所示：

```
server {
    listen      80;
    server_name demo.renren.io;
    location /security-enterprise {
        alias /data/security-enterprise-admin;
        index index.html;
    }

    location / {
        if ($request_method = 'OPTIONS') {
            add_header 'Access-Control-Allow-Origin' '$http_origin';
            add_header 'Access-Control-Allow-Credentials' 'true';
            add_header 'Access-Control-Allow-Methods' 'GET, POST, PUT, DELETE, OPTIONS';
            add_header 'Access-Control-Allow-Headers' 'DNT,X-CustomHeader,Keep-Alive,User-Agent,X-Requested-With,If-Modified-Since,Cache-Control,Content-Type,token';
            add_header 'Access-Control-Max-Age' 1728000;
            add_header 'Content-Type' 'text/plain charset=UTF-8';
            add_header 'Content-Length' 0;
            return 204;
        }

        add_header 'Access-Control-Allow-Origin' '$http_origin';
        add_header 'Access-Control-Allow-Credentials' 'true';
        add_header 'Access-Control-Allow-Methods' 'GET, POST, PUT, DELETE, OPTIONS';
        add_header 'Access-Control-Allow-Headers' 'DNT,X-CustomHeader,Keep-Alive,User-Agent,X-Requested-With,If-Modified-Since,Cache-Control,Content-Type,token';

        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;
    }
}
```