

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 生产环境部署	1.6
5.1 jar包部署	1.6.1
5.2 docker部署	1.6.2
5.3 跨域配置	1.6.3

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。

文档更新

本文档由人人开源于2018年10月25日最后修订。

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。

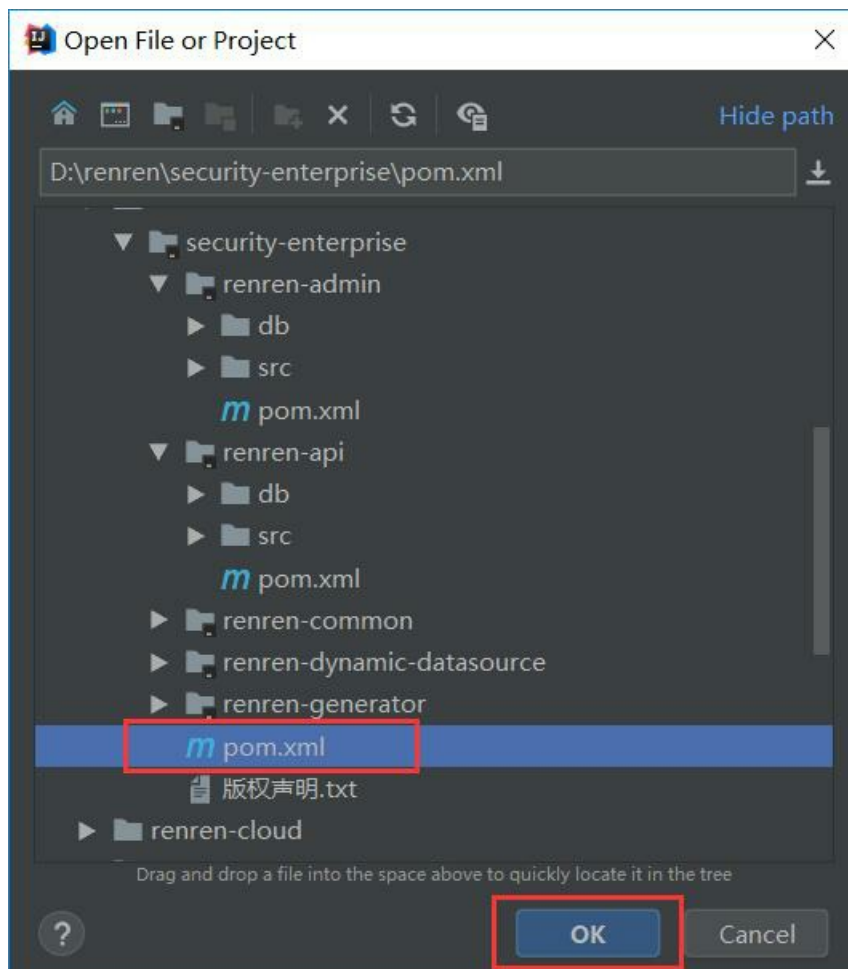
1.3 开发环境搭建

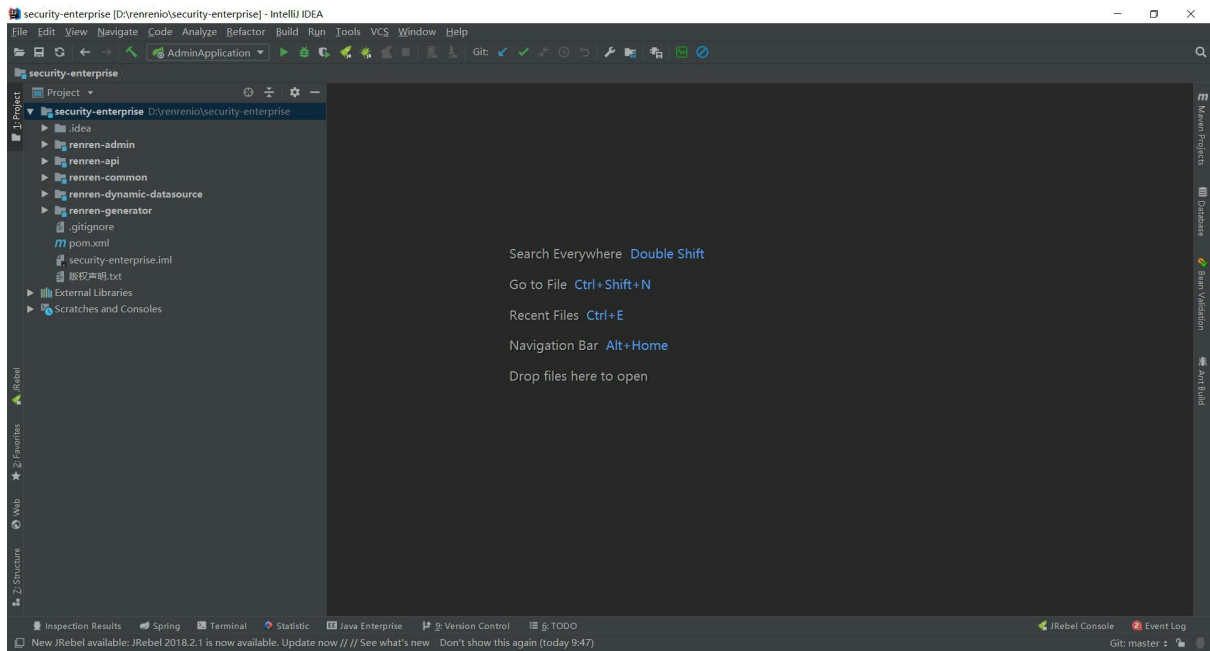
1.3.1 软件需求

- JDK 1.8+
- Maven 3.0+
- MySQL 5.5+
- Oracle 11g+
- SQL Server 2012+
- PostgreSQL 9.4+

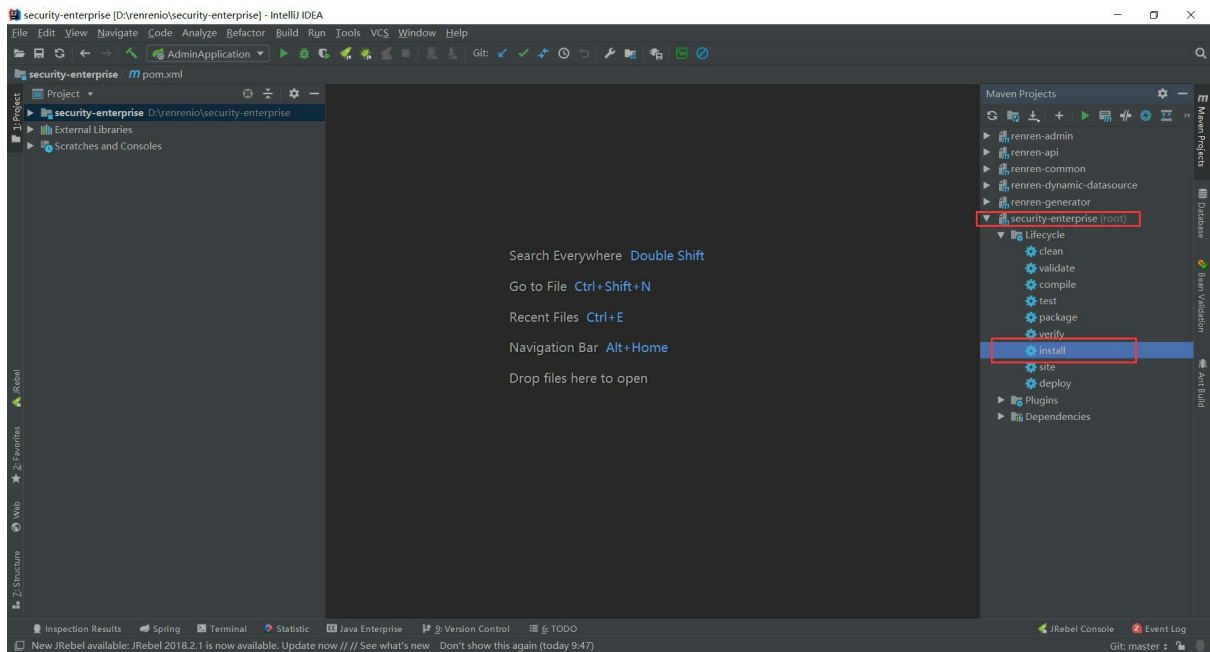
1.3.2 IDEA开发工具

- IDEA打开项目， `File -> Open` 如下图：



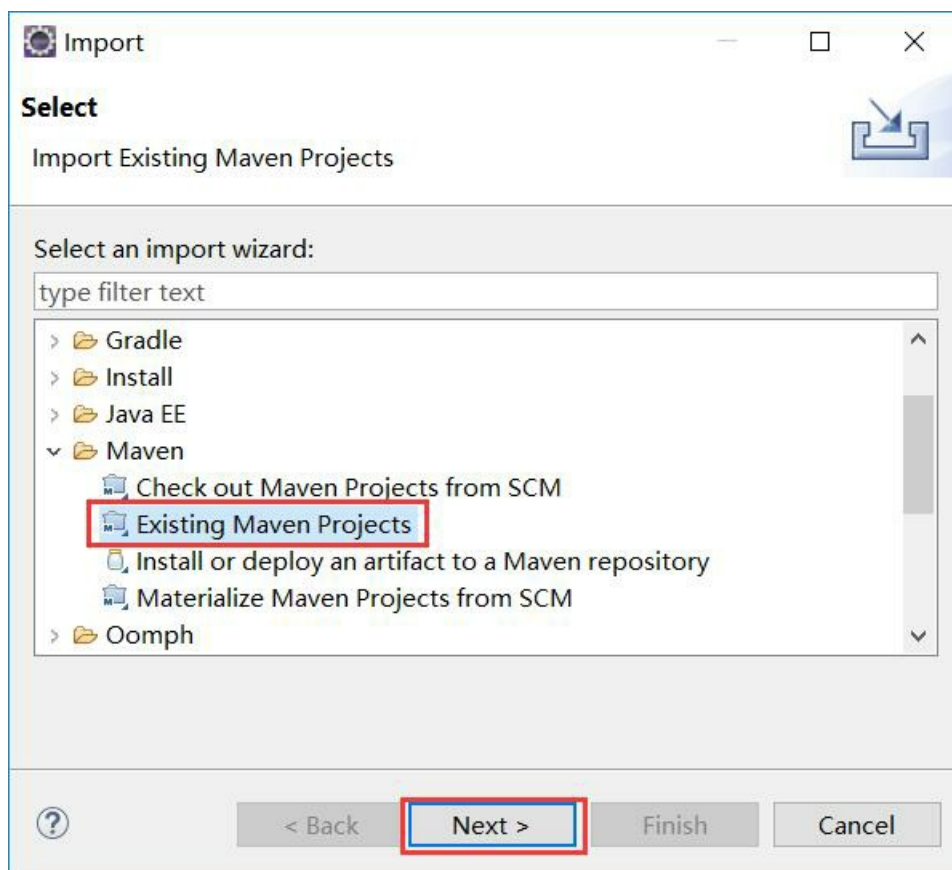
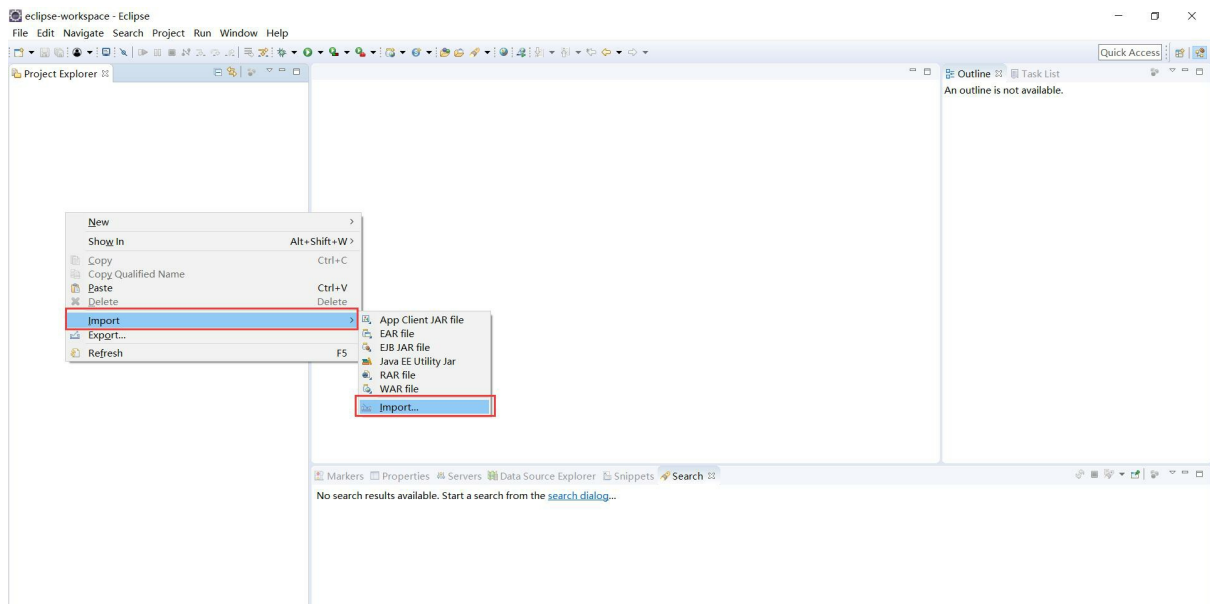


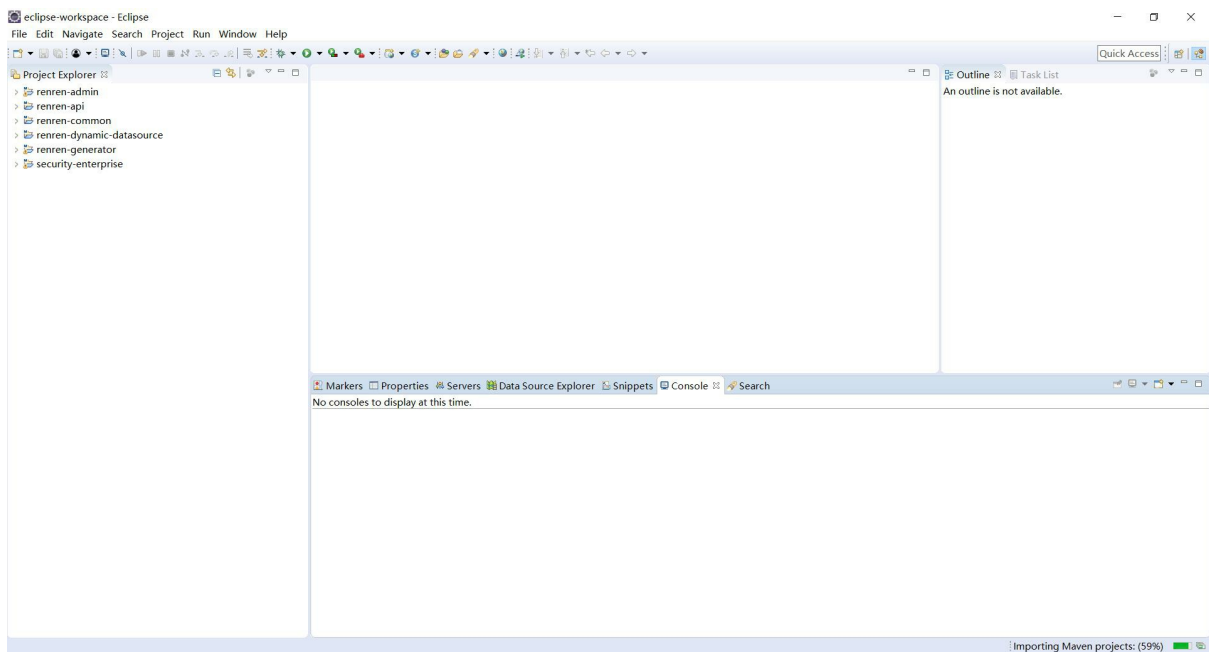
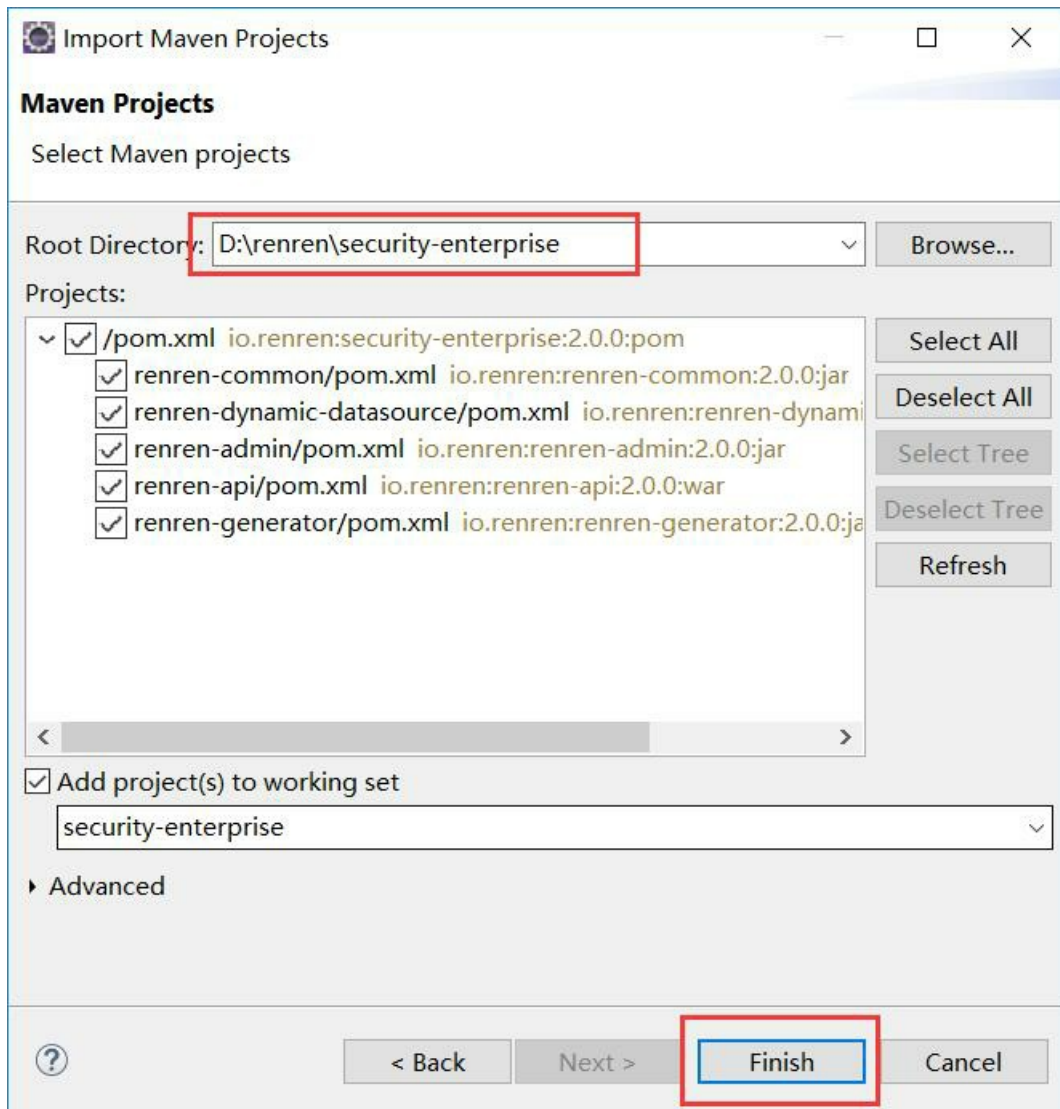
- 在security-enterprise目录下，执行mvn install，如下图所示：



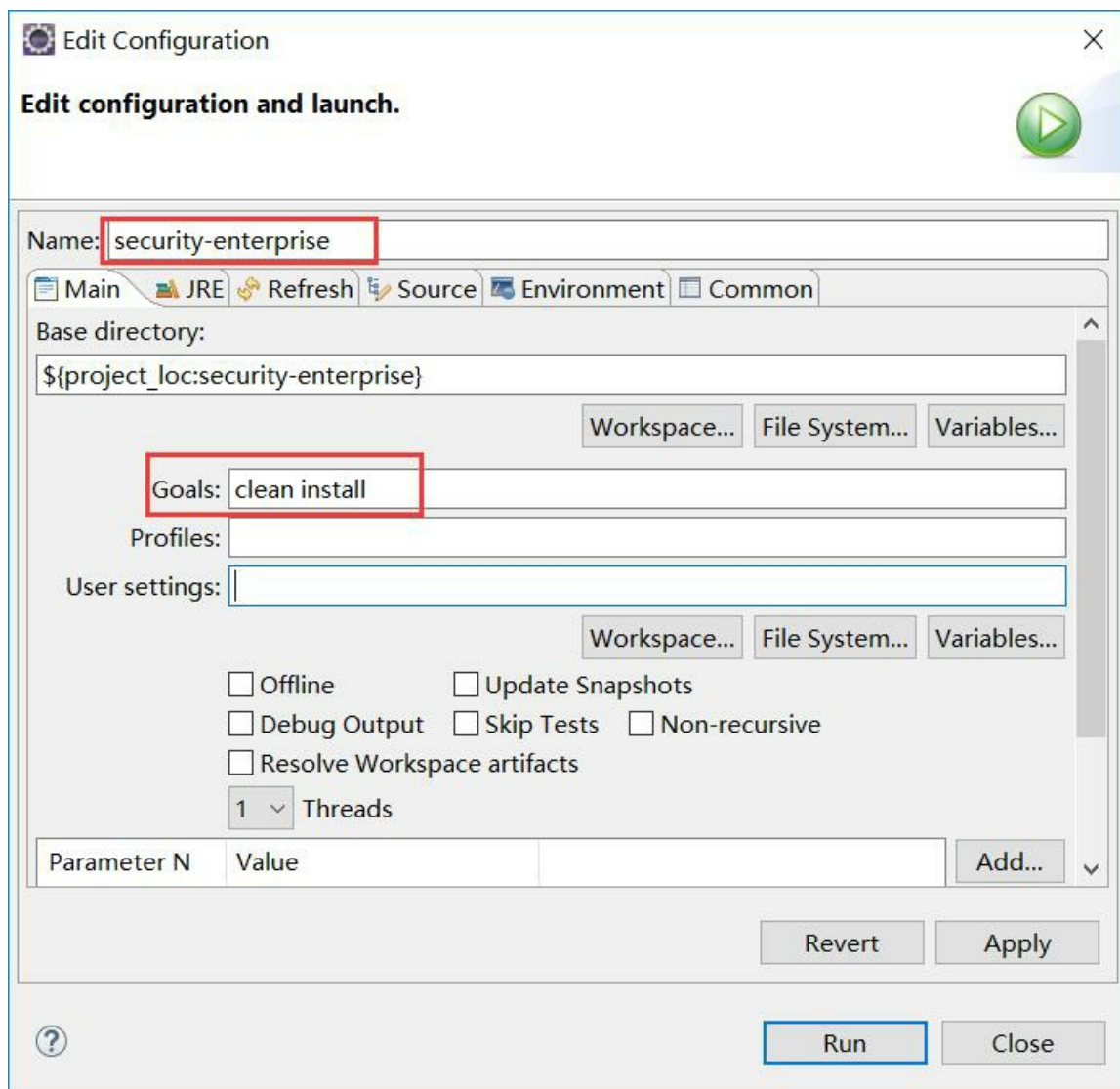
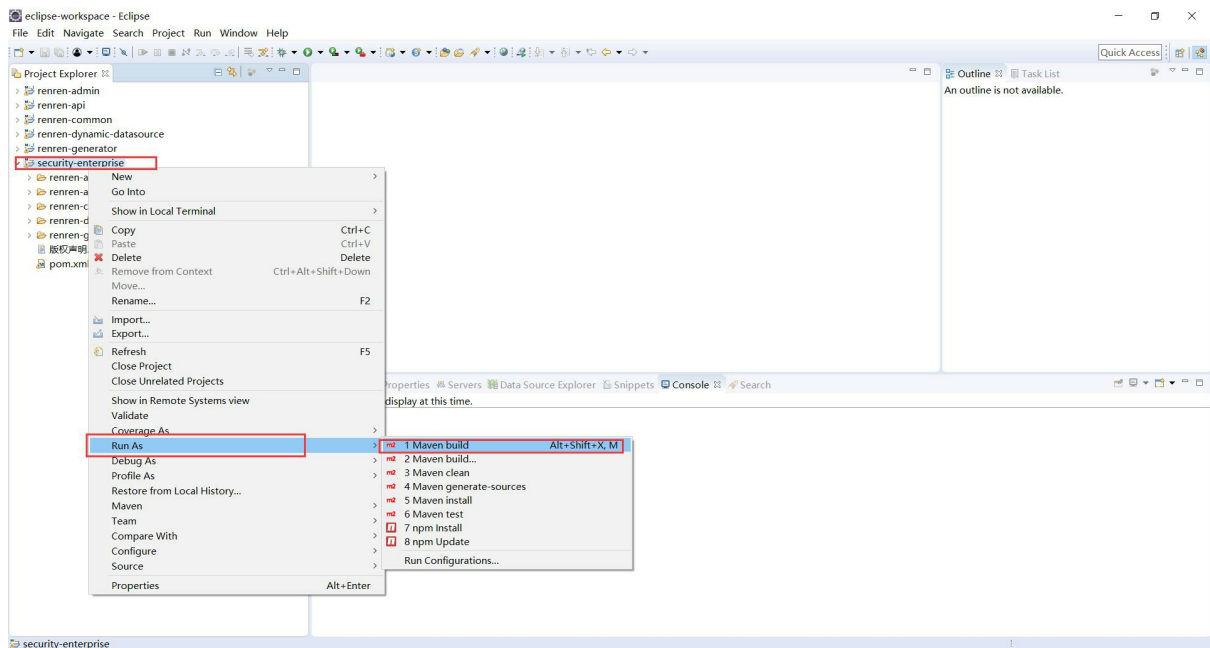
1.3.3 Eclipse开发工具

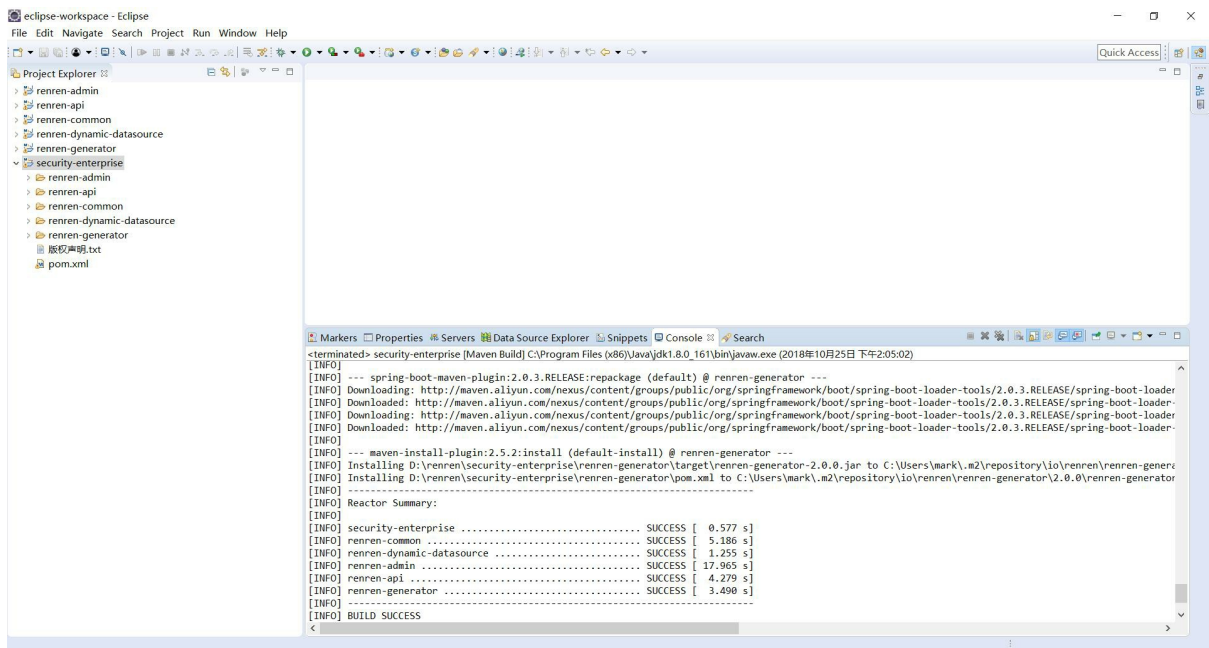
- Eclipse导入项目，如下图：





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



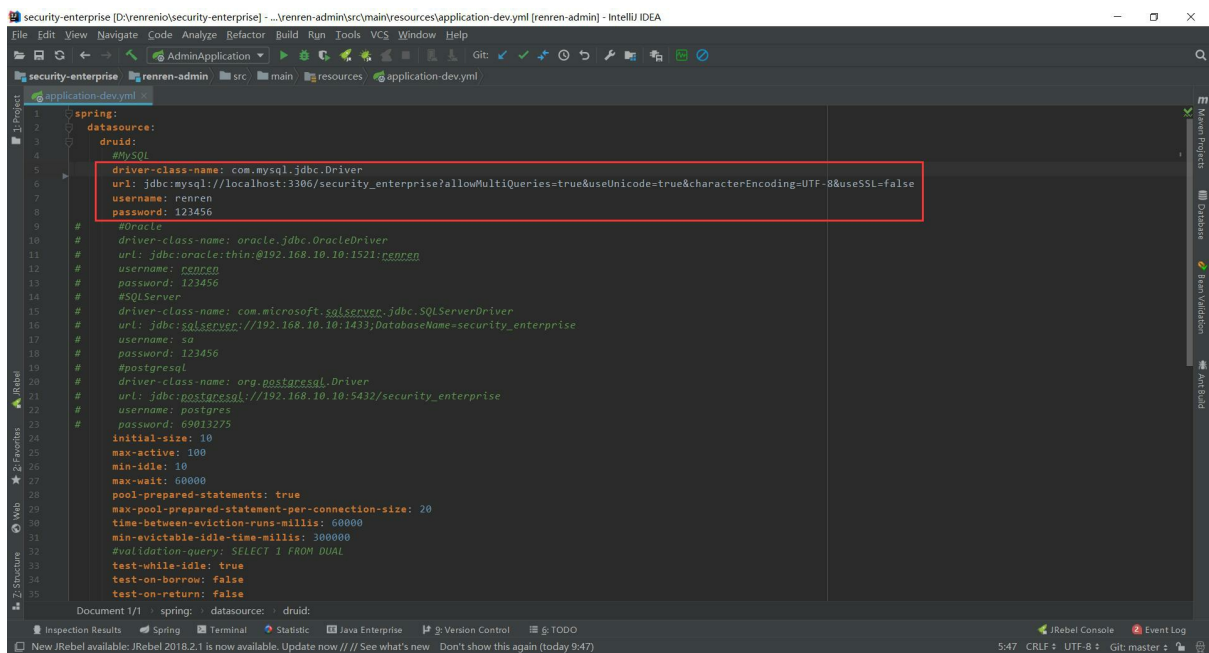


1.3.4 创建数据库

- 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

1.3.5 修改配置文件

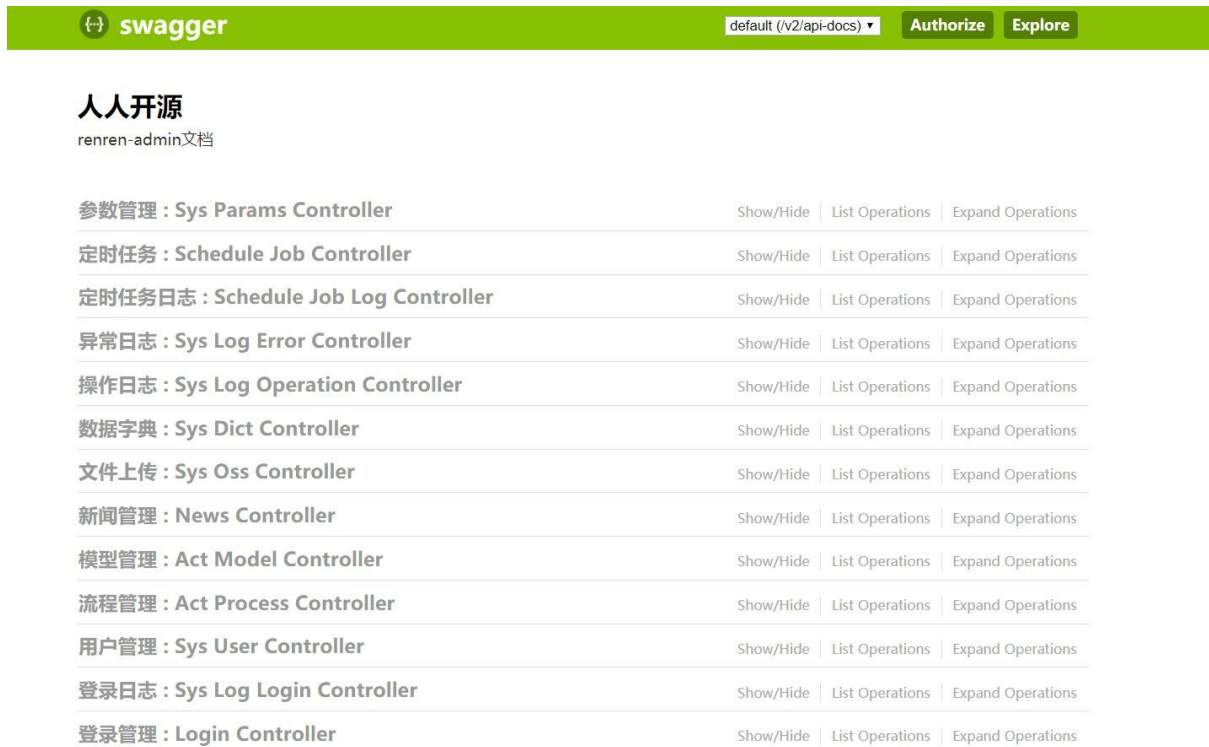
修改renren-admin配置文件 application-dev.yml，如下图所示：



1.3.6 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
- Swagger地址: <http://localhost:8080/renren-admin/swagger-ui.html>



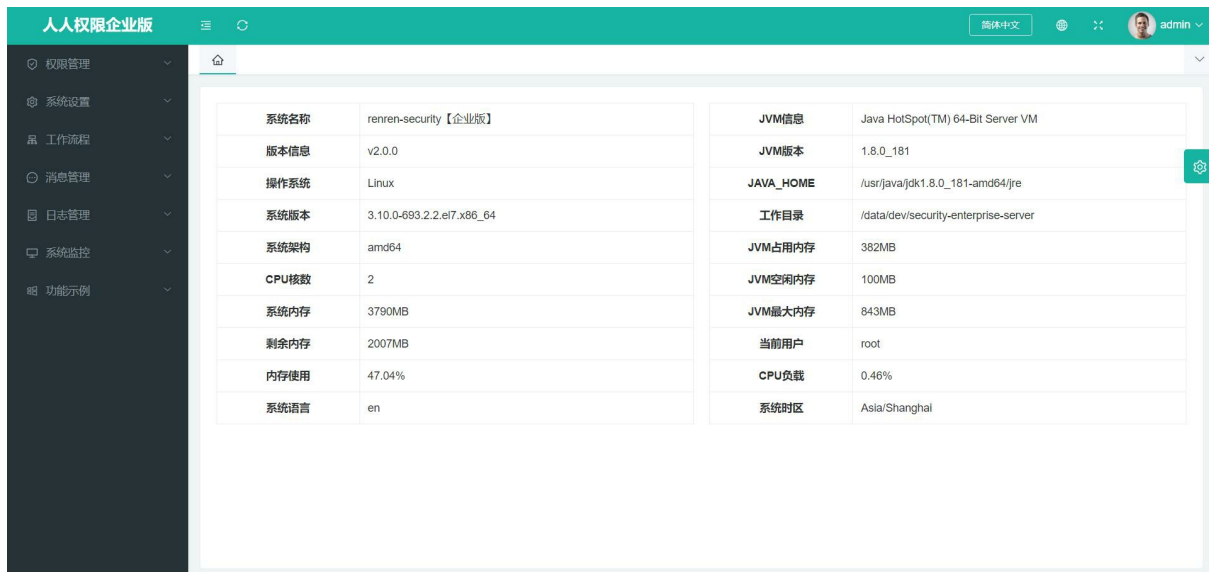
The image shows the Swagger UI for the renren-admin API. The header is green with the 'swagger' logo and 'default (v2/api-docs)' selected. Below the header, the title '人人开源' (Renren Open Source) is displayed, followed by 'renren-admin文档'. A table lists various API endpoints with their corresponding controllers and actions.

Controller	Show/Hide	List Operations	Expand Operations
参数管理 : Sys Params Controller	Show/Hide	List Operations	Expand Operations
定时任务 : Schedule Job Controller	Show/Hide	List Operations	Expand Operations
定时任务日志 : Schedule Job Log Controller	Show/Hide	List Operations	Expand Operations
异常日志 : Sys Log Error Controller	Show/Hide	List Operations	Expand Operations
操作日志 : Sys Log Operation Controller	Show/Hide	List Operations	Expand Operations
数据字典 : Sys Dict Controller	Show/Hide	List Operations	Expand Operations
文件上传 : Sys Oss Controller	Show/Hide	List Operations	Expand Operations
新闻管理 : News Controller	Show/Hide	List Operations	Expand Operations
模型管理 : Act Model Controller	Show/Hide	List Operations	Expand Operations
流程管理 : Act Process Controller	Show/Hide	List Operations	Expand Operations
用户管理 : Sys User Controller	Show/Hide	List Operations	Expand Operations
登录日志 : Sys Log Login Controller	Show/Hide	List Operations	Expand Operations
登录管理 : Login Controller	Show/Hide	List Operations	Expand Operations

- 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

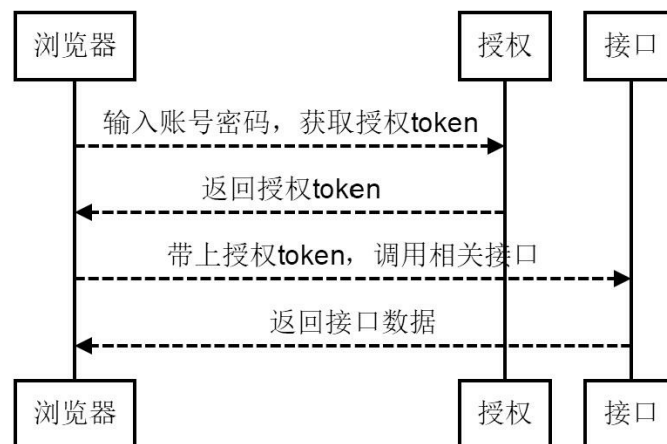
- Eclipse、IDEA运行ApiApplication.java，则可启动项目【renren-api】
- Swagger路径：<http://localhost:8081/renren-api/swagger-ui.html>

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/renren_cloud?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: renren_cloud
      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=renren_cloud
      username: sa
      password: 123456
```

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

2.4 PostgreSQL数据库支持

1) 修改数据库配置信息，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/renren_cloud
      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，测试多数据源及事物

```
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(String id){
        SysUserEntity user = new SysUserEntity();
        user.setId(id);
        user.setMobile("13500000000");
        sysUserDao.updateById(user);
    }

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



```

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

        //测试事物
        int i = 1/0;
    }
}

```

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

```

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(){
        String id = "fdc8e752cc7f41828f3b1d5f7effc7dd";

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

```

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

```

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
```

3.3 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
 * @since 1.0.0
 */
@Target({ElementType.METHOD, ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface DataSource {
    String value() default "";
}
```

2) 定义读取多数据源配置文件的类，如下所示：

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

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

```

private boolean poolPreparedStatements = false;
private int maxOpenPreparedStatements = -1;
private boolean sharePreparedStatements = false;
private String filters = "stat,wall";

public String getDriverClassName() {
    return driverClassName;
}

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

public String getUrl() {
    return url;
}

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

public String getUsername() {
    return username;
}

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

public String getPassword() {
    return password;
}

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

public int getInitialSize() {
    return initialSize;
}

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

public int getMaxActive() {
    return maxActive;
}

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

```

```

public int getMinIdle() {
    return minIdle;
}

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

public long getMaxWait() {
    return maxWait;
}

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

public long getTimeBetweenEvictionRunsMillis() {
    return timeBetweenEvictionRunsMillis;
}

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

public long getMinEvictableIdleTimeMillis() {
    return minEvictableIdleTimeMillis;
}

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

public long getMaxEvictableIdleTimeMillis() {
    return maxEvictableIdleTimeMillis;
}

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

public String getValidationQuery() {
    return validationQuery;
}

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

public int getValidationQueryTimeout() {
    return validationQueryTimeout;
}

```

```

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

public boolean isTestOnBorrow() {
    return testOnBorrow;
}

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

public boolean isTestOnReturn() {
    return testOnReturn;
}

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

public boolean isTestWhileIdle() {
    return testWhileIdle;
}

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

public boolean isPoolPreparedStatements() {
    return poolPreparedStatements;
}

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

public int getMaxOpenPreparedStatements() {
    return maxOpenPreparedStatements;
}

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

public boolean isSharePreparedStatements() {
    return sharePreparedStatements;
}

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

```

```

    public String getFilters() {
        return filters;
    }

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

```

```

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

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

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

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

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

```

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

```

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

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

    @Override

```

```

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

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

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

```

4) 配置多数据源，如下所示：

```

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

```

```

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

```

```

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

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

```

//因为DynamicDataSource是继承与AbstractRoutingDataSource，而AbstractRoutingDataSource又是继承于AbstractDataSource，AbstractDataSource实现了统一的DataSource接口，所以DynamicDataSource也可以当做DataSource使用

```

@Bean

```



```

    public DynamicDataSource dynamicDataSource(DataSourceProperties dataSourceProperties) {
        DynamicDataSource dynamicDataSource = new DynamicDataSource();
        dynamicDataSource.setTargetDataSources(getDynamicDataSource());

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

        return dynamicDataSource;
    }

    private Map<Object, Object> getDynamicDataSource(){
        Map<Object, Object> targetDataSources = new HashMap<>();
        properties.getDatasource().forEach((k, v) -> {
            DruidDataSource druidDataSource = DynamicDataSourceFactory.buildDruidDataSource(v);
            targetDataSources.put(k, druidDataSource);
        });

        return targetDataSources;
    }
}

```

```

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

import java.sql.SQLException;

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

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

        druidDataSource.setInitialSize(properties.getInitialSize());
        druidDataSource.setMaxActive(properties.getMaxActive());
        druidDataSource.setMinIdle(properties.getMinIdle());
        druidDataSource.setMaxWait(properties.getMaxWait());
    }
}

```

```

        druidDataSource.setTimeBetweenEvictionRunsMillis(properties.getTimeBetweenEvictionRunsMillis());
        druidDataSource.setMinEvictableIdleTimeMillis(properties.getMinEvictableIdleTimeMillis());
        druidDataSource.setMaxEvictableIdleTimeMillis(properties.getMaxEvictableIdleTimeMillis());
        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.DynamicDataSource;
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
 * @since 1.0.0
 */
@Aspect
@Component
@Order(Ordered.HIGHEST_PRECEDENCE)

```

```

public class DataSourceAspect {
    protected Logger logger = LoggerFactory.getLogger(getClass());

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

    }

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

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

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

        try {
            return point.proceed();
        } finally {
            DynamicDataSource.clearDataSource();
            logger.debug("clean datasource");
        }
    }
}

```

第4章 基础知识讲解

4.1 Spring MVC使用

4.2 Swagger使用

4.3 Mybatis-plus使用

4.4 Hibernate Validator使用

4.1 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 {

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



```

resources/webjars/");
    }

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

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

```

4.2.2 Swagger常用注解

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

```

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

}

```

- @ApiOperation注解用在方法上，说明该方法的作用，如下所示：

```

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

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

```

```
}
```

- @ApiModelProperty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.entity
  global-config:
    #数据库相关配置
    db-config:
      #主键类型  AUTO:"数据库ID自增", INPUT:"用户输入ID", ID_WORKER:"全局唯一ID (数字类型唯一ID)"
      , UUID:"全局唯一ID UUID";
      id-type: UUID
      #字段策略 IGNORED:"忽略判断",NOT_NULL:"非 NULL 判断"),NOT_EMPTY:"非空判断"
      field-strategy: NOT_NULL
      #驼峰下划线转换
      column-underline: true
      db-type: mysql
      banner: false
  #原生配置
  configuration:
    map-underscore-to-camel-case: true
    cache-enabled: false
    call-setters-on-nulls: true
```

4.4 Hibernate Validator使用

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

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

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

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

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

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

- 使用方式如下所示：

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

    @NotNull(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 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。

5.1 jar包部署

5.2 docker部署

5.3 跨域配置

5.1 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
```

5.2 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 -s -o /usr/local/bin/docker-compose
```

% Total	% Received	% Xferd	Average Speed	Time	Time	Time	Current
			Dload Upload	Total	Spent	Left	Speed
100 617	0 617	0 0	555	0	--:--:--	0:00:01	--:--:-- 556
100 11.2M	100 11.2M	0 0	1661k	0	0:00:06	0:00:06	--:--:-- 2845k

#增加可执行权限

```
[root@renren ~]# chmod +x /usr/local/bin/docker-compose
```

#查看版本信息

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

如果下载不了，可以用迅雷将 https://github.com/docker/compose/releases/download/1.22.0/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 ago

```
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/>

5.3 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;
    }
}
```