platform-wechat-mall 商业版文档

1. 版权说明

1.1. 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 pwm 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，不承担任何责任。

1.2. 文档更新

- 本文档由李鹏军于 2018 年 9 月 25 日最后修订。

2. 介绍

2.1. 项目背景

12 月 18 日，腾讯 CEO 马化腾在第二届深商大会“互联与时代”论坛上透露，微信小程序将会在 2017 年春节前推出。移动互联网繁荣发展的背景下，app 产品在应用市场数量众多，且更新换代速度极快，这就带来了下载与卸载的烦恼。小程序是一种新的开放能力，开发者可以快速地开发一个小程序。小程序可以在微信内被便捷地获取和传播，同时具有出色的使用体验。

2.1.1. 全新服务体验

微信小程序是一种全新的连接用户与服务的方式， 它可以在微信内被便捷地获取和传播，同时具有出色的使用体验。

2.1.2. 庞大用户量

微信 2016 年已突破 9 亿用户，月活用户已近 7 亿，现 在每天每人花费在微信上的时间占了所有程序的 30%以上，其庞大用户 群体以及生态链条，是商家抢占流量入口的必争之地。

2.1.3. 流量引导

微信小程序作为连接微信与 APP 之间的桥梁，可以起到流量引导作用。

2.2. 项目描述

pwm 是一套极速开发微信小程序商城框架，主要包括用户管理、角色管理、部门管理、菜单管理、定时任务、文件上传、数据权限、Redis 缓存、前后台统一异常处理等系统通用功能，还拥有一套完整的商城后台管理系统、微信小程序源码、小程序接口服务、以及完善的支付流程，极大缩短项目的开发周期。

2.3. 项目特点

- platform-wechat-mall 采用 Spring、MyBatis、Shiro、swagger 框架开发。
- 灵活的权限控制，可控制到页面或按钮，满足绝大部分的权限需求。
- 完善的部门管理及数据权限，通过注解实现数据权限的控制。
- 支持 MySQL 数据库。

2.4. 项目介绍

2.4.1. platform-admin

后台模块，也是系统的核心，用来开发后台管理系统和商城的后台管理功能。

2.4.2. platform-api

接口模块，是小程序商城的接口开发模块。实现了微信用户登录、接口权限认证、获取登录用户、商城首页、专题、分类、购物车、个人中心等功能，为小程序商城接口的安全调用，提供一套完整的解决方案。

2.4.3. platform-common

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 platform-admin、platform-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。

2.4.4. platform-framework

系统 web 合并模块，最终项目打包部署模块。最后会介绍为什么会设计此模块，以及设计此模块的意图。

2.4.5. platform-gen

代码生成器模块，只需在数据库里，创建好表结构，就可以生成增、删、改、查等操作的代码，包括 entity、mapper、dao、service、controller、页面等所有代码，项目开发神器。

2.4.6. platform-schedule

定时任务模块，使用开源框架 quartz 实现分布式定时任务，动态添加、修改、删除、暂停、恢复、立即执行定时任务。

2.4.7. wx-mall

商城小程序端源码

2.5. 开发使用到的软件和工具

Xshell6、Xftp6、Git、Tomcat8.0.33、jdk1.8、MySQL5.7、redis4.0.1

2.6. 本地部署

- 配置环境（推荐 jdk1.8、maven3.3、tomcat8、mysql5.5+、redis4.0.1）

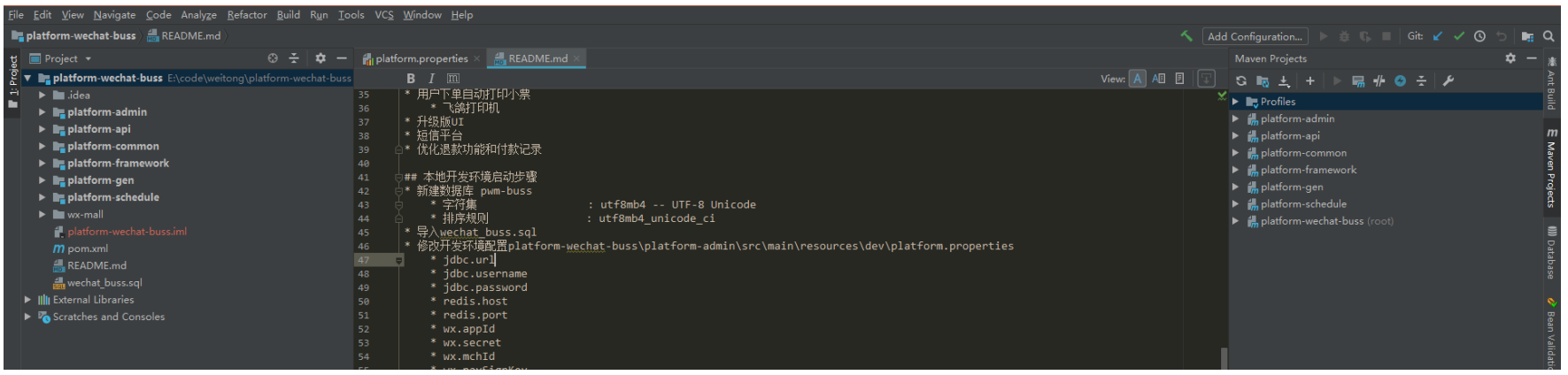
2.6.1. 本机启动 redis 服务、mysql 数据库

- 启动 redis 服务
- 启动 mysql 服务

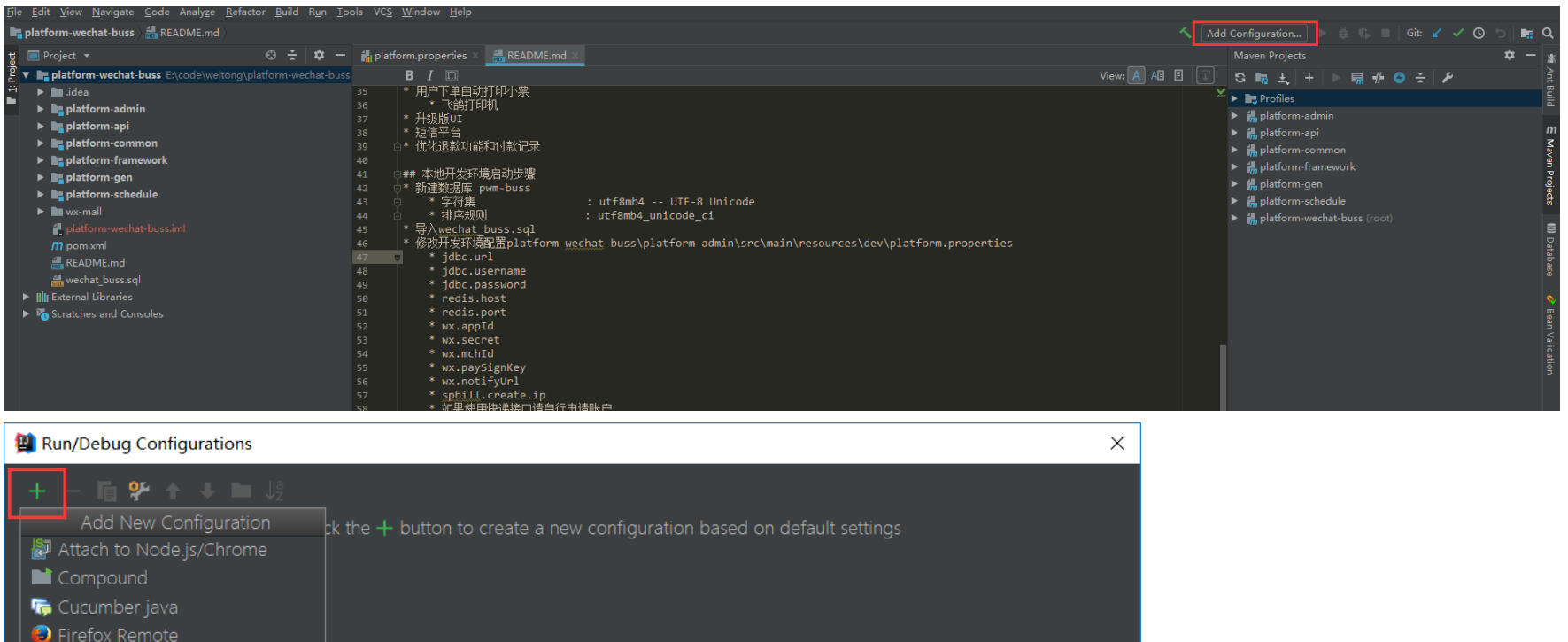
2.6.2. 初始化项目

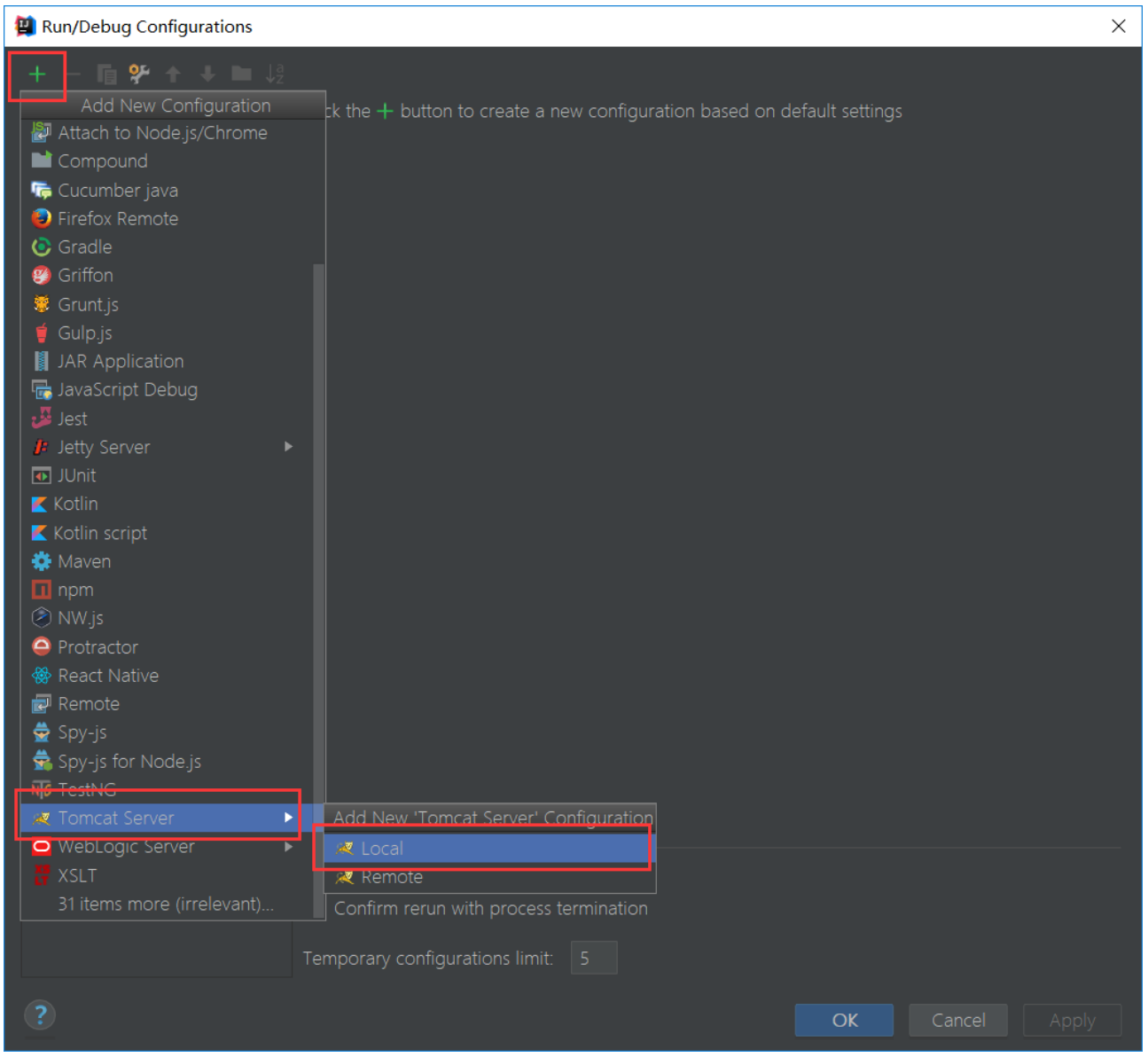
- 创建数据库 platform-shop，数据库编码为 utf8mb4，执行数据库脚本/wechat_buss.sql
- 启动项目之前修改 platform-wechat-buss\platform-admin\src\main\resources\dev\platform.properties，详情请查看 README.md

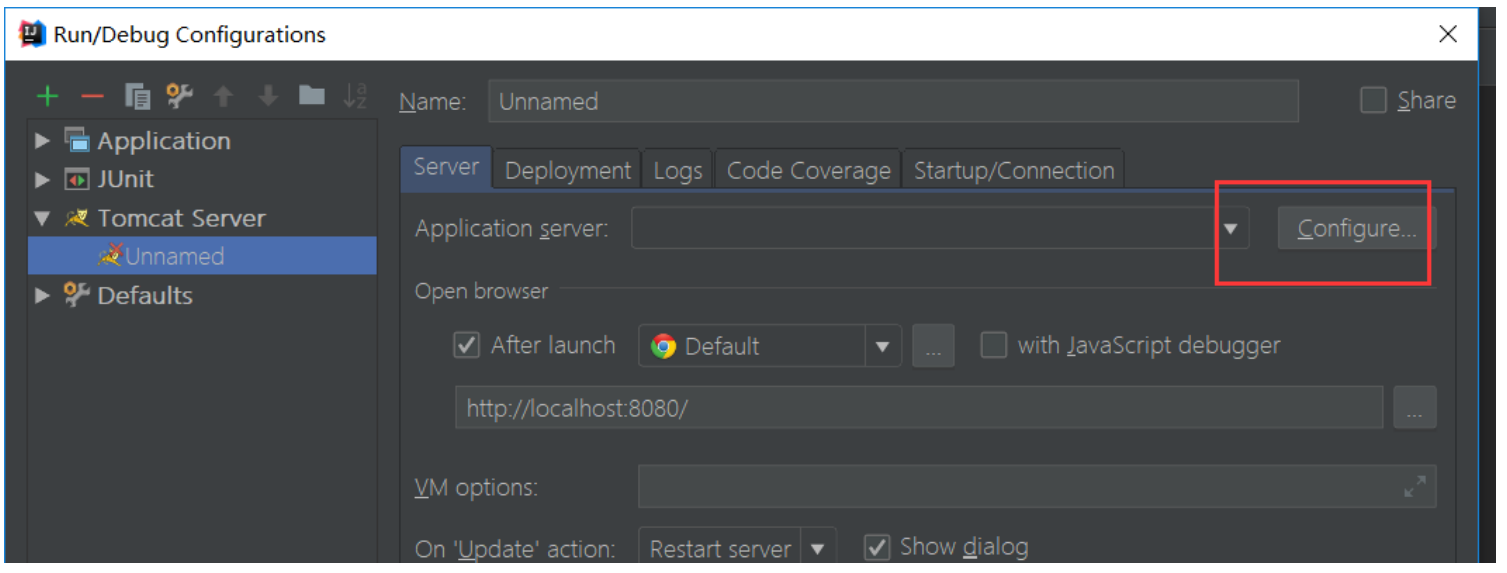
2.6.3. 使用 IDEA 启动项目

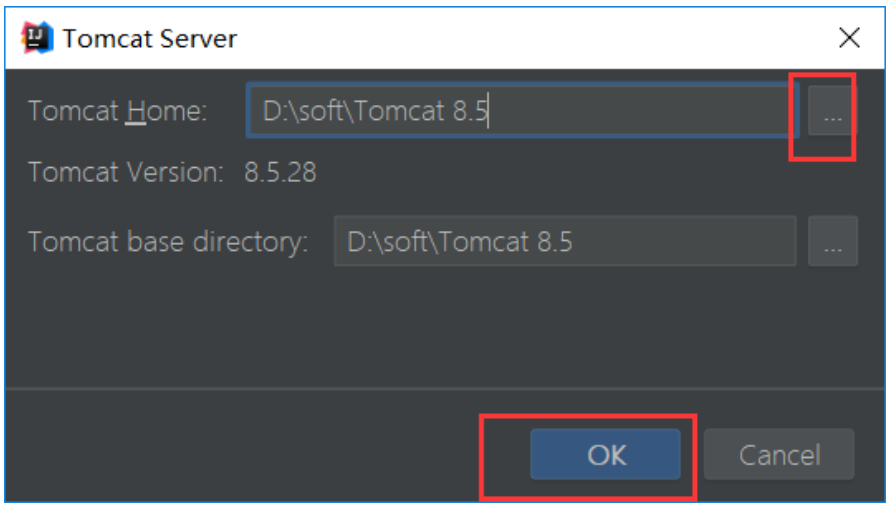

platform-wechat-buss
README.md
Add Configuration...
Project
platform-wechat-buss E:\code\weitong\platform-wechat-buss
.idea
platform-admin
platform-api
platform-common
platform-framework
platform-gen
platform-schedule
wx-mall
platform-wechat-buss.iml
pom.xml
README.md
wechat_buss.sql
External Libraries
Scratches and Consoles
platform.properties
README.md
Maven Projects
Profiles
platform-admin
platform-api
platform-common
platform-framework
platform-gen
platform-schedule
platform-wechat-buss (root)
* 用户下单自动打印小票
* 飞鹅打印机
* 升级版UI
* 短信平台
* 优化退款功能和付款记录
本地开发环境启动步骤
* 新建数据库 pwm-buss
* 字符集 : utf8mb4 -- UTF-8 Unicode
* 排序规则 : utf8mb4_unicode_ci
* 导入wechat_buss.sql
* 修改开发环境配置platform-wechat-buss\platform-admin\src\main\resources\dev\platform.properties
* jdbc.url
* jdbc.username
* jdbc.password
* redis.host
* redis.port
* wx.appId
* wx.secret
* wx.mchId

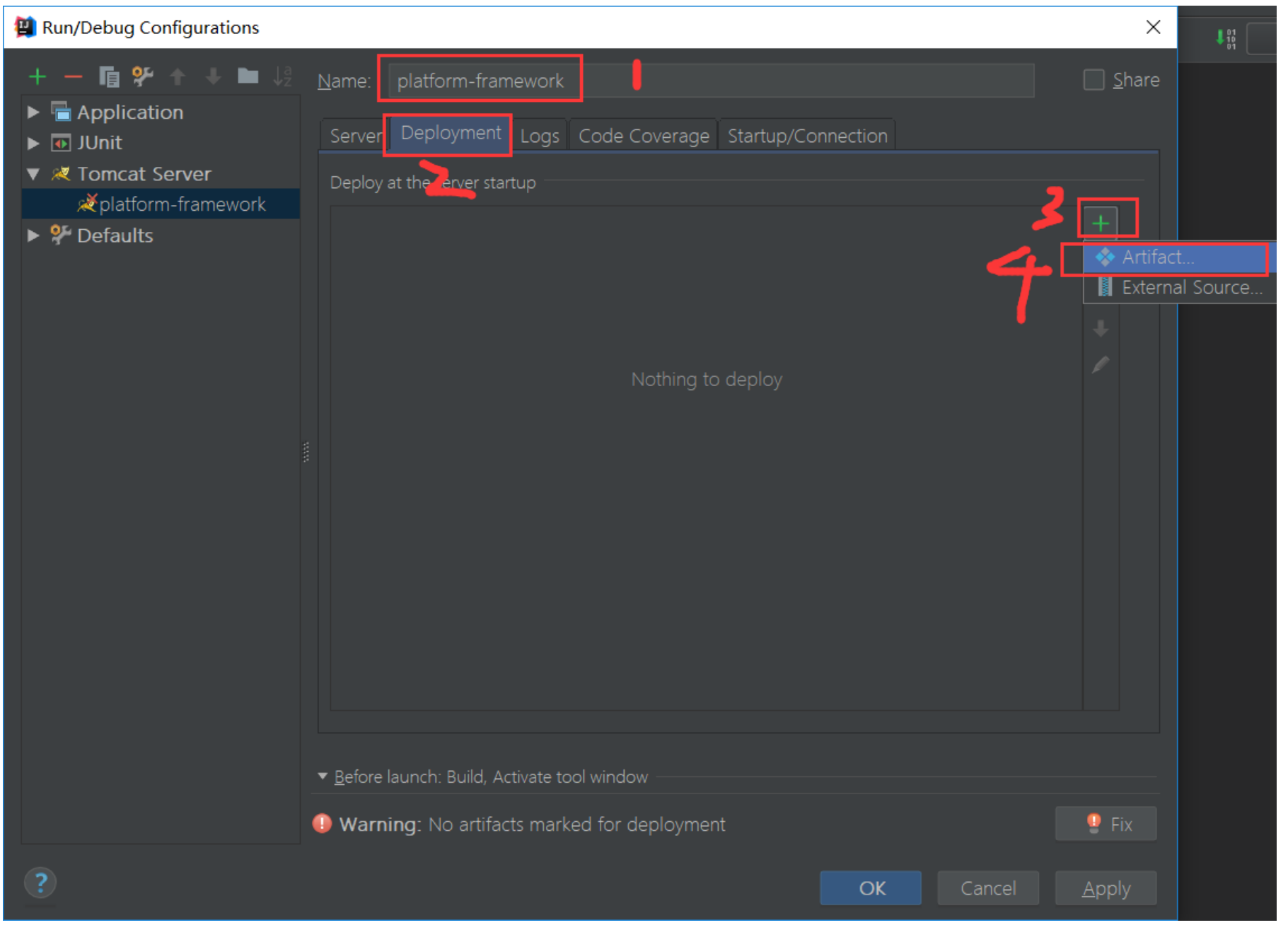
配置 tomcat

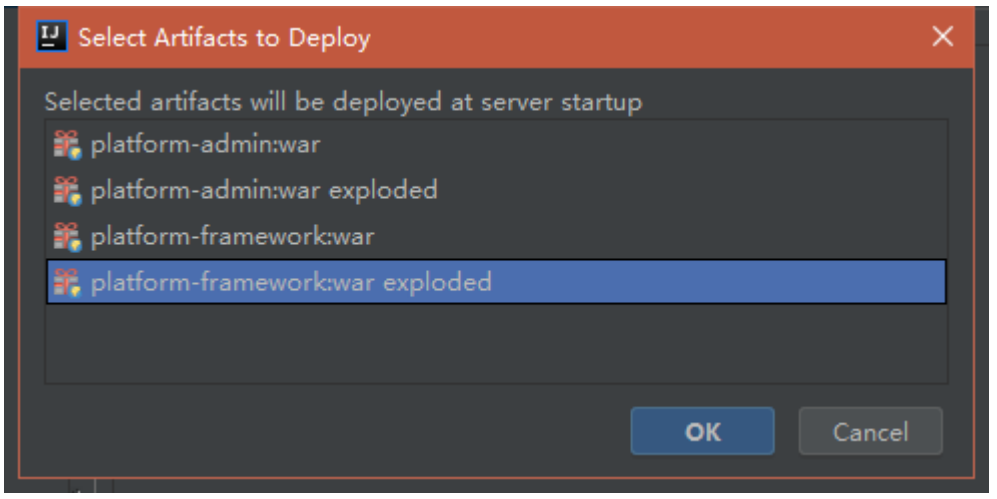

Add Configuration...
platform-wechat-buss
README.md
Project
platform-wechat-buss E:\code\weitong\platform-wechat-buss
.idea
platform-admin
platform-api
platform-common
platform-framework
platform-gen
platform-schedule
wx-mall
platform-wechat-buss.iml
pom.xml
README.md
wechat_buss.sql
External Libraries
Scratches and Consoles
platform.properties
README.md
Maven Projects
Profiles
platform-admin
platform-api
platform-common
platform-framework
platform-gen
platform-schedule
platform-wechat-buss (root)
* 用户下单自动打印小票
* 飞鹅打印机
* 升级版UI
* 短信平台
* 优化退款功能和付款记录
本地开发环境启动步骤
* 新建数据库 pwm-buss
* 字符集 : utf8mb4 -- UTF-8 Unicode
* 排序规则 : utf8mb4_unicode_ci
* 导入wechat_buss.sql
* 修改开发环境配置platform-wechat-buss\platform-admin\src\main\resources\dev\platform.properties
* jdbc.url
* jdbc.username
* jdbc.password
* redis.host
* redis.port
* wx.appId
* wx.secret
* wx.mchId
* wx.paySignKey
* wx.notifyUrl
* spbill.create.ip

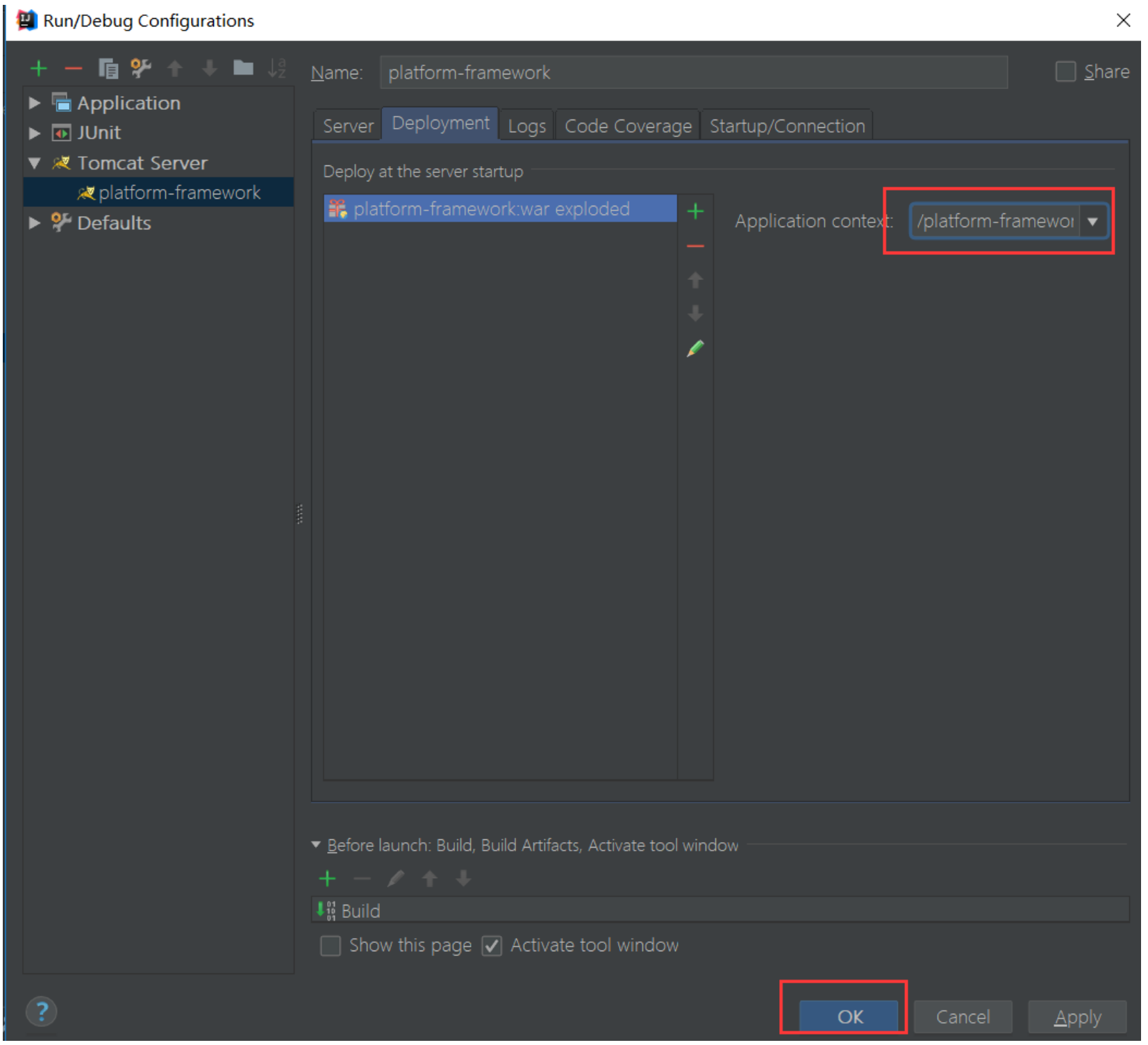

Run/Debug Configurations
Add New Configuration
Attach to Node.js/Chrome
Compound
Cucumber java
Firefox Remote
Gradle
Griffon
Grunt.js
Gulp.js
JAR Application
JavaScript Debug
Jest
Jetty Server
JUnit
Kotlin
Kotlin script
Maven
npm
NW.js
Protractor
React Native
Remote
Spy-js
Spy-js for Node.js
TestNG
Tomcat Server
WebLogic Server
XSLT
31 items more (irrelevant)...
Add New 'Tomcat Server' Configuration
Local
Remote
Confirm rerun with process termination
Temporary configurations limit: 5
OK
Cancel
Apply


Run/Debug Configurations
Application
JUnit
Tomcat Server
Unnamed
Defaults
Name: Unnamed
Share
Server
Deployment
Logs
Code Coverage
Startup/Connection
Application server:
Configure...
Open browser
After launch
Default
with JavaScript debugger
http://localhost:8080/
VM options:
On 'Update' action:
Restart server
Show dialog

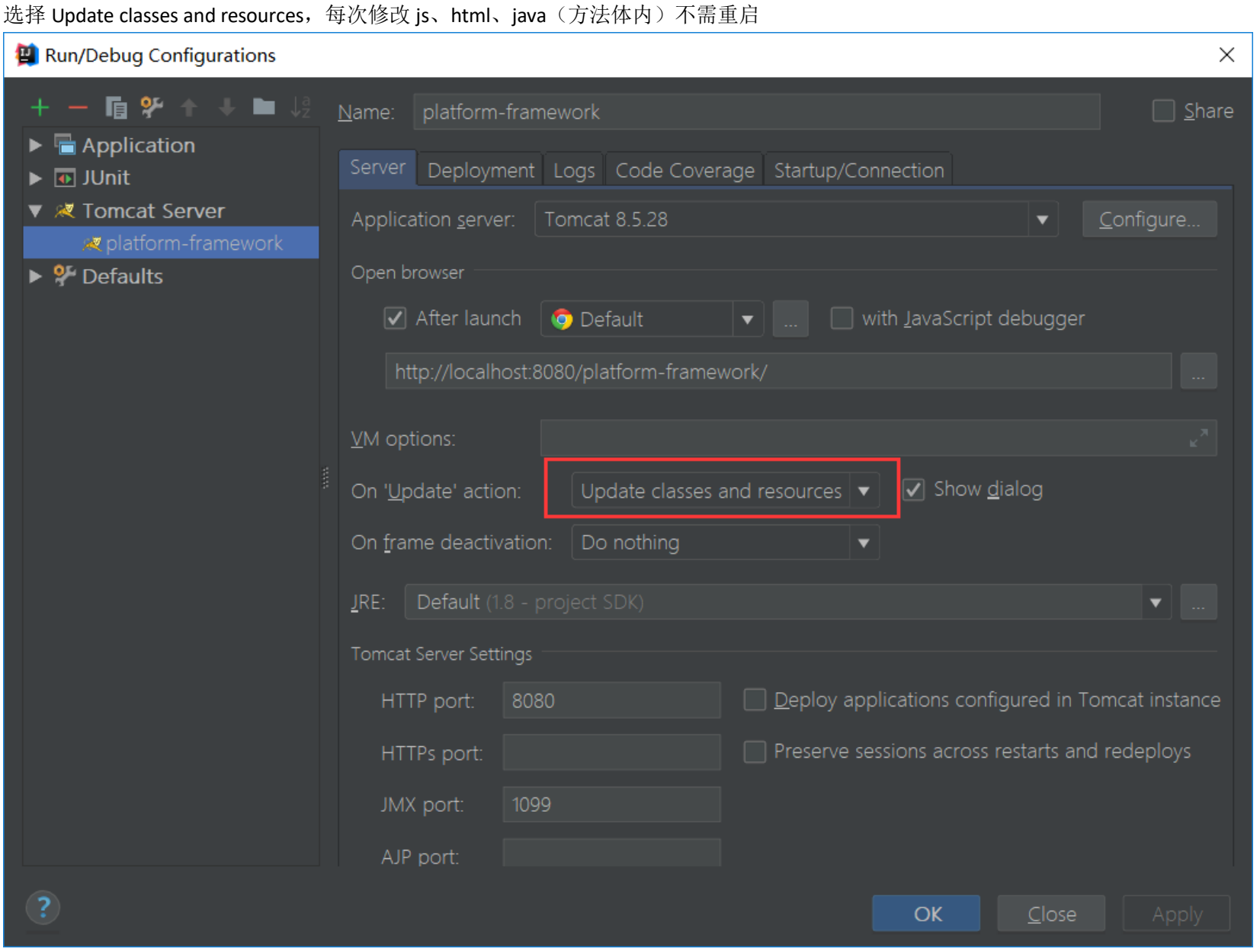

Tomcat Server
Tomcat Home:
D:\soft\Tomcat 8.5
Tomcat Version: 8.5.28
Tomcat base directory:
D:\soft\Tomcat 8.5
OK
Cancel

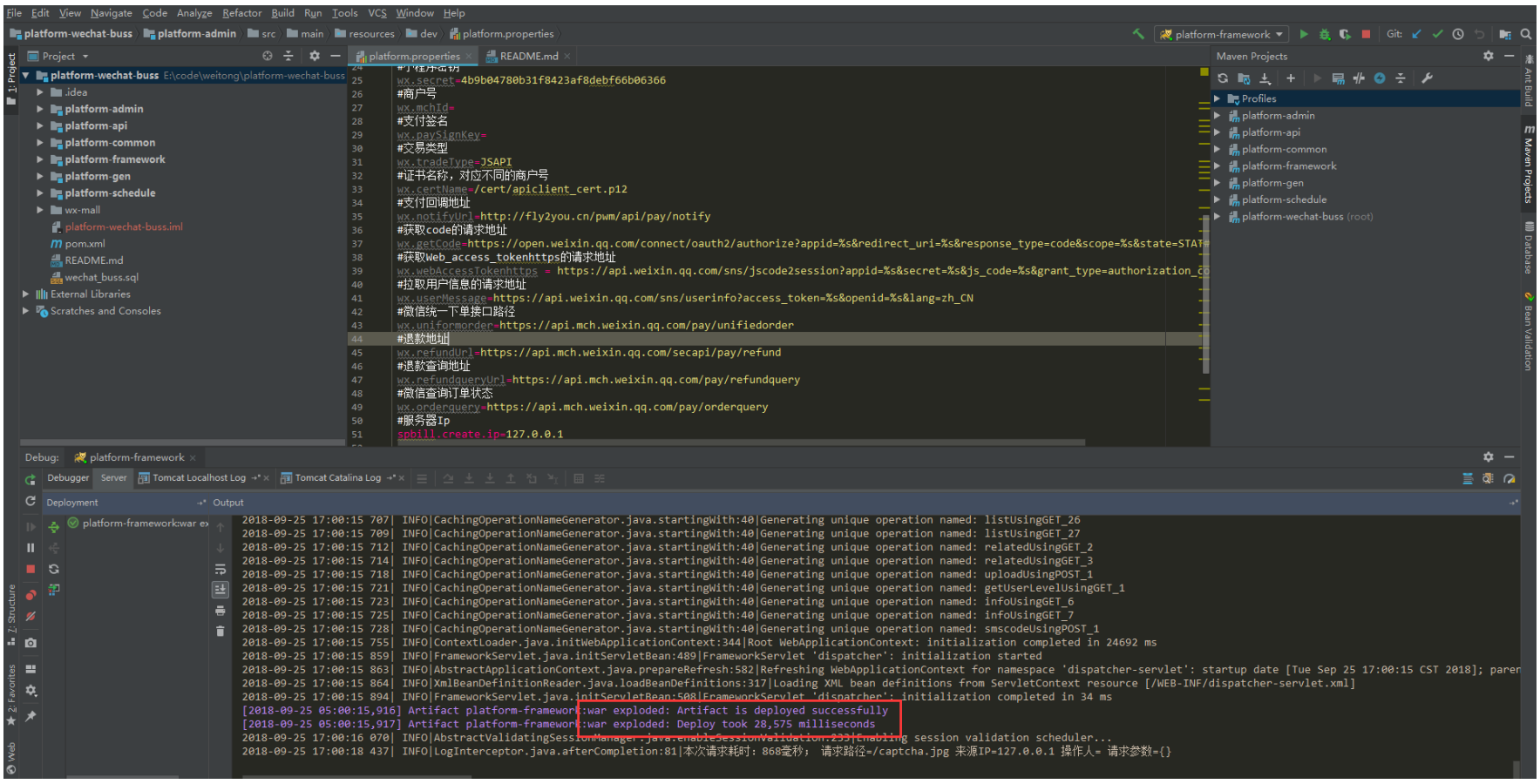

Run/Debug Configurations
Name: platform-framework
Share
Application
JUnit
Tomcat Server
platform-framework
Defaults
Server
Deployment
Logs
Code Coverage
Startup/Connection
Deploy at the server startup
Artifact...
External Source...
Nothing to deploy
Before launch: Build, Activate tool window
Warning: No artifacts marked for deployment
Fix
OK
Cancel
Apply


Select Artifacts to Deploy
Selected artifacts will be deployed at server startup
platform-admin:war
platform-admin:war exploded
platform-framework:war
platform-framework:war exploded
OK
Cancel

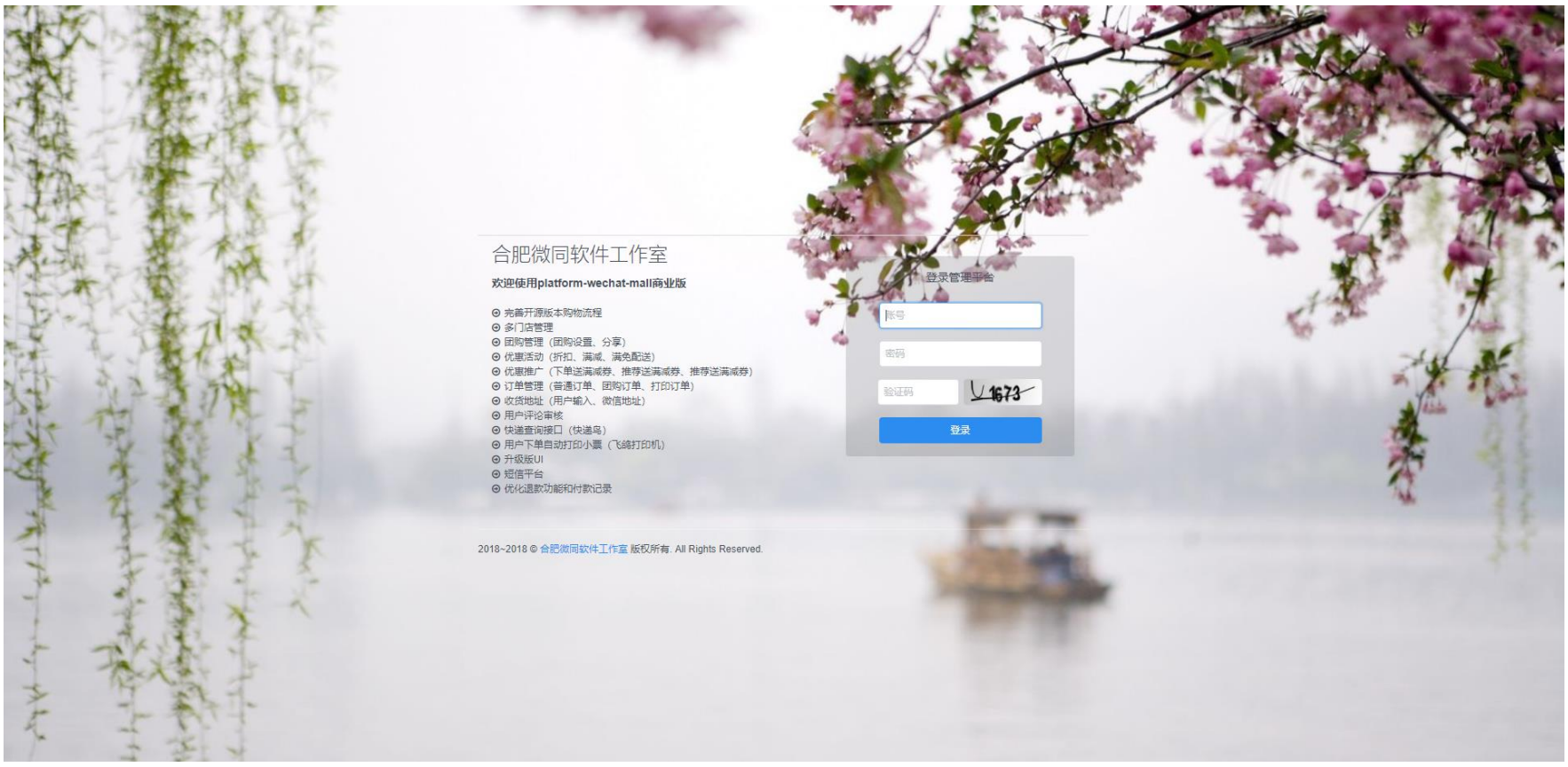


选择 Update classes and resources，每次修改 js、html、java（方法体内）不需重启





如上图所示，启动成功，访问 http://localhost:8080/platform-framework



2.6.4. 项目访问路径

http://localhost:8080/platform-framework

账号密码：admin/admin

2.6.5. Swagger 路径

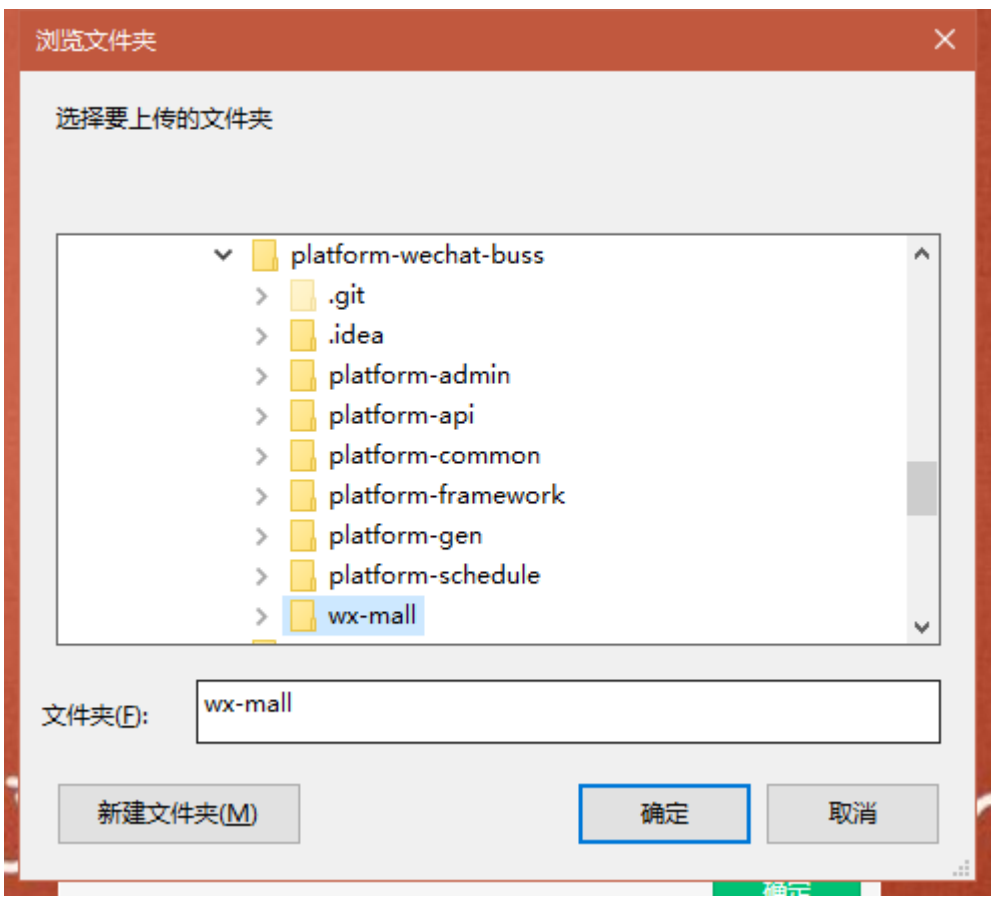
http://localhost:8080/platform-framework/swagger-ui.html

2.6.6. 小程序接口路径

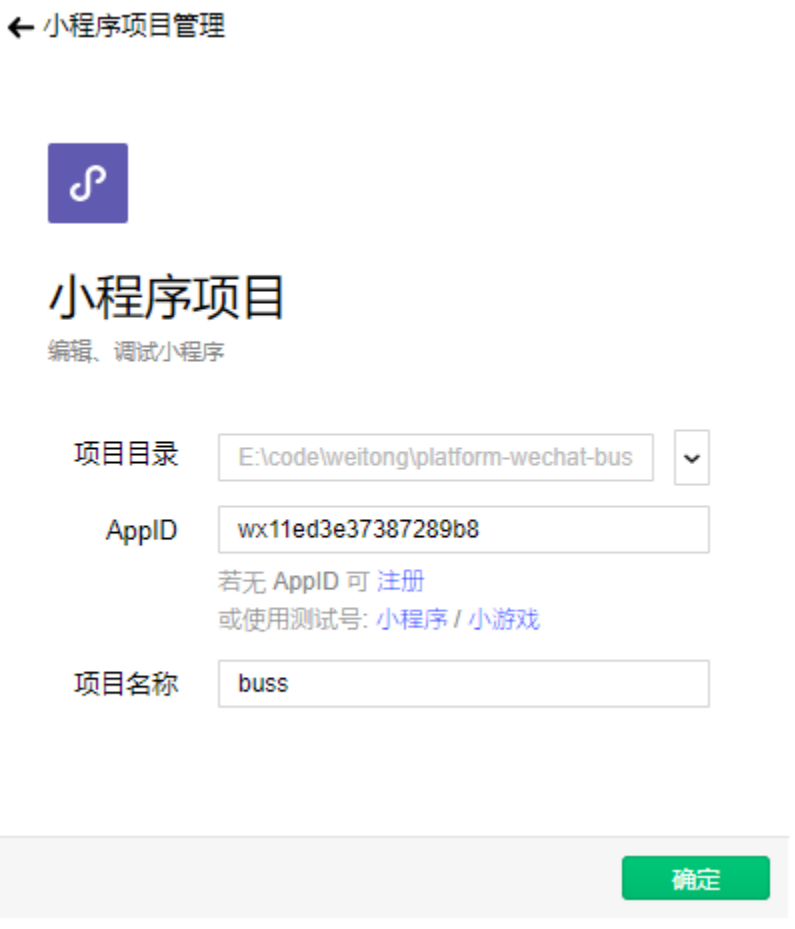
http://localhost:8080/platform-framework/api/

2.6.7. 使用微信 web 开发者工具启动 wx-mall

2.6.7.1. 导入 wx-mall 到微信 web 开发者工具

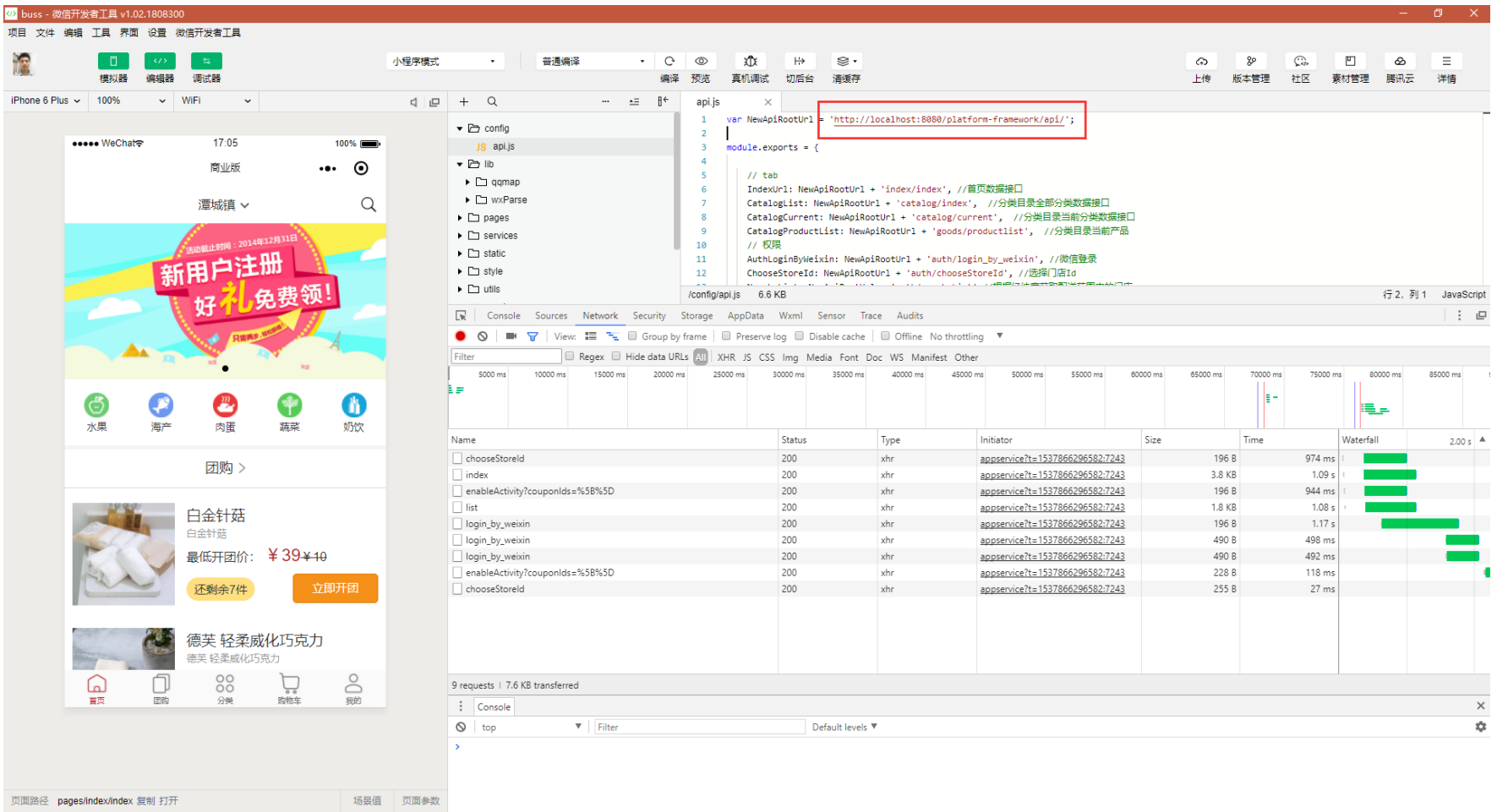


2.6.7.2. 填写自己在微信公众平台申请的小程序 AppID(与 platform.properties 里的 wx.appId 保持一致)

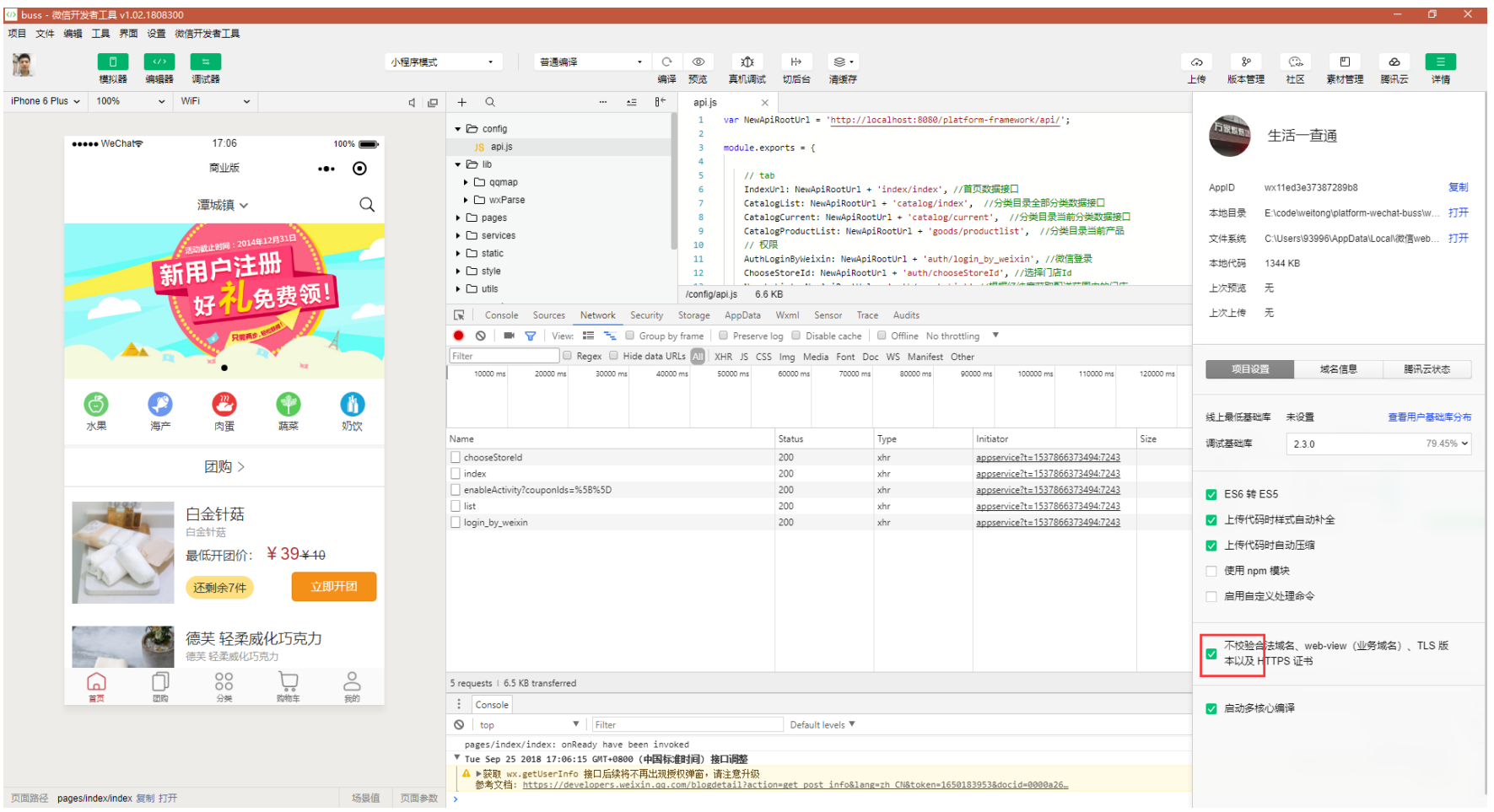


2.6.7.3. 修改 config/api.js 配置

```
var NewApiRootUrl = 'http://localhost:8080/platform-framework/api/';
```

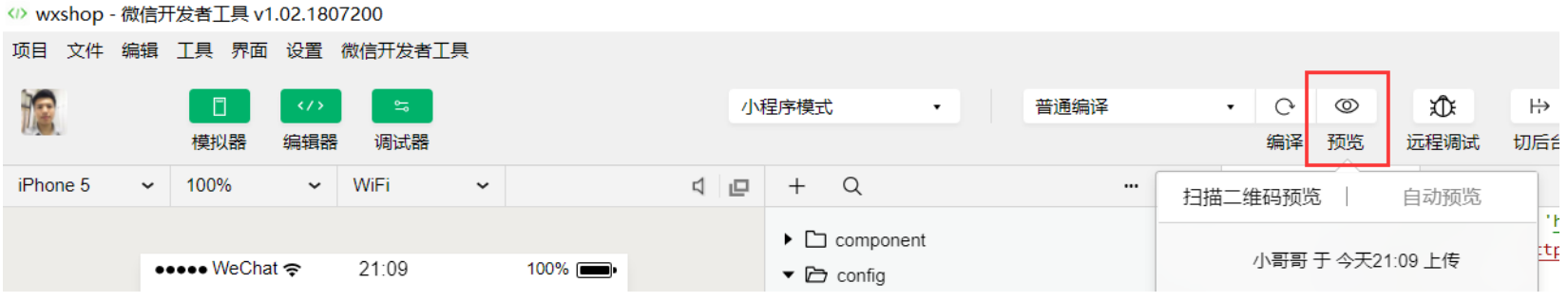


2.6.7.4. 开发模式设置

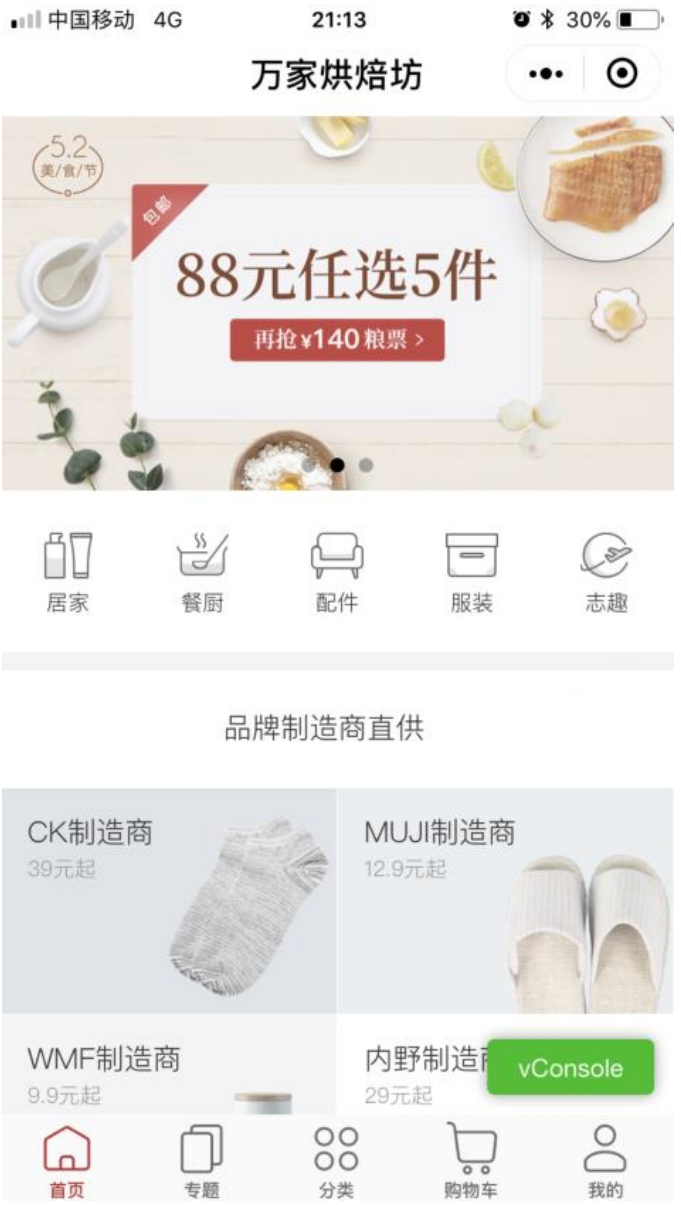


2.6.7.5. 关于发布到手机端测试

若想发布到手机端测试请求数据，后台项目必须发布到云服务器（使用内网穿透工具亦可），然后在微信开发工具点击预览，如下图



然后使用手机微信扫码，注意此时还是请求不到数据，需要打开调试模式才能正常访问接口数据，操作如下图



2.6.8. 官网

http://fly2you.cn

3. 项目实战

3.1. 功能描述

开发一个商品评论的列表、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。

◆ 我们先建一个商品评论表 nideshop_comment，表结构如下所示：

```
CREATE TABLE `nideshop_comment` (
  `id` int(11) NOT NULL AUTO_INCREMENT COMMENT '主键',
  `type_id` tinyint(3) unsigned NOT NULL DEFAULT '0' COMMENT '类型',
  `value_id` int(11) DEFAULT '0',
  `content` varchar(6550) COLLATE utf8mb4_unicode_ci DEFAULT NULL COMMENT '储存为 base64 编码',
  `add_time` bigint(12) unsigned DEFAULT '0' COMMENT '记录时间',
  `status` tinyint(3) unsigned DEFAULT '0' COMMENT '状态',
  `user_id` int(11) DEFAULT '0' COMMENT '会员 Id',
  PRIMARY KEY (`id`),
  KEY `id_value` (`value_id`)
) ENGINE=InnoDB CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

3.2. 使用代码生成器

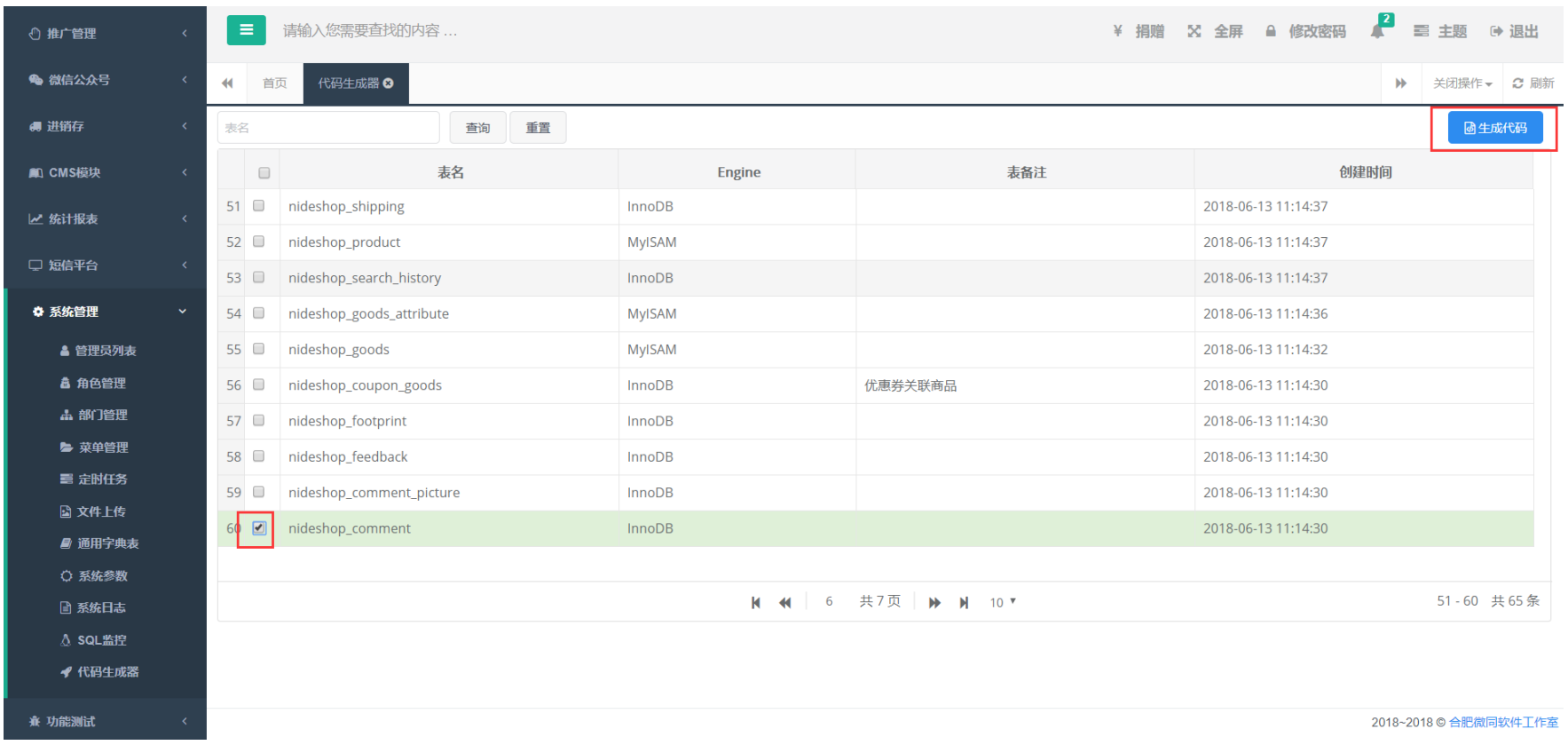
◆ 使用代码生成器前，我们先来看下代码生成器的配置，看看那些是可配置的，打开 platform-gen 模块的配置文件 generator.properties，如下所示：

generator.properties

```
#\u4EE3\u7801\u751F\u6210\u5668\uFF0C\u914D\u7F6E\u4FE1\u606F


#\u5305\u540D
package=com.platform
#\u4F5C\u8005
author=lipengjun
#Email
email=939961241@qq.com
#\u8868\u524D\u7F00(\u7C7B\u540D\u4E0D\u4F1A\u5305\u542B\u8868\u524D\u7F00)
tablePrefix=nideshop_

#\u7C7B\u578B\u8F6C\u6362\uFF0C\u914D\u7F6E\u4FE1\u606F
tinyint=Integer
smallint=Integer
mediumint=Integer
int=Integer
integer=Integer
bigint=Long
float=Float
double=Double
decimal=BigDecimal

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

date=Date
datetime=Date
timestamp=Date
```

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。如果有缺少的类型，可自行在 generator.properties 文件中补充。

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



生成的代码结构，如下所示：

```
AutoCode
├─menu.sql                          创建菜单的sql
│
└─main
    ├─java                          生成的java代码
    │    └─com
    │       └─platform
    │            ├─controller
    │            ├─dao
    │            ├─entity
    │            └─service
    │                └─impl
    └─webapp                        生成的页面文件
        ├─js
        │  └─shop
        └─WEB-INF
            └─page
                └─shop
```

生成好代码后，我们只需在数据库 platform-shop 中，执行 menu.sql 语句

再把生成的文件覆盖到项目，重新启动 platform-framework 项目即可。现在，我们就可以新增、修改、删除等操作。

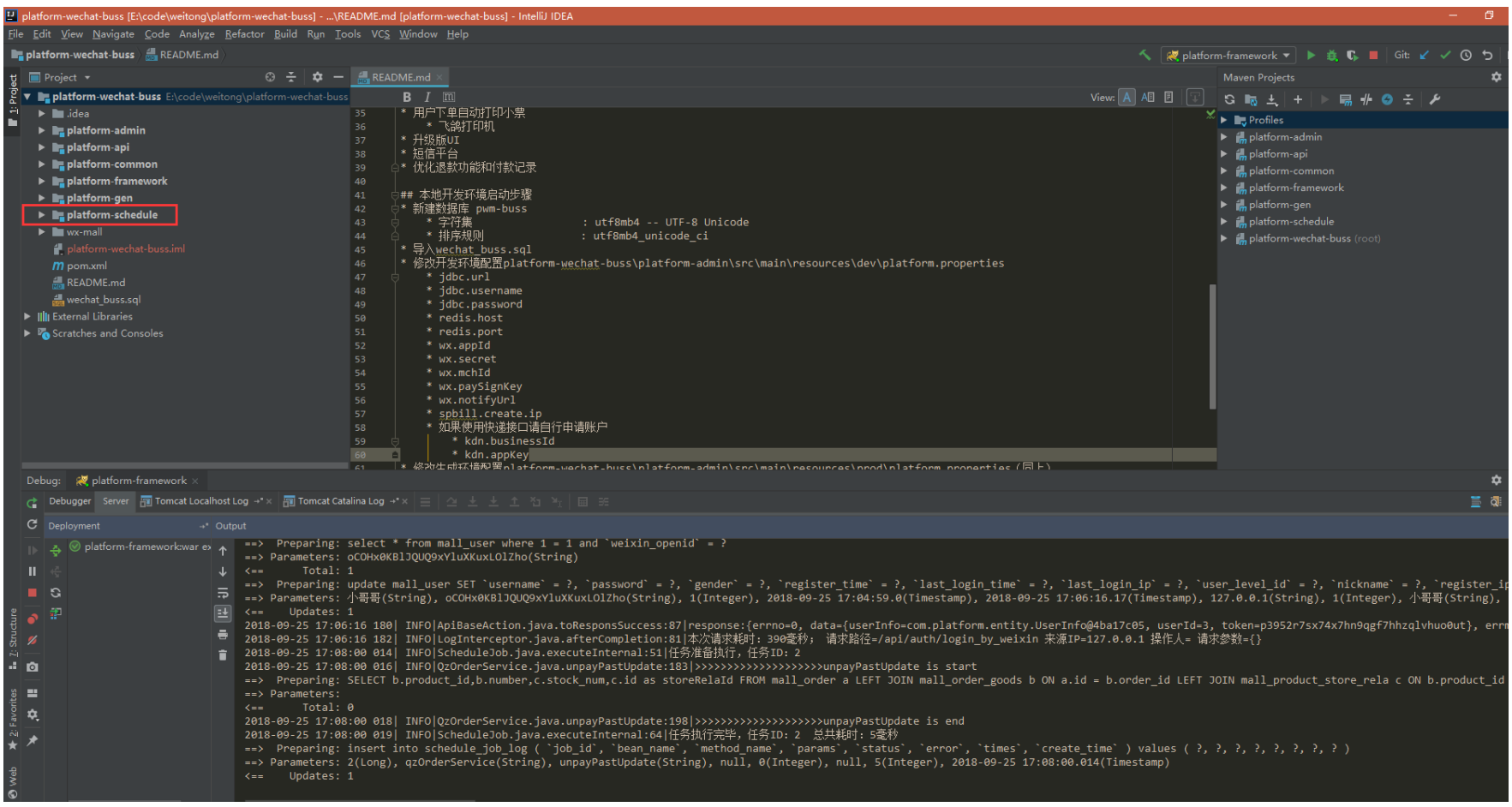
4. 后端源码分析

4.1. 功能模块移除

移除定时任务功能点，项目结构是按模块划分的，所以想要移除某个功能，只需删除 modules 目录即可；

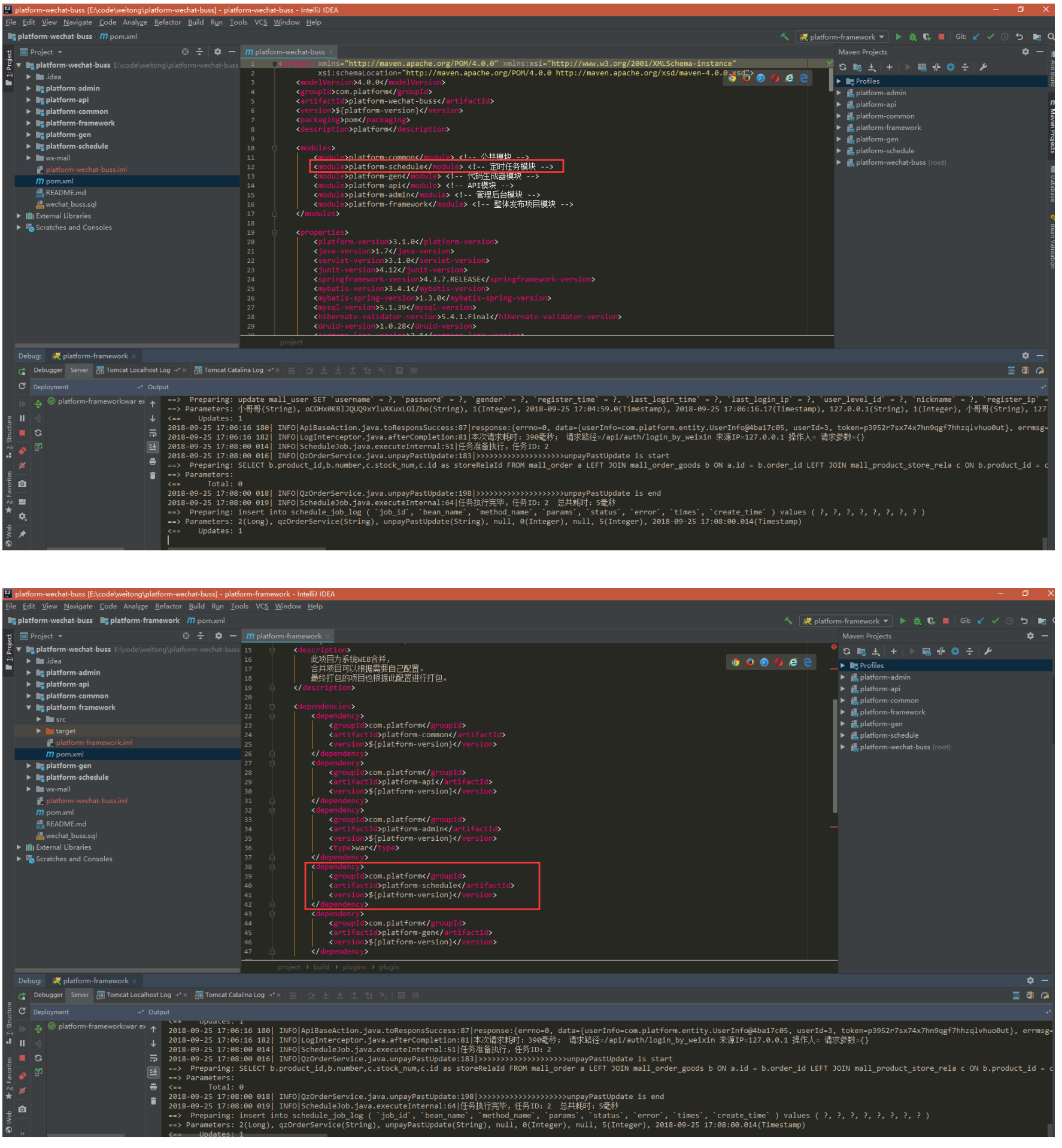
4.1.1. 删除定时任务的 module

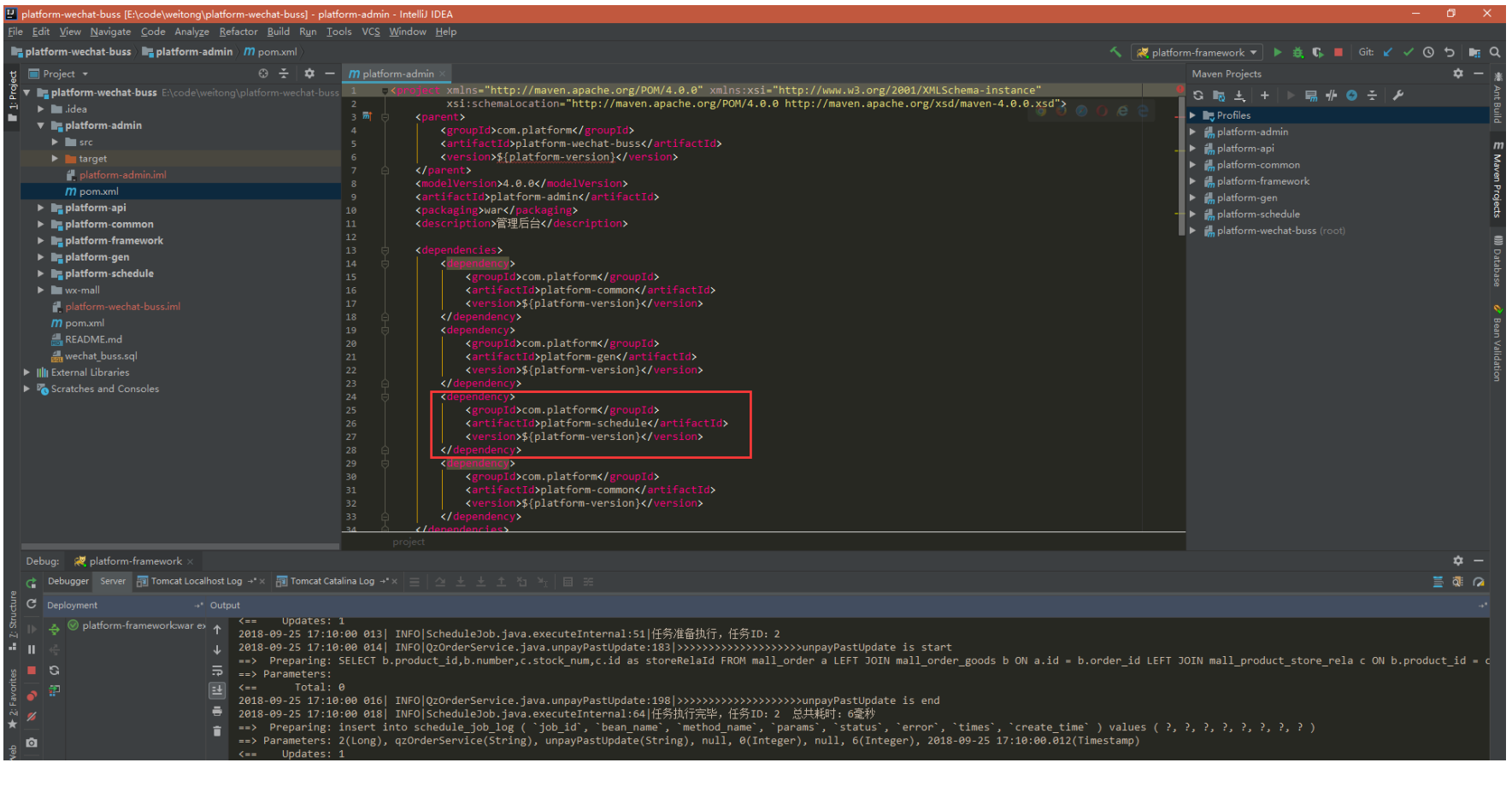
如下图



4.1.2. 删除项目依赖

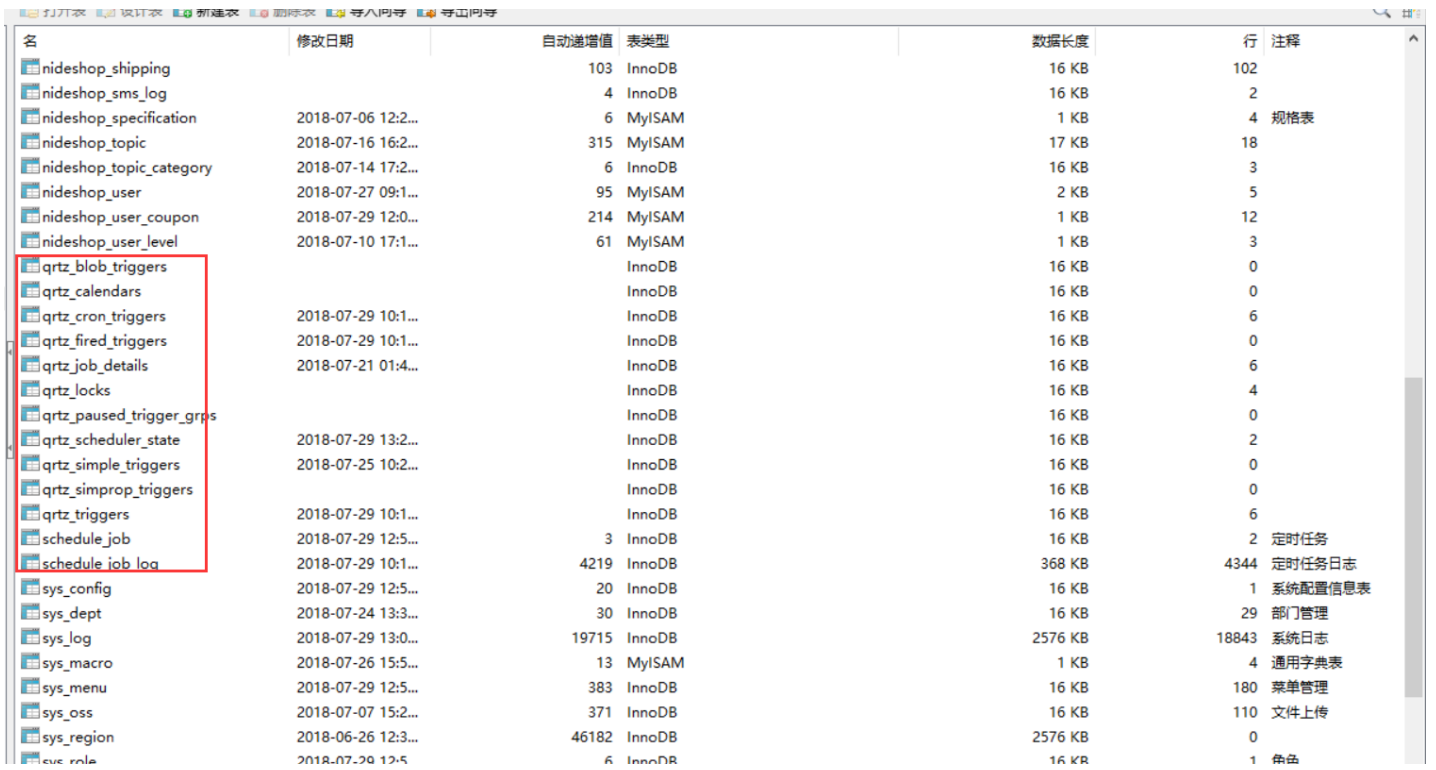
如下图


platform-wechat-buss [E:\code\weitong\platform-wechat-buss] - platform-wechat-buss - IntelliJ IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
platform-wechat-buss pom.xml
platform-framework
Git:
Project
platform-wechat-buss E:\code\weitong\platform-wechat-buss
.idea
platform-admin
platform-api
platform-common
platform-framework
platform-gen
platform-schedule
wx-mall
platform-wechat-buss.iml
pom.xml
README.md
wechat_buss.sql
External Libraries
Scratches and Consoles
platform-wechat-buss
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>com.platform</groupId>
<artifactId>platform-wechat-buss</artifactId>
<version>${platform-version}</version>
<packaging>pom</packaging>
<description>platform</description>
<modules>
<module>platform-common</module> <!-- 公共模块 -->
<module>platform-schedule</module> <!-- 定时任务模块 -->
<module>platform-gen</module> <!-- 代码生成器模块 -->
<module>platform-api</module> <!-- API模块 -->
<module>platform-admin</module> <!-- 管理后台模块 -->
<module>platform-framework</module> <!-- 整体发布项目模块 -->
</modules>
<properties>
<platform-version>3.1.0</platform-version>
<java-version>1.7</java-version>
<servlet-version>3.1.0</servlet-version>
<junit-version>4.12</junit-version>
<springframework-version>4.3.7.RELEASE</springframework-version>
<mybatis-version>3.4.1</mybatis-version>
<mybatis-spring-version>1.3.0</mybatis-spring-version>
<mysql-version>5.1.39</mysql-version>
<hibernate-validator-version>5.4.1.Final</hibernate-validator-version>
<druid-version>1.0.28</druid-version>
Maven Projects
Profiles
platform-admin
platform-api
platform-common
platform-framework
platform-gen
platform-schedule
platform-wechat-buss (root)
Debug: platform-framework
Debugger Server Tomcat Localhost Log Tomcat Catalina Log
Deployment Output
platform-framework:war ex
==> Preparing: update mall_user SET `username` = ?, `password` = ?, `gender` = ?, `register_time` = ?, `last_login_time` = ?, `last_login_ip` = ?, `user_level_id` = ?, `nickname` = ?, `register_ip` =
==> Parameters: 小哥哥(String), oCOHx0KBlJQUQ9xYluXKuxLOlZho(String), 1(Integer), 2018-09-25 17:04:59.0(Timestamp), 2018-09-25 17:06:16.17(Timestamp), 127.0.0.1(String), 1(Integer), 小哥哥(String), 127
<== Updates: 1
2018-09-25 17:06:16 180| INFO|ApiBaseAction.java.toResponsSuccess:87|response:{errno=0, data={userInfo=com.platform.entity.UserInfo@4ba17c05, userId=3, token=p3952r7sx74x7hn9qgf7hhzqlvhuo0ut}, errmsg=
2018-09-25 17:06:16 182| INFO|LogInterceptor.java.afterCompletion:81|本次请求耗时: 390毫秒; 请求路径=/api/auth/login_by_weixin 来源IP=127.0.0.1 操作人= 请求参数={}
2018-09-25 17:08:00 014| INFO|ScheduleJob.java.executeInternal:51|任务准备执行, 任务ID: 2
2018-09-25 17:08:00 016| INFO|QzOrderService.java.unpayPastUpdate:183|>>>>>>>>>>>>>>>>>>>>unpayPastUpdate is start
==> Preparing: SELECT b.product_id,b.number,c.stock_num,c.id as storeRelaId FROM mall_order a LEFT JOIN mall_order_goods b ON a.id = b.order_id LEFT JOIN mall_product_store_rela c ON b.product_id = c
==> Parameters:
<== Total: 0
2018-09-25 17:08:00 018| INFO|QzOrderService.java.unpayPastUpdate:198|>>>>>>>>>>>>>>>>>>>>unpayPastUpdate is end
2018-09-25 17:08:00 019| INFO|ScheduleJob.java.executeInternal:64|任务执行完毕, 任务ID: 2 总共耗时: 5毫秒
==> Preparing: insert into schedule_job_log (`job_id`, `bean_name`, `method_name`, `params`, `status`, `error`, `times`, `create_time`) values (?, ?, ?, ?, ?, ?, ?, ?)
==> Parameters: 2(Long), qzOrderService(String), unpayPastUpdate(String), null, 0(Integer), null, 5(Integer), 2018-09-25 17:08:00.014(Timestamp)
<== Updates: 1
platform-framework
<description>
此项目为系统WEB合并,
合并项目可以根据需要自己配置。
最终打包的项目也根据此配置进行打包。
</description>
<dependencies>
<dependency>
<groupId>com.platform</groupId>
<artifactId>platform-common</artifactId>
<version>${platform-version}</version>
</dependency>
<dependency>
<groupId>com.platform</groupId>
<artifactId>platform-api</artifactId>
<version>${platform-version}</version>
</dependency>
<dependency>
<groupId>com.platform</groupId>
<artifactId>platform-admin</artifactId>
<version>${platform-version}</version>
<type>war</type>
</dependency>
<dependency>
<groupId>com.platform</groupId>
<artifactId>platform-schedule</artifactId>
<version>${platform-version}</version>
</dependency>
<dependency>
<groupId>com.platform</groupId>
<artifactId>platform-gen</artifactId>
<version>${platform-version}</version>
</dependency>
project › build › plugins › plugin



4.1.3. 删除对应的表结构

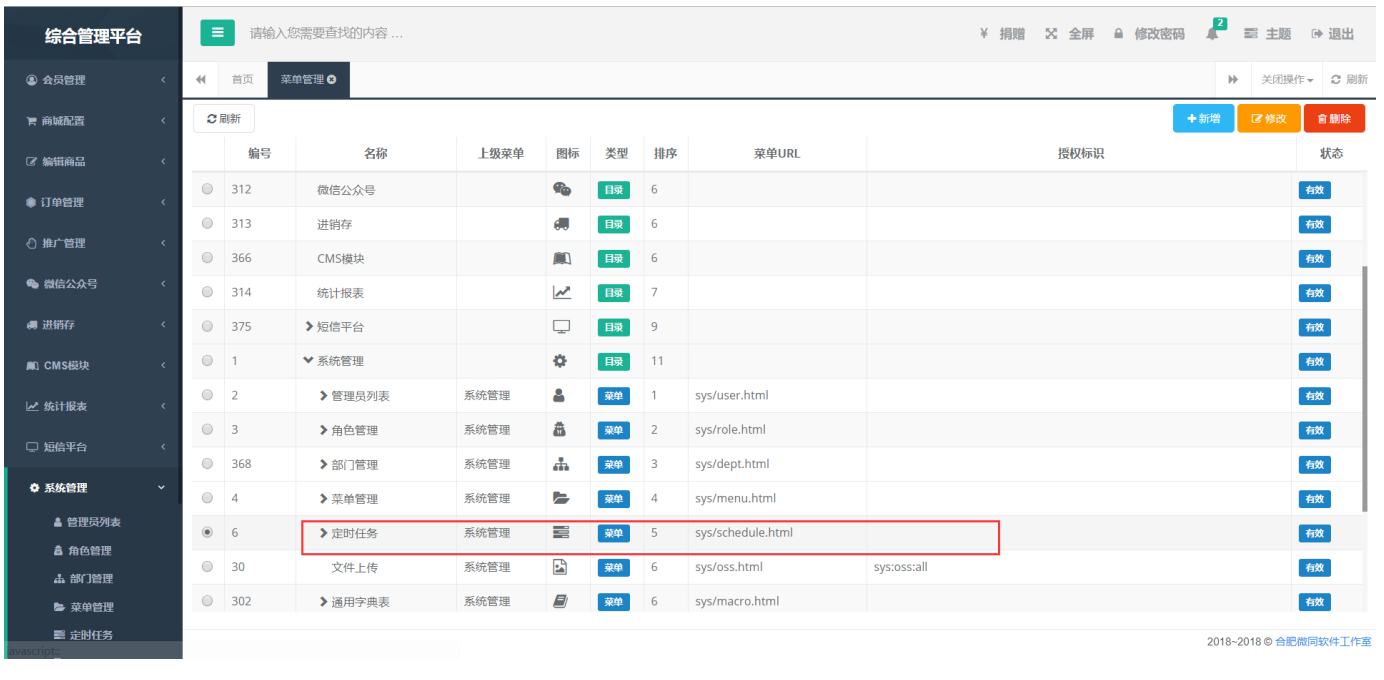
如下图



名	修改日期	自动递增值	表类型	数据长度	行	注释
nideshop_shipping		103	InnoDB	16 KB	102	
nideshop_sms_log		4	InnoDB	16 KB	2	
nideshop_specification	2018-07-06 12:2...	6	MyISAM	1 KB	4	规格表
nideshop_topic	2018-07-16 16:2...	315	MyISAM	17 KB	18	
nideshop_topic_category	2018-07-14 17:2...	6	InnoDB	16 KB	3	
nideshop_user	2018-07-27 09:1...	95	MyISAM	2 KB	5	
nideshop_user_coupon	2018-07-29 12:0...	214	MyISAM	1 KB	12	
nideshop_user_level	2018-07-10 17:1...	61	MyISAM	1 KB	3	
qrtz_blob_triggers			InnoDB	16 KB	0	
qrtz_calendars			InnoDB	16 KB	0	
qrtz_cron_triggers	2018-07-29 10:1...		InnoDB	16 KB	6	
qrtz_fired_triggers	2018-07-29 10:1...		InnoDB	16 KB	0	
qrtz_job_details	2018-07-21 01:4...		InnoDB	16 KB	6	
qrtz_locks			InnoDB	16 KB	4	
qrtz_paused_trigger_grps			InnoDB	16 KB	0	
qrtz_scheduler_state	2018-07-29 13:2...		InnoDB	16 KB	2	
qrtz_simple_triggers	2018-07-25 10:2...		InnoDB	16 KB	0	
qrtz_simprop_triggers			InnoDB	16 KB	0	
qrtz_triggers	2018-07-29 10:1...		InnoDB	16 KB	6	
schedule_job	2018-07-29 12:5...	3	InnoDB	16 KB	2	定时任务
schedule_job_log	2018-07-29 10:1...	4219	InnoDB	368 KB	4344	定时任务日志
sys_config	2018-07-29 12:5...	20	InnoDB	16 KB	1	系统配置信息表
sys_dept	2018-07-24 13:3...	30	InnoDB	16 KB	29	部门管理
sys_log	2018-07-29 13:0...	19715	InnoDB	2576 KB	18843	系统日志
sys_macro	2018-07-26 15:5...	13	MyISAM	1 KB	4	通用字典表
sys_menu	2018-07-29 12:5...	383	InnoDB	16 KB	180	菜单管理
sys_oss	2018-07-07 15:2...	371	InnoDB	16 KB	110	文件上传
sys_region	2018-06-26 12:3...	46182	InnoDB	2576 KB	0	

4.1.4. 删除对应的菜单

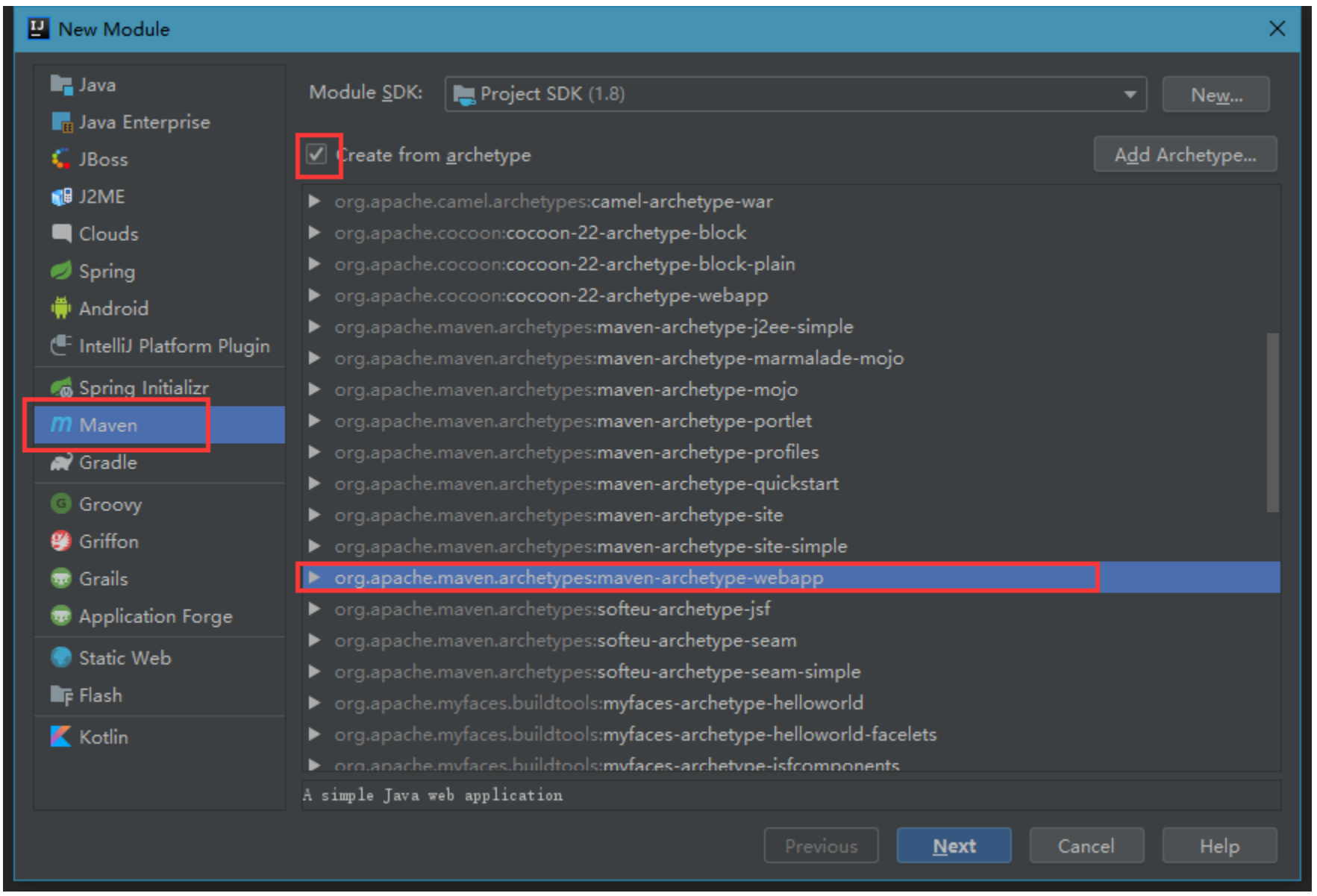
如下图



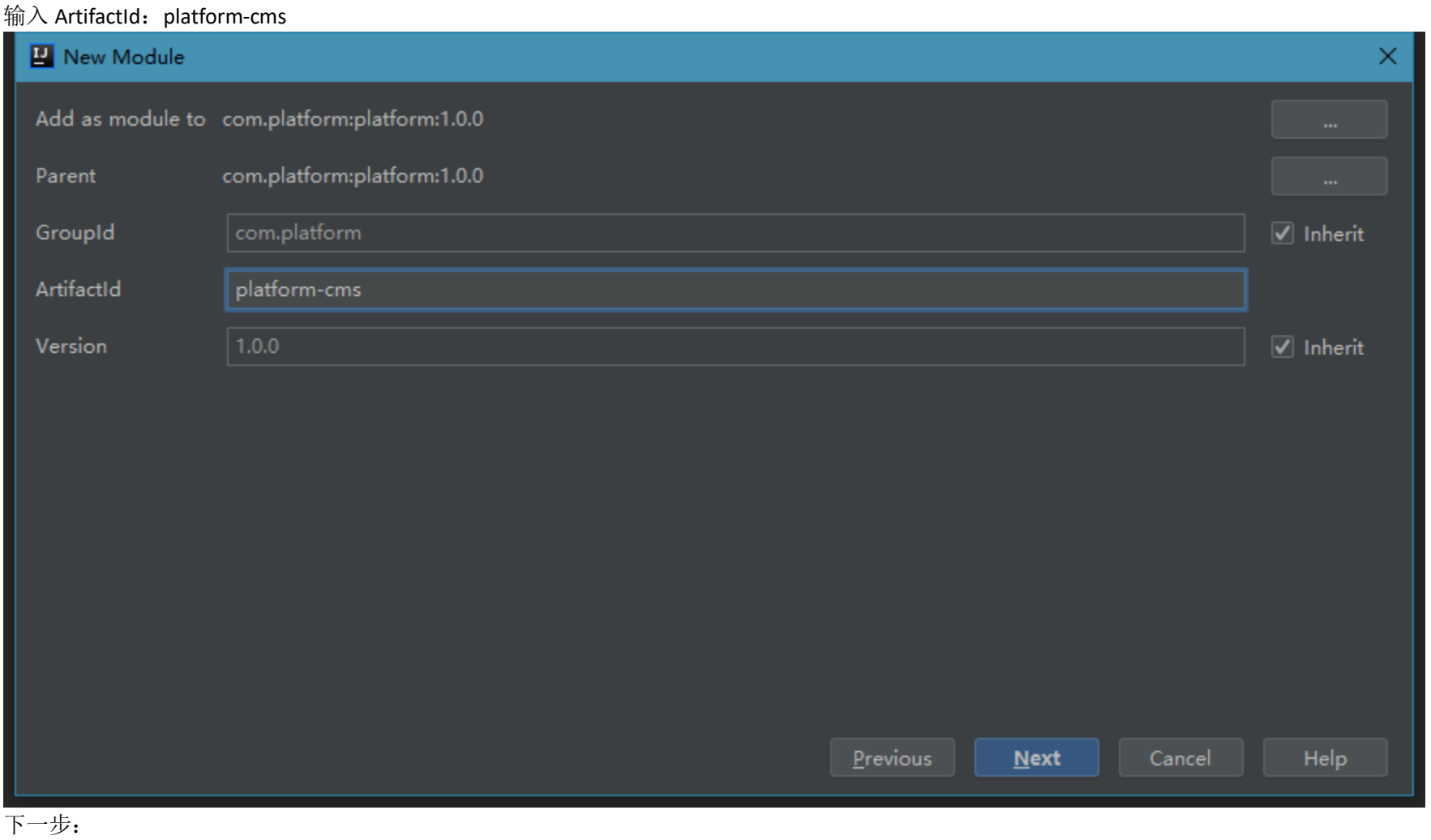
4.2. 功能模块添加（eg：cms）

4.2.1. 新增 module

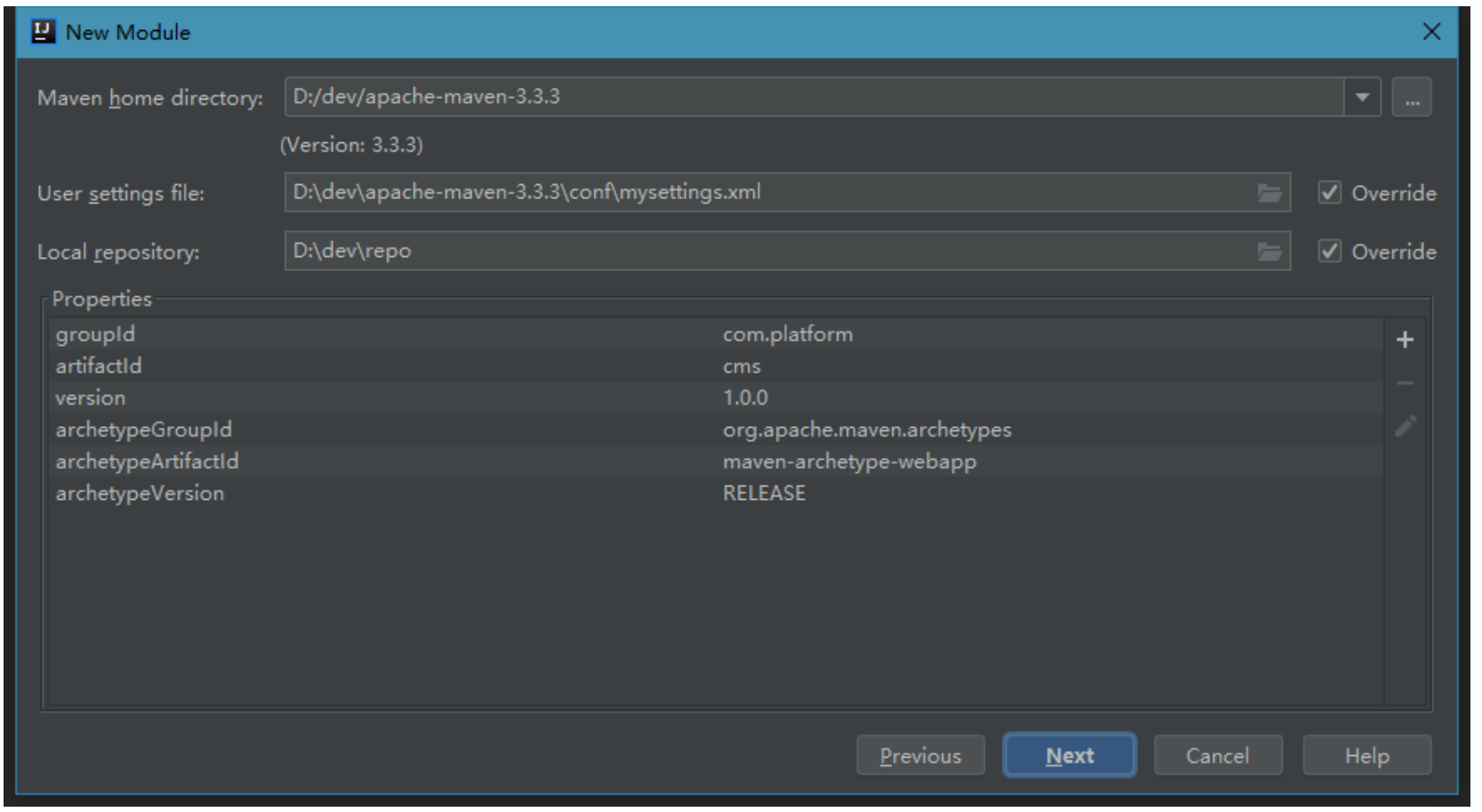
File->New->Module...

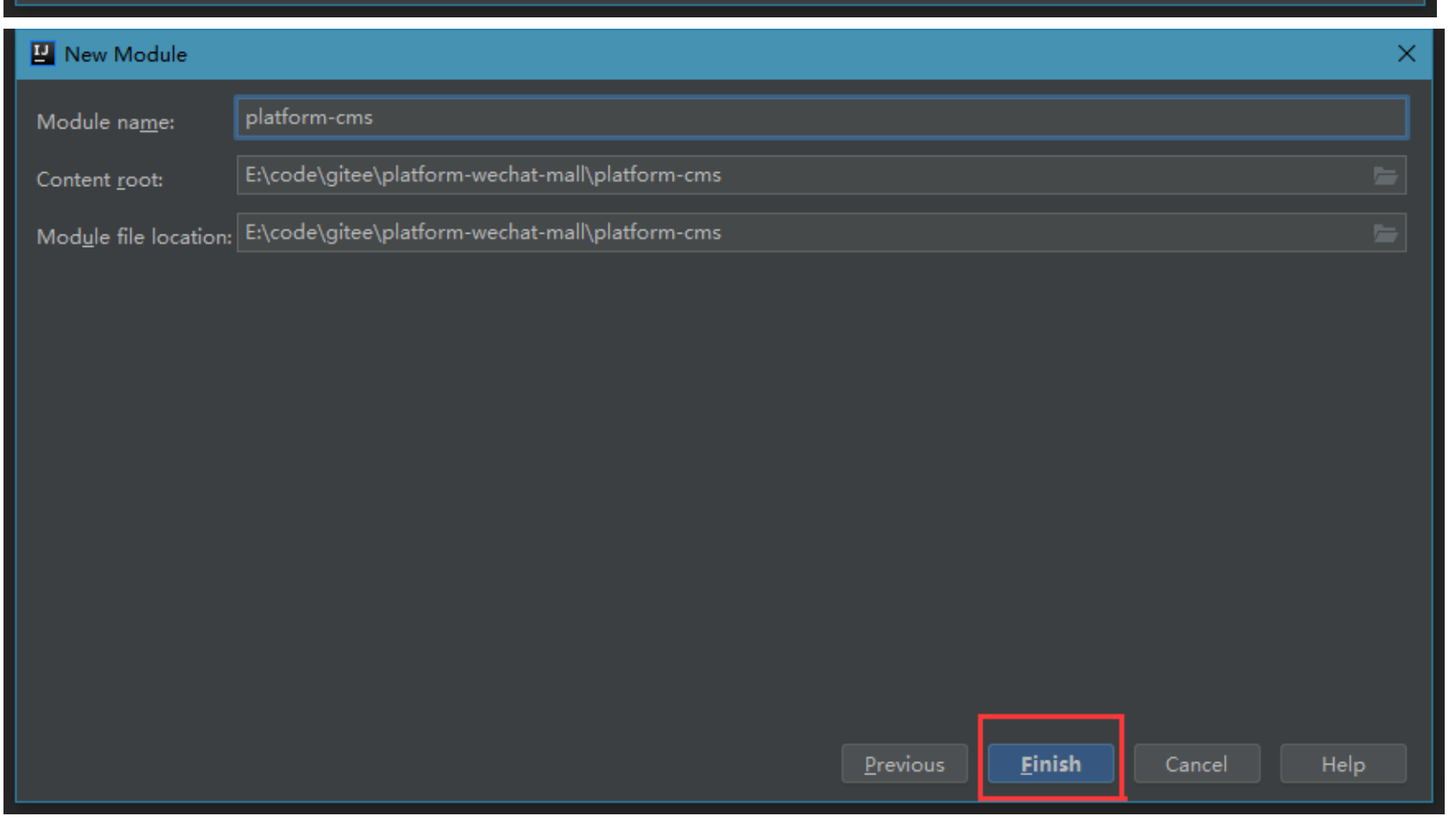


输入 ArtifactId：platform-cms

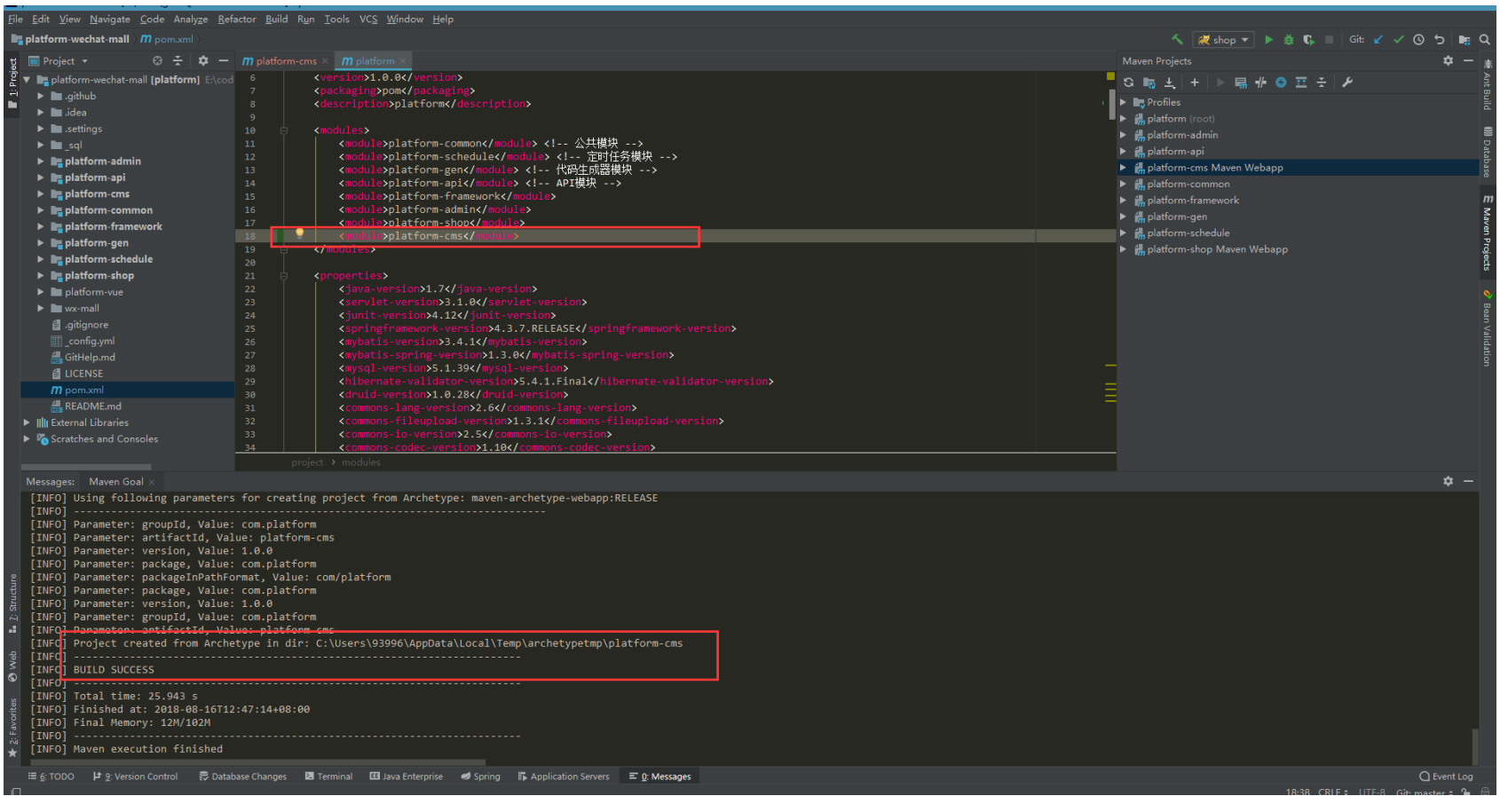


下一步：





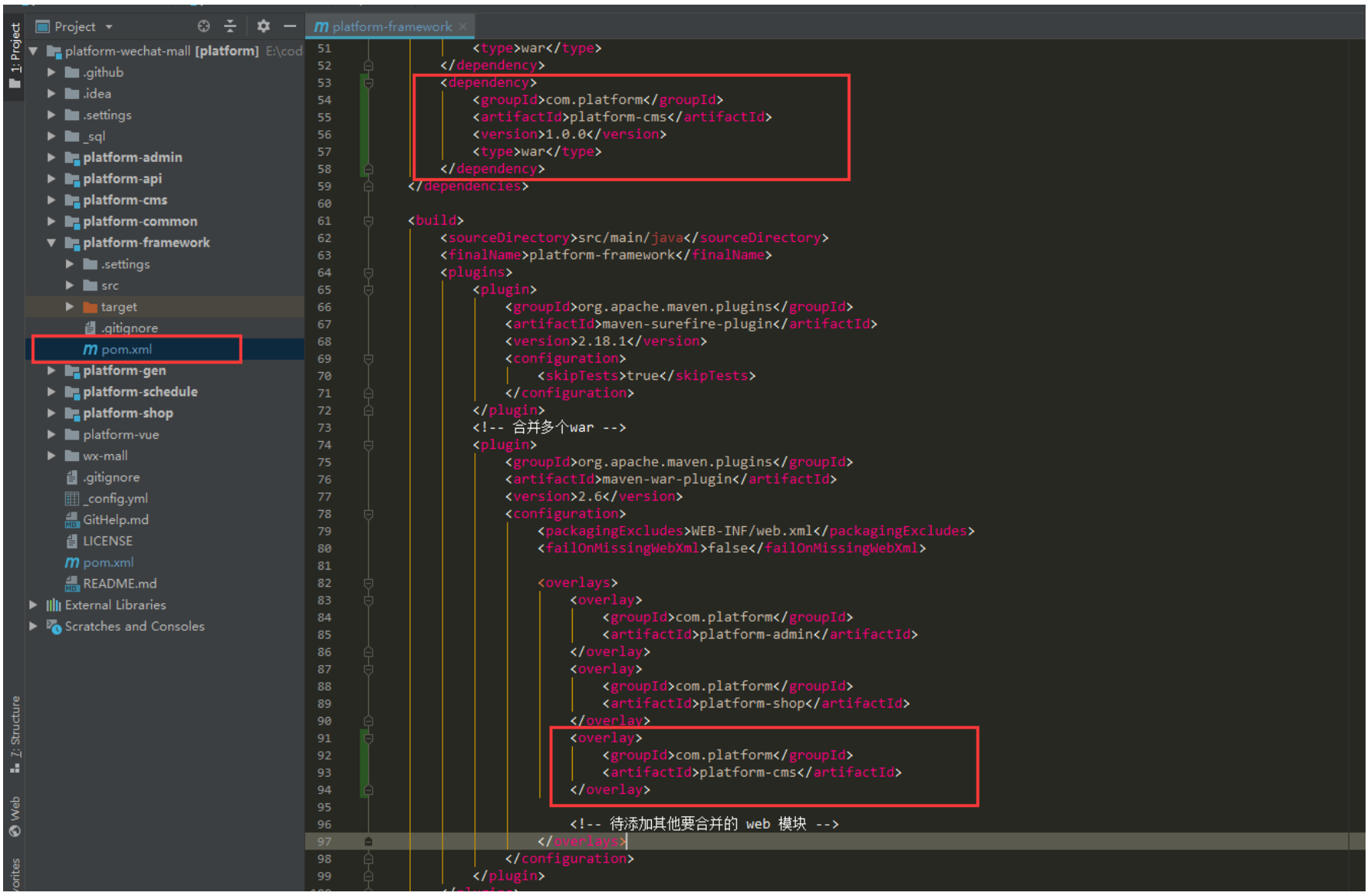
使用 IDEA 创建 module，根目录会自动添加<module>platform-cms</module>;其他工具请自行添加。



4.2.2. 将 cms 模块添加到 platform-framework

至此模块已新增成功，下面还需要修改配置，将该项目一起打包到 platform-framework

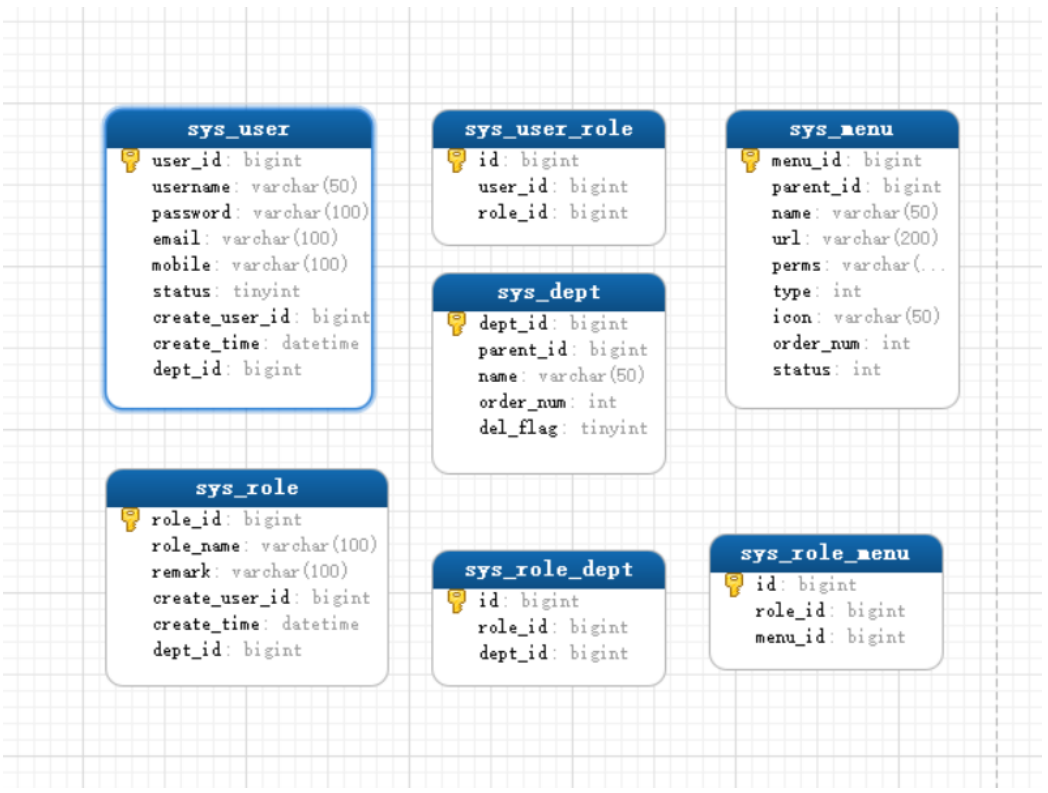
platform-framework/pom.xml



5. 核心模块

5.1. 功能权限设计

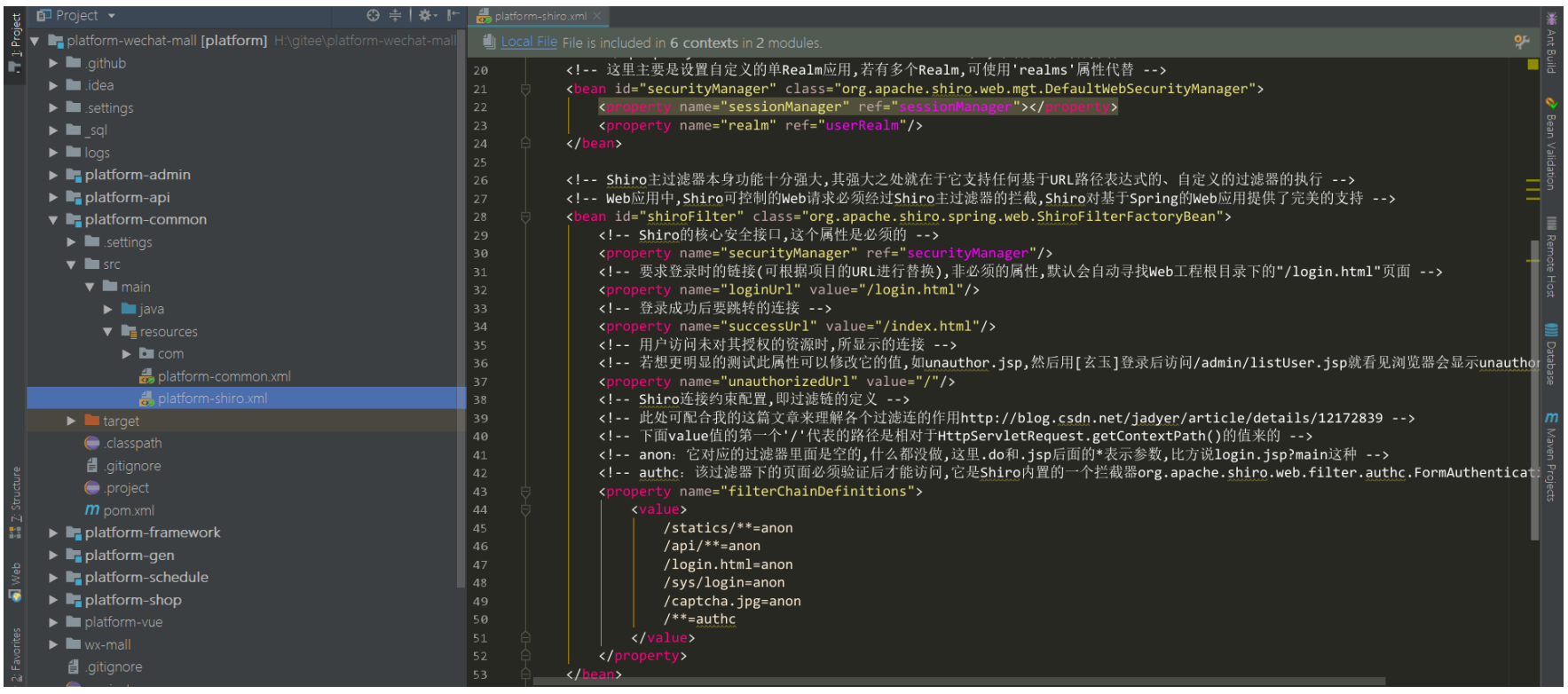
权限相关的表结构，如下图所示



- sys_user[用户]表，保存用户相关数据，通过 sys_user_role[用户与角色关联]表，与 sys_role[角色]表关联；sys_menu[菜单]表通过 sys_role_menu[菜单与角色关联]表，与 sys_role[角色]表关联
- sys_menu 表，保存菜单相关数据，并在 perms 字段里，保存了 shiro 的权限标识，也就是说拥有此菜单，就拥有 perms 字段里的所有权限，比如，某用户拥有的菜单权限标识 sys:menu:list，就可以访问下面的方法

```java
/**
 * 所有菜单列表
 */
@RequestMapping("/list")
@RequiresPermissions("sys:menu:list")
public R list(@RequestParam Map<String, Object> params) {
    //查询列表数据
    Query query = new Query(params);
    List<SysMenuEntity> menuList = sysMenuService.queryList(query);
    int total = sysMenuService.queryTotal(query);

    PageUtils pageUtil = new PageUtils(menuList, total, query.getLimit(), query.getPage());

    return R.ok().put("page", pageUtil);
}
```

- sys_dept 表，保存部门相关数据，数据权限也是根据部门进行过滤的，下一小节具体讲解
- 在配置文件里，anon 表示不经过 shiro 处理，authc 表示经过 shiro 处理，这样就保证没有权限的请求有效的拒绝。



- 接下来，我们看看 shiro 框架，具体是怎么效验功能权限的，系统登录代码如下

```java
@Controller
public class SysLoginController {
    @Autowired
    private Producer producer;
    @RequestMapping("captcha.jpg")
    public void captcha(HttpServletResponse response) throws ServletException, IOException {
        response.setHeader("Cache-Control", "no-store, no-cache");
        response.setContentType("image/jpeg");
        //生成文字验证码
        String text = producer.createText();
        //生成图片验证码
        BufferedImage image = producer.createImage(text);
        //保存到 shiro session
        ShiroUtils.setSessionAttribute(Constants.KAPTCHA_SESSION_KEY, text);

        ServletOutputStream out = response.getOutputStream();
        ImageIO.write(image, "jpg", out);
    }
    /**
     * 登录
     */
    @SysLog("登录")
    @ResponseBody
    @RequestMapping(value = "/sys/login", method = RequestMethod.POST)
    public R login(String username, String password, String captcha) throws IOException {
        String kaptcha = ShiroUtils.getKaptcha(Constants.KAPTCHA_SESSION_KEY);
        if(null == kaptcha){
            return R.error("验证码已失效");
        }
        if (!captcha.equalsIgnoreCase(kaptcha)) {
```

```
            return R.error("验证码不正确");
        }
        try {
            Subject subject = ShiroUtils.getSubject();
            //sha256 加密
            password = new Sha256Hash(password).toHex();
            UsernamePasswordToken token = new UsernamePasswordToken(username, password);
            subject.login(token);
        } catch (UnknownAccountException e) {
            return R.error(e.getMessage());
        } catch (IncorrectCredentialsException e) {
            return R.error(e.getMessage());
        } catch (LockedAccountException e) {
            return R.error(e.getMessage());
        } catch (AuthenticationException e) {
            return R.error("账户验证失败");
        }
        return R.ok();
    }
    /**
     * 退出
     */
    @RequestMapping(value = "logout", method = RequestMethod.GET)
    public String logout() {
        ShiroUtils.logout();
        return "redirect:/";
    }
}
```

◆ 登录的时候 subject.login(token)调用自定义的 Realm 的 doGetAuthorizationInfo 方法，方法如下

```
public class UserRealm extends AuthorizingRealm {
    @Autowired
    private SysUserDao sysUserDao;
    @Autowired
    private SysMenuDao sysMenuDao;

    /**
     * 授权(验证权限时调用)
     */
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {
        SysUserEntity user = (SysUserEntity) principals.getPrimaryPrincipal();
        Long userId = user.getUserId();

        List<String> permsList = (List<String>) J2CacheUtils.get(Constant.PERMS_LIST + userId);
        //用户权限列表
        Set<String> permsSet = new HashSet<String>();
        if (permsList != null && permsList.size() != 0) {
            for (String perms : permsList) {
                if (StringUtils.isBlank(perms)) {
                    continue;
                }
                permsSet.addAll(Arrays.asList(perms.trim().split(",")));
            }
        }
        SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();
        info.setStringPermissions(permsSet);
        return info;
    }
    /**
     * 认证(登录时调用)
     */
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(
            AuthenticationToken token) throws AuthenticationException {
        String username = (String) token.getPrincipal();
        String password = new String((char[]) token.getCredentials());
```

```
        //查询用户信息
        SysUserEntity user = sysUserDao.queryByUserName(username);
        //账号不存在
        if (user == null) {
            throw new UnknownAccountException("账号或密码不正确");
        }
        //密码错误
        if (!password.equals(user.getPassword())) {
            throw new IncorrectCredentialsException("账号或密码不正确");
        }
        //账号锁定
        if (user.getStatus() == 0) {
            throw new LockedAccountException("账号已被锁定,请联系管理员");
        }
        // 把当前用户放入到 session 中
        Subject subject = SecurityUtils.getSubject();
        Session session = subject.getSession(true);
        session.setAttribute(Global.CURRENT_USER, user);

        List<String> permsList;
        //系统管理员，拥有最高权限
        if (Constant.SUPER_ADMIN == user.getUserId()) {
            List<SysMenuEntity> menuList = sysMenuDao.queryList(new HashMap<String, Object>());
            permsList = new ArrayList<>(menuList.size());
            for (SysMenuEntity menu : menuList) {
                permsList.add(menu.getPerms());
            }
        } else {
            permsList = sysUserDao.queryAllPerms(user.getUserId());
        }
        J2CacheUtils.put(Constant.PERMS_LIST + user.getUserId(), permsList);
        SimpleAuthenticationInfo info = new SimpleAuthenticationInfo(user, password, getName());
        return info;
    }
}
```

- 在 doGetAuthorizationInfo 方法里，会通过用户名，查询用户信息，如果用户不存在则直接抛出 UnknownAccountException 异常，如果查询到用户信息，则封装成 SimpleAuthenticationInfo 对象并返回，为了减少对数据库的压力，将用户权限存入缓存中。
- 登录验证通过后，每次向后台请求调用有@RequiresPermissions 注解的请求时 shiro 会调用自定义 Realm 的 doGetAuthorizationInfo 方法验证当前登录用户是否拥有该请求的权限。

5.2. 数据权限设计

使用注解的方式实现数据权限的功能。

5.2.1. 通过@DataFilter 注解实现

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

    /**
     * sql 中数据创建用户（通常传入 CREATE_USER_ID）的别名
     */
    String userAlias() default "";

    /**
     * sql 中数据 deptId 的别名
     */
```

```
    String deptAlias() default "";

    /**
     * true：没有部门数据权限，也能查询本人数据
     */
    boolean self() default true;
}
```

5.2.2. 具体实现

```
@Aspect
@Component
public class DataFilterAspect {
    @Autowired
    private SysRoleDeptService sysRoleDeptService;

    /**
     * 切点
     */
    @Pointcut("@annotation(com.platform.annotation.DataFilter)")
    public void dataFilterCut() {

    }

    /**
     * 前置通知
     *
     * @param point 连接点
     */
    @Before("dataFilterCut()")
    public void dataFilter(JoinPoint point) {
        //获取参数
        Object params = point.getArgs()[0];
        if (params != null && params instanceof Map) {
            SysUserEntity user = ShiroUtils.getUserEntity();

            //如果不是超级管理员，则只能查询本部门及子部门数据
            if (user.getUserId() != Constant.SUPER_ADMIN) {
                Map map = (Map) params;
                map.put("filterSql", getFilterSQL(user, point));
            }
            return;
        }
        throw new RRException("数据权限接口的参数必须为Map类型，且不能为NULL");
    }

    /**
     * 获取数据过滤的SQL
     *
     * @param user  登录用户
     * @param point 连接点
     * @return sql
     */
    private String getFilterSQL(SysUserEntity user, JoinPoint point) {
        MethodSignature signature = (MethodSignature) point.getSignature();
        DataFilter dataFilter = signature.getMethod().getAnnotation(DataFilter.class);

        String userAlias = dataFilter.userAlias();
        String deptAlias = dataFilter.deptAlias();

        StringBuilder filterSql = new StringBuilder();
        filterSql.append(" and ( ");
        if (StringUtils.isNotEmpty(deptAlias) || StringUtils.isNotBlank(userAlias)) {
            if (StringUtils.isNotEmpty(deptAlias)) {
                //取出登录用户部门权限
                String alias = getAliasByUser(user.getUserId());
                filterSql.append(deptAlias);
```

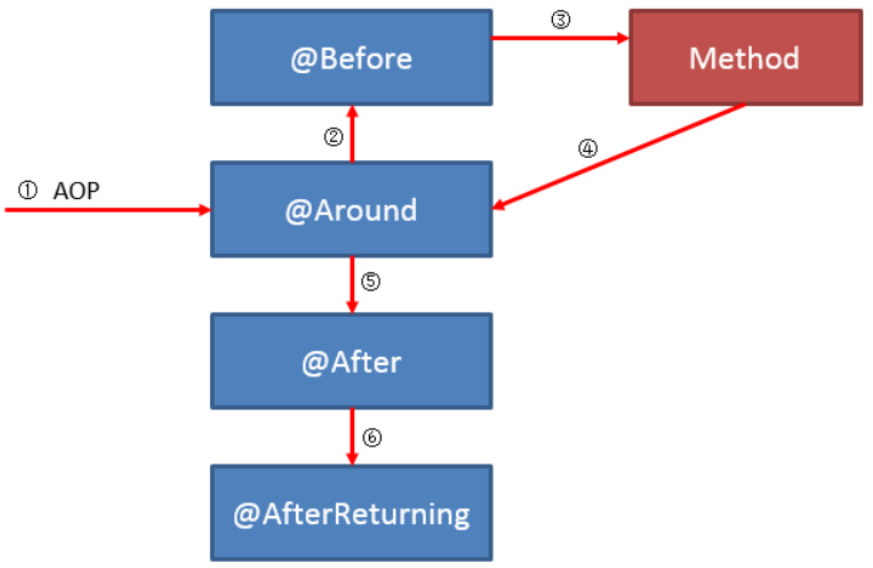
```
            filterSql.append(" in ");
            filterSql.append(" ( ");
            filterSql.append(alias);
            filterSql.append(" ) ");
            if (StringUtils.isNotBlank(userAlias)) {
                if (dataFilter.self()) {
                    filterSql.append(" or ");
                } else {
                    filterSql.append(" and ");
                }
            }
        }
        if (StringUtils.isNotBlank(userAlias)) {
            //没有部门数据权限，也能查询本人数据
            filterSql.append(userAlias);
            filterSql.append(" = ");
            filterSql.append(user.getUserId());
            filterSql.append(" ");
        }
    } else {
        return "";
    }
    filterSql.append(" ) ");
    return filterSql.toString();
}

/**
 * 取出用户权限
 *
 * @param userId 登录用户 Id
 * @return 权限
 */
private String getAliasByUser(Long userId) {
    @SuppressWarnings("unchecked")
    List<Long> roleOrglist = sysRoleDeptService.queryDeptIdListByUserId(userId);
    StringBuilder roleStr = new StringBuilder();
    String alias = "";
    if (roleOrglist != null && !roleOrglist.isEmpty()) {
        for (Long roleId : roleOrglist) {
            roleStr.append(",");
            roleStr.append("'");
            roleStr.append(roleId);
            roleStr.append("'");
        }
        alias = roleStr.toString().substring(1, roleStr.length());
    }
    return alias;
}
}
```

5.2.3. 说明

该实现类中，定义了一个切入点，只要方法上加@DataFilter 注解，执行添加注解的方法执行之前会进入 dataFilter 方法。可以看到上面代码 map.put("filterSql"，getFilterSQL(user，point))；把查询的过滤条件存入方法的 map 参数中，key 值 filterSql，所以使用此注解的方法第一个参数必须是 Map 类型。

执行顺序：

正常：



5.2.4. 生成过滤条件的 SQL

```
/**
 * 获取数据过滤的 SQL
 *
 * @param user  登录用户
 * @param point 连接点
 * @return sql
 */
private String getFilterSQL(SysUserEntity user, JoinPoint point) {
    MethodSignature signature = (MethodSignature) point.getSignature();
    DataFilter dataFilter = signature.getMethod().getAnnotation(DataFilter.class);

    String userAlias = dataFilter.userAlias();
    String deptAlias = dataFilter.deptAlias();

    StringBuilder filterSql = new StringBuilder();
    filterSql.append(" and ( ");
    if (StringUtils.isNotEmpty(deptAlias) || StringUtils.isNotBlank(userAlias)) {
        if (StringUtils.isNotEmpty(deptAlias)) {
            //取出登录用户部门权限
            String alias = getAliasByUser(user.getUserId());
            filterSql.append(deptAlias);
            filterSql.append(" in ");
            filterSql.append(" ( ");
            filterSql.append(alias);
            filterSql.append(" ) ");
            if (StringUtils.isNotBlank(userAlias)) {
                if (dataFilter.self()) {
                    filterSql.append(" or ");

                } else {
                    filterSql.append(" and ");
                }
            }
        }
        if (StringUtils.isNotBlank(userAlias)) {
            //没有部门数据权限，也能查询本人数据
            filterSql.append(userAlias);
            filterSql.append(" = ");
            filterSql.append(user.getUserId());
            filterSql.append(" ");
        }
    } else {
        return "";
    }
    filterSql.append(" ) ");
    return filterSql.toString();
}
```

5.2.5. 数据权限实现案例

◆ 对商品列表的查询，代码如下

GoodsServiceImpl.java

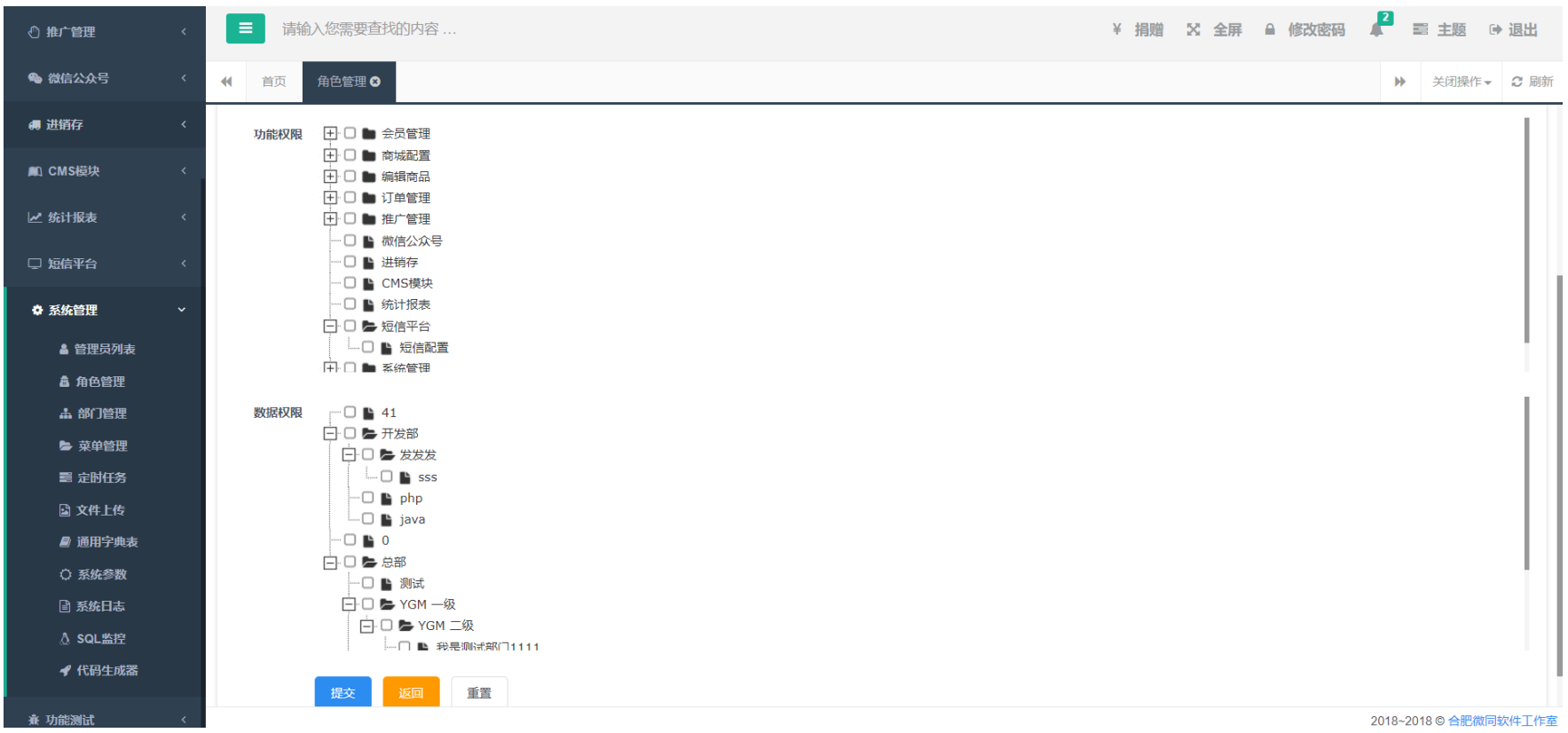
```
@Override
@DataFilter(userAlias = "nideshop_goods.create_user_id", deptAlias = "nideshop_goods.create_user_dept_id")
public List<GoodsEntity> queryList(Map<String, Object> map) {
    return goodsDao.queryList(map);
}
```

GoodsDao.xml

```
<select id="queryList" resultType="com.platform.entity.GoodsEntity">
    select
    nideshop_goods.id,
    nideshop_goods.category_id,
    nideshop_goods.goods_sn,
    nideshop_goods.name,
    nideshop_goods.brand_id,
    nideshop_goods.goods_number,
    nideshop_goods.keywords,
    nideshop_goods.goods_brief,
    nideshop_goods.goods_desc,
    nideshop_goods.is_on_sale,
    nideshop_goods.add_time,
    nideshop_goods.update_time,
    nideshop_goods.sort_order,
    nideshop_goods.is_delete,
    nideshop_goods.attribute_category,
    nideshop_goods.counter_price,
    nideshop_goods.extra_price,
    nideshop_goods.is_new,
    nideshop_goods.goods_unit,
    nideshop_goods.primary_pic_url,
    nideshop_goods.list_pic_url,
    nideshop_goods.retail_price,
    nideshop_goods.sell_volume,
    nideshop_goods.primary_product_id,
    nideshop_goods.unit_price,
    nideshop_goods.promotion_desc,
    nideshop_goods.promotion_tag,
    nideshop_goods.app_exclusive_price,
    nideshop_goods.is_app_exclusive,
    nideshop_goods.is_limited,
    nideshop_goods.is_hot,
    nideshop_goods.market_price,
    nideshop_goods.create_user_id,
    nideshop_goods.create_user_dept_id,
    nideshop_goods.update_user_id,
    nideshop_category.name category_name,
    nideshop_attribute_category.name attribute_category_name,
    nideshop_brand.name brand_name
    from nideshop_goods
    LEFT JOIN nideshop_category
    ON nideshop_goods.category_id = nideshop_category.id
    LEFT JOIN nideshop_attribute_category ON nideshop_goods.attribute_category = nideshop_attribute_category.id
    LEFT JOIN nideshop_brand ON nideshop_brand.id = nideshop_goods.brand_id
    WHERE 1=1
    <!--  数据过滤  -->
    ${filterSql}
    <if test="name != null and name != ''">
        AND nideshop_goods.name LIKE concat('%',#{name},'%')
    </if>
    AND nideshop_goods.is_Delete = #{isDelete}
    <choose>
        <when test="sidx != null and sidx.trim() != ''">
```

```
            order by ${sidx} ${order}
        </when>
        <otherwise>
            order by nideshop_goods.id desc
        </otherwise>
    </choose>
    <if test="offset != null and limit != null">
        limit #{offset}, #{limit}
    </if>
</select>
```

◆ 在 sql 语句的 where 条件中，加入${filterSql}，这样就完成了数据权限的代码实现，最后在角色管理页面配置数据权限。



5.3. XSS 脚本过滤

XSS 攻击全称跨站脚本攻击，是为不和层叠样式表(Cascading Style Sheets, CSS)的缩写混淆，故将跨站脚本攻击缩写为 XSS，XSS 是一种在 web 应用中的计算机安全漏洞，它允许恶意 web 用户将代码植入到提供给其它用户使用的页面中。本系统针对 XSS 攻击，提供了过滤功能，可以有效防止 XSS 攻击，代码请参照 XssFilter.java

5.3.1. 富文本数据处理

在 web.xml 配置处理带有富文本参数的请求，代码如下

```
<filter>
    <filter-name>xssFilter</filter-name>
    <filter-class>com.platform.xss.XssFilter</filter-class>
    <init-param>
        <!--凡是提交包含 html 内容的请求都要写在这里，若有多个以逗号隔开-->
        <param-name>excludedPages</param-name>
        <param-value>/topic/update,/topic/save,/goods/save,/goods/update</param-value>
    </init-param>
</filter>
```

具体实现代码

```
public class XssFilter implements Filter {
    //需要排除过滤的 url
    private String excludedPages;
    private String[] excludedPageArray;

    @Override
    public void init(FilterConfig config) throws ServletException {
        excludedPages = config.getInitParameter("excludedPages");
        if (StringUtils.isNotEmpty(excludedPages)) {
            excludedPageArray = excludedPages.split(",");
```

```
        }
    }

    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
            throws IOException, ServletException {
        XssHttpServletRequestWrapper xssRequest = new XssHttpServletRequestWrapper((HttpServletRequest) request);

        boolean isExcludedPage = false;
        for (String page : excludedPageArray) {//判断是否在过滤 url 之外
            if (((HttpServletRequest) request).getServletPath().equals(page)) {
                isExcludedPage = true;
                break;
            }
        }
        if (isExcludedPage) {//排除过滤 url
            chain.doFilter(request, response);
        } else {
            chain.doFilter(xssRequest, response);
        }
    }

    @Override
    public void destroy() {
    }
}
```

5.4. SQL 注入

本系统使用的是 Mybatis，如果使用${}拼接 SQL，则存在 SQL 注入风险，可以对参数进行过滤，避免 SQL 注入，具体代码实现请参照 SQLFilter.java

5.4.1. 处理 SQL 注入风险

Query.java

```
public class Query extends LinkedHashMap<String, Object> {
    private static final long serialVersionUID = 1L;
    //当前页码
    private int page;
    //每页条数
    private int limit = 10;

    public Query(Map<String, Object> params) {
        this.putAll(params);

        //分页参数
        this.page = Integer.parseInt(params.get("page").toString());
        this.limit = Integer.parseInt(params.get("limit").toString());
        this.put("offset", (page - 1) * limit);
        this.put("page", page);
        this.put("limit", limit);

        //防止SQL注入（因为sidx、order是通过拼接SQL实现排序的，会有SQL注入风险）
        String sidx = params.get("sidx").toString();
        String order = params.get("order").toString();
        this.put("sidx", SQLFilter.sqlInject(sidx));
        this.put("order", SQLFilter.sqlInject(order));
    }


    public int getPage() { return page; }

    public void setPage(int page) { this.page = page; }

    public int getLimit() { return limit; }

    public void setLimit(int limit) { this.limit = limit; }
}
```

5.5. 日志拦截器

为了方便开发调试需要日志拦截器。（登录拦截和权限拦截已在 shiro 实现，日志拦截器只做控制台输出日志，不做任何拦截

处理。）输出打印除/statics/**、*.html、*.js 以外的所有请求。

```
<mvc:interceptors>
    <!-- 使用 bean 定义一个 Interceptor，直接定义在 mvc:interceptors 根下面的 Interceptor 将拦截所有的请求 -->
    <!--<bean class="com.platform.interceptor.LogInterceptor"/>-->
    <mvc:interceptor>
        <mvc:mapping path="/**"/>
        <mvc:exclude-mapping path="/statics/**"/>
        <mvc:exclude-mapping path="/**/**.html"/>
        <mvc:exclude-mapping path="/**/**.js"/>
        <bean class="com.platform.interceptor.LogInterceptor"/>
    </mvc:interceptor>
</mvc:interceptors>
```

5.5.1. 实现类

在 preHandle 记录本次请求的时间，在 afterCompletion 中取出，然后对比当前时间，即可计算出本次请求的耗时。

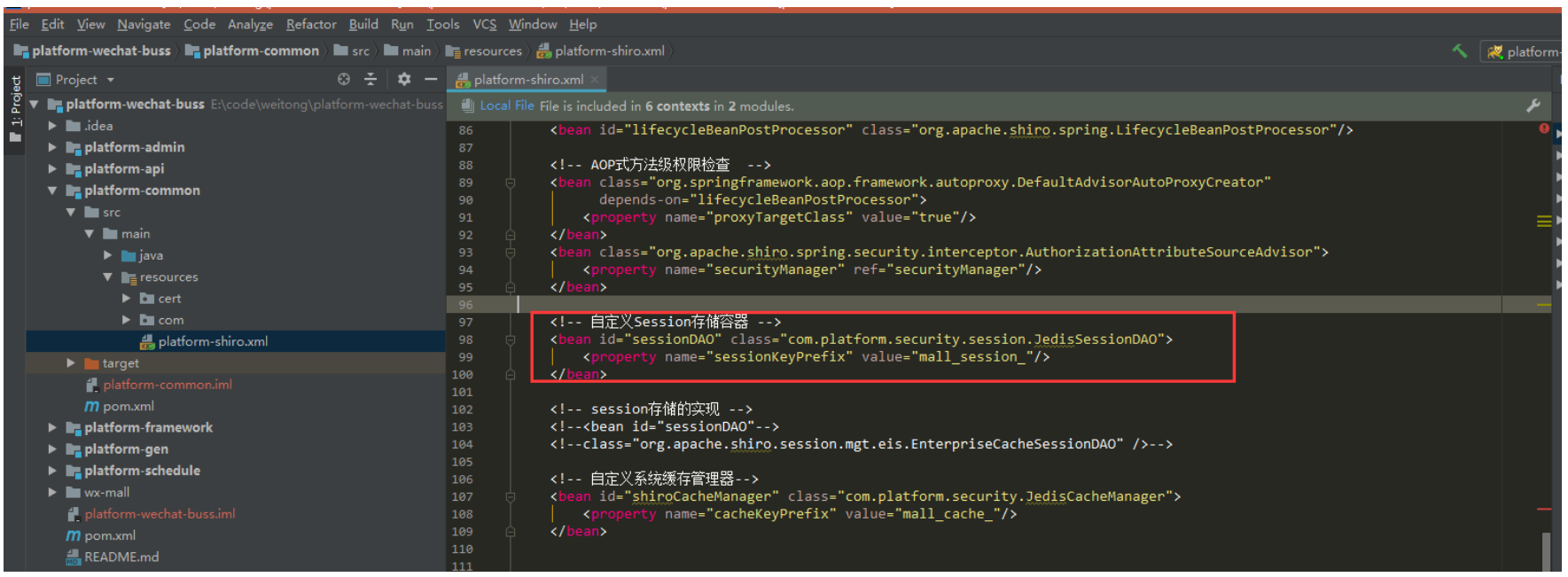
```
public class LogInterceptor extends HandlerInterceptorAdapter {
    private static final Log log = LogFactory.getLog(LogInterceptor.class);
    /*
     * (non-Javadoc)
     * @see org.springframework.web.servlet.handler.HandlerInterceptorAdapter#
     * preHandle(javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse,
     * java.lang.Object)
     */
    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {
        request.setAttribute("REQUEST_START_TIME", new Date());
        return true;
    }
    /*
     * (non-Javadoc)
     * @see org.springframework.web.servlet.handler.HandlerInterceptorAdapter#
     * postHandle(javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse,
     * java.lang.Object, org.springframework.web.servlet.ModelAndView)
     */
    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler,
                           ModelAndView modelAndView) throws Exception {
    }
    /*
     * (non-Javadoc)
     * @see org.springframework.web.servlet.handler.HandlerInterceptorAdapter#
     * afterCompletion(javax.servlet.http.HttpServletRequest,
     * javax.servlet.http.HttpServletResponse, java.lang.Object, java.lang.Exception)
     */
    @Override
    public void afterCompletion(HttpServletRequest request,
                                HttpServletResponse response, Object handler,
                                Exception ex)
            throws Exception {
        Date start = (Date) request.getAttribute("REQUEST_START_TIME");
        Date end = new Date();
        log.info("本次请求耗时：" + (end.getTime() - start.getTime()) + "毫秒：" + getRequestInfo(request).toString());
    }
    @Override
    public void afterConcurrentHandlingStarted(HttpServletRequest request,
                                               HttpServletResponse response,
                                               Object handler)
            throws Exception {
        super.afterConcurrentHandlingStarted(request, response, handler);
    }
    /**
     * 主要功能:获取请求详细信息
     * 注意事项:无
     *
     * @param request 请求
     * @return 请求信息
```

```
     */
    private StringBuilder getRequestInfo(HttpServletRequest request) {
        StringBuilder reqInfo = new StringBuilder();
        UrlPathHelper urlPathHelper = new UrlPathHelper();
        String urlPath = urlPathHelper.getLookupPathForRequest(request);
        reqInfo.append(" 请求路径=" + urlPath);
        reqInfo.append(" 来源 IP=" + RequestUtil.getIpAddrByRequest(request));
        String userName = "";
        try {
            SysUserEntity sysUser = (SysUserEntity) SecurityUtils.getSubject().getSession().getAttribute(Global.CURRENT_USER);
            if (sysUser != null) {
                userName = (sysUser.getUsername());
            }
        } catch (Exception e) {

        }
        reqInfo.append(" 操作人=" + (userName));
        reqInfo.append(" 请求参数=" + RequestUtil.getParameters(request).toString());
        return reqInfo;
    }
}
```

5.6. 分布式 session 处理

支持分布式部署，将 session 存入 redis 中。

Shiro 中的 session 默认是在容器内（Tomcat），我们只需继承 EnterpriseCacheSessionDAO 类，重写 doCreate、doReadSession、doUpdate、doDelete，将 session 存入 redis 中即可。实现代码如下：




```

/**
 * 自定义授权会话管理类
 *
 * @author zhuliyun
 */
public class JedisSessionDAO extends EnterpriseCacheSessionDAO {

    private Logger logger = LoggerFactory.getLogger(getClass());

    private String sessionKeyPrefix = "shiro_session_";
    // session 在 redis 过期时间是 30 分钟 30*60
    private static int expireTime = 1800;

    // 创建 session，保存到数据库
    @Override
    protected Serializable doCreate(Session session) {
        Serializable sessionId = super.doCreate(session);
        logger.debug("创建 session:{}", session.getId().toString());
        HttpServletRequest request = ServletUtils.getRequest();
```

```
        if (request != null) {
            String uri = request.getServletPath();
            // 如果是静态文件，则不创建 SESSION
            if (ServletUtils.isStaticFile(uri)) {
                return null;
            }
        }

        JedisUtil.setObject(sessionKeyPrefix + session.getId().toString(), session, expireTime);
        return sessionId;
    }

    // 获取 session
    @Override
    protected Session doReadSession(Serializable sessionId) {
        logger.debug("获取 session:{}", sessionId);
        // 先从缓存中获取 session，如果没有再去数据库中获取
        Session session = (Session) JedisUtil.getObject(sessionKeyPrefix + sessionId.toString());
        return session;
    }

    // 更新 session 的最后一次访问时间
    @Override
    protected void doUpdate(Session session) {
//        super.doUpdate(session);
        logger.debug("获取 session:{}", session.getId());
        String key = sessionKeyPrefix + session.getId().toString();

        HttpServletRequest request = ServletUtils.getRequest();
        if (request != null) {
            String uri = request.getServletPath();
            // 如果是静态文件，则不更新 SESSION
            if (ServletUtils.isStaticFile(uri)) {
                return;
            }
            // 手动控制不更新 SESSION
            if (Global.NO.equals(request.getParameter("updateSession"))) {
                return;
            }
        }
        if (null == JedisUtil.getObject(key)) {
            return;
        }
        JedisUtil.expire(key, expireTime);
    }

    // 删除 session
    @Override
    protected void doDelete(Session session) {
        if (session == null || session.getId() == null) {
            return;
        }
        logger.debug("删除 session:{}", session.getId());
//        super.doDelete(session);
        JedisUtil.del(sessionKeyPrefix + session.getId().toString());
    }

    public String getSessionKeyPrefix() {
        return sessionKeyPrefix;
    }

    public void setSessionKeyPrefix(String sessionKeyPrefix) {
        this.sessionKeyPrefix = sessionKeyPrefix;
    }

    public static int getExpireTime() {
        return expireTime;
```

```
    }

    public static void setExpireTime(int expireTime) {
        JedisSessionDAO.expireTime = expireTime;
    }
}
```

5.7. 统一异常处理

5.7.1. 后台异常处理

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

```
public class RRException extends RuntimeException {
    private static final long serialVersionUID = 1L;
    private String msg;
    private int code = 500;

    public RRException(String msg) {
        super(msg);
        this.msg = msg;
    }
    public RRException(String msg, Throwable e) {
        super(msg, e);
        this.msg = msg;
    }

    public RRException(String msg, int code) {
        super(msg);
        this.msg = msg;
        this.code = code;
    }

    public RRException(String msg, int code, Throwable e) {
        super(msg, e);
        this.msg = msg;
        this.code = code;
    }

    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;
    }
}
```

我们定义了 RRExceptionHandler 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(value = {"com.platform"})
public class RRExceptionHandler {
    private Logger logger = LoggerFactory.getLogger(getClass());

    /**
     * 自定义异常
     */
    @ExceptionHandler(RRException.class)
```

```
    public R handleRRException(RRException e) {
        R r = new R();
        r.put("code", e.getCode());
        r.put("msg", e.getMessage());

        return r;
    }

    @ExceptionHandler(DuplicateKeyException.class)
    public R handleDuplicateKeyException(DuplicateKeyException e) {
        logger.error(e.getMessage(), e);
        return R.error("数据库中已存在该记录");
    }

    @ExceptionHandler(AuthorizationException.class)
    public R handleAuthorizationException(AuthorizationException e) {
        logger.error(e.getMessage(), e);
        return R.error("没有权限，请联系管理员授权");
    }

    @ExceptionHandler(Exception.class)
    public R handleException(Exception e) {
        logger.error(e.getMessage(), e);
        return R.error();
    }

    @ExceptionHandler(ApiRRException.class)
    public Object handleApiRRException(ApiRRException e) {
        HashMap result = new HashMap();
        result.put("errno", e.getErrno());
        result.put("errmsg", e.getErrmsg());
        return result;
    }
}
```

5.7.2. 前端统一异常处理

前端请求统一调用 Ajax.request

```
/**
 *
 Ajax.request({
        url: '', //访问路径
        dataType: 'json', //访问类型 'json','html'等
        params: getJson(form),
        resultMsg: true, false, //是否需要提示信息
        type: 'GET',//,'get','post'
        beforeSubmit: function (data) {},//提交前处理
        successCallback: function (data) {} //提交后处理
    });
 */
Ajax = function () {
    //var opt = { type:'GET',dataType:'json',resultMsg:true };
    function request(opt) {
        //添加遮罩层
        dialogLoading(true);

        if (typeof opt.cache == 'undefined') {
            opt.cache = false;
        }

        if (typeof opt == 'undefined') {
            return;
        }
        //opt = $.extend(opt, p);
        if (typeof(opt.type) == 'undefined') {
            opt.type = 'GET'
        }
```

```
        if (typeof(opt.async) == 'undefined') {
            opt.async = false;
        }
        if (typeof(opt.dataType) == 'undefined') {
            opt.dataType = 'json'
        }
        if (typeof(opt.contentType) == 'undefined') {
            opt.contentType = 'application/x-www-form-urlencoded;chartset=UTF-8'
        }
        if (typeof(opt.params) == 'undefined' || opt.params == null) {
            opt.params = {};
        }
        opt.params.date = new Date();
        if (typeof(opt.beforeSubmit) != 'undefined') {
            var flag = opt.beforeSubmit(opt);
            if (!flag) {
                return;
            }
        }
        if (typeof(opt.resultMsg) == 'undefined') {
            opt.resultMsg = true;
        }

        $.ajax({
            async: opt.async,
            url: opt.url,
            dataType: opt.dataType,
            contentType: opt.contentType,
            data: opt.params,
            crossDomain: opt.crossDomain || false,
            type: opt.type,
            cache: opt.cache,
            success: function (data) {
                //关闭遮罩
                dialogLoading(false);
                //后台抛出自定义异常处理
                if (typeof data == 'string' && data.indexOf("exception") > 0 || typeof data.code != 'undefined' && data.code != '0') {
                    var result = {code: null};
                    if (typeof data == 'string') {
                        result = eval('(' + data + ')')
                    } else if (typeof data == 'object') {
                        result = data;
                    }
                    if (opt.resultMsg && result.msg) {
                        layer.alert(result.msg, {icon: 5});
                    }
                    return;
                }
                if (opt.resultMsg && data.msg) {
                    layer.alert(data.msg, {icon: 6}, function () {
                        if (typeof(opt.successCallback) != 'undefined') {
                            opt.successCallback(data);
                        }
                    });
                    return;
                }
                if (typeof(opt.successCallback) != 'undefined') {
                    opt.successCallback(data);
                }
            },
            error: function () {
                //关闭遮罩
                dialogLoading(false);

                layer.alert("此页面发生未知异常,请联系管理员", {icon: 5});
            }
        });
```

```
    }
    return {
        /**
         * Ajax 调用 request
         */
        request: request
    };
}();
```

后台抛出未知异常，进入 error，提示此页面发生未知异常，请联系管理员。当后台抛出自定义异常，由 RRExceptionHandler 类捕获，返回异常信息 Json 数据。

5.8. 系统日志

系统日志是通过 Spring AOP 实现的，我们自定义了注解@SysLog，此注解只能用于方法上。

5.8.1. 定义注解

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface SysLog {
    String value() default "操作日志";
}
```

5.8.2. 具体实现

```
@Aspect
@Component
public class SysLogAspect {
    @Autowired
    private SysLogService sysLogService;
    /**
     * 切点
     */
    @Pointcut("@annotation(com.platform.annotation.SysLog)")
    public void logPointCut() {}
    /**
     * 前置通知
     *
     * @param joinPoint 连接点
     */
    @Before("logPointCut()")
    public void saveSysLog(JoinPoint joinPoint) {
        MethodSignature signature = (MethodSignature) joinPoint.getSignature();
        Method method = signature.getMethod();
        SysLogEntity sysLog = new SysLogEntity();
        SysLog syslog = method.getAnnotation(SysLog.class);
        if (syslog != null) {
            //注解上的描述
            sysLog.setOperation(syslog.value());
        }
        //请求的方法名
        String className = joinPoint.getTarget().getClass().getName();
        String methodName = signature.getName();
        sysLog.setMethod(className + "." + methodName + "()");
        //请求的参数
        Object[] args = joinPoint.getArgs();
        String params = JSON.toJSONString(args[0]);
        sysLog.setParams(params);
        //获取 request
        HttpServletRequest request = HttpContextUtils.getHttpServletRequest();
        //设置 IP 地址
        sysLog.setIp(IPUtils.getIpAddr(request));
        //用户名
        SysUserEntity sysUserEntity = ShiroUtils.getUserEntity();
        String username = "";
```

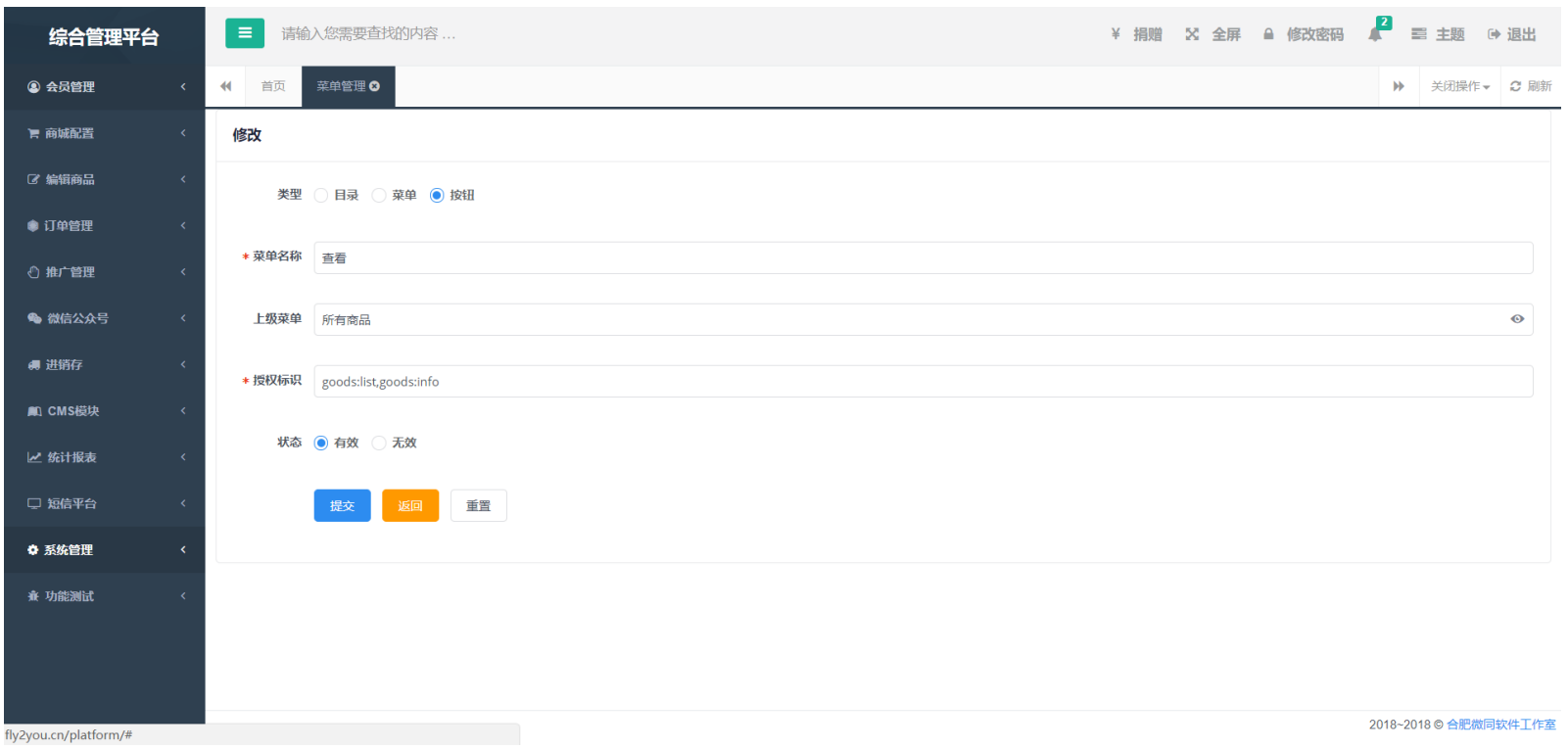
```
        if ("login".equals(methodName)) {
            username = params;
        }
        if (null != sysUserEntity) {
            username = ShiroUtils.getUserEntity().getUsername();
        }
        sysLog.setUsername(username);
        sysLog.setCreateDate(new Date());
        //保存系统日志
        sysLogService.save(sysLog);
    }
}
```

5.8.3. 使用方式

```
/**
 * 保存
 */
@SysLog("保存菜单")
@RequestMapping("/save")
@RequiresPermissions("sys:menu:save")
public R save(@RequestBody SysMenuEntity menu) {
    //数据校验
    verifyForm(menu);
    sysMenuService.save(menu);
    return R.ok();
}
```

5.9. 添加菜单

菜单管理，主要是对【目录、菜单、按钮】进行动态的新增、修改、删除等操作，方便开发者管理菜单。



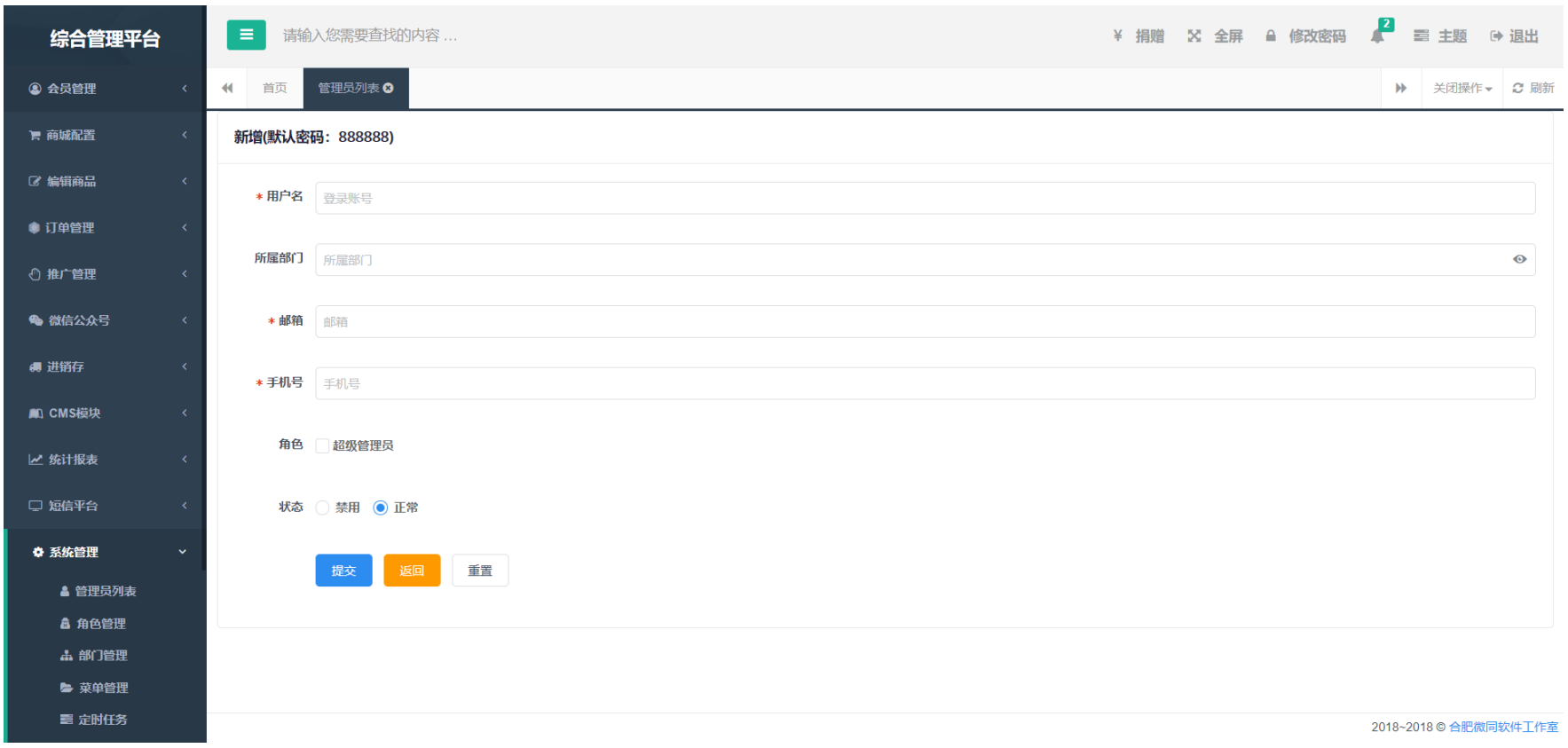
上图是拿现有的菜单进行讲解。其中，授权标识与 shiro 中的注解@RequiresPermissions，定义的授权标识是一一对应的。

GoodsController.java

```java
    /**
     * 查看列表
     */
    @RequestMapping("/list")
    @RequiresPermissions("goods:list")
    public R list(@RequestParam Map<String, Object> params) {
        //查询列表数据
        Query query = new Query(params);

        query.put("isDelete", 0);
        List<GoodsEntity> goodsList = goodsService.queryList(query);
        int total = goodsService.queryTotal(query);

        PageUtils pageUtil = new PageUtils(goodsList, total, query.getLimit(), query.getPage());

        return R.ok().put("page", pageUtil);
    }

    /**
     * 查看信息
     */
    @RequestMapping("/info/{id}")
    @RequiresPermissions("goods:info")
    public R info(@PathVariable("id") Integer id) {
        GoodsEntity goods = goodsService.queryObject(id);

        return R.ok().put("goods", goods);
    }
```

5.10. 添加管理员

本系统默认就创建了 admin 账号，无需分配任何角色，就拥有最高权限。一个管理员是可以拥有多个角色的。创建的管理员默认密码是 888888



5.11. 定时任务模块

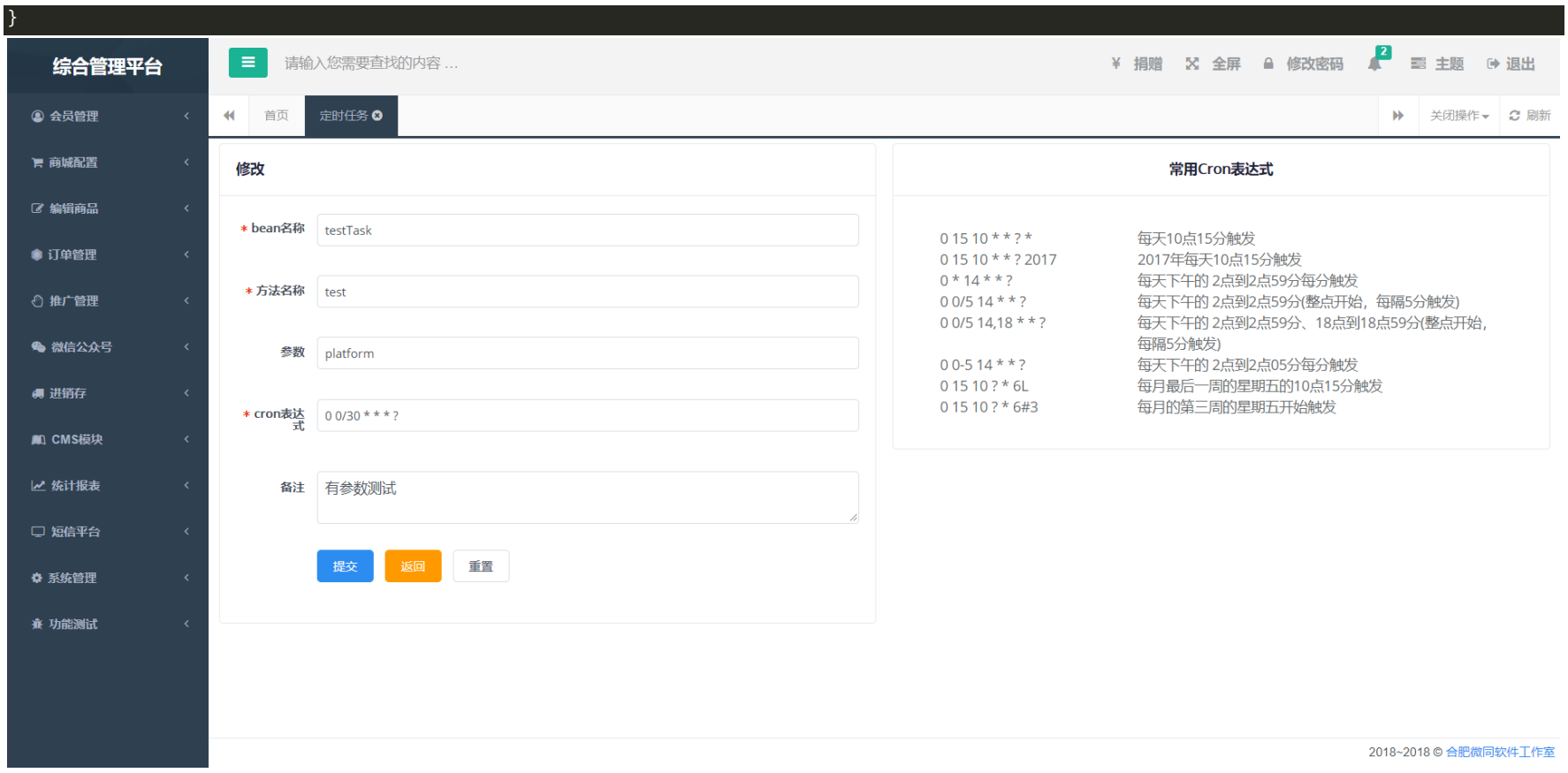
本系统使用开源框架 Quartz，实现的定时任务，已实现分布式定时任务，可部署多台服务器，不重复执行，以及动态增加、修改、删除、暂停、恢复、立即执行定时任务。

5.11.1.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，然后在页面添加定时任务

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

    @Autowired
    private SysUserService sysUserService;

    public void test(String params) {
        logger.info("我是带参数的 test 方法，正在被执行，参数为：" + params);
```

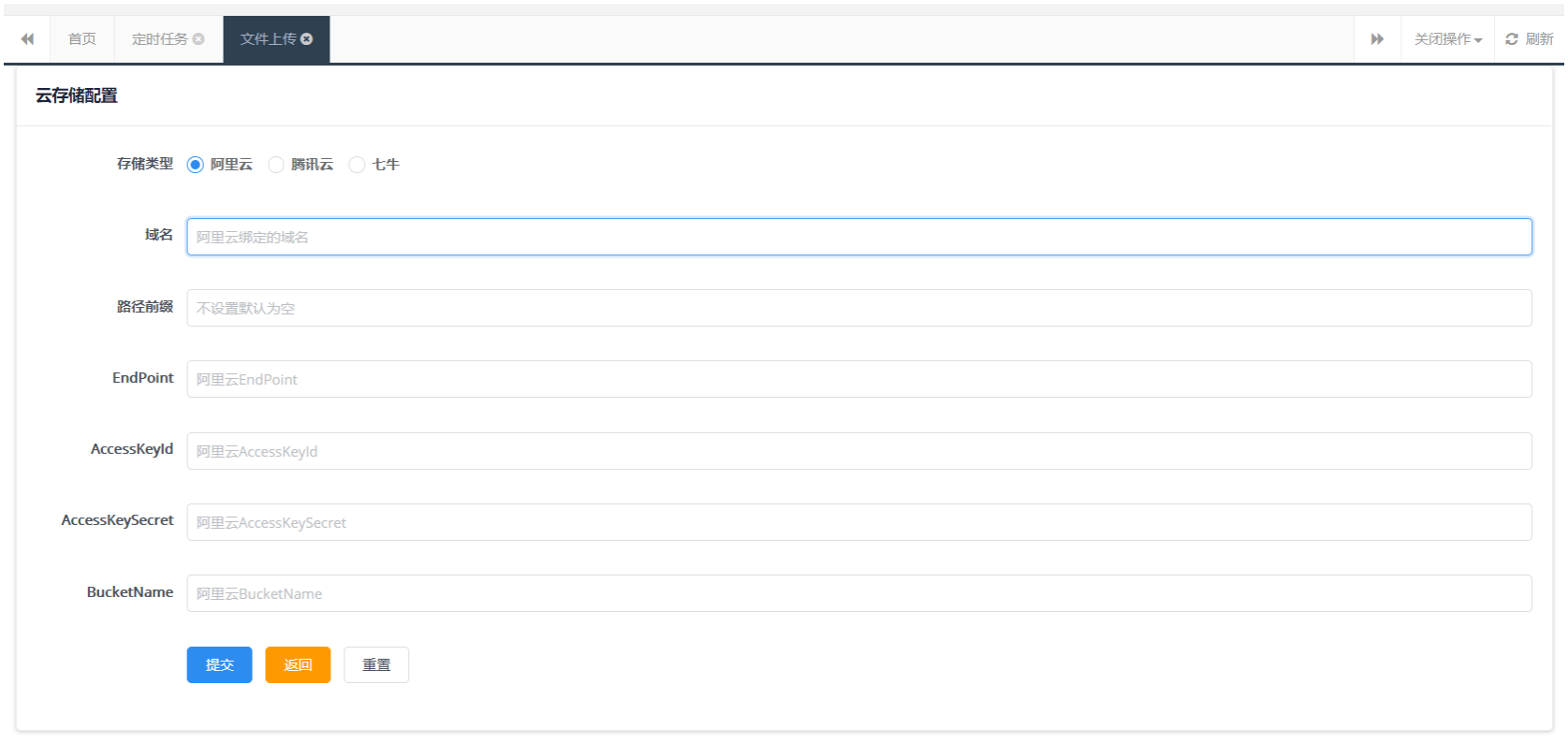
```
        try {
            Thread.sleep(1000L);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        SysUserEntity user = sysUserService.queryObject(1L);
        System.out.println(ToStringBuilder.reflectionToString(user));

    }

    public void test2() {
        logger.info("我是不带参数的test2方法，正在被执行");
    }
}
```



5.12. 云存储模块

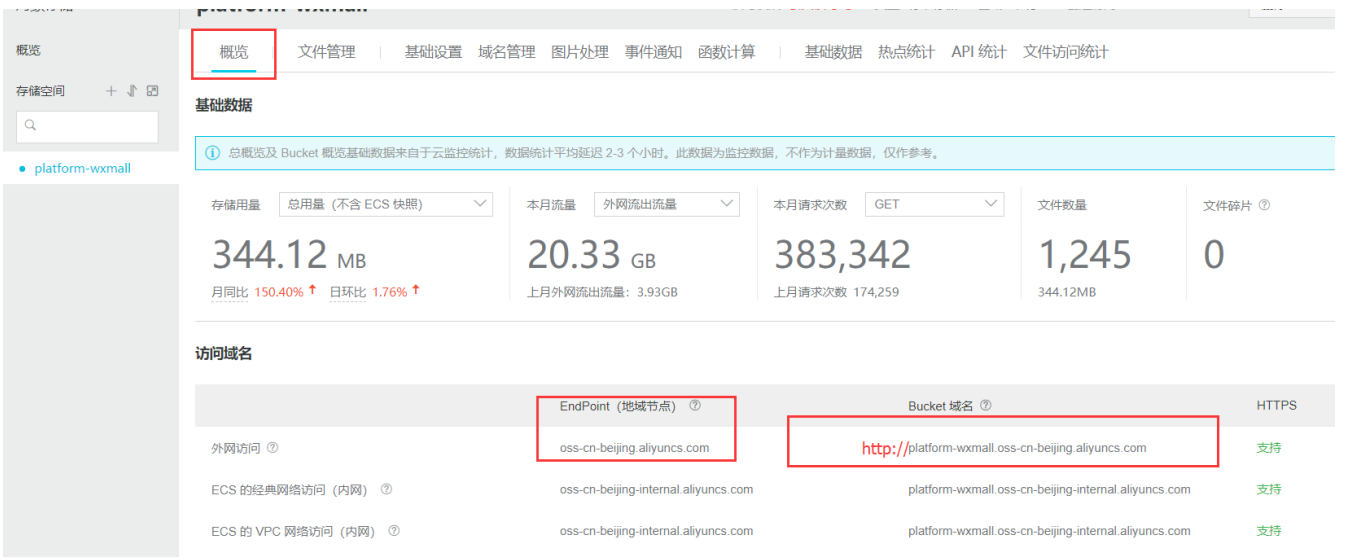
图片、文件上传，使用的是七牛、阿里云、腾讯云的存储服务，不能上传到本地服务器。上传到本地服务器，不利于维护，访问速度慢、占用服务器带宽等缺点，所以推荐使用云存储服务。

5.12.1.阿里云配置



登录阿里云控制台，获取配置信息

域名（Bucket 域名）、EndPoint：

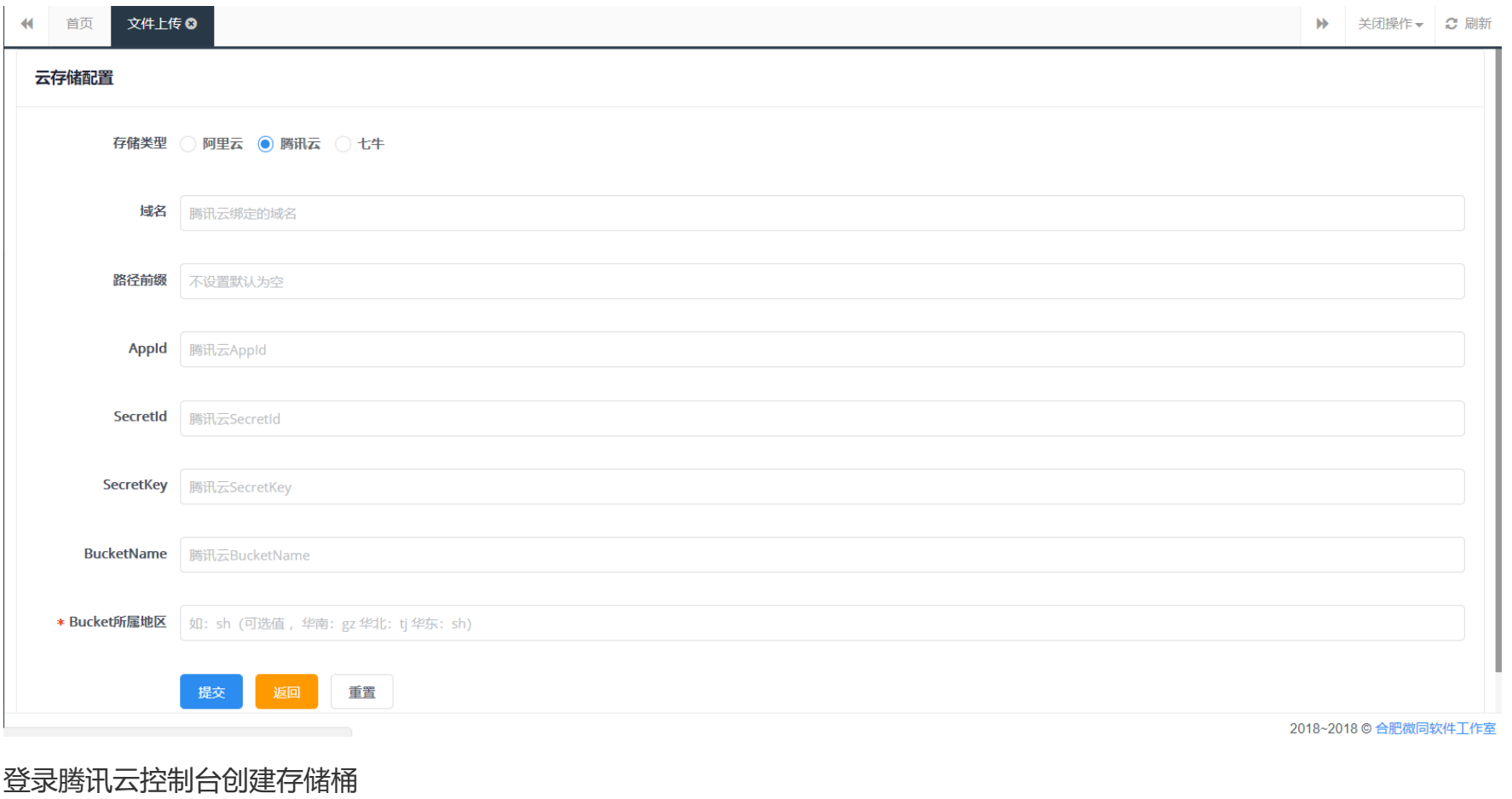


路径前缀：自己定义（如 upload）

Bucket：

AccessKeyId、AccessKeySecret：

5.12.2.腾讯云配置

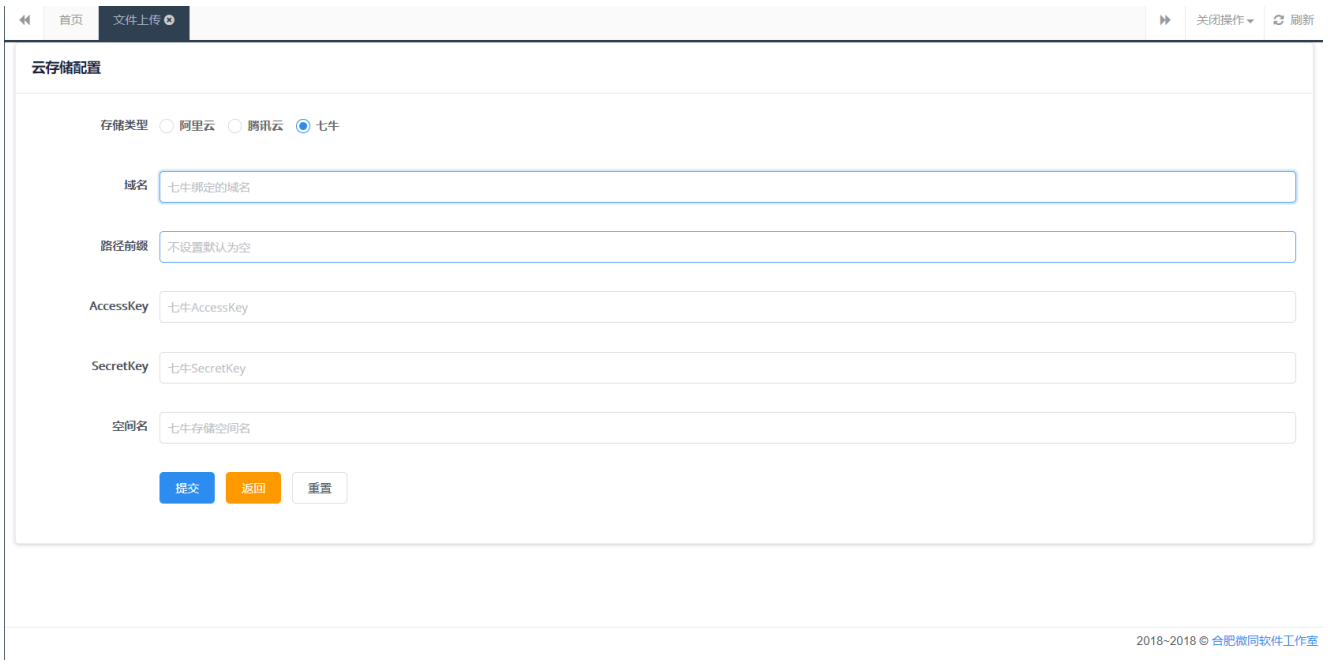


登录腾讯云控制台创建存储桶

域名、Bucket 所属地区、BucketName

AppId、SecretId、SecretKey

5.12.3.七牛云配置



登录七牛云控制台，创建公开空间

空间名、域名：

AccessKey、SecretKey

5.12.4.文件上传示例

```
/**
 * 上传文件
 *
 * @param file 文件
 * @return R
 * @throws Exception 异常
 */
@RequestMapping("/upload")
public R upload(@RequestParam("file") MultipartFile file) throws Exception {
    if (file.isEmpty()) {
        throw new RRException("上传文件不能为空");
    }
    //上传文件
    String url = OSSFactory.build().upload(file);

    //保存文件信息
    SysOssEntity ossEntity = new SysOssEntity();
    ossEntity.setUrl(url);
    ossEntity.setCreateDate(new Date());
    sysOssService.save(ossEntity);

    R r = new R();
    r.put("url", url);
    r.put("link", url);
    return r;
}
```

5.13. 短信平台

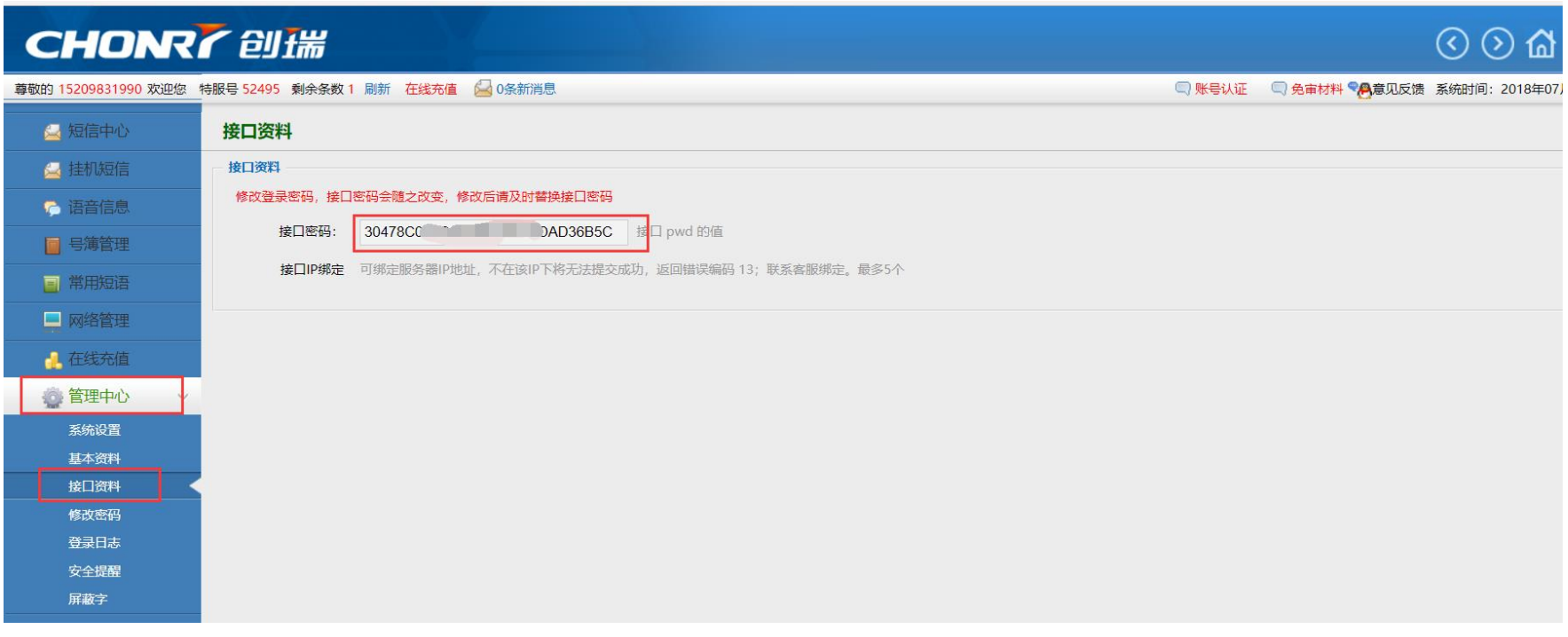
本系统已集成创瑞云短信平台功能。

5.13.1.短信配置项

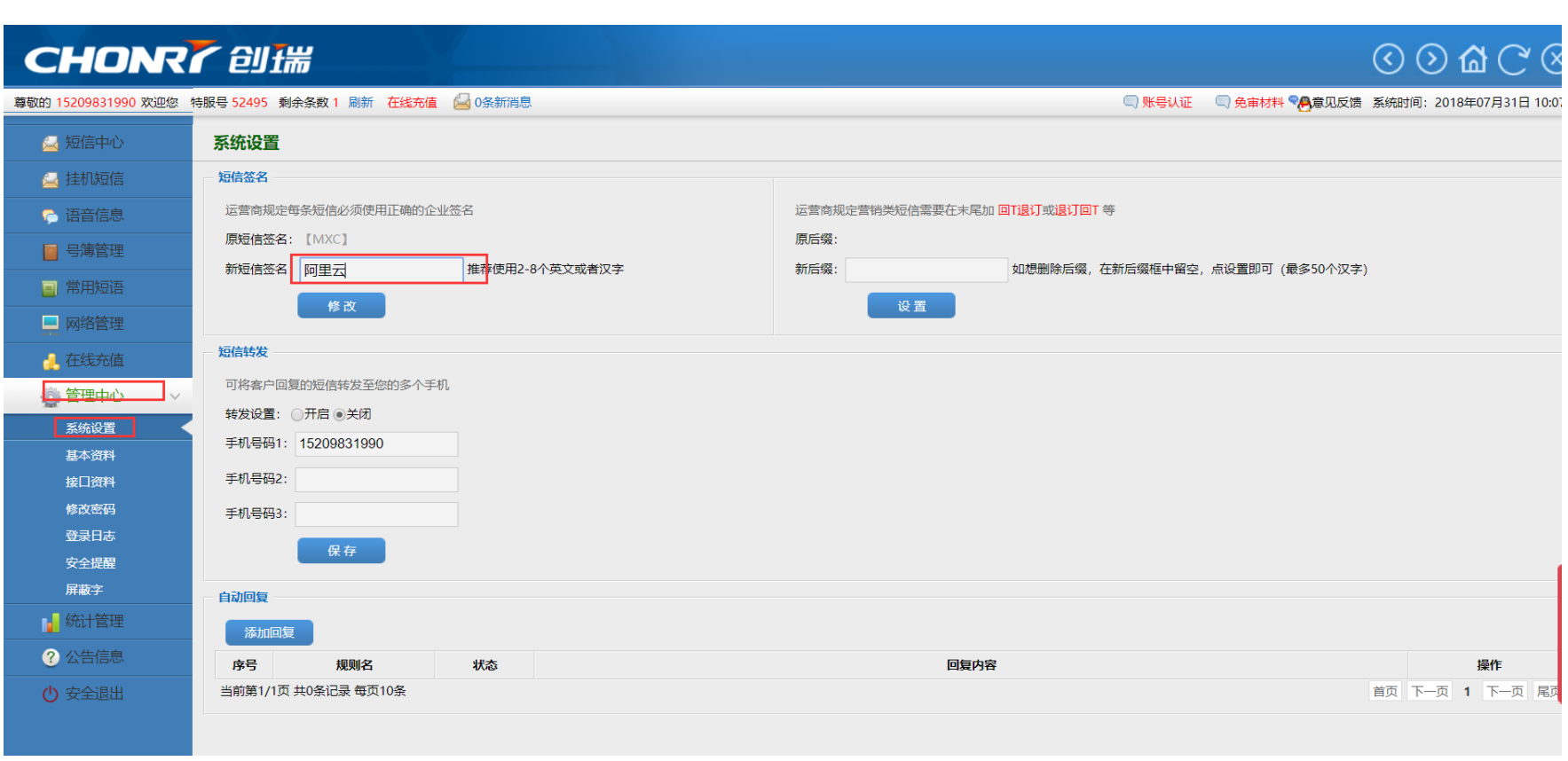
登录创瑞云平台 http://web.cr6868.com/

- 短信类型：目前只支持创瑞云 SMS
- 发送域名：
  - UTF-8 编码地址为：http://web.cr6868.com/asmx/smsservice.aspx（本系统使用 UTF-8 编码）
  - GB2312 编码地址为：http://web.cr6868.com/gbk/smsservice.aspx
- 用户名：平台登录用户名

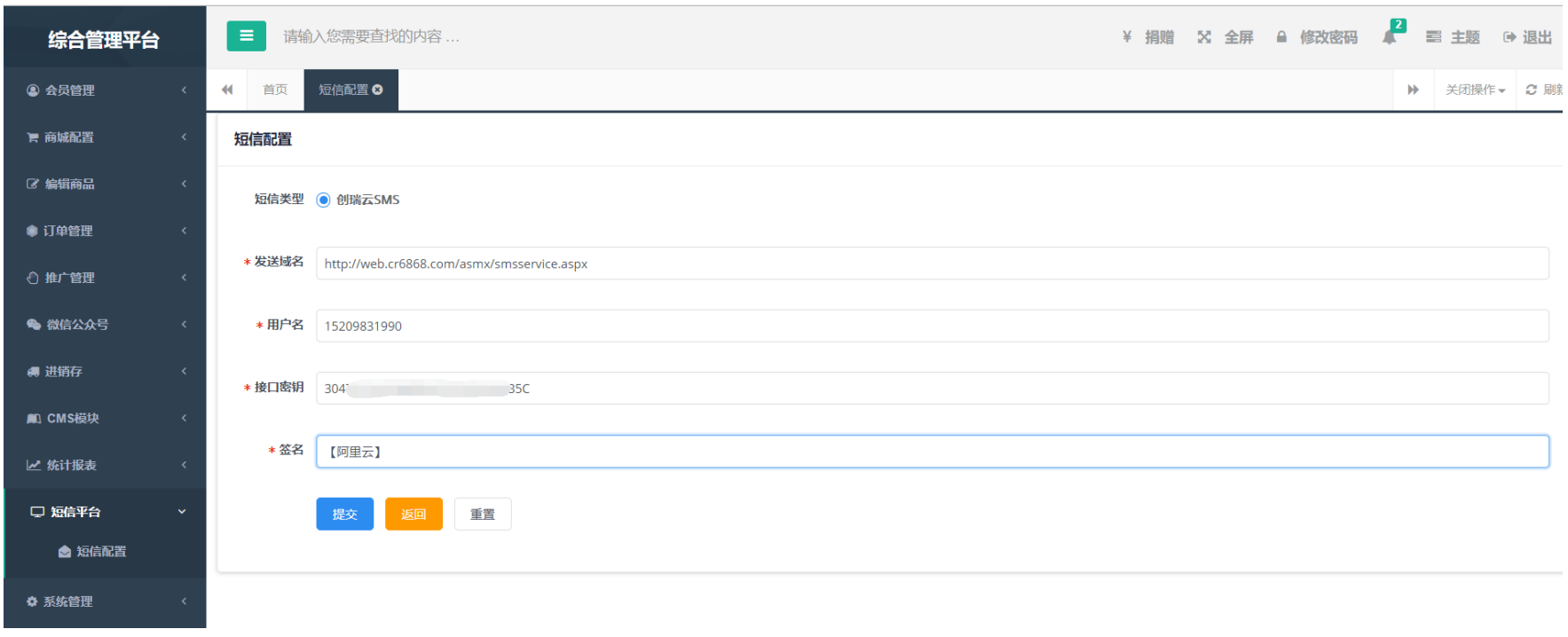
◆ 接口密钥：如下图



◆ 签名：一般签名用公司简称。如下图，注意，这里填写的签名不需要【】符号！

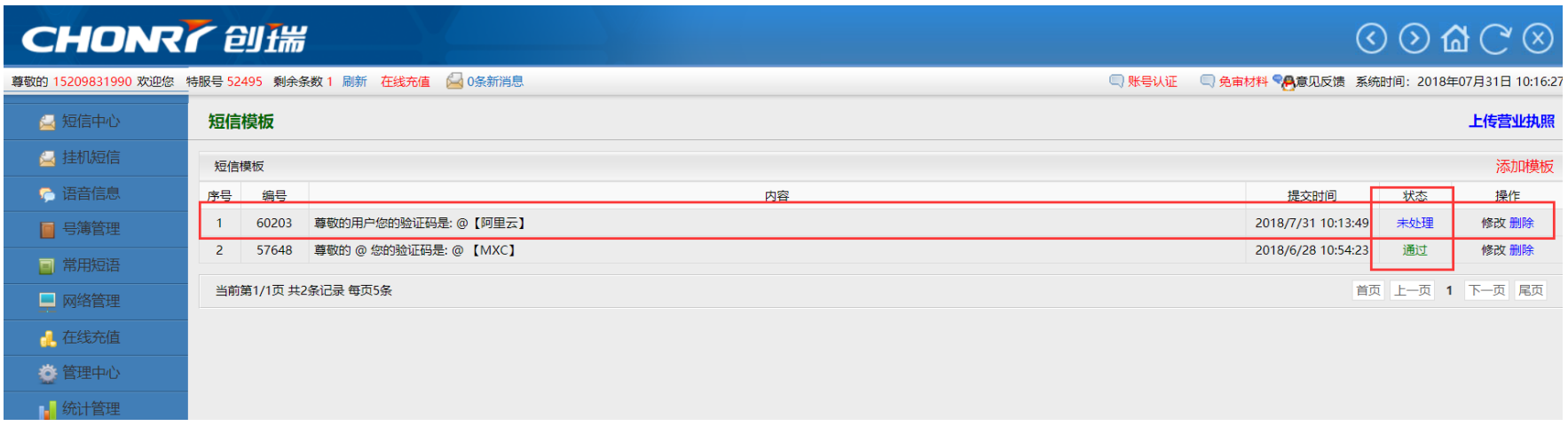


配置完成如下图：

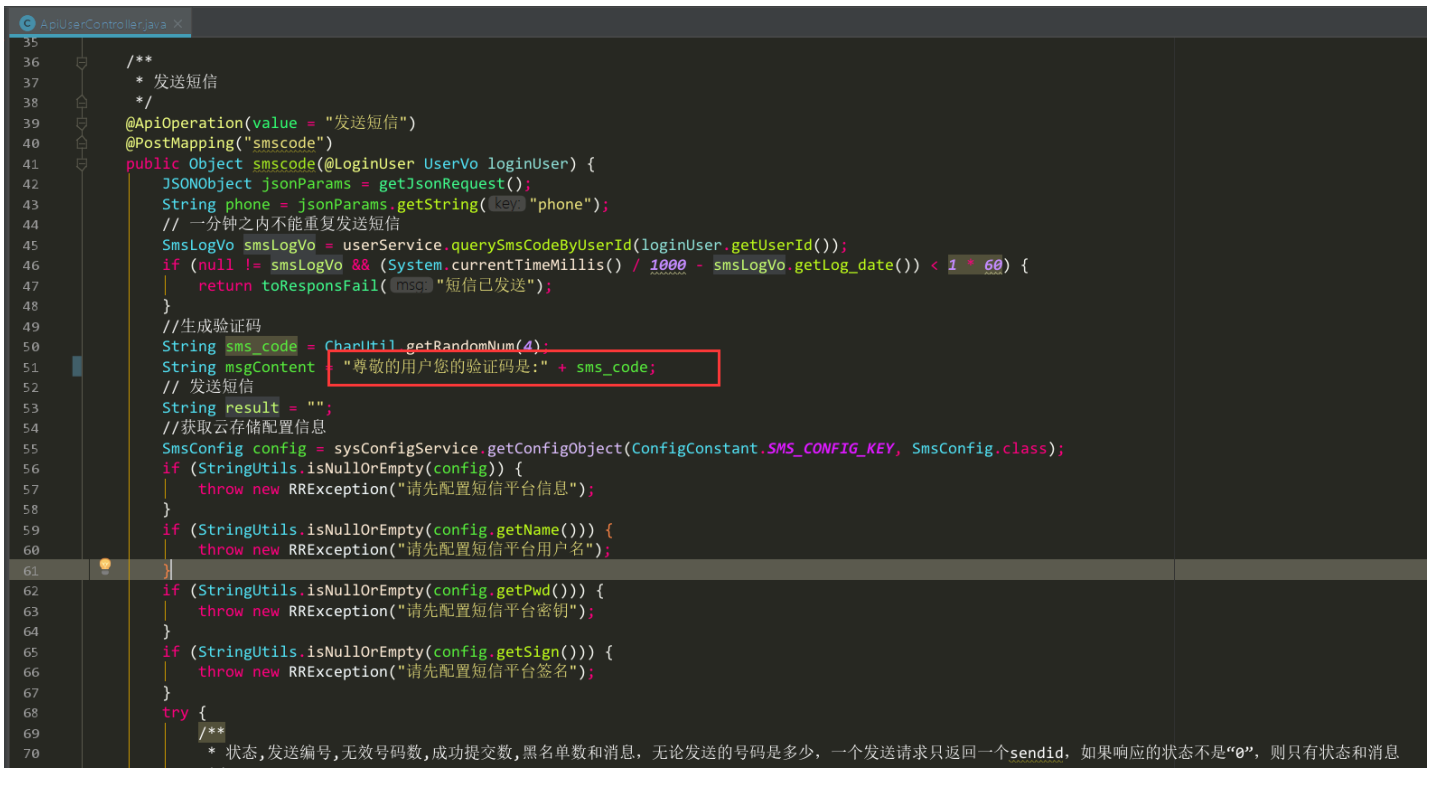


5.13.2.设置免审短信模版

在首页点击免审材料->添加模版



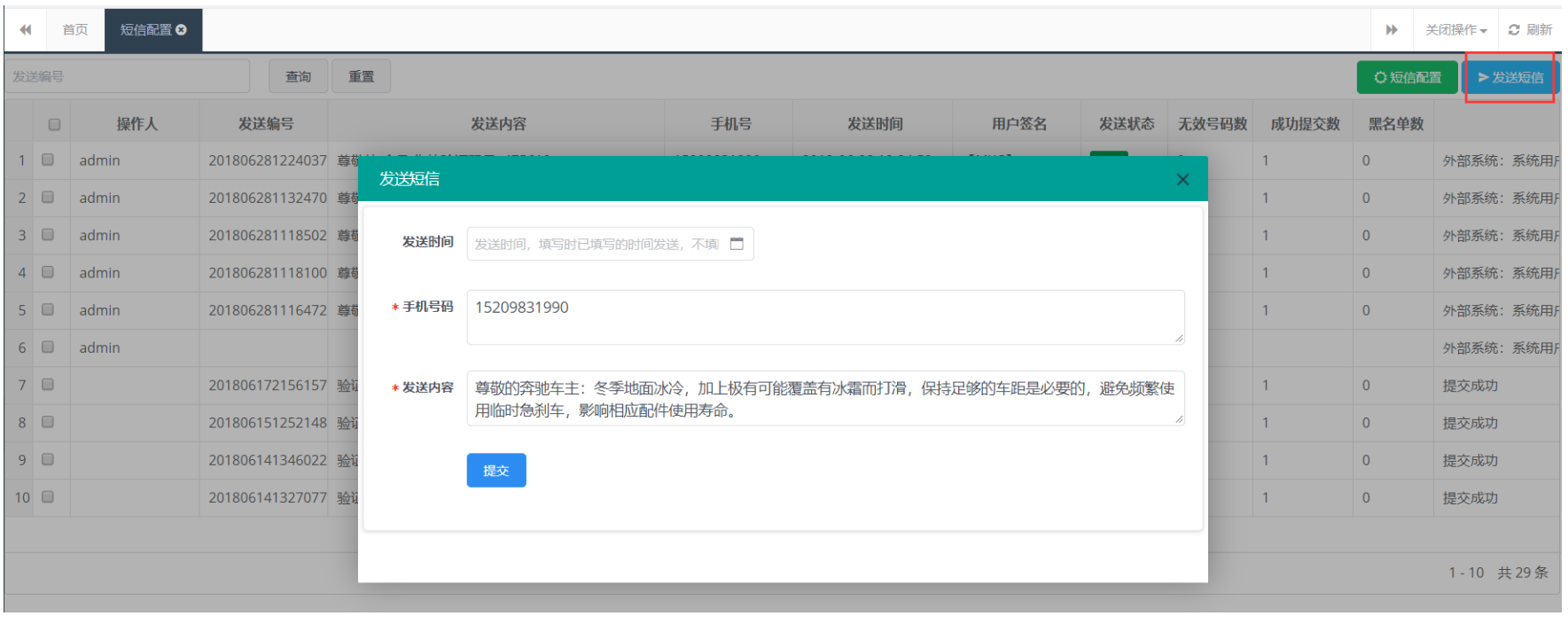
审核通过后，修改 ApiUserController 类。免审短信是根据内容匹配，一定要确保发送的短信内容和模版内容保持一致！



```java
/**
 * 发送短信
 */
@ApiOperation(value = "发送短信")
@PostMapping("smscode")
public Object smscode(@LoginUser UserVo loginUser) {
    JSONObject jsonParams = getJsonRequest();
    String phone = jsonParams.getString( key: "phone");
    // 一分钟之内不能重复发送短信
    SmsLogVo smsLogVo = userService.querySmsCodeByUserId(loginUser.getUserId());
    if (null != smsLogVo && (System.currentTimeMillis() / 1000 - smsLogVo.getLog_date()) < 1 * 60) {
        return toResponsFail( msg: "短信已发送");
    }
    //生成验证码
    String sms_code = CharUtil.getRandomNum(4);
    String msgContent = "尊敬的用户您的验证码是:" + sms_code;
    // 发送短信
    String result = "";
    //获取云存储配置信息
    SmsConfig config = sysConfigService.getConfigObject(ConfigConstant.SMS_CONFIG_KEY, SmsConfig.class);
    if (StringUtils.isNullOrEmpty(config)) {
        throw new RRException("请先配置短信平台信息");
    }
    if (StringUtils.isNullOrEmpty(config.getName())) {
        throw new RRException("请先配置短信平台用户名");
    }
    if (StringUtils.isNullOrEmpty(config.getPwd())) {
        throw new RRException("请先配置短信平台密钥");
    }
    if (StringUtils.isNullOrEmpty(config.getSign())) {
        throw new RRException("请先配置短信平台签名");
    }
    try {
        /**
         * 状态,发送编号,无效号码数,成功提交数,黑名单数和消息，无论发送的号码是多少，一个发送请求只返回一个sendid，如果响应的状态不是"0"，则只有状态和消息
```

5.13.3.在本系统中发送自定义短信

短信配置页面点击发送短信，编辑发送时间（若不填写，实时提交到后台审核），手机号码如果有多个以英文逗号隔开，发送内容 500 字以内（每 70 字符计算为一条短信，自定义发送内容不需要写签名信息）



发送自定义短信后，需要创瑞平台审核，审核通过后才能发送。注意，如果发送营销类短信，在短信最后加上‘回 T 退订’。

提交成功之后登陆创瑞平台查看发送情况。

5.13.4.在创瑞云平台发送自定义短信

步骤如下图

5.13.5.其他系统调用短信接口

在实际的应用中，我们可能会有多个系统需要短信服务，这个时候，本系统就可以对外提供短信服务接口。为了安全起见，我们需要配置有效的服务器 IP，需要配置的 IP 为其他系统所在的服务器 IP。具体代码实现如下

```
#安全起见，暴露的短信接口需要配置有效的请求 IP
sms.validIp=127.0.0.1
```

```
/**
 * 发送短信
 *
 * @param request request
 * @param params 请求参数{mobile：电话号码字符串，中间用英文逗号间隔,content：内容字符串,stime：追加发送时间，可为空，为空为及时发送}
 * @return R
 */
@IgnoreAuth
@RequestMapping("/sendSms")
public R sendSms(HttpServletRequest request, @RequestParam Map<String, String> params) {
    SysSmsLogEntity smsLog = new SysSmsLogEntity();
    String validIP = RequestUtil.getIpAddrByRequest(request);
    if (ResourceUtil.getConfigByName("sms.validIp").indexOf(validIP) < 0) {
        throw new RRException("非法 IP 请求！");
    }
    smsLog.setMobile(params.get("mobile"));
    smsLog.setContent(params.get("content"));
    String stime = params.get("stime");
    if (StringUtils.isNotEmpty(stime)) {
        smsLog.setStime(DateUtils.convertStringToDate(stime));
    }
    SysSmsLogEntity sysSmsLogEntity = smsLogService.sendSms(smsLog);
    return R.ok().put("result", sysSmsLogEntity);
}
```

5.13.6.申请注册创瑞云

提供以下信息，发送给我（QQ：939961241，微信：15209831990）

5.14. API 模块

为微信小程序商城提供接口服务，platform-api 实现了微信登录、接口权限验证、商城所有接口。

5.14.1.API 的使用

小程序端通过微信授权登录，系统会生成与登录用户对应的 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。

主要使用两个自定义注解实现，@LoginUser 注解是获取当前登录用户的信息@IgnoreAuth 是忽略用户登录，是可以直接访

问的请求。

```
/**
 * 获取用户的收货地址
 */
@ApiOperation(value = "获取用户的收货地址接口", response = Map.class)
@GetMapping("list")
public Object list(@LoginUser UserVo loginUser) {
    Map<String, Object> param = new HashMap<~>();
    param.put("user_id", loginUser.getUserId());
    List<AddressVo> addressEntities = addressService.queryList(param);
    return toResponsSuccess(addressEntities);
}
```

不用登录也能访问

```
    @ApiOperation(value = "新热门商品信息")
    @IgnoreAuth
    @GetMapping(value = "hotGoods")
    public Object hotGoods() {
        Map<String, Object> resultObj = new HashMap<~>();
        //
        Map<String, Object> param = new HashMap<~>();
        param.put("is_hot", "1");
        param.put("is_delete", 0);
        PageHelper.startPage( pageNum: 0, pageSize: 3, count: false);
        List<GoodsVo> hotGoods = goodsService.queryHotGoodsList(param);
        resultObj.put("hotGoodsList", hotGoods);
        //

        return toResponsSuccess(resultObj);
    }
```

5.15. Swagger 接口文档

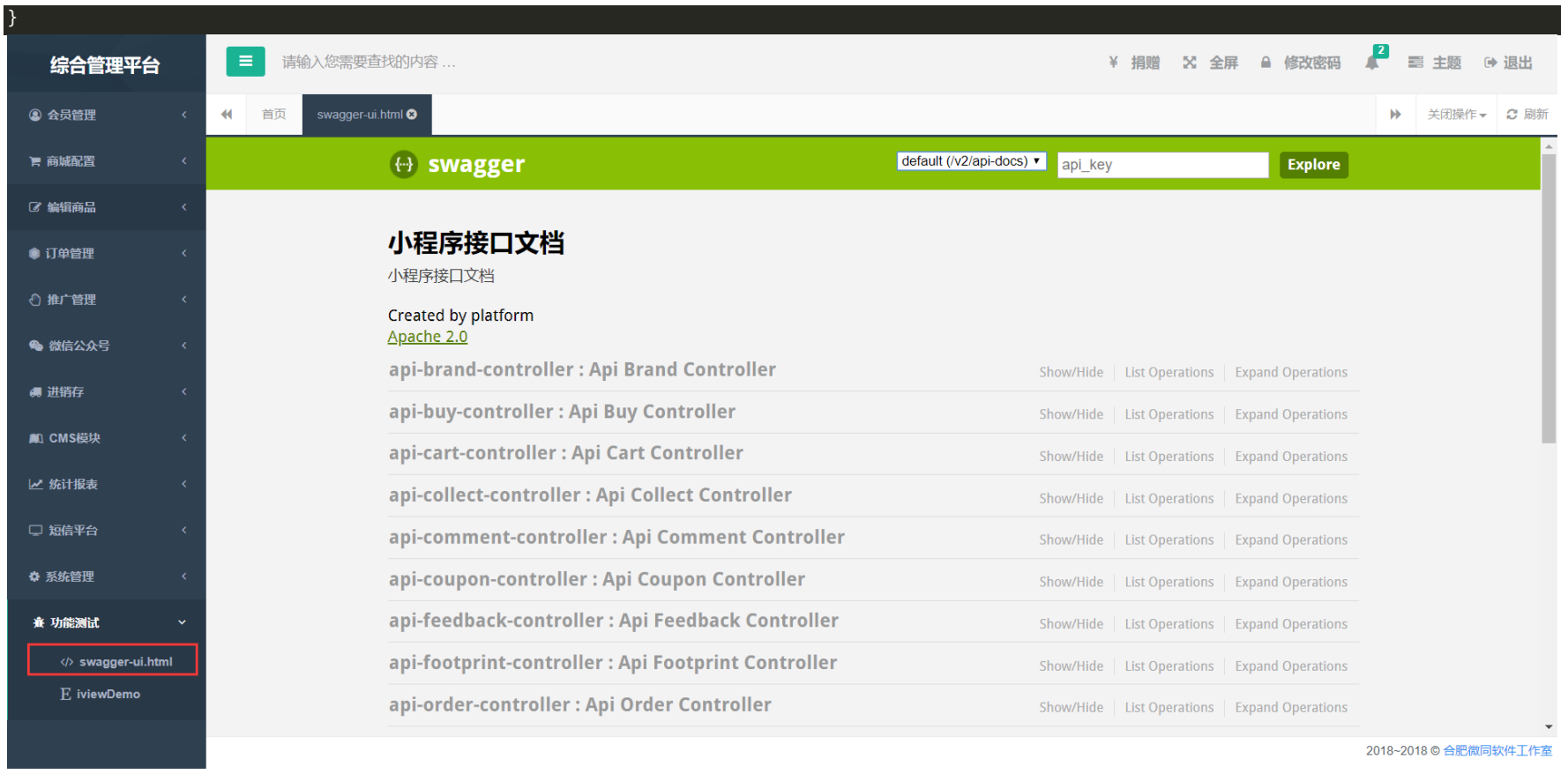
支持 Swagger 注解的方式，生成在线接口文档，使用方法：

```
@Api(tags = "收货地址")
@RestController
@RequestMapping("/api/address")
public class ApiAddressController extends ApiBaseAction {
    @Autowired
    private ApiAddressService addressService;

    /**
     * 获取用户的收货地址
     */
    @ApiOperation(value = "获取用户的收货地址接口", response = Map.class)
    @GetMapping("list")
    public Object list(@LoginUser UserVo loginUser) {
        Map<String, Object> param = new HashMap<~>();
        param.put("user_id", loginUser.getUserId());
        List<AddressVo> addressEntities = addressService.queryList(param);
        return toResponsSuccess(addressEntities);
    }

    /**
     * 获取收货地址的详情
     */
    @ApiOperation(value = "获取收货地址的详情", response = Map.class)
    @ApiImplicitParams({@ApiImplicitParam(name = "id", value = "收获地址ID", required = true, dataType = "Integer")})
    @GetMapping("detail")
    public Object detail(Integer id, @LoginUser UserVo loginUser) {
        AddressVo entity = addressService.queryObject(id);
        //判断越权行为
        if (!entity.getUserId().equals(loginUser.getUserId())) {
            return toResponsObject( requestCode: 403, msg: "您无权查看", data: "");
        }
        return toResponsSuccess(entity);
    }
```

Swagger 配置

```
@Configuration
@EnableWebMvc
@EnableSwagger2
@ComponentScan(basePackages="com.platform.api")
public class SwaggerConfig {
    @Bean
    public Docket api(){
        return new Docket(DocumentationType.SWAGGER_2)
                .apiInfo(this.apiInfo())
```

```
                .select()
                //需要生成接口文档的包
                .apis(RequestHandlerSelectors.basePackage("com.platform.api"))
                .paths(PathSelectors.any())
                .build();
    }
    private ApiInfo apiInfo(){
        @SuppressWarnings("deprecation")
        ApiInfo info=new ApiInfo(
                "pwm 接口文档",
                "pwm 接口文档",
                "1.0",
                "urn:tos",
                "platform",
                "Apache 2.0",
                "http://www.apache.org/licenses/LICENSE-2.0");
        return info;
    }
}
```



5.16. 日志分级输出

log4j 默认是判断优先级，如果设置 info，会打印 info 及优先级小于 info 的的日志，这样所有类型的日志都输出到一个文件，不便于分析。经过分析源码得知，只需更改 DailyRollingFileAppender 的 isAsSevereAsThreshold 方法即可实现日志分类打印，代码实现如下：

```
public class GradeLogDailyRollingFileAppender extends DailyRollingFileAppender {
    @Override
    public boolean isAsSevereAsThreshold(Priority priority) {
        //只判断是否相等，而不判断优先级
        return this.getThreshold().equals(priority);
    }
}
```

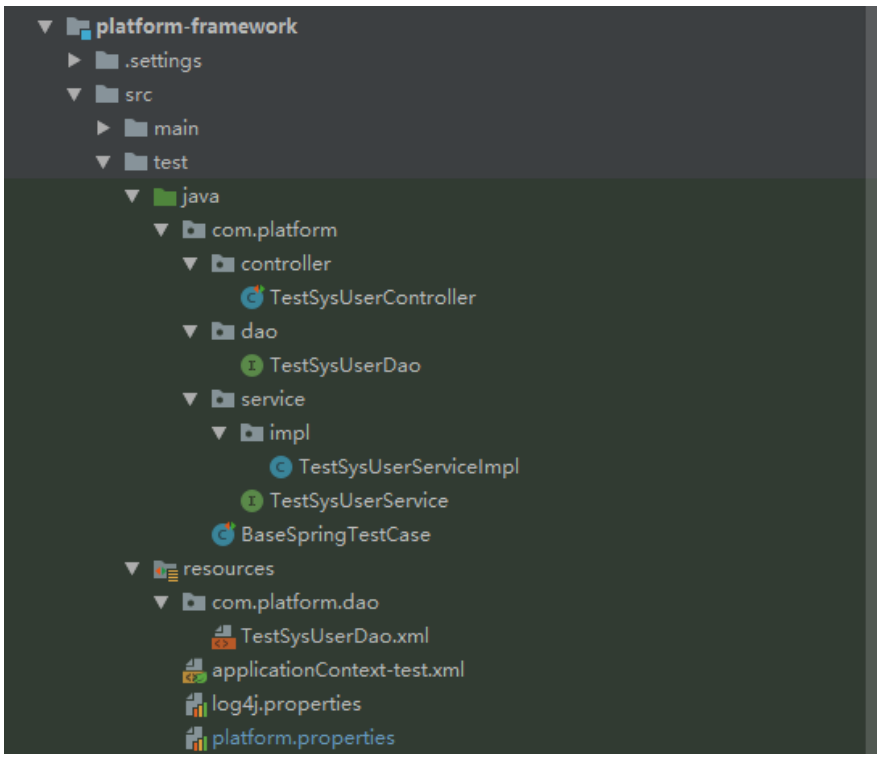
```
log4j.rootLogger=INFO,stdout,info,warn,error,file
#控制台输出
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.Threshold=INFO
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss SSS}|%5p|%F.%M:%L|%m%n
#INFO 所有日志
log4j.logger.file=info
log4j.appender.file=org.apache.log4j.DailyRollingFileAppender
log4j.appender.file.File=../logs/info.log
```

```
log4j.appender.file.datePattern='.'yyyy-MM-dd'.log'
log4j.appender.file.append=true
log4j.appender.file.Threshold=INFO
log4j.appender.file.encoding=UTF-8
log4j.appender.file.ImmediateFlush=true
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss SSS}|%5p|%F.%M:%L|%m%n
#INFO 日志
log4j.logger.info=info
log4j.appender.info=com.platform.log4j.GradeLogDailyRollingFileAppender
log4j.appender.info.File=../logs/info/info.log
log4j.appender.info.datePattern='.'yyyy-MM-dd'.log'
log4j.appender.info.append=true
log4j.appender.info.Threshold=INFO
log4j.appender.info.encoding=UTF-8
log4j.appender.info.ImmediateFlush=true
log4j.appender.info.layout=org.apache.log4j.PatternLayout
log4j.appender.info.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss SSS}|%5p|%F.%M:%L|%m%n
#WARN 日志
log4j.appender.warn=com.platform.log4j.GradeLogDailyRollingFileAppender
log4j.appender.warn.File=../logs/warn/warn.log
log4j.appender.warn.datePattern='.'yyyy-MM-dd'.log'
log4j.appender.warn.append=true
log4j.appender.warn.Threshold=WARN
log4j.appender.warn.encoding=UTF-8
log4j.appender.warn.ImmediateFlush=true
log4j.appender.warn.layout=org.apache.log4j.PatternLayout
log4j.appender.warn.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss SSS}|%5p|%F.%M:%L|%m%n
#ERROR 日志
log4j.appender.error=com.platform.log4j.GradeLogDailyRollingFileAppender
log4j.appender.error.File=../logs/error/error.log
log4j.appender.error.datePattern='.'yyyy-MM-dd'.log'
log4j.appender.error.append=true
log4j.appender.error.Threshold=ERROR
log4j.appender.error.encoding=UTF-8
log4j.appender.error.ImmediateFlush=true
log4j.appender.error.layout=org.apache.log4j.PatternLayout
log4j.appender.error.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss SSS}|%5p|%F.%M:%L|%m%n
```

6. junit 单元测试

该项目已集成基于 spring 的 junit 单元测试，方便开发人员进行功能测试。

6.1. 单元测试代码结构



6.2. 实现原理

```
/**
 * 基于 spring 的单元测试基类
```

```
 *
 * @author lipengjun
 * @email 939961241@qq.com
 * @date 2018-07-09 10:06:23
 */
@ContextConfiguration(locations = {"classpath:applicationContext-test.xml"})
public class BaseSpringTestCase extends AbstractJUnit4SpringContextTests {

    protected Logger logger = LoggerFactory.getLogger(this.getClass());

    /**
     * 获取 Logger
     */
    public Logger getLogger() {
        return logger;
    }
}
```

使用@contextConfiguration 注解引入 spring 的配置文件，如果有多个配置文件用逗号隔开。

6.3. 使用方法

继承 BaseSpringTestCase 类就可以开发测试类。用于测试的配置文件已经扫描项目中的 bean，所以可以直接使用项目中的 service 进行测试，demo 如下：

```
public class TestSysUserController extends BaseSpringTestCase {
    @Autowired
    TestSysUserService testSysUserService;
    @Autowired
    SysUserService sysUserService;
    private Logger logger = getLogger();

    /**
     * 使用测试类
     */
    @Test
    public void queryTestSysUserList() {
        Map params = new HashMap();
        List<SysUserEntity> list = testSysUserService.queryList(params);
        if (list != null && list.size() != 0) {
            for (SysUserEntity userEntity : list) {
                logger.info("username: " + userEntity.getUsername() + "; mobile: " + userEntity.getMobile());
            }
        }
    }

    /**
     * 使用项目中的 service
     */
    @Test
    public void querySysUserList() {
        Map params = new HashMap();
        List<SysUserEntity> list = sysUserService.queryList(params);
        if (list != null && list.size() != 0) {
            for (SysUserEntity userEntity : list) {
                logger.info("username: " + userEntity.getUsername() + "; mobile: " + userEntity.getMobile());
            }
        }
    }
}
```

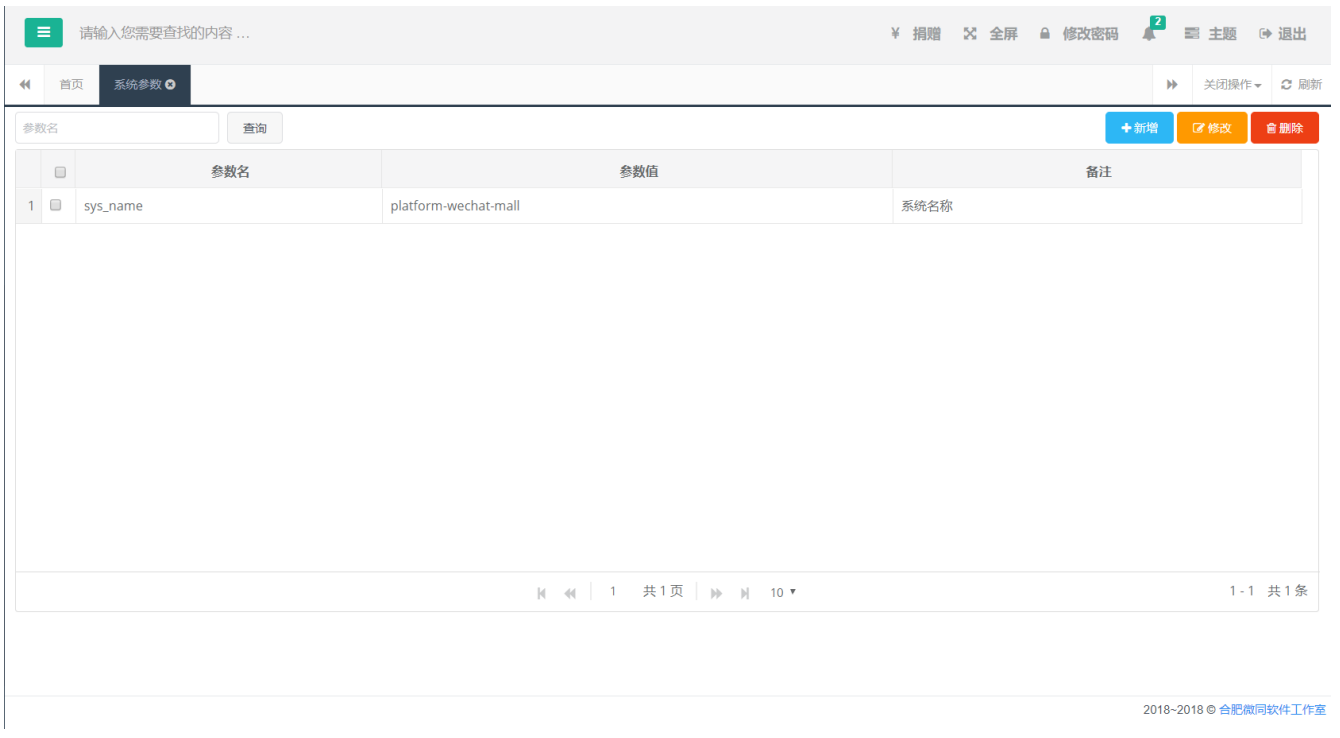
7. 前端源码分析

前端是用 hplus 风格主题。

7.1. 页面源码分析

下面分析系统参数相关前端代码，包括数据列表、查询、新增、编辑、删除等功能，掌握了这部分内容，再开发其他前端页面，

就会得心应手，界面如下：



7.1.1. 列表查询

下面的这段代码，用来展示数据列表的，定义了数据列表 ID 为 jqGrid，分页 ID 为 jqGridPager

```
<table id="jqGrid"></table>
<div id="jqGridPager"></div>
```

下面的代码，就是具体的数据列表 Grid

```
$(function () {
    $("#jqGrid").Grid({
        url: '../sys/config/list',
        colModel: [
            {label: 'ID', name: 'id', key: true, hidden: true},
            {label: '参数名', name: 'key', index: 'key', width: 60},
            {label: '参数值', name: 'value', index: 'value', width: 100},
            {label: '备注', name: 'remark', index: 'remark', width: 80}
        ]
    });
});
```

上面这些代码，就实现了数据列表的功能，下面就来看看，查询的实现，页面代码如下

```
<div v-show="showList">
    <Row :gutter="16">
        <i-col span="4">
            <i-input v-model="q.key" @on-enter="query" placeholder="参数名"/>
        </i-col>
        <i-button @click="query">查询</i-button>
    </Row>
</div>
```

我们定义了查询参数 key，点击查询，就会调用 vue 的 query 方法，其中一定要定义 key 变量，不然页面会报错，代码如下：

```
var vm = new Vue({
    el: '#rrapp',
    data: {
        q: {key: null},
        showList: true,
        title: null,
        config: {},
        ruleValidate: {
            key: [
                {required: true, message: '参数名不能为空', trigger: 'blur'}
            ],
            value: [
                {required: true, message: '参数值不能为空', trigger: 'blur'}
            ]
```

```
        }
    },
    methods: {
        query: function () {
            vm.reload();
        },
        reload: function (event) {
            vm.showList = true;
            var page = $("#jqGrid").jqGrid('getGridParam', 'page');
            $("#jqGrid").jqGrid('setGridParam', {
                postData: {'key': vm.q.key},
                page: page
            }).trigger("reloadGrid");
        },
        reloadSearch: function () {
            vm.q = {
                confKey: ''
            }
            vm.reload();
        }
    }
});
```

7.1.2. 新增、修改、删除功能

```
<div id="rrapp" v-cloak>
    <div v-show="showList">
        <Row :gutter="16">
            <div class="buttons-group">
                <i-button type="info" @click="add"><i class="fa fa-plus"></i> 新增</i-button>
                <i-button type="warning" @click="update"><i class="fa fa-pencil-square-o"></i> 修改</i-button>
                <i-button type="error" @click="del"><i class="fa fa-trash-o"></i> 删除</i-button>
            </div>
        </Row>
        <table id="jqGrid"></table>
        <div id="jqGridPager"></div>
    </div>
    <Card v-show="!showList">
        <p slot="title">{{title}}</p>
        <i-form ref="formValidate" :model="config" :rules="ruleValidate" :label-width="80">
            <Form-item label="参数名" prop="key">
                <i-input v-model="config.key" placeholder="参数名"/>
            </Form-item>
            <Form-item label="参数值" prop="value">
                <i-input v-model="config.value" placeholder="参数值"/>
            </Form-item>
            <Form-item label="备注" prop="remark">
                <i-input type="textarea" v-model="config.remark" placeholder="备注"/>
            </Form-item>
            <Form-item>
                <i-button type="primary" @click="handleSubmit('formValidate')">提交</i-button>
                <i-button type="warning" @click="reload" style="margin-left: 8px">返回</i-button>
                <i-button type="ghost" @click="handleReset('formValidate')" style="margin-left: 8px">重置</i-button>
            </Form-item>
        </i-form>
    </Card>
</div>
var vm = new Vue({
    el: '#rrapp',
    data: {
        q: {
            key: null
        },
        showList: true,
        title: null,
        config: {},
        ruleValidate: {
```

```
            key: [
                {required: true, message: '参数名不能为空', trigger: 'blur'}
            ],
            value: [
                {required: true, message: '参数值不能为空', trigger: 'blur'}
            ]
        }
    },
    methods: {
        query: function () {
            vm.reload();
        },
        add: function () {
            vm.showList = false;
            vm.title = "新增";
            vm.config = {};
        },
        update: function () {
            var id = getSelectedRow("#jqGrid");
            if (id == null) {
                return;
            }
            Ajax.request({
                url: "../sys/config/info/" + id,
                async: true,
                successCallback: function (r) {
                    vm.showList = false;
                    vm.title = "修改";
                    vm.config = r.config;
                }
            });
        },
        del: function (event) {
            var ids = getSelectedRows("#jqGrid");
            if (ids == null) {
                return;
            }
            confirm('确定要删除选中的记录？', function () {
                Ajax.request({
                    url: "../sys/config/delete",
                    params: JSON.stringify(ids),
                    contentType: "application/json",
                    type: 'POST',
                    successCallback: function () {
                        alert('操作成功', function (index) {
                            vm.reload();
                        });
                    }
                });
            });
        },
        saveOrUpdate: function (event) {
            var url = vm.config.id == null ? "../sys/config/save" : "../sys/config/update";
            Ajax.request({
                url: url,
                params: JSON.stringify(vm.config),
                contentType: "application/json",
                type: 'POST',
                successCallback: function () {
                    alert('操作成功', function (index) {
                        vm.reload();
                    });
                }
            });
        },
        handleSubmit: function (name) {
            handleSubmitValidate(this, name, function () {
```

```
                vm.saveOrUpdate()
            });
        },
        handleReset: function (name) {
            handleResetForm(this, name);
        }
    }
});
```

7.1.3. 表单验证

```
<i-form ref="formValidate" :model="user" :rules="ruleValidate" :label-width="80">
```

属性	说明	类型	默认值
model	表单数据对象	Object	-
rules	表单验证规则，具体配置查看 async-validator	Object	-
inline	是否开启行内表单模式	Boolean	false
label-position	表单域标签的位置，可选值为 left、right、top	String	right
label-width	表单域标签的宽度，所有的 FormItem 都会继承 Form 组件的 label-width 的值	Number	-
show-message	是否显示校验错误信息	Boolean	true
autocomplete	原生的 autocomplete 属性，可选值为 off 或 on	String	off

```
<Form-item label="用户名" prop="username">
```

属性	说明	类型	默认值
prop	对应表单域 model 里的字段	String	-
label	标签文本	String	-
label-width	表单域标签的的宽度	Number	-
label-for	指定原生的 label 标签的 for 属性，配合控件的 element-id 属性，可以点击 label 时聚焦控件。	String	-
required	是否必填，如不设置，则会根据校验规则自动生成	Boolean	-
rules	表单验证规则	Object | Array	-
error	表单域验证错误信息，设置该值会使表单验证状态变为 error，并显示该错误信息	String	-
show-message	是否显示校验错误信息		

```
var vm = new Vue({
    el: '#rrapp',
    data: {
        q: {
            username: null
        },
        showList: true,
        title: null,
        roleList: {},
        user: {
            status: 1,
            deptName: '',
            roleIdList: []
        },
        ruleValidate: {
            username: [
                {required: true, message: '姓名不能为空', trigger: 'blur'}
            ],
            email: [
                {required: true, message: '邮箱不能为空', trigger: 'blur'},
                {type: 'email', message: '邮箱格式不正确', trigger: 'blur'}
            ],
            mobile: [
                {required: true, message: '手机号不能为空', trigger: 'blur'}
            ]
        }
```

7.1.4. 自定义字段验证

请参照以下例子

```
const validateTel = (rule, value, callback) => {
    //可以使用正则匹配验证
    if (!value) {
        callback(new Error('名称不能为空'));
    } else {
        callback();
    }
};
var vm = new Vue({
    el: '#rrapp',
    data: {
        showList: true,
        title: null,
        macro: {type: 0, status: 1},
        ruleValidate: {
            name: [
                {required: true, validator: validateTel, trigger: 'blur'}
            ]
        },
        q: {
            name: ''
        },
        macros: []
    },
    methods: {
        query: function () {
            vm.reload();
        }
```

7.2. 富文本编辑器 wysiwyg-editor

WYSIWYG HTML 编辑器是一款有史以来最强大的 JavaScript 富文本编辑器之一。它采用了最新的技术，并利用 jQuery 和 HTML5 的巨大优势，创造了出色的编辑体验。拥有非常多的示例让你轻松集成，让你的用户爱上它清晰的设计。它能和 Ruby On Rails，Django，Angular.js，Meteor，Symgony.CakePHP 等集成，具有如下特点。

- 微小 - 只需添加您需要的插件(30+ 官方插件)
- 客户端框架集成
- 可以向如 PHP, Node.JS, .NET, Java, 和 Python 提供服务端开发工具包
- 代码注释精美
- 在线文档更新
- 简单可扩展- 良好的插件注释使你更容易使用和开发自己的插件
- 良好的维护 - 持续更新
- 很好的支持 - 帮助中心

7.2.1. 项目中的使用

- 导入到项目中

header.html

```
<!--富文本-->
<link rel="stylesheet" href='m/statics/plugins/froala_editor/css/froala_editor.min.css'/>
<script src='/statics/plugins/froala_editor/js/froala_editor.min.js'></script>
```

- 初始化

```
$(function () {
    $('#goodsDesc').editable({
        inlineMode: false,
```

```
        alwaysBlank: true,
        height: '500px', //高度
        minHeight: '200px',
        language: "zh_cn",
        spellcheck: false,
        plainPaste: true,
        enableScript: false,
        imageButtons: ["floatImageLeft", "floatImageNone", "floatImageRight", "linkImage", "replaceImage", "removeImage"],
        allowedImageTypes: ["jpeg", "jpg", "png", "gif"],
        imageUploadURL: '../sys/oss/upload',
        imageUploadParams: {id: "edit"},
        imagesLoadURL: '../sys/oss/queryAll'
    })
});
```

- 赋值

```
$('#goodsDesc').editable('setHTML', '');
```

- 获取文本内容

```
$('#goodsDesc').editable('getHTML');
```

8. 使用 postman 对接口调试

在日常开发中，好的工具往往起到事半功倍的效果，尤其是在这个服务接口满天飞的时代，为了更加方便开发人员对接口的调试，这里推介大家使用 postman 这款工具。

8.1. 下载安装 postman

地址：https://www.getpostman.com/apps (文档最后一节有百度网盘地址)

8.2. 获取 token

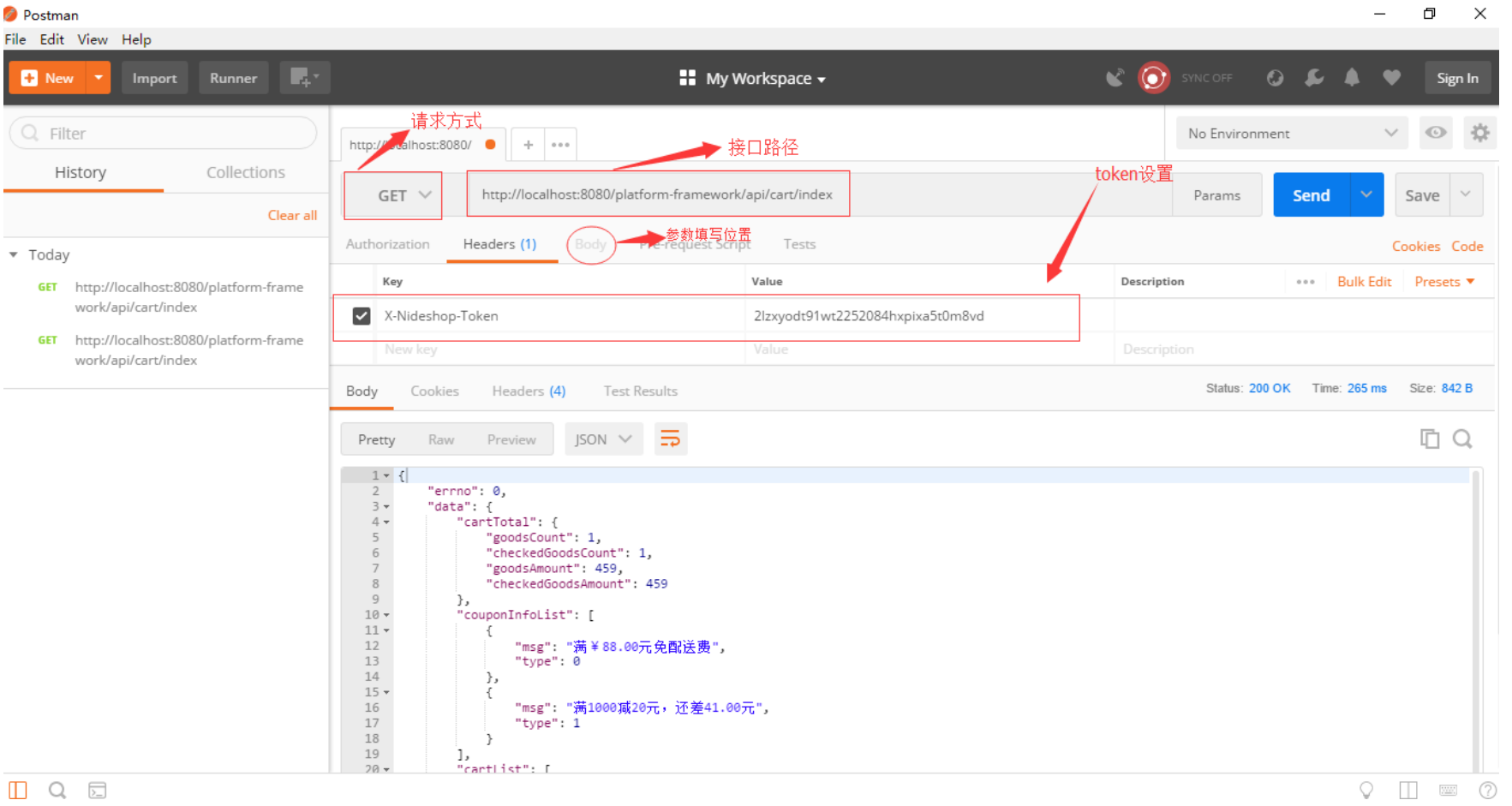
进入微信开发者工具，进入小程序 -> 点击“我的”栏目页面 -> 点击最上面 “Hi，游客”进行登陆。登陆成功后，可以在 IDE 的控制台可以看到 ”INFO|ApiBaseAction.java.toResponsSuccess:....” 中有 token 信息，或者直接从数据库的“tb_token”表中复制该 token 信息保存备用。

8.3. 配置 postman 调试接口

- 打开 postman，找到 Headers，添加一栏参数：Key 的位置填写：X-Nideshop-Token，value 的位置填写上面 8.2 获取到的 token 值。
- 在 postman 地址栏首先选择接口的请求方式，在地址栏填写要调用的接口地址，在 body 中填写需要的参数并正确填写。
- 点击 Send.

8.4. 请求用户购物车示例

这里是用 postman 请求用户购物车的一个示例：查询用户购物车首先需要用户处于登陆状态，这里在 Headers 中已经将用户 token 保存，在向后台发送请求的时候，会使用 token 对用户状态做校验来决定是否可以访问业务数据接口。



9. 小程序介绍

9.1. 产品介绍及功能介绍

微信小程序是一种全新的连接用户与服务的方式，它可以在微信内被便捷地获取和传播，同时具有出色的使用体验。

9.2. 小程序注册

查看注册详情

9.3. 小程序申请微信认证

政府、媒体、其他组织类型帐号，必须通过微信认证验证主体身份。企业类型帐号，可以根据需要确定是否申请微信认证。已认证帐号可使用微信支付权限。

个人类型帐号暂不支持微信认证。

认证入口：登录小程序—设置—基本设置—微信认证—详情

设置

基本设置　开发设置

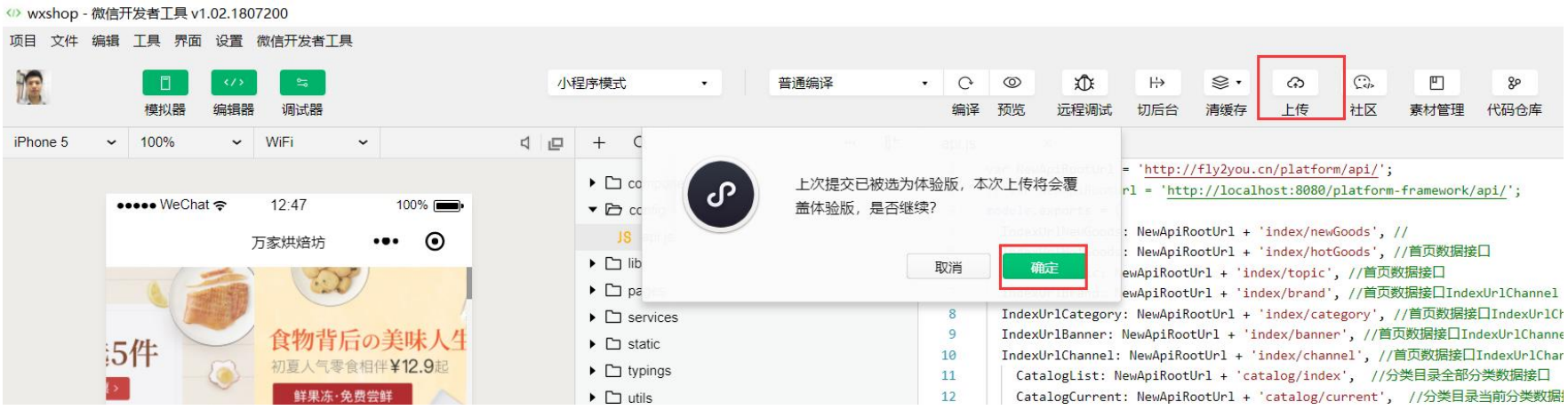
基本信息		说明	操作
小程序名称		一年内可申请修改1次 本年还可修改1次	修改
小程序头像		一个月内可申请修改5次 本月还可修改5次	修改头像
二维码			下载更多尺寸
介绍	仅用于测试	一个月内可申请5次修改 本月还可修改5次	修改
微信认证	未认证		详情
主体信息		企业	详情
服务范围	出行与交通 > 市内公交	一个月内可申请修改1次 本月还可修改1次	详情

9.4. 小程序申请微信支付

已认证的小程序可申请微信支付。

9.5. 代码审核与发布

9.5.1. 微信开发工具上传



9.5.2. 提交审核

登录微信公众平台小程序，进入开发管理，开发版本中展示已上传的代码，管理员可提交审核或是删除代码。

9.5.3. 代码发布

代码审核通过，需要开发者手动点击发布，小程序才会发布到线上提供服务。

9.6. 小程序开发 API

https://developers.weixin.qq.com/miniprogram/dev/api/

10. 生成环境部署

10.1. 打包

```
mvn package -P prod
```

```
[INFO] --- maven-war-plugin:2.6:war (default-war) @ platform-framework ---
[INFO] Packaging webapp
[INFO] Assembling webapp [platform-framework] in [E:\code\weitong\platform-wechat-buss\platform-framework\target\pwm]
[INFO] Processing war project
[INFO] Copying webapp resources [E:\code\weitong\platform-wechat-buss\platform-framework\src\main\webapp]
[INFO] Processing overlay [ id com.platform:platform-admin]
[INFO] Webapp assembled in [7271 msecs]
[INFO] Building war: E:\code\weitong\platform-wechat-buss\platform-framework\target\pwm.war
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building platform-wechat-buss 3.1.0
[INFO] ------------------------------------------------------------------------
Downloading: http://maven.aliyun.com/nexus/content/groups/public/org/apache/maven/plugins/maven-compiler-plugin/maven-metadata.xml
Downloaded: http://maven.aliyun.com/nexus/content/groups/public/org/apache/maven/plugins/maven-compiler-plugin/maven-metadata.xml (2 KB at 0.5 KB/sec)
Downloading: http://maven.aliyun.com/nexus/content/groups/public/org/apache/maven/plugins/maven-resources-plugin/maven-metadata.xml
Downloaded: http://maven.aliyun.com/nexus/content/groups/public/org/apache/maven/plugins/maven-resources-plugin/maven-metadata.xml (844 B at 0.9 KB/sec)
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO]
[INFO] platform-common ..................................... SUCCESS [  7.784 s]
[INFO] platform-schedule ................................... SUCCESS [  0.577 s]
[INFO] platform-gen ........................................ SUCCESS [  0.563 s]
[INFO] platform-api ........................................ SUCCESS [  3.022 s]
[INFO] platform-admin ...................................... SUCCESS [ 11.653 s]
[INFO] platform-framework .................................. SUCCESS [ 11.270 s]
[INFO] platform-wechat-buss ................................ SUCCESS [  6.041 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 41.199 s
[INFO] Finished at: 2018-09-25T17:16:56+08:00
[INFO] Final Memory: 37M/208M
[INFO] ------------------------------------------------------------------------

E:\code\weitong\platform-wechat-buss>
```

拷贝 pwm.war 到云服务器 Tomcat 的 webapps 目录下。

10.2. 启动 Tomcat

```
[root@iZ2ze6jga1y3xg3vatxz9mZ ~]# cd /usr/local/tomcat/tomcat8.0.33/bin/
[root@iZ2ze6jga1y3xg3vatxz9mZ bin]# ./startup.sh
```

启动成功之后访问 http://ip:port/pwm

10.3. 查看实时日志

```
[root@iZ2ze6jga1y3xg3vatxz9mZ bin]# cd /usr/local/tomcat/tomcat8.0.33/logs/
[root@iZ2ze6jga1y3xg3vatxz9mZ logs]# tail -222f catalina.out
```

```
sys_menu m WHERE 1=1 order by m.order_num asc,domain_id
==> Parameters:
<==      Total: 58
==>  Preparing: select m.*,(select p.name from sys_menu p where p.menu_id = m.parent_id) as parentName, (select d.domain_name from sys_domain d where d.id = m.domain_id) domain_name from
sys_menu m WHERE 1=1 order by m.order_num asc,domain_id
==> Parameters:
<==      Total: 58
==>  Preparing: select m.*,(select p.name from sys_menu p where p.menu_id = m.parent_id) as parentName, (select d.domain_name from sys_domain d where d.id = m.domain_id) domain_name from
sys_menu m WHERE 1=1 order by m.order_num asc,domain_id
==> Parameters:
<==      Total: 58
==>  Preparing: select m.*,(select p.name from sys_menu p where p.menu_id = m.parent_id) as parentName, (select d.domain_name from sys_domain d where d.id = m.domain_id) domain_name from
sys_menu m WHERE 1=1 order by m.order_num asc,domain_id
==> Parameters:
<==      Total: 58
==>  Preparing: select m.*,(select p.name from sys_menu p where p.menu_id = m.parent_id) as parentName, (select d.domain_name from sys_domain d where d.id = m.domain_id) domain_name from
sys_menu m WHERE 1=1 order by m.order_num asc,domain_id
==> Parameters:
<==      Total: 58
==>  Preparing: select * from schedule_job order by job_id desc limit ?, ?
==> Parameters: 0(Integer), 10(Integer)
<==      Total: 1
==>  Preparing: select count(1) from schedule_job
==> Parameters:
<==      Total: 1
2018-07-29 23:13:35 650| INFO|LogInterceptor.java.afterCompletion:68|本次请求耗时：19毫秒；  请求路径=/sys/schedule/list 来源IP=223.73.212.94 操作人=admin 请求参数={nd=1532877212587, limit
=10, page=1, _search=false, sidx=, order=asc}
2018-07-29 23:30:00 247| INFO|ScheduleJob.java.executeInternal:50|任务准备执行，任务ID：2
2018-07-29 23:30:00 247| INFO|TestTask.java.test2:42|我是不带参数的test2方法，正在被执行
2018-07-29 23:30:00 247| INFO|ScheduleJob.java.executeInternal:63|任务执行完毕，任务ID：2  总共耗时：0毫秒
==>  Preparing: insert into schedule_job_log ( `job_id`, `bean_name`, `method_name`, `params`, `status`, `error`, `times`, `create_time` ) values ( ?, ?, ?, ?, ?, ?, ?, ? )
==> Parameters: 2(Long), testTask(String), test2(String), null, 0(Integer), null, 0(Integer), 2018-07-29 23:30:00.247(Timestamp)
<==    Updates: 1
2018-07-29 23:53:52 003| INFO|AbstractValidatingSessionManager.java.validateSessions:275|Validating all active sessions...
2018-07-29 23:53:52 004| INFO|AbstractValidatingSessionManager.java.validateSessions:308|Finished session validation.  No sessions were stopped.
2018-07-30 00:00:00 097| INFO|ScheduleJob.java.executeInternal:50|任务准备执行，任务ID：2
2018-07-30 00:00:00 110| INFO|TestTask.java.test2:42|我是不带参数的test2方法，正在被执行
2018-07-30 00:00:00 112| INFO|ScheduleJob.java.executeInternal:63|任务执行完毕，任务ID：2  总共耗时：15毫秒
==>  Preparing: insert into schedule_job_log ( `job_id`, `bean_name`, `method_name`, `params`, `status`, `error`, `times`, `create_time` ) values ( ?, ?, ?, ?, ?, ?, ?, ? )
==> Parameters: 2(Long), testTask(String), test2(String), null, 0(Integer), null, 15(Integer), 2018-07-30 00:00:00.097(Timestamp)
<==    Updates: 1
```

11. 代码生成工具-IDEA 插件使用

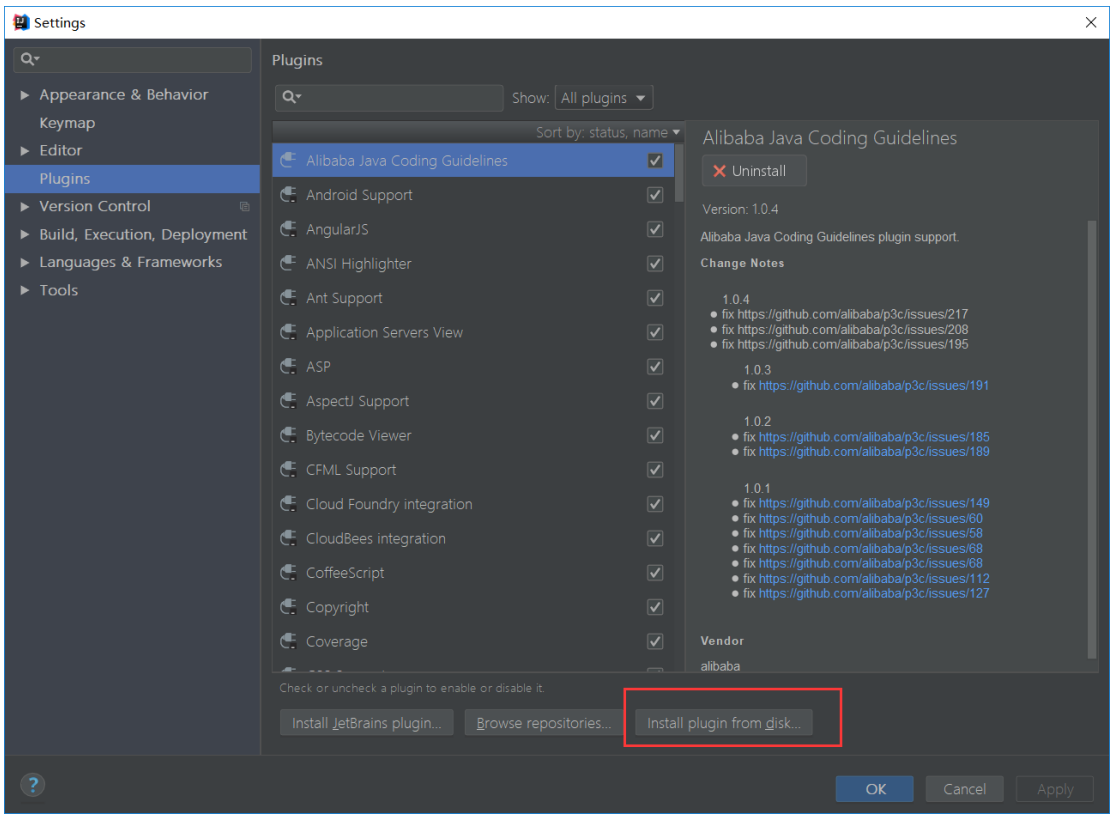
11.1. 下载

https://gitee.com/fuyang_lipengjun/platform-gen

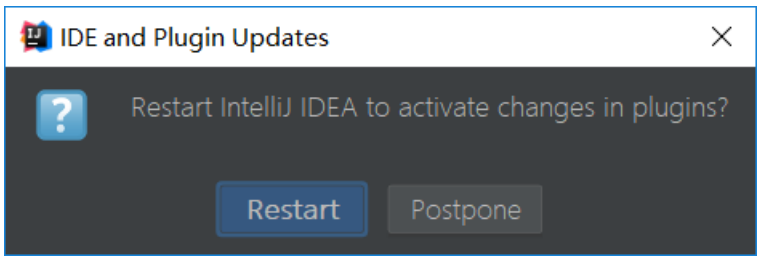
下载的 zip 解压缩到电脑

11.2. 安装

File->Settings->Plugins

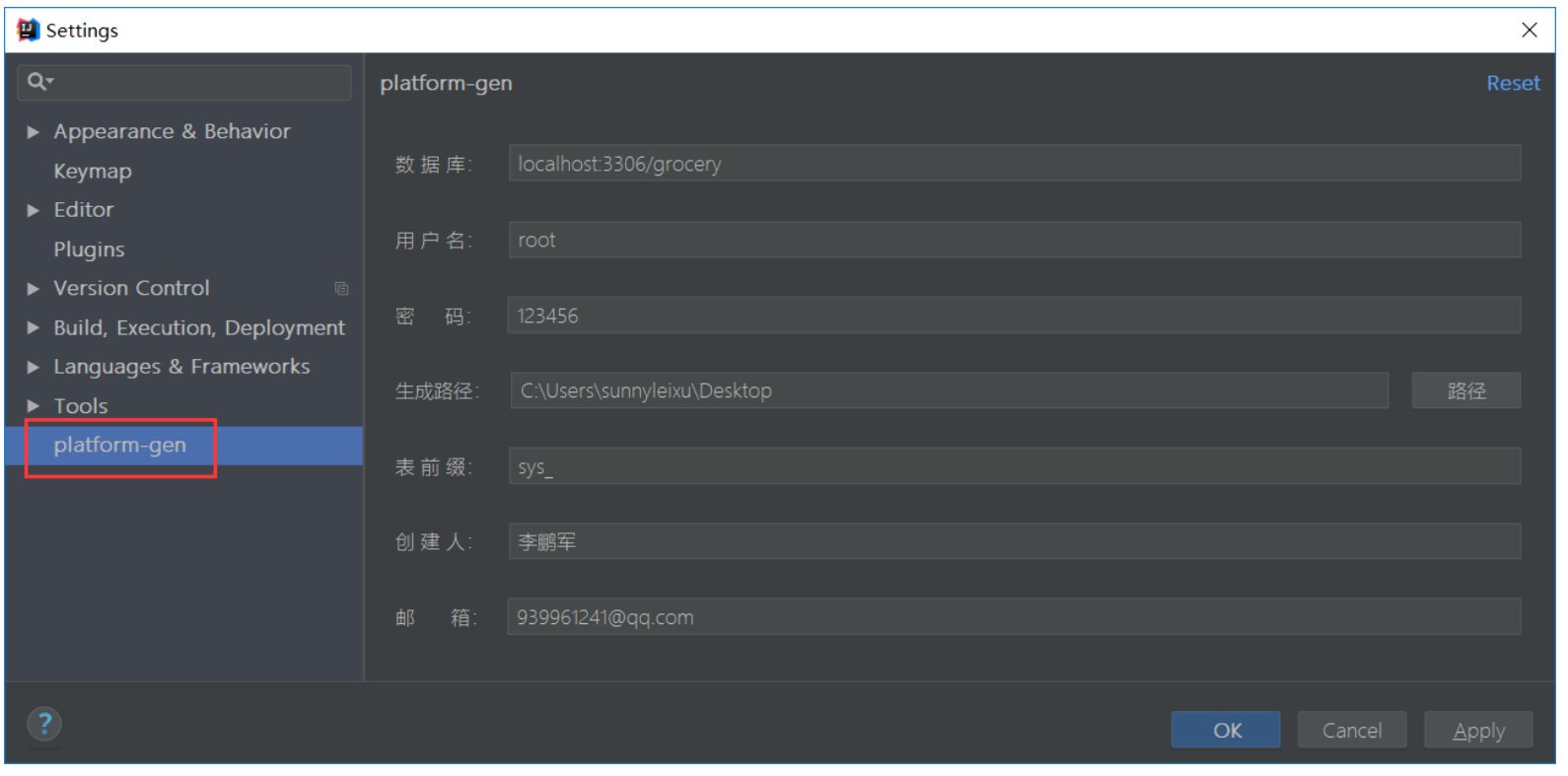


重启

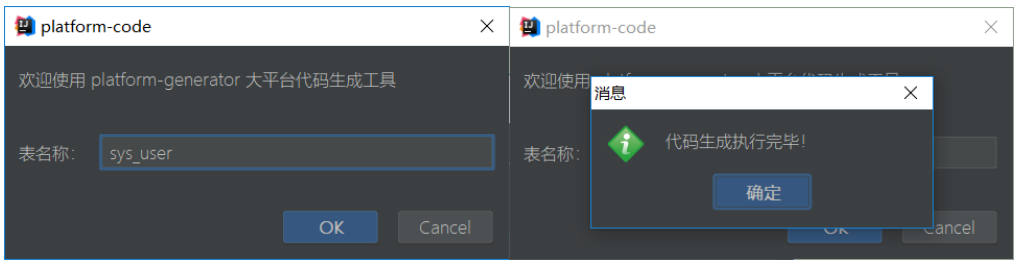


11.3. 使用

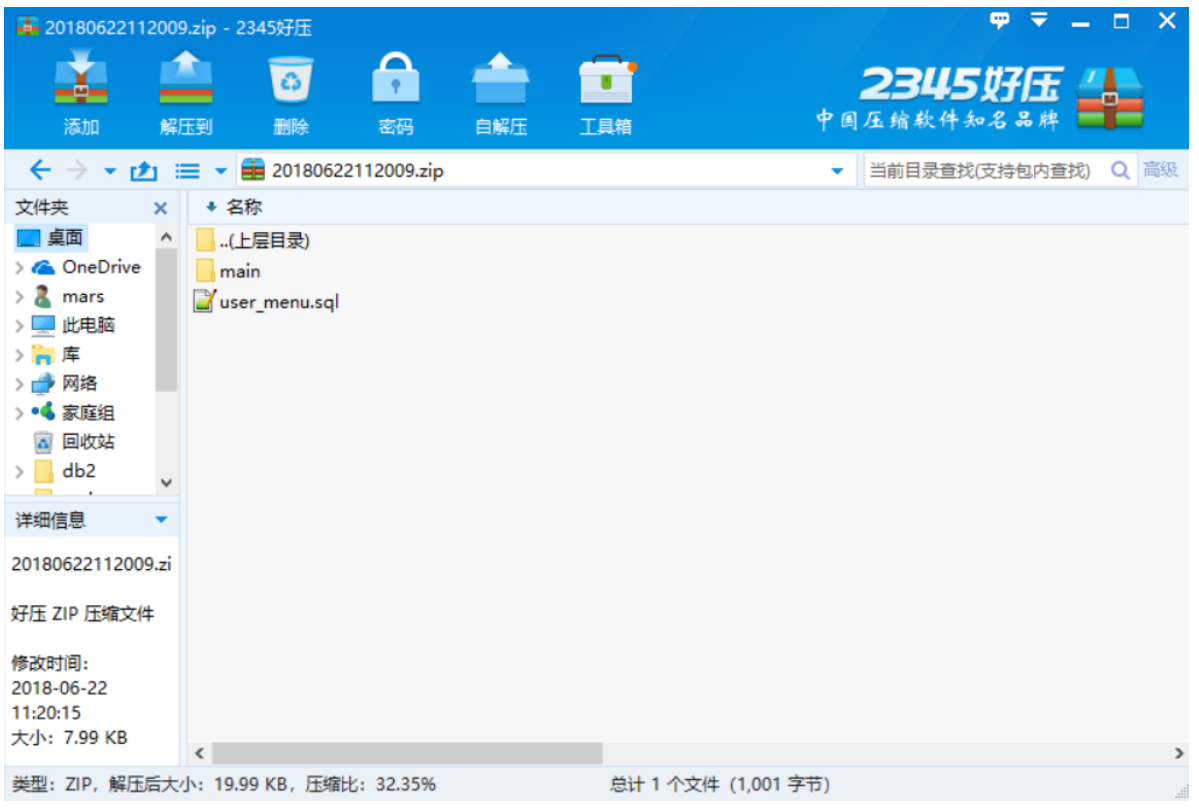
File->Settings->platform-code



使用快捷键 ctrl+alt+u



代码生成完毕



12. 常见问题

12.1. 开发阶段需要注意的问题

12.1.1.关于微信支付回调的问题

在本地开发的环境中，微信支付成功后是无法正常调用支付回调的。这是因为当我们支付时是通知微信，而支付成功之后的回调是微信发起的，所以必须是线上环境才能正常调用支付回调！详情请参照微信开发文档之支付结果通知

12.1.2.关于图片上传的问题

图片、文件上传，使用的是七牛、阿里云、腾讯云的存储服务，不能上传到本地服务器。所以上传图片之前一定要先配置云存储。

12.1.3.关于 404 问题

不明白为什么会有人问这个问题，我还是说一下吧！后台管理系统项目的访问路径是 localhost:port/contextPath

API 的请求路径是 localhost:port/contextPath/api/

12.1.4.为什么要设计 platform-framework 模块

- 此模块包含所有项目，只需要部署 platform-framework.war 即可启动整个项目
- 目前后台管理系统包含两个大模块，platform-admin 和 platform-shop，当有更多业务系统时候容易扩展。
- 多个团队协作开发，可以更有效的防止核心代码泄露。假如 A 团队只负责开发 platform-admin，就可以在给代码权限时候屏蔽 platform-shop，然后只需要修改根目录下 pom.xml、platform-framework 目录下 pom.xml 和 platform-admin 目录下 pom.xml，将 platform-shop 依赖删除。

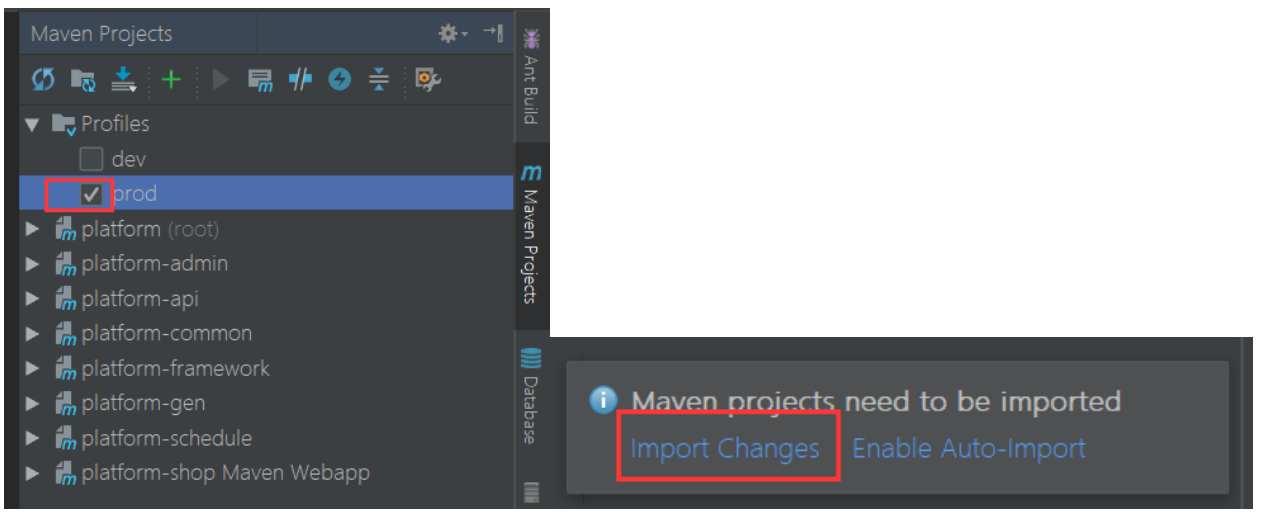
12.1.5.为什么可以’直接访问’ WEB-INF 目录下的 html

从浏览器的请求中可以看到我们是直接请求的 html 文件：http://localhost/platform-framework/sys/main.html，给我们产生一种是直接请求 html 文件的错觉，实际上这次请求是通过 SysPageController.java 返回的，具体的实现代码如下：

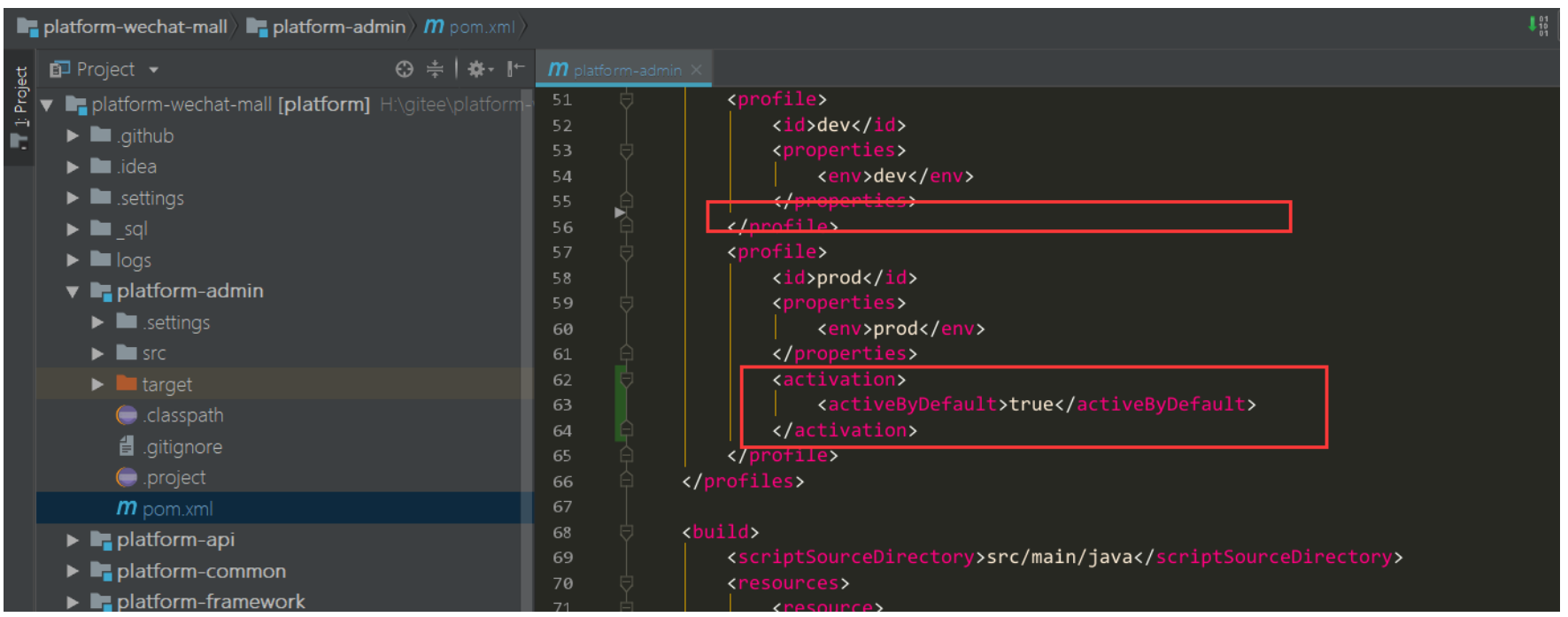
```
@Controller
public class SysPageController {

    /**
     * 视图路径
     *
     * @param module 模块
     * @param url    url
     * @return 页面视图路径
     */
    @RequestMapping("{module}/{url}.html")
    public String page(@PathVariable("module") String module, @PathVariable("url") String url) {
        return module + "/" + url + ".html";
    }

}
```

12.1.6.dev 和 prod 如何切换

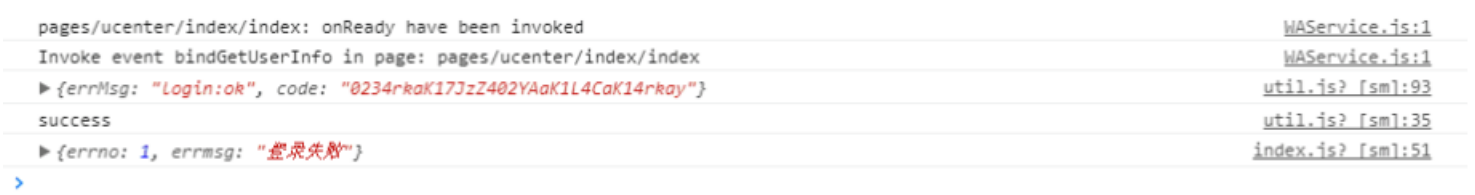
◆ 使用 IDEA



◆ 修改 admin 模块 pom，如下图所示



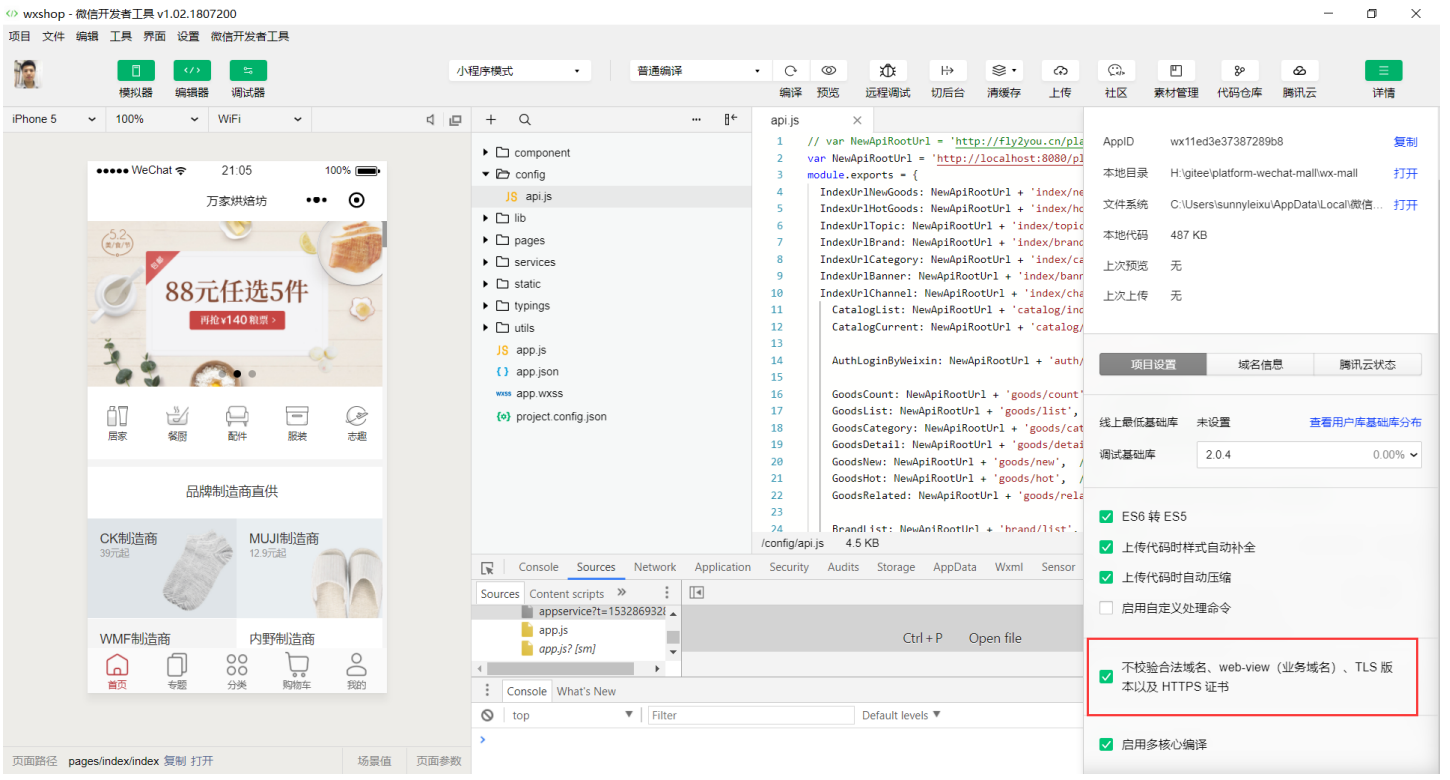
12.2. 小程序登录失败



解决方法

◆ 更新至最新代码

◆ 不校验合法域名、web-view（业务域名）、TLS 版本以及 HTTPS 证书

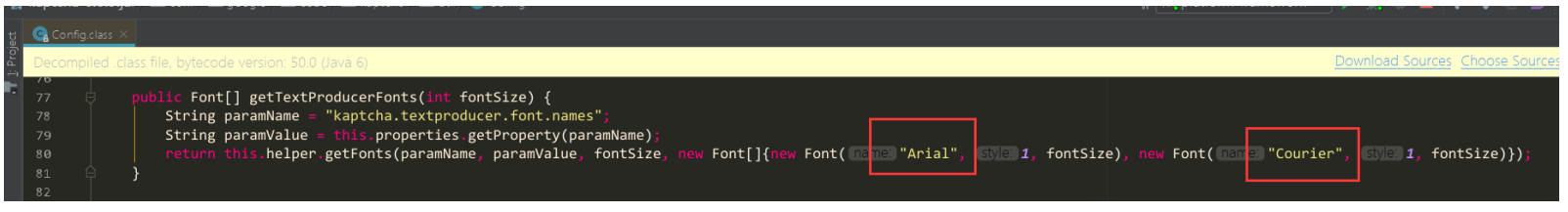


◆ 小程序端 AppId 和后台配置的 AppId 保持一致

◆ 删除 nide_shop_user、tb_token 表中对应登录用户的数据。

12.3. 登录验证码无法正常显示

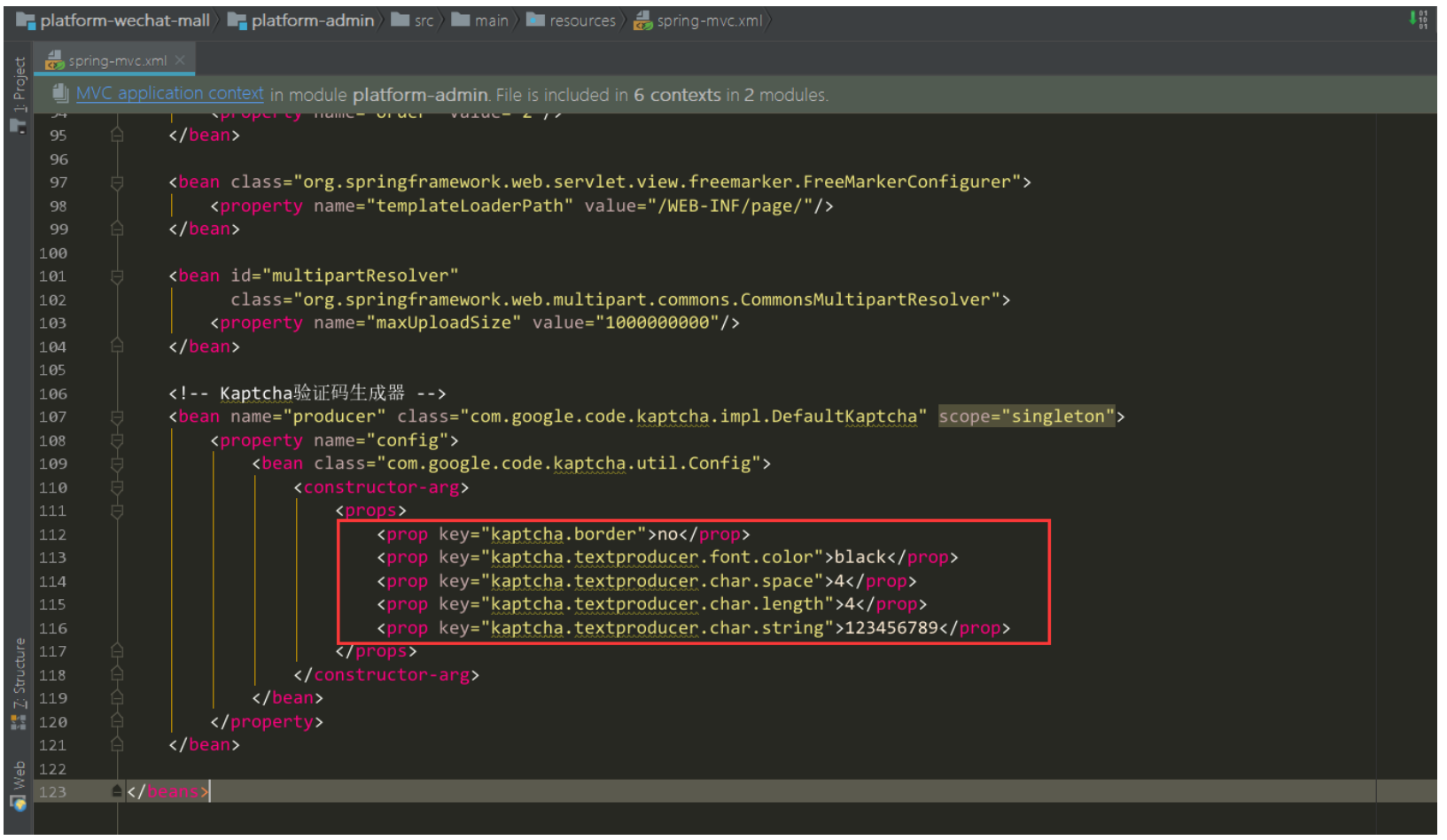
解决方法：出现这种情况是因为服务器缺少相应的字体库，有两种解决方案。分析源码得知，kaptcha 默认字体为 Arial、Courier。



解决方法

◆ 在服务器安装该字体库

◆ 修改默认的字体



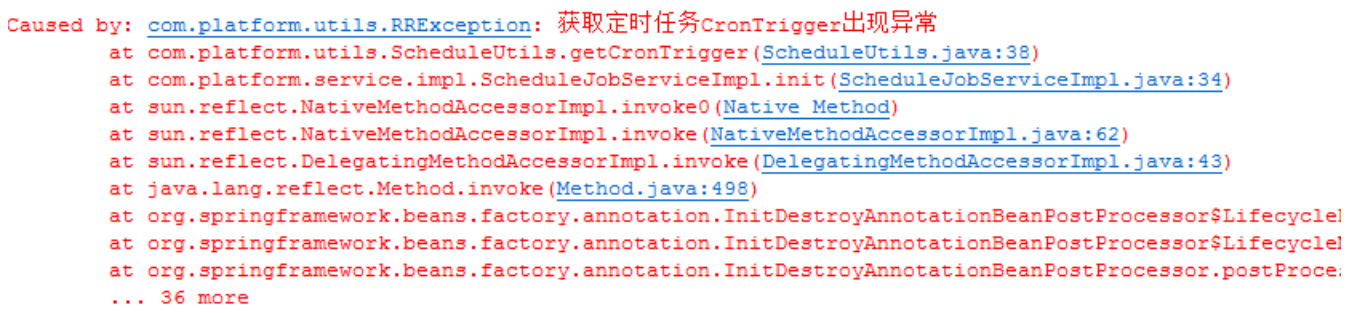
12.4. Error creating bean with name ‘cacheUtil’

解决方法：本地启动 redis 服务，确保 redis 服务是正常运行状态。

12.5. Failed to load image

解决方法：该图片资源不存在。这个错误对程序没有影响。

12.6. Error creating bean with name ‘scheduleJobController’ 获取定时任务 CronTrigger 出现异常



解决方法：按照顺序清空以下表数据 qrtz_cron_triggers、qrtz_locks、qrtz_scheduler_state、qrtz_triggers、qrtz_job_details

12.7. 通过 Nginx 代理之后验证码已失效

解决方法：

- 1、将 platform-framework.war 改为 ROOT.war 放入 Tomcat 的 webapps 下

◆ 2、nginx 配置如下，这两处保持一致

```
location /platform-framework {
    #root   html;
        proxy_pass  http://fly2you.cn:8083/platform-framework;
        proxy_set_header  Host             $host;
        proxy_set_header  X-Real-IP        $remote_addr;
        proxy_set_header  X-Forwarded-For  $proxy_add_x_forwarded_for;
}
```

◆ 设置 proxy_cookie_path

```
location /pwm {
        proxy_pass http://127.0.0.1:8080/platform-framework;
        proxy_connect_timeout 600;
        proxy_read_timeout 600;
        proxy_cookie_path /platform-framework /pwm
}
```

13. 常用工具下载

链接：https://pan.baidu.com/s/17MUNxkPVuMihMp-kY138rw 密码：6c3w

apache-maven-3.3.9.rar	2018-07-30 13:34	8.12MB
apache-tomcat-8.0.30-windows-x64.zip	2018-07-30 13:34	10.02MB
apache-tomcat-8.0.33.tar.gz	2018-07-30 13:34	8.82MB
eclipse-jee-mars-R-win32-x86_64.zip	2018-07-30 13:34	269.44MB
Git-2.7.2-32-bit_setup.1457942412.exe	2018-07-30 13:34	29.33MB
ideaIU-2017.2.2.exe	2018-07-30 13:34	505.62MB
jdk-8u102-windows-x64.exe	2018-07-30 13:34	194.68MB
jdk-8u151-linux-x64.tar.gz	2018-07-30 13:34	180.95MB
mysql-5.7.17.msi	2018-07-30 13:34	386.64MB
Navicat Premium_11.2.7简体中文版.zip	2018-07-30 13:34	68.48MB
Postman-win64-6.2.4-Setup.exe	2018-08-16 13:46	64.80MB
redis-4.0.1.tar.gz	2018-08-07 11:00	1.63MB
redis-desktop-manager.rar	2018-07-30 13:34	37.52MB
Redis-x64-3.2.100.msi	2018-07-30 13:34	5.80MB
TortoiseSVN-1.8.4.24972-x64-svn-1.8.5.msi	2018-07-30 13:34	18.46MB
wechat_devtools_1.02.1807200_x64.exe	2018-07-30 13:34	78.04MB
Xftp-6.0.0083p.exe	2018-07-30 13:34	25.44MB
Xshell-6.0.0089p.exe	2018-07-30 13:34	36.49MB