Guns 技术文档 v1.0

---

1. 序言

1.1 文档简介

本文档基于最新的Guns版本，集`Guns使用手册`，`Guns开发手册`，`Guns核心思想`等于一体，并整理了`qq群`和`gitee`上用户经常反馈的问题的答疑！本文档最好的阅读方式是从上到下依次阅读(推荐)，也可根据需要直接从目录查看相关文档！感谢您对Guns的支持！

1.2 Guns教程

教程采用视频的形式，讲述了Guns作者近年来工作经验的总结，以及自2017年3月份编写Guns的感悟。教程历时两个月精心打造，希望大家多多支持！

点击查看教程详细介绍

教程零售价格：199元

如何获取教程?
请添加作者(stylefeng)qq 332464581(请备注购买教程)或加入下方qq群联系群主购买

1.3 获取帮助

> - Guns官方qq交流群：**254550081**
> - Guns官方git地址： https://gitee.com/naan1993/guns

---

2. 使用手册

注意：

> - Guns运行环境：JDK1.8
> - maven 3.3.9或更高
> - 请使用阿里云maven镜像
> - 作者当前使用开发工具为Eclipse oxygen.2 和 IDEA 2017.3

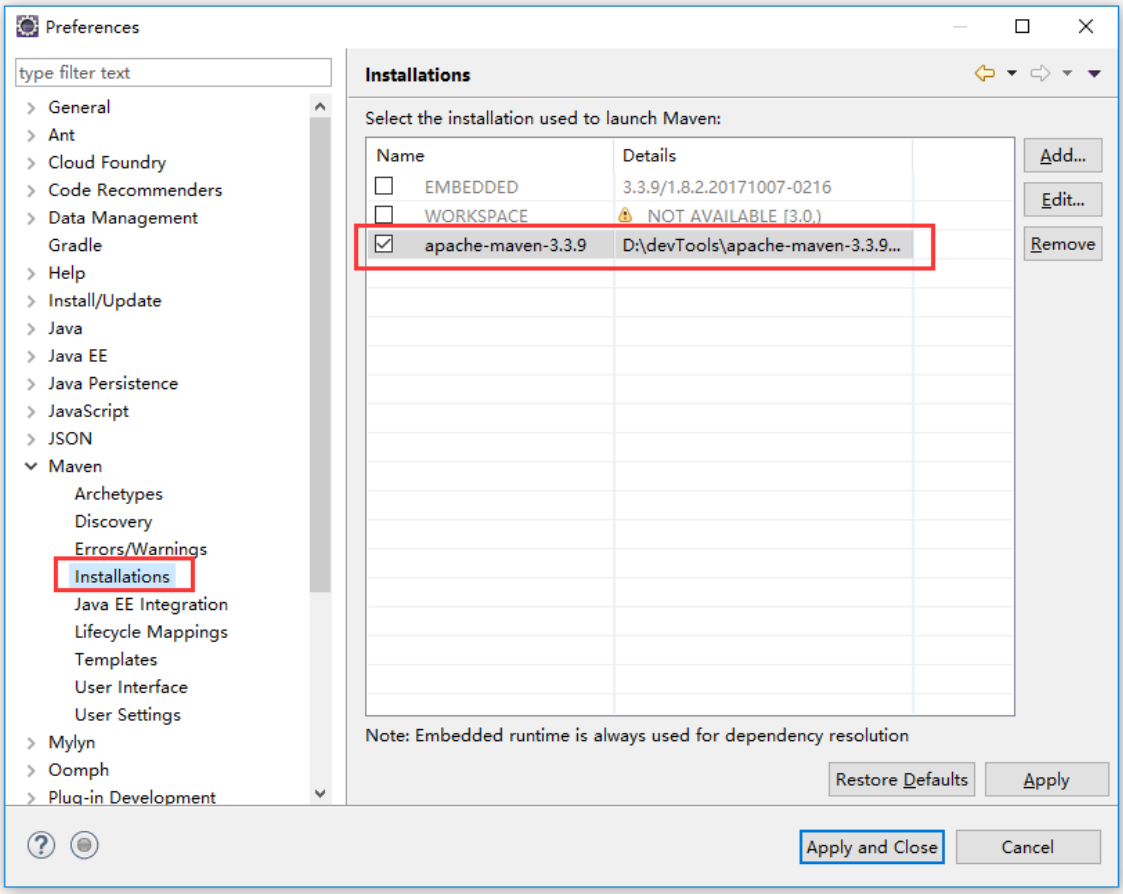
2.1 下载项目

登录码云平台，打开Guns主页，点击下载按钮下载

2.2 导入项目

2.2.1 eclipse导入

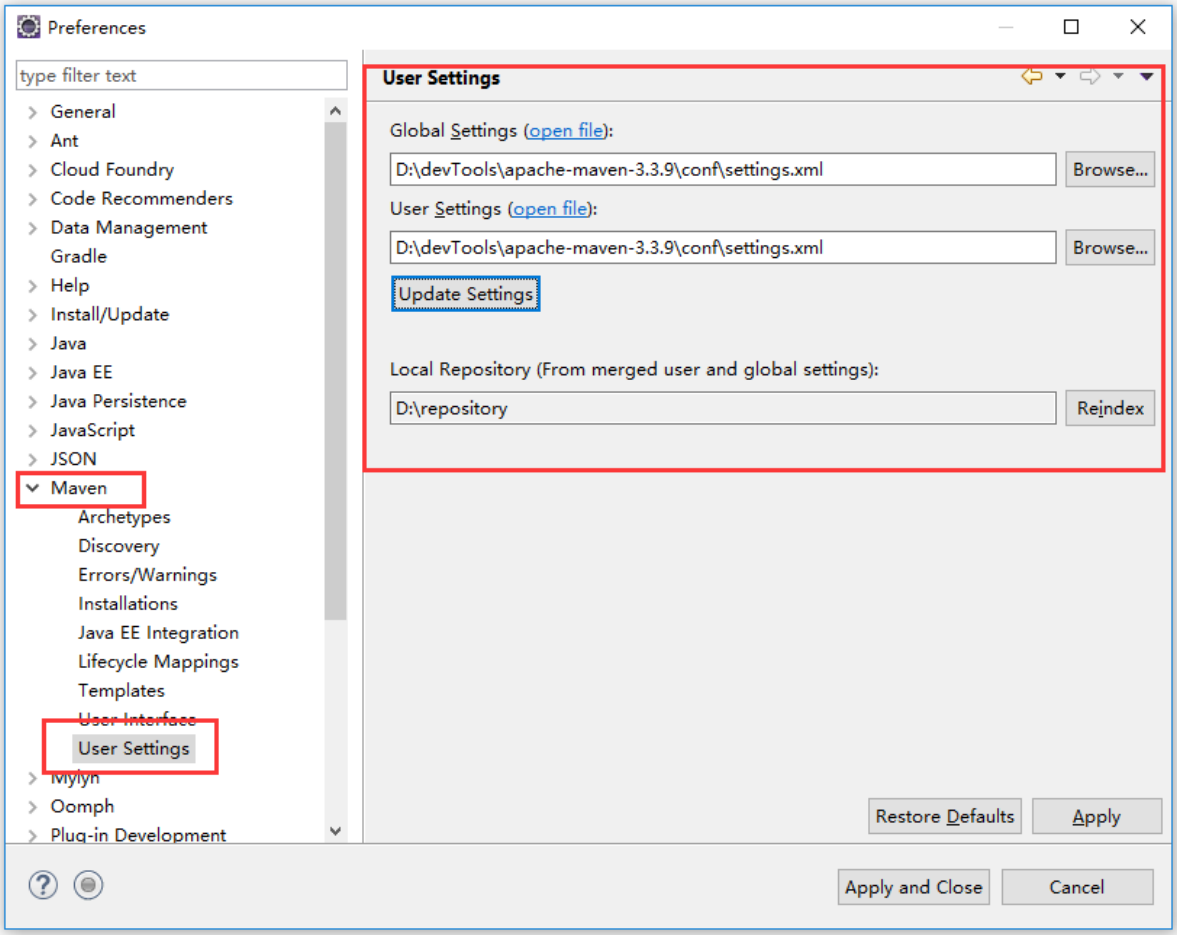
1. 导入之前请检查eclipse的maven配置是否本机所安装的maven（一般不用eclipse自带的maven），如下



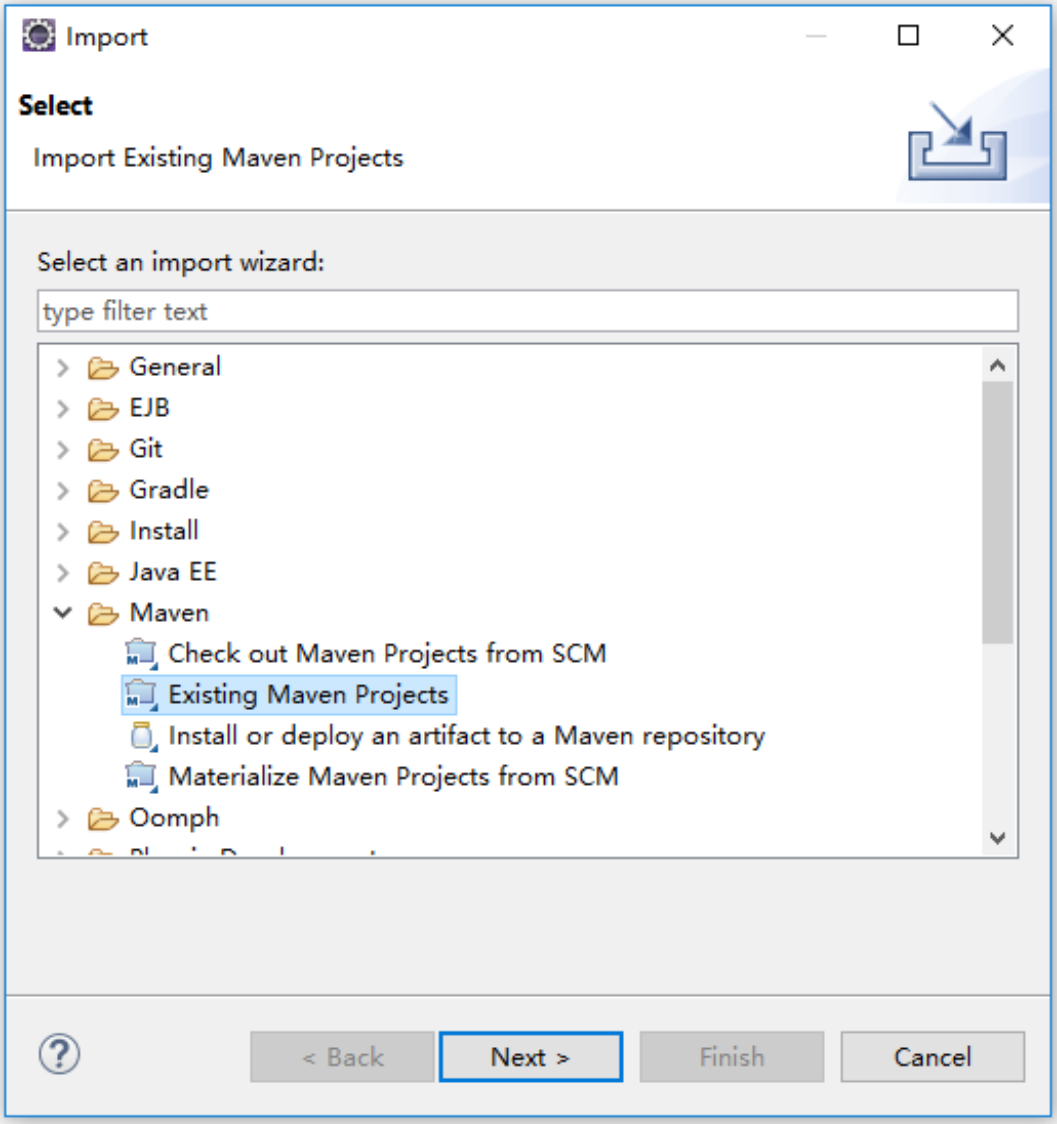
2. 检查maven安装目录下的settings.xml是否配置了阿里云镜像

```
</mirror>
 -->
 <mirror>
    <id>nexus-aliyun</id>
    <mirrorOf>*</mirrorOf>
    <name>Nexus aliyun</name>
    <url>http://maven.aliyun.com/nexus/content/groups/public/</url>
</mirror>
```

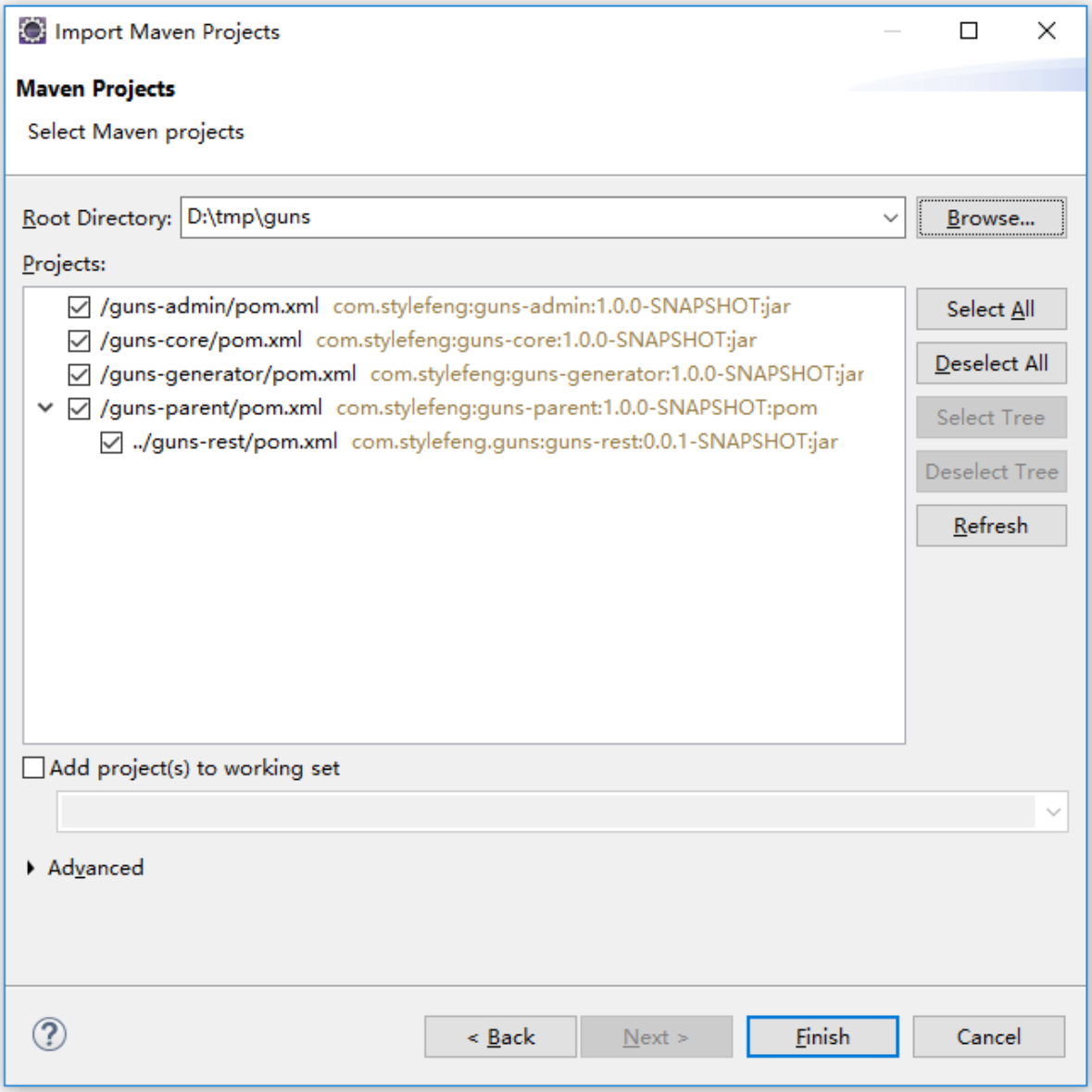
3. 再次检查eclipse中maven的配置是否应用了当前maven安装目录的settings.xml配置文件（个人习惯全局和用户配置设置为一个），如下



4. 以上设置完成，需要重启一下eclipse
5. 点击eclipse菜单File->import，出现如下界面，选择`Existing maven project`

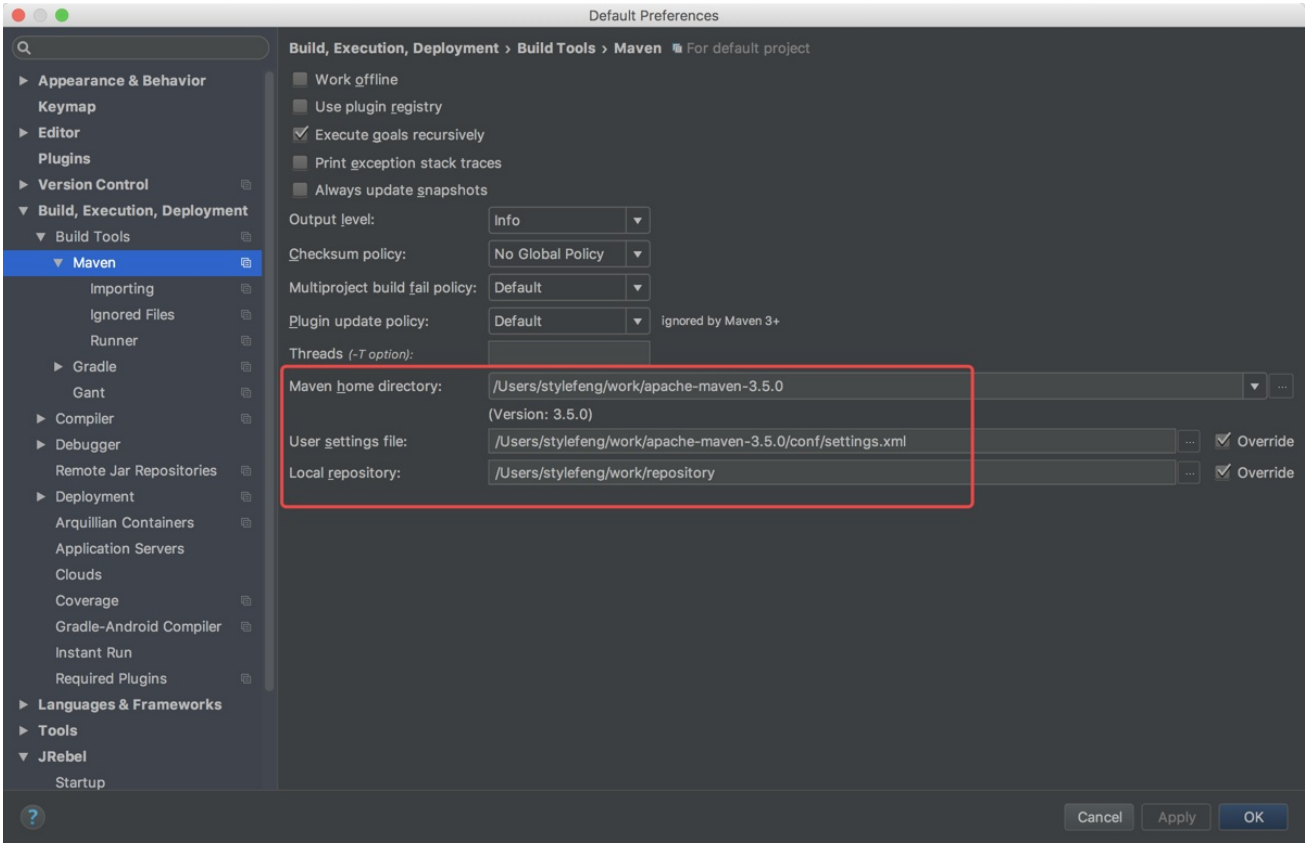


6. 找到下载的项目目录，并点击所有模块，之后点击Finish，导入成功



2.2.2 IDEA导入

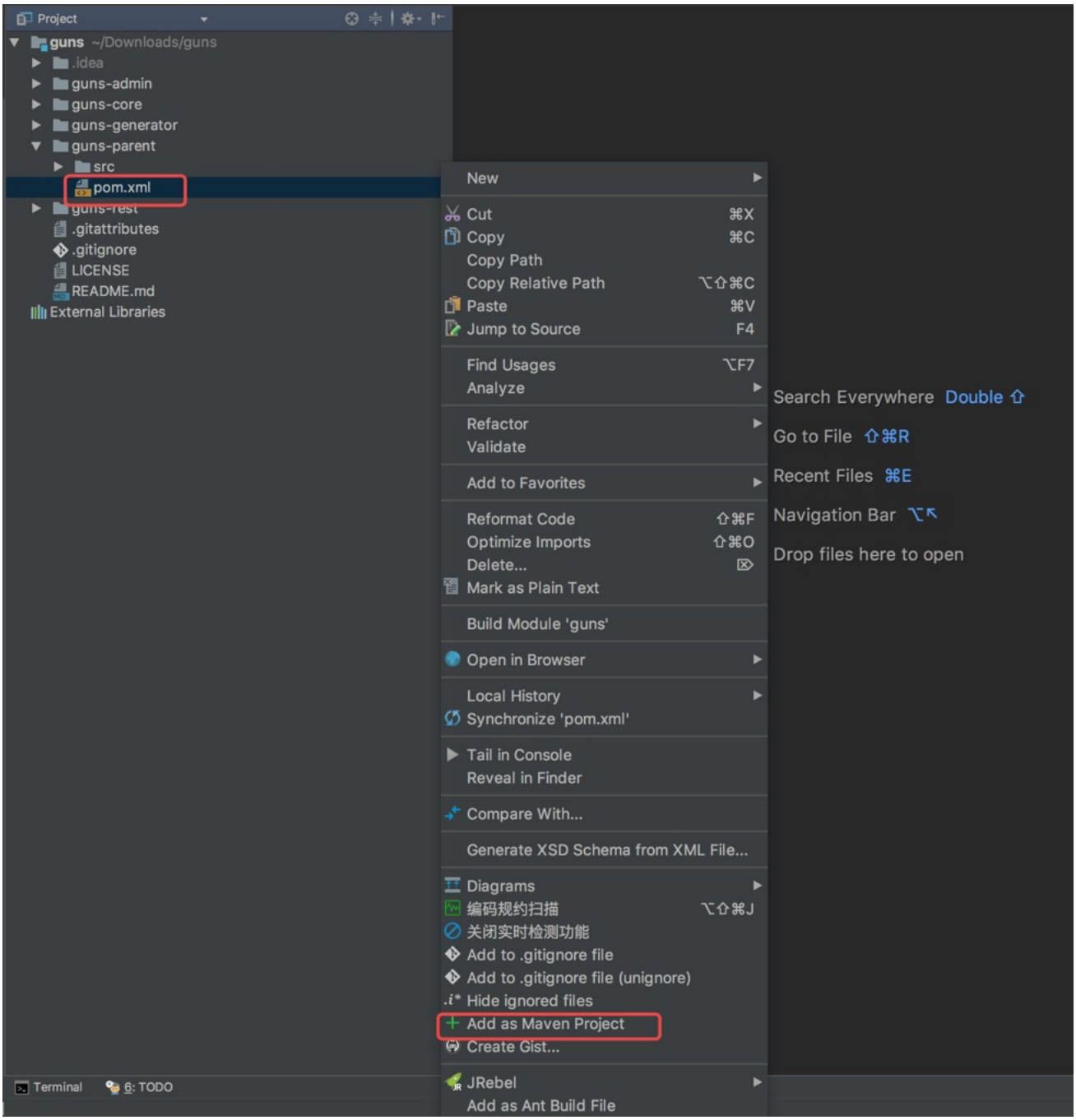
1. 同样，导入前检查IDEA的maven配置是否正确



2. 检查maven安装目录下的settings.xml是否配置了阿里云镜像(同`2.2.1节`第`2`步)

3. 进入IDEA主界面，点击open，并选择下载好的guns代码的根目录

4. 进入IDEA之后，右击`guns-parent`，选择`add as maven project`，即可完成导入



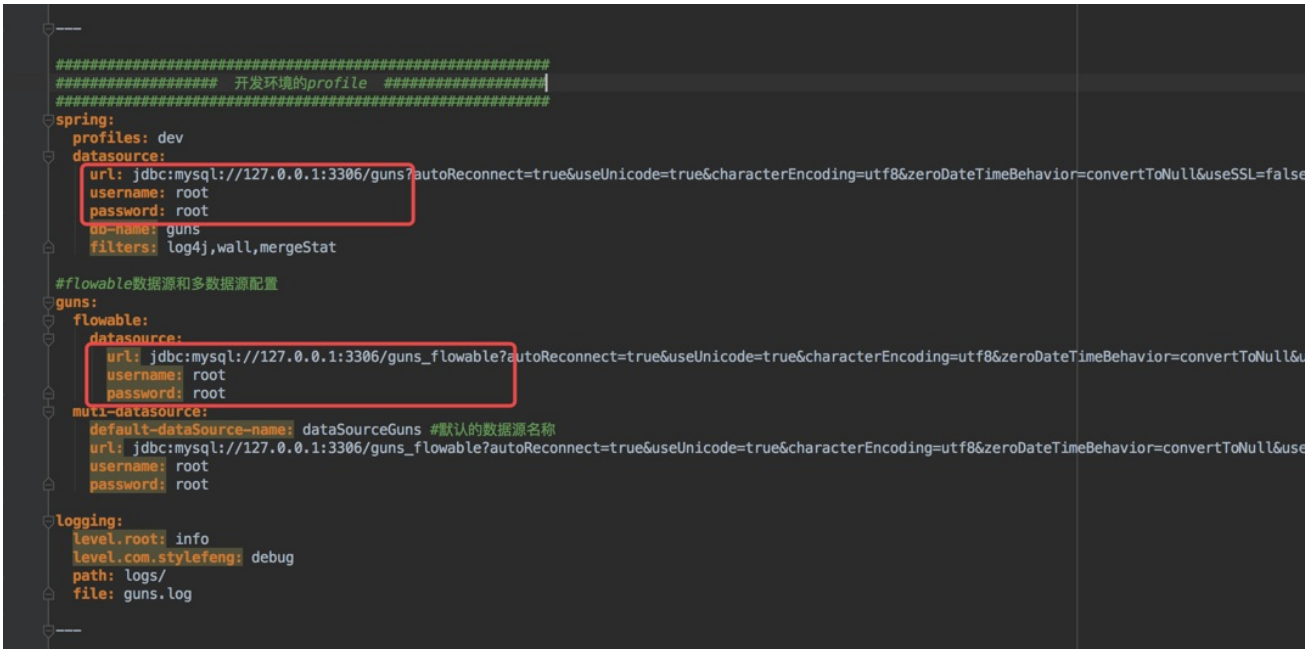
2.3 运行项目

运行前的准备:

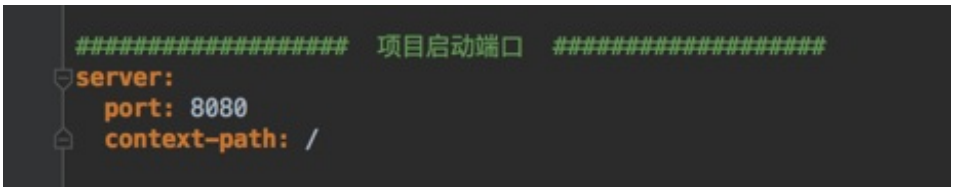
> - 安装mysql数据库，作者所用mysql版本为5.7

1. 执行`guns-admin`模块下的`sql/guns.sql`脚本，初始化guns的数据库环境

2. 打开`guns-admin/src/main/resources/application.yml`配置文件，修改`数据连接`，`账号`和`密码`，改为您所连接数据库的配置



3. 如需修改服务器端口或者context-path，可参考下图，修改相应属性即可

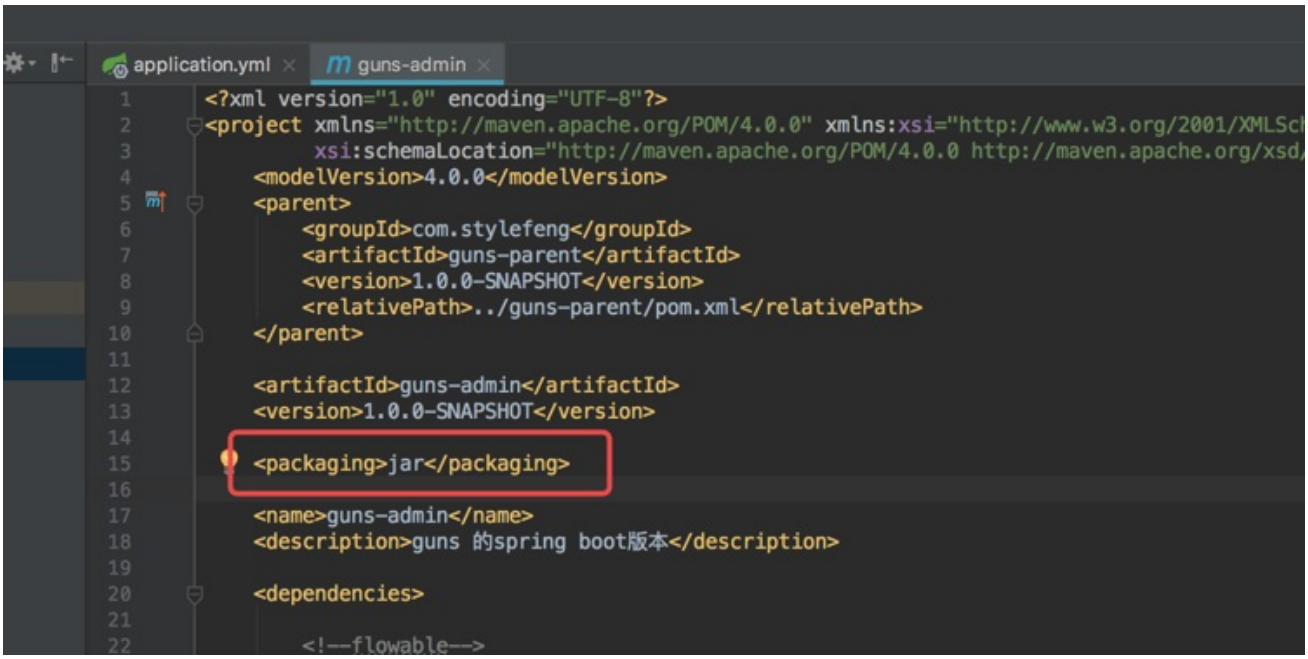


4. 执行`GunsApplication`类中的main方法，即可运行Guns系统
5. 打开浏览器，输入`localhost:8080`，即可访问到Guns的登录页面，默认登录账号密码:`admin/111111`

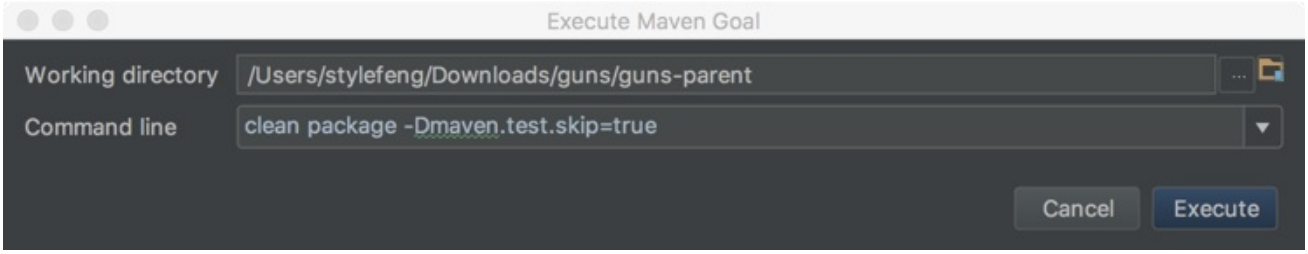
2.4 打包部署

目前Guns支持两种打包方式，即`jar包`和`war包`

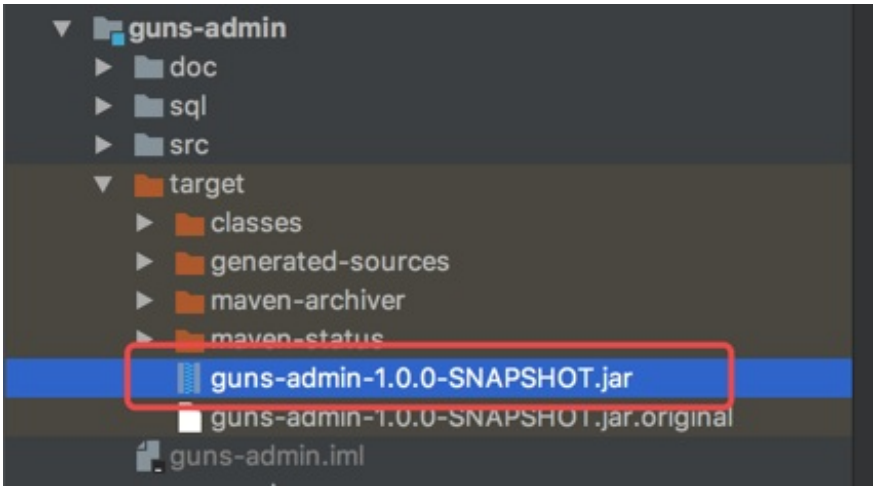
1. 打包之前修改`guns-admin.pom`中的`packaging`节点，改为`jar`或者`war`



2. 在项目的`guns-parent`目录执行`clean package -Dmaven.test.skip=true`，即可打包，如下



3. 命令执行成功后，在`guns-admin/target`目录下即可看到打包好的文件



提示：若打的包为jar包，可通过`java -jar guns-admin-1.0.0-SNAPSHOT.jar`来启动Guns系统

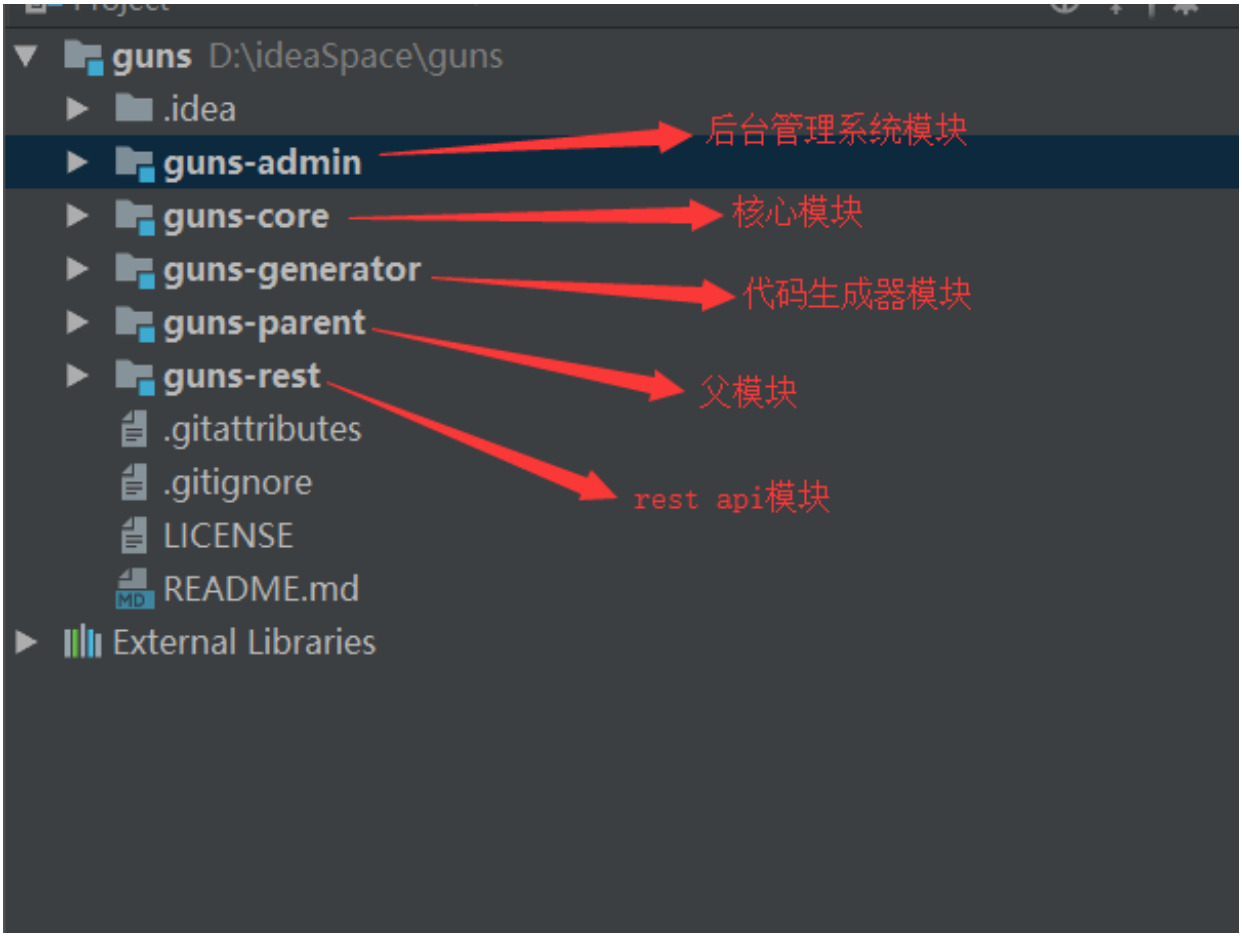
3. 开发手册

用Guns开发手头常备如下几个工具:

> - H+ 4.2源代码： 群文件里有
> - mybatis-plus文档：http://mp.baomidou.com/
> - beetl文档：http://ibeetl.com/guide/#beetl
> - Spring Boot文档：https://docs.spring.io/spring-boot/docs/current/reference/html/

3.1 了解Guns

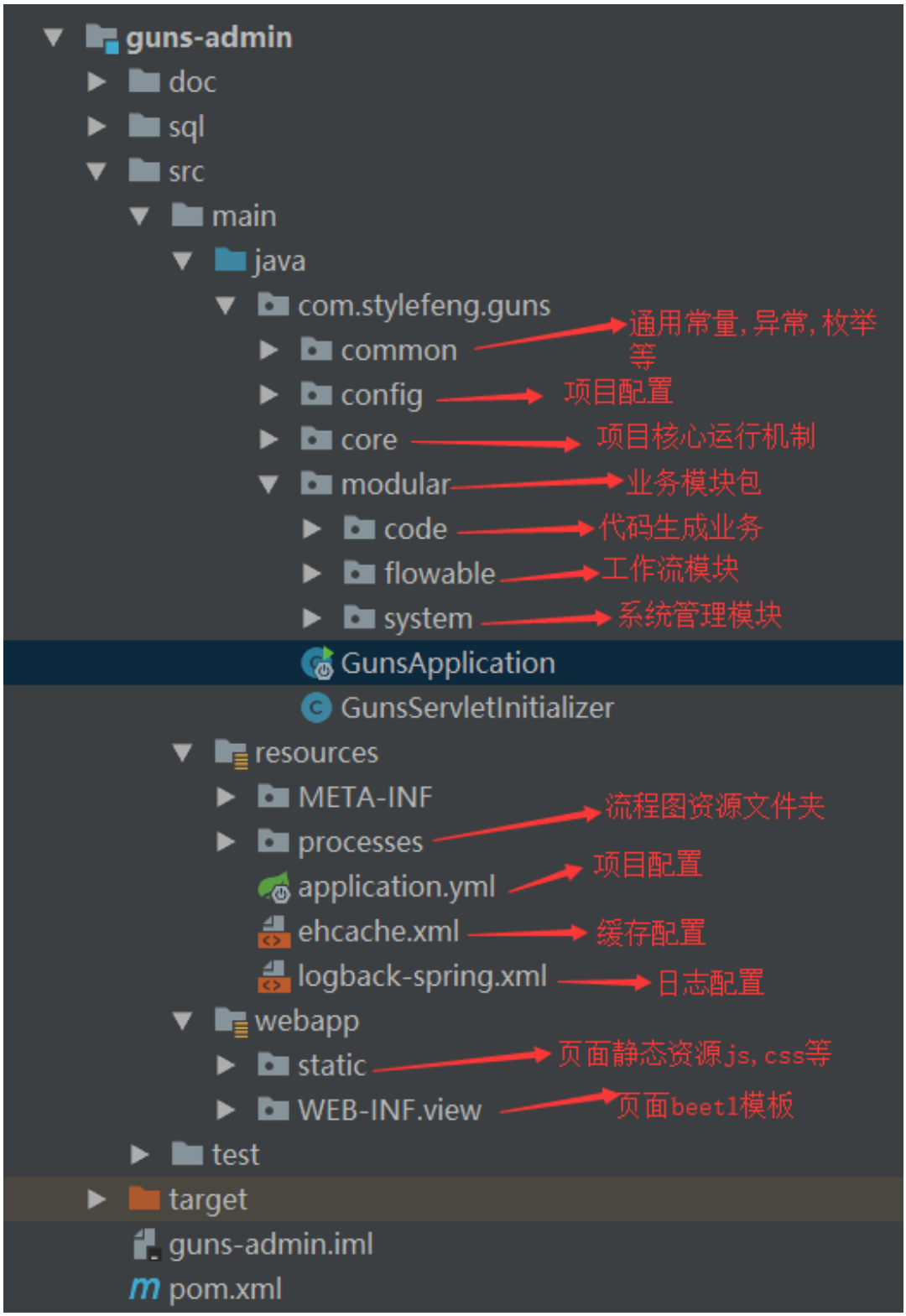
3.1.1 模块结构



1. `guns-admin`模块为后台管理系统模块，包括管理系统的业务代码，前端页面，项目的配置信息等等

2. `guns-core`模块为抽象出的核心（通用）模块，以供其他模块调用，此模块主要封装了一些通用的工具类，公共枚举，常量，配置等等
3. `guns-generator`为代码生成模块，其中代码生成模块整合了mybatis-plus的代码生成器和guns独有的代码生成器，可以一键生成entity，dao，service，html，js等代码，可减少新业务`70%`的工作量
4. `guns-parent`模块为其他所有模块的父模块，主要功能是管理项目中所有用到的jar，以及聚合其他模块
5. `guns-rest`为专门提供restful api的模块，该模块中主要实现了jwt鉴权和传输数据签名的机制

3.1.2 包结构



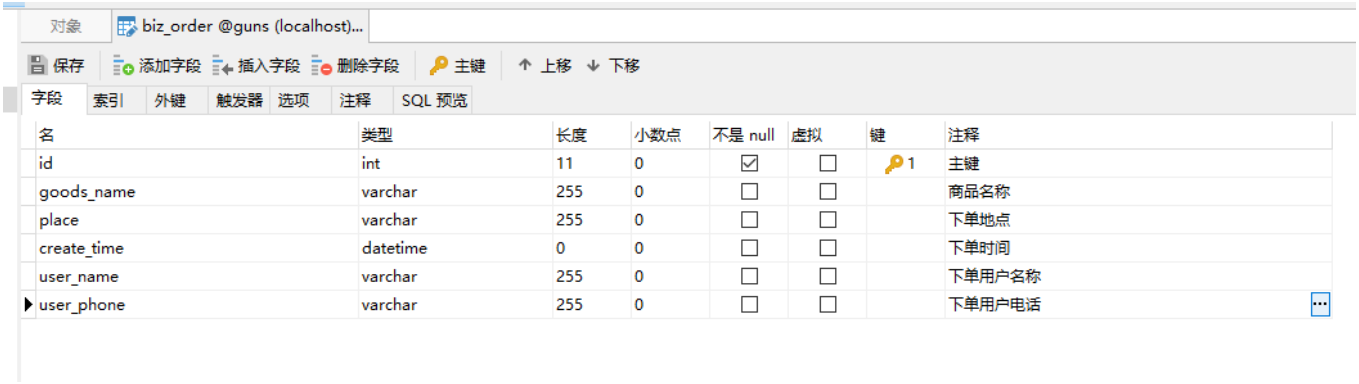
3.2 实战开发

Guns开发三部曲 -> 1.建表 2.代码生成 3.添加菜单 4.适配业务代码

下面以一个`订单业务`为例，实战演练如何用Guns编写简单的增删改查业务

3.2.1 建表

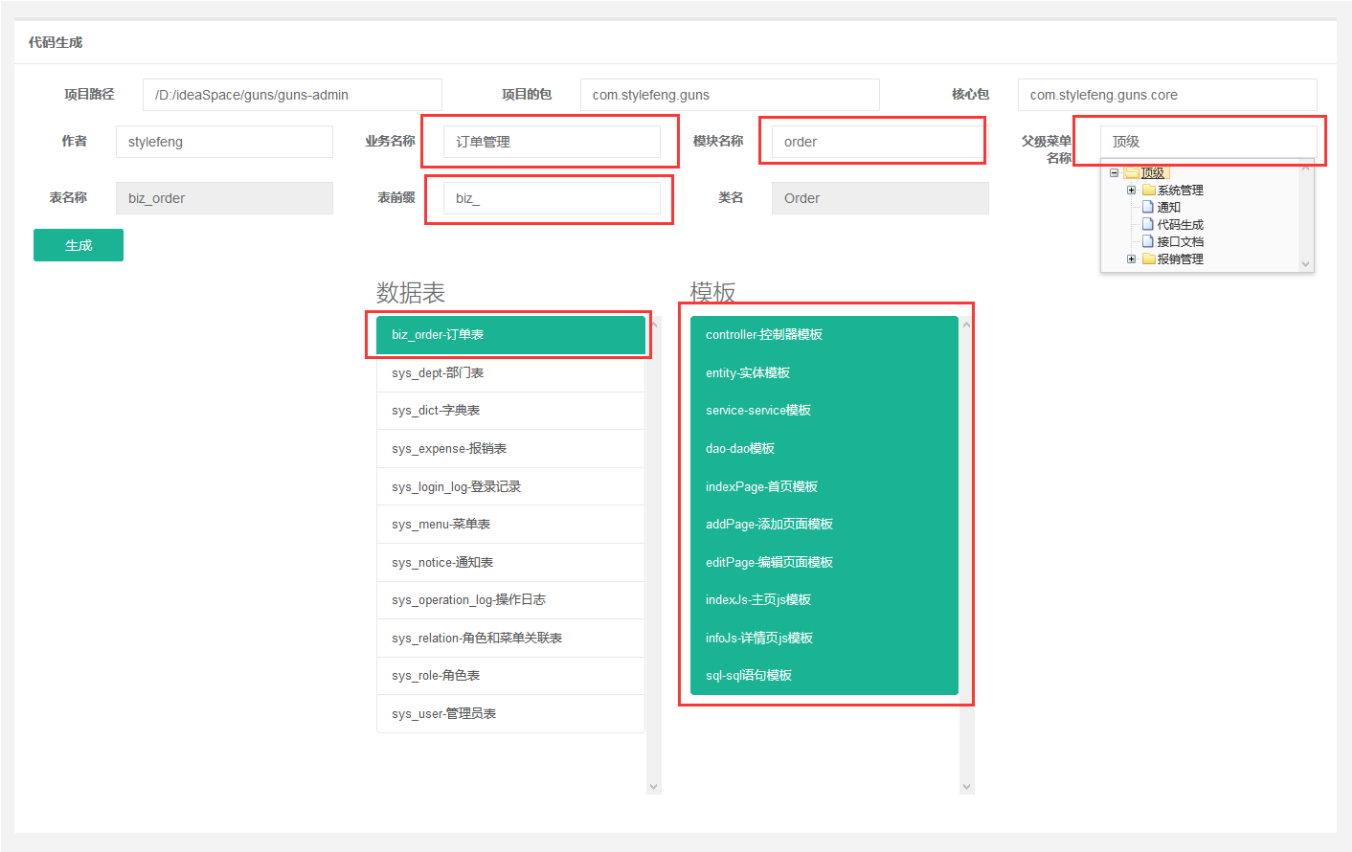
新建订单表如下:




```
DROP TABLE IF EXISTS `biz_order`;
CREATE TABLE `biz_order` (
  `id` int(11) NOT NULL AUTO_INCREMENT COMMENT '主键',
  `goods_name` varchar(255) DEFAULT NULL COMMENT '商品名称',
  `place` varchar(255) DEFAULT NULL COMMENT '下单地点',
  `create_time` datetime DEFAULT NULL COMMENT '下单时间',
  `user_name` varchar(255) DEFAULT NULL COMMENT '下单用户名称',
  `user_phone` varchar(255) DEFAULT NULL COMMENT '下单用户电话',
  PRIMARY KEY (`id`) USING BTREE
) ENGINE=InnoDB DEFAULT CHARSET=utf8 ROW_FORMAT=DYNAMIC COMMENT='订单表
';

SET FOREIGN_KEY_CHECKS = 1;
```

3.2.2 代码生成

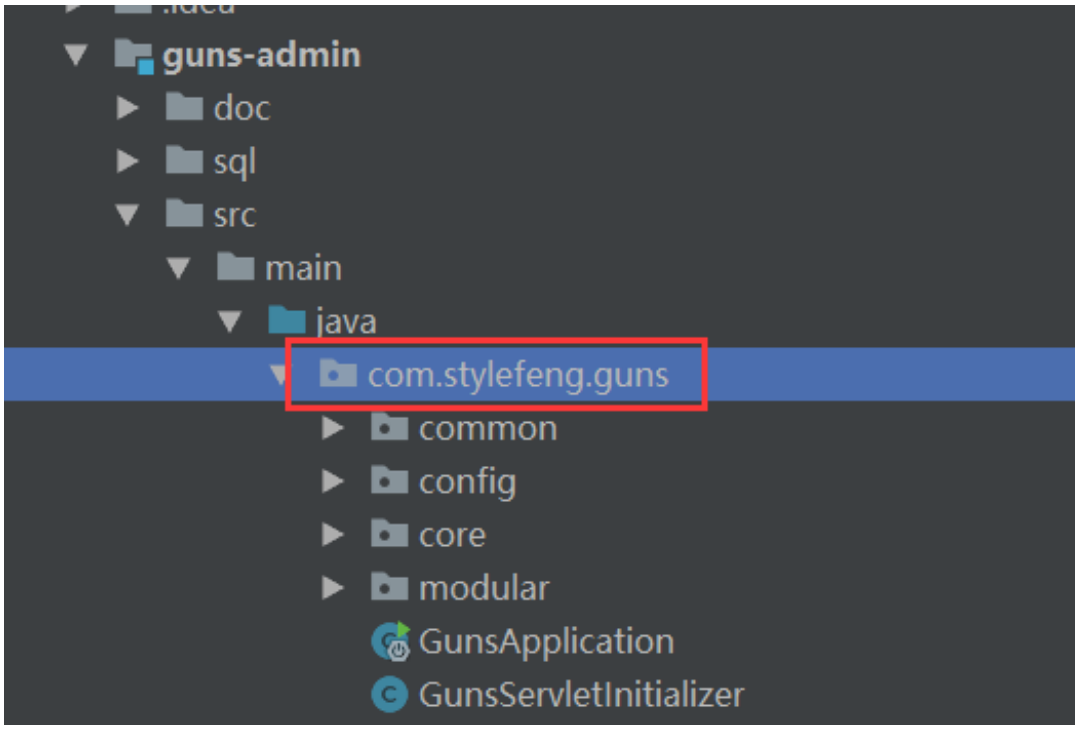
登录管理系统，打开代码生成页面，填写如下内容，注意看`红线部分`内容



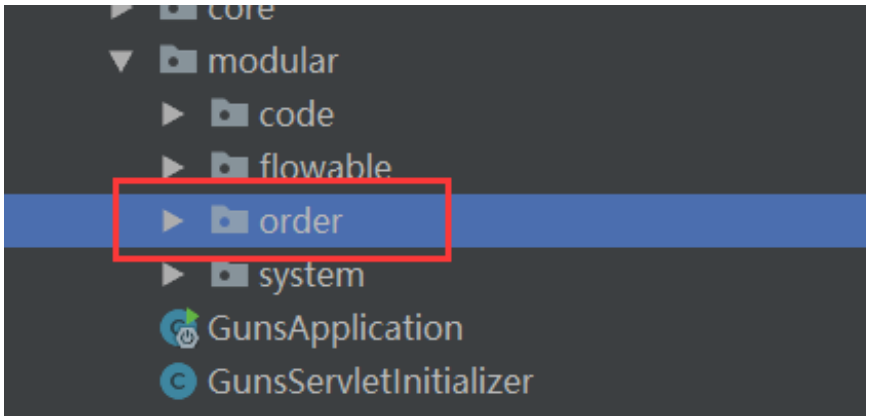
下面详细讲解代码生成使用:

1. **项目路径:** 代码生成的路径，具体到guns-admin模块的绝对路径，`一般不需要修改`，因为程序会自动计算出guns-admin的绝对路径

2. **项目的包:** 为guns-admin的同`GunsApplication`类同一目录的包，如下图，一般也不需要修改



3. **核心包:** gun-core的包，一般也不需要修改
4. **作者:** 填写代码生成出的注释上的作者
5. **业务名称:** 生成业务的中午名称
6. **模块名称:** 对应代码中modular包下的模块名称，如下图，若模块名称填order，则生成出的业务代码回到order包下



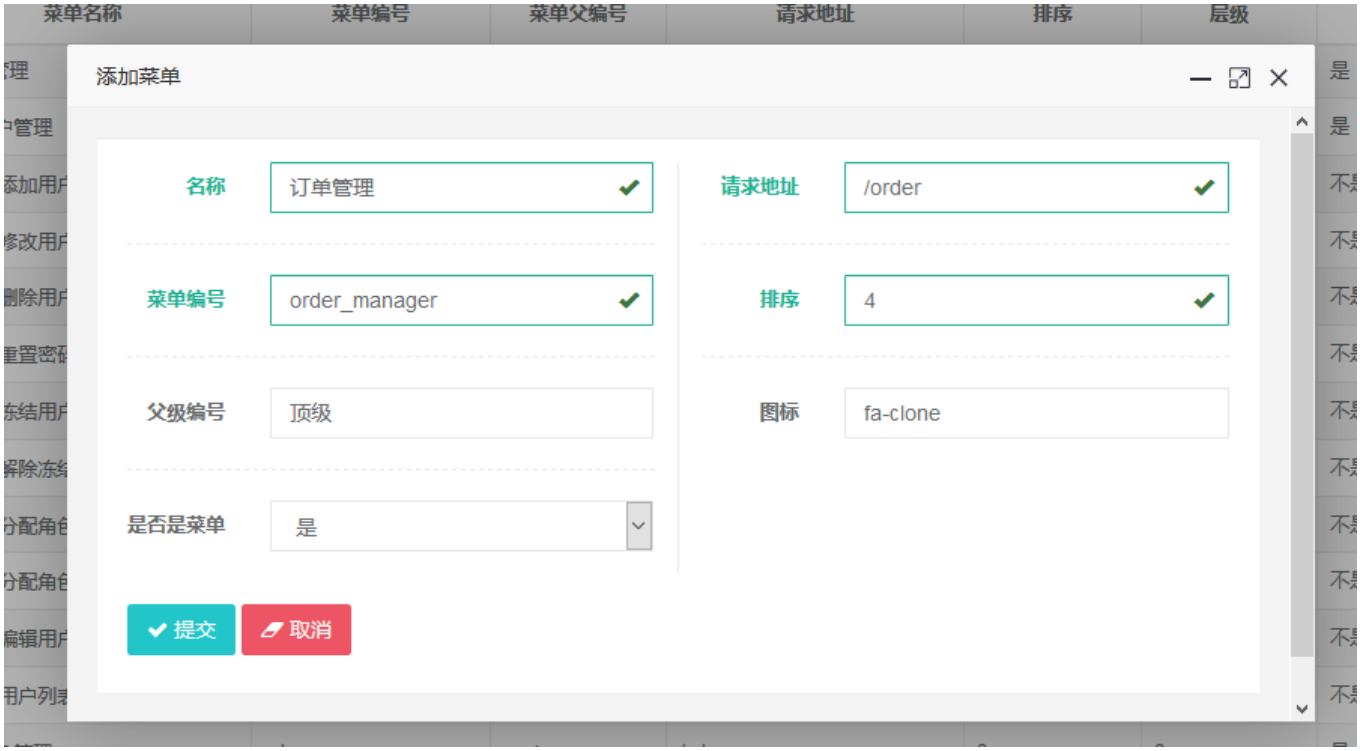
7. **父级菜单:** 此项的选择会影响生成sql添加菜单项的切入点，生成出的sql文件执行后可自动增加到sys_menu菜单项，省去手动添加菜单的繁琐
8. **表前缀:** 填写此项会自动移除生成实体，mapper和service类的名称中包含的重复前缀，例如生成订单表业务代码时，填写`biz_`，则生成的实体中不会包含Biz前缀名称，若不填写，则生成的实体类为BizOrder
9. **数据表:** 选择即为生成该表所对应的实体，dao，service等类
10. **模板:** 选择后生成相应的控制器，实体，service，dao代码等等

生成代码之后需要重启一下管理系统，生成的代码才可以生效!

3.3.3 添加菜单与分配权限

生成代码之后，需要为管理系统添加菜单，才可以让新增加的业务显示到页面上，添加菜单有两种方式:

第一种为手动添加菜单，依次点击`系统管理`->`菜单管理`->`点击添加`，打开添加页面，如下



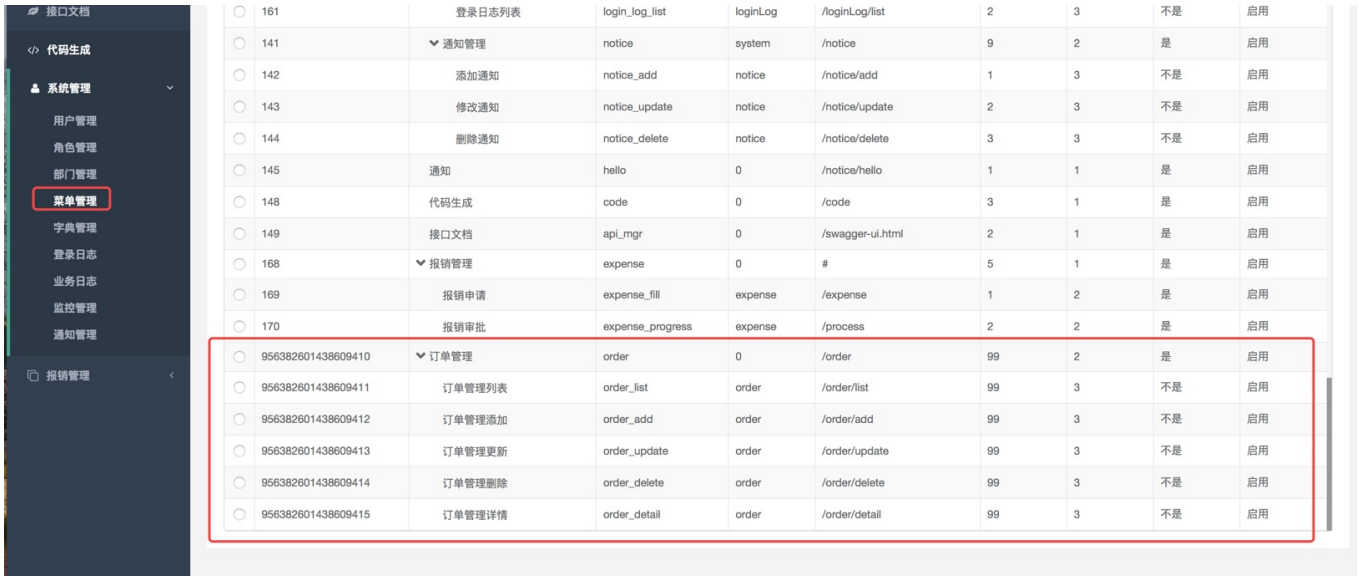
这里需要注意如下几点:

> - `请求地址`需要和Controller中的RequestMapping的值一致
> - `排序`为同层级菜单中显示菜单的顺序
> - `父级编号`的选择可以更改菜单插入的位置
> - `图标`可以从H+的资源库中获取
> - 因为菜单管理不单单是对管理系统中的菜单管理，也包含权限的管理，所以需要选择是否是菜单这个选项

第二种添加菜单的方式为直接执行代码生成中的sql脚本，默认生成的sql文件在`src/main/java`目录下，如下所示

```
1. INSERT INTO `guns`.`sys_menu` (`id`, `code`, `pcode`, `pcodes`, `name`, `icon`, `url`, `num`, `levels`, `ismenu`, `tips`, `status`, `isopen`) VALUES ('956388083570089986', 'order', '0', '[0],', '订单管理', '', '/order', '99', '1', '1', NULL, '1', '0');
2. INSERT INTO `guns`.`sys_menu` (`id`, `code`, `pcode`, `pcodes`, `name`, `icon`, `url`, `num`, `levels`, `ismenu`, `tips`, `status`, `isopen`) VALUES ('956388083570089987', 'order_list', 'order', '[0],[order],', '订单管理列表', '', '/order/list', '99', '2', '0', NULL, '1', '0');
3. INSERT INTO `guns`.`sys_menu` (`id`, `code`, `pcode`, `pcodes`, `name`, `icon`, `url`, `num`, `levels`, `ismenu`, `tips`, `status`, `isopen`) VALUES ('956388083570089988', 'order_add', 'order', '[0],[order],', '订单管理添加', '', '/order/add', '99', '2', '0', NULL, '1', '0');
4. INSERT INTO `guns`.`sys_menu` (`id`, `code`, `pcode`, `pcodes`, `name`, `icon`, `url`, `num`, `levels`, `ismenu`, `tips`, `status`, `isopen`) VALUES ('956388083570089989', 'order_update', 'order', '[0],[order],', '订单管理更新', '', '/order/update', '99', '2', '0', NULL, '1', '0');
5. INSERT INTO `guns`.`sys_menu` (`id`, `code`, `pcode`, `pcodes`, `name`, `icon`, `url`, `num`, `levels`, `ismenu`, `tips`, `status`, `isopen`) VALUES ('956388083570089990', 'order_delete', 'order', '[0],[order],', '订单管理删除', '', '/order/delete', '99', '2', '0', NULL, '1', '0');
6. INSERT INTO `guns`.`sys_menu` (`id`, `code`, `pcode`, `pcodes`, `name`, `icon`, `url`, `num`, `levels`, `ismenu`, `tips`, `status`, `isopen`) VALUES ('956388083570089991', 'order_detail', 'order', '[0],[order],', '订单管理详情', '', '/order/detail', '99', '2', '0', NULL, '1', '0');
```

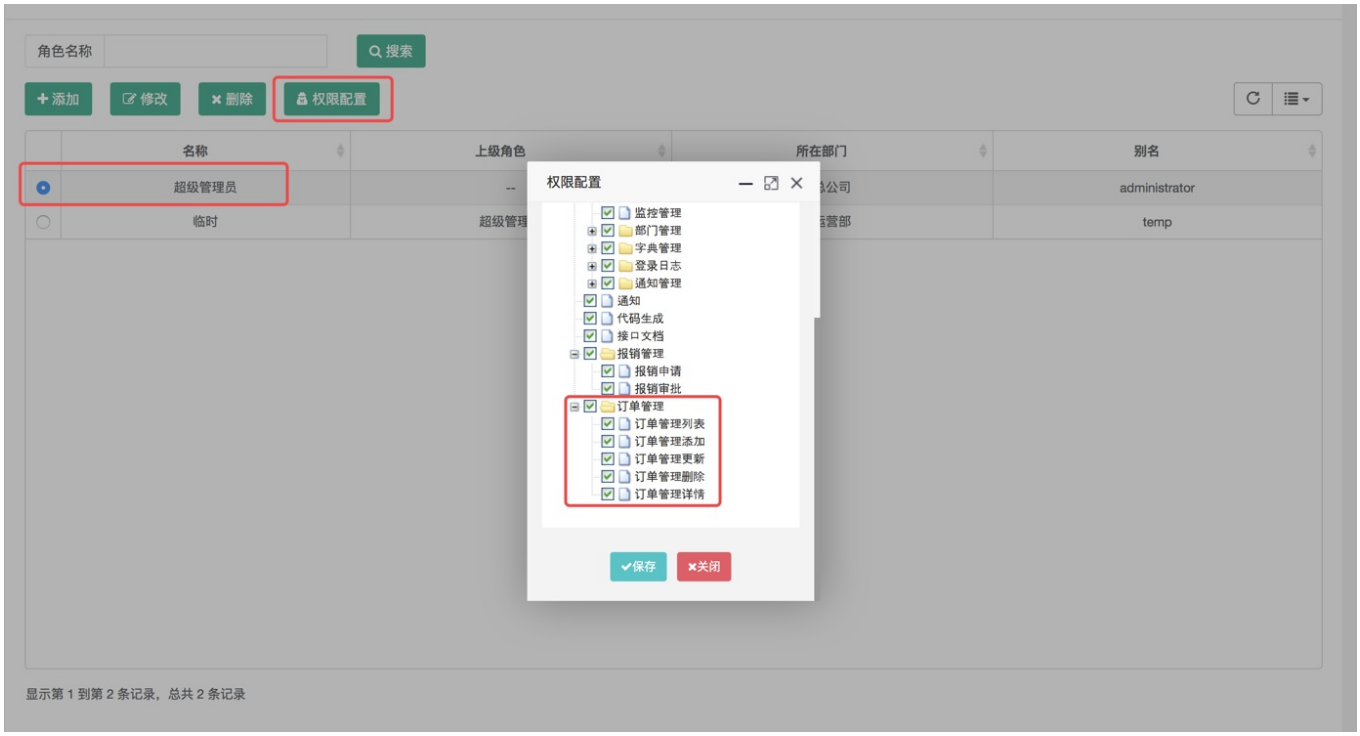
执行完成后可以看到，菜单管理页面中已经有了新添加的订单相关的菜单和资源，如下



161	登录日志列表	login_log_list	loginLog	/loginLog/list	2	3	不是	启用
141	通知管理	notice	system	/notice	9	2	是	启用
142	添加通知	notice_add	notice	/notice/add	1	3	不是	启用
143	修改通知	notice_update	notice	/notice/update	2	3	不是	启用
144	删除通知	notice_delete	notice	/notice/delete	3	3	不是	启用
145	通知	hello	0	/notice/hello	1	1	是	启用
148	代码生成	code	0	/code	3	1	是	启用
149	接口文档	api_mgr	0	/swagger-ui.html	2	1	是	启用
168	报销管理	expense	0	#	5	1	是	启用
169	报销申请	expense_fill	expense	/expense	1	2	是	启用
170	报销审批	expense_progress	expense	/process	2	2	是	启用
956382601438609410	订单管理	order	0	/order	99	2	是	启用
956382601438609411	订单管理列表	order_list	order	/order/list	99	3	不是	启用
956382601438609412	订单管理添加	order_add	order	/order/add	99	3	不是	启用
956382601438609413	订单管理更新	order_update	order	/order/update	99	3	不是	启用
956382601438609414	订单管理删除	order_delete	order	/order/delete	99	3	不是	启用
956382601438609415	订单管理详情	order_detail	order	/order/detail	99	3	不是	启用

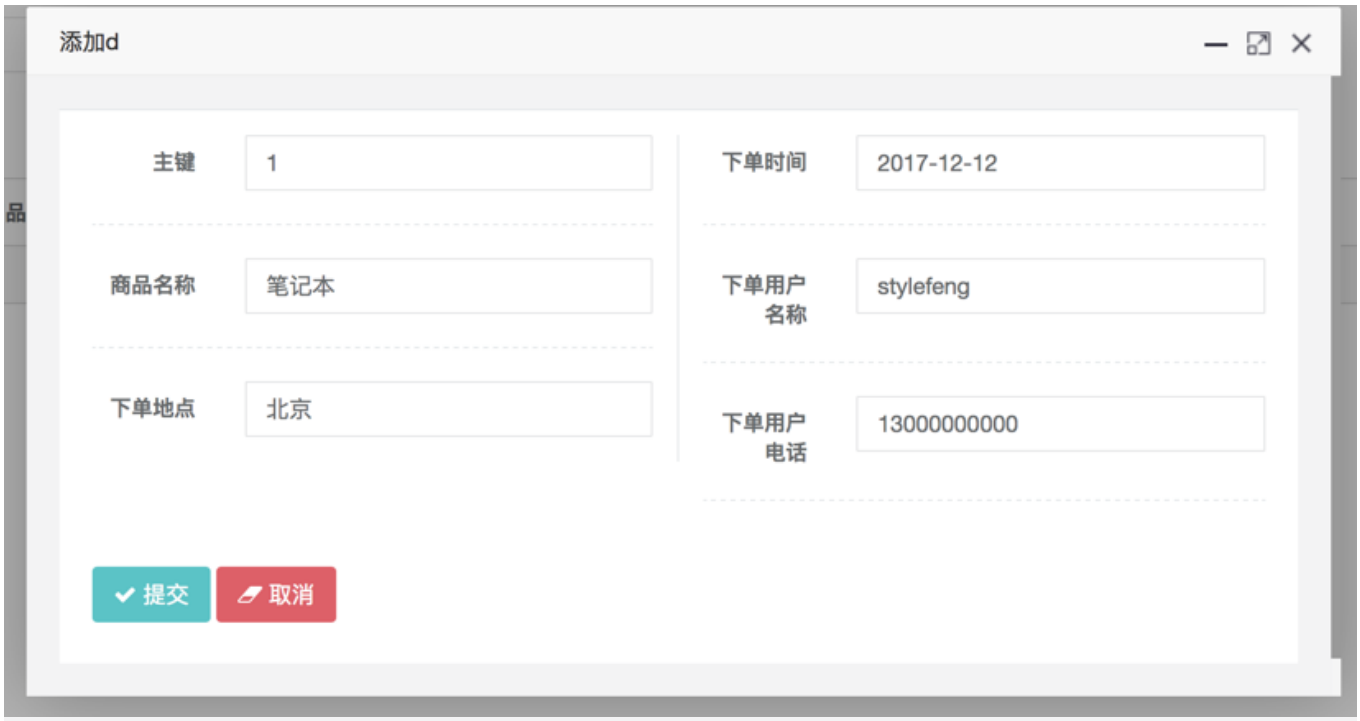
在添加完菜单只有，还需要给角色分配相关的菜单权限，才可以把新增的业务显示到菜单上

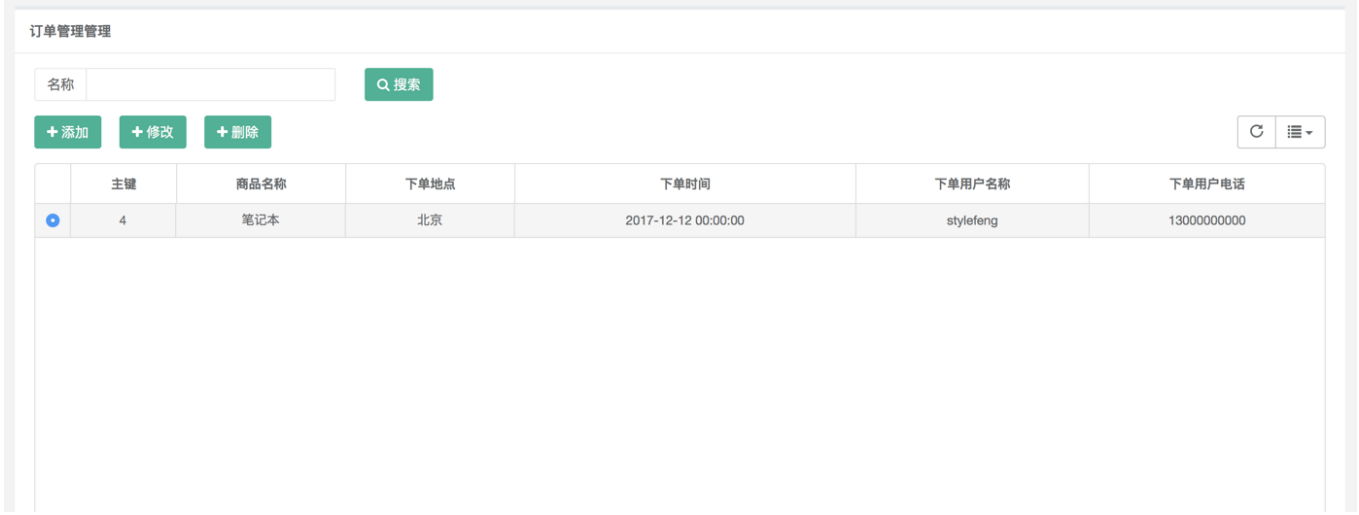
打开`系统管理`->`角色管理`，给当前的登录的超级管理员，增加刚才新增的权限，如下图



配置完成刷新页面即可看到，即可看到新增加的菜单，如下图，若看不到请重新登录

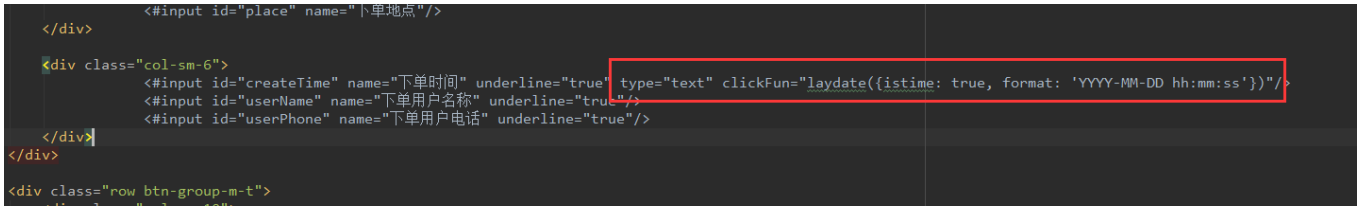
到这里，基本的增删改查功能就实现了，如下图





3.3.4 编写业务代码

由于Guns的代码生成器还不能实现100%的智能，所以生成之后还需要对生成的代码做一些完善，如果有除了增删改查以外的业务，还需要手动编写。例如，上面编写的添加订单和修改订单里，下单时间默认是text文本框，这里需要手动改为laydate样式的日期框，如下图



至此，guns的开发流程介绍完毕~！

3.3 权限控制于校验

3.3.1 用户，角色和资源

用户、角色和资源（或者说权限），这三者的关系是`用户对应角色`，`角色对应资源`，菜单和所有的按钮都可以看做是`资源`(或`权限`)，把某一个角色赋予相应的资源，那么该角色就会有访问该资源的权限，否则，该角色访问这些被管控的资源就会被服务器返回`403 没有权限`，当角色绑定资源后还需要给`用户赋予角色`才可以让登录的用户访问相关服务器接口。

一句话概括: 用户对应角色，角色对应资源

3.3.2 如何对资源进行权限控制

Guns系统中，通过在控制器上加`@Permission`注解进行权限校验，如下所示，该接口在被访问的时候，就会进行权限校验

```
/**
 * 获取所有部门列表
 */
@RequestMapping(value = "/list")
@Permission
@ResponseBody
public Object list(String condition) {
    List<Map<String, Object>> list = this.deptDao.list(condition);
    return super.warpObject(new DeptWarpper(list));
}
```

通过我们查找`用户对应的角色`，并查找`角色对应的资源`，可以找到，当前用户(admin)有该资源的权限，如下

152	to_dept_update	dept	[0],[system],[dept],	修改部门跳转		/dept/dept_update	4	3	0	(Null)	1
153	dept_list	dept	[0],[system],[dept],	部门列表		/dept/list	5	3	0	(Null)	1
154	dept_detail	dept	[0],[system],[dept],	部门详情		/dept/detail	6	3	0	(Null)	1

`@Permission`注解中可以带一个String数组类型的参数，如下，加上该参数，则接口被限制为只有某个或某些角色才可访问

```java
/**
 * 添加管理员
 */
@RequestMapping("/add")
@BussinessLog(value = "添加管理员", key = "account", dict = UserDict.class)
@Permission(Const.ADMIN_NAME)
@ResponseBody
public Tip add(@Valid UserDto user, BindingResult result) {
    if (result.hasErrors()) {
        throw new GunsException(BizExceptionEnum.REQUEST_NULL);
    }

    // 判断账号是否重复
```

权限的检查是通过`AOP`拦截`@Permission`注解完成的，当访问受权限控制的资源时，`AOP`对当前请求的`servletPath`和数据库中`sys_menu`表的`url`字段进行匹配，如果当前用户所拥有的权限包含当前请求的`servletPath`，则访问这个接口成功

3.3.3 前端页面对权限资源的显示

在前端页面中，如果增删改查等按钮受权限控制，则我们需要对资源进行一个权限检查，如果有该资源的权限，才能让该按钮显示，通过`beetl`的`shiro注册方法`即可完成该项的检查

```
@if(shiro.hasPermission("/menu/add")){
    <#button name="添加" icon="fa-plus" clickFun="Menu.openAddMenu()"/>
@}
@if(shiro.hasPermission("/menu/edit")){
    <#button name="修改" icon="fa-edit" clickFun="Menu.openChangeMenu()" space="true"/>
@}
@if(shiro.hasPermission("/menu/remove")){
    <#button name="删除" icon="fa-remove" clickFun="Menu.delMenu()" space="true"/>
@}
```

其中`shiro.hasPermission()`起到了权限检查的作用，如果有该资源对应的权限，则被检查的资源显示，若没有该资源的权限，则按钮不显示

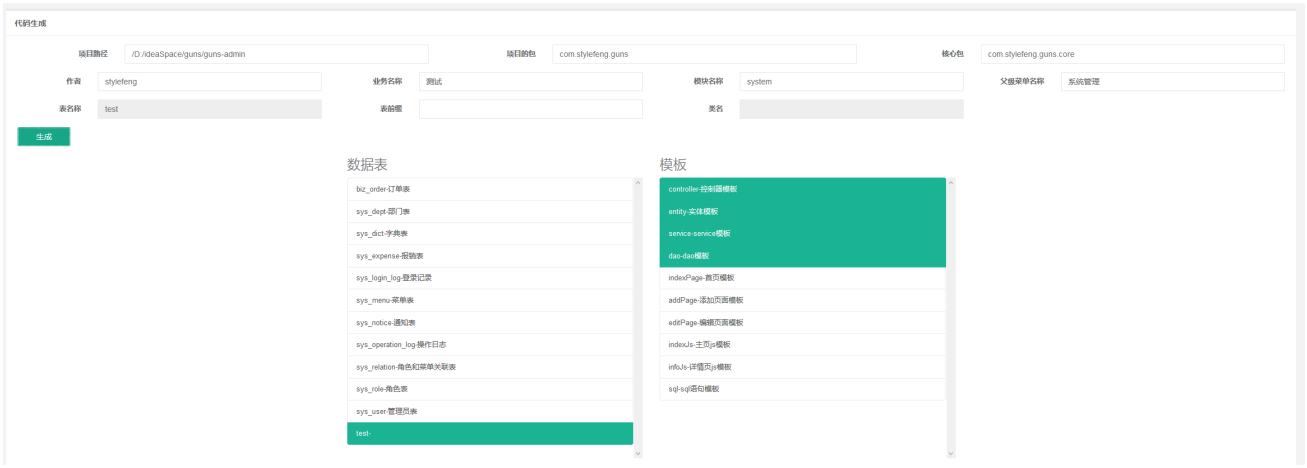
若想深入了解shiro和权限控制的实现原理，可参考视频教程第`12节 shiro与权限系统`，内有70分钟详细的讲解

3.4 多数据源的使用

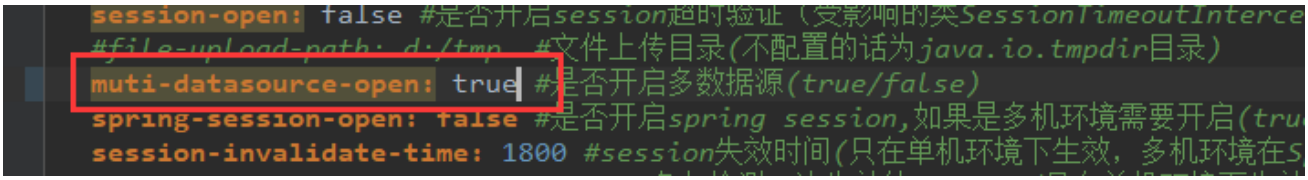
首先，我们新建一个数据库`guns_test`，并分别在`guns`数据库和`guns_test`数据库中分别新增同样结构的两个表`test`

```
DROP DATABASE IF EXISTS guns_test;
CREATE DATABASE IF NOT EXISTS guns_test DEFAULT CHARSET utf8 COLLATE utf8_general_ci;

USE guns_test;

SET NAMES utf8mb4;
SET FOREIGN_KEY_CHECKS = 0;

-- ----------------------------
-- Table structure for test
-- ----------------------------
DROP TABLE IF EXISTS `test`;
CREATE TABLE `test`  (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `value` varchar(255) CHARACTER SET utf8 COLLATE utf8_general_ci DEFAULT NULL,
  PRIMARY KEY (`id`) USING BTREE
) ENGINE = InnoDB AUTO_INCREMENT = 1 CHARACTER SET = utf8 COLLATE = utf8_general_ci ROW_FORMAT = Dynamic;

SET FOREIGN_KEY_CHECKS = 1;
```

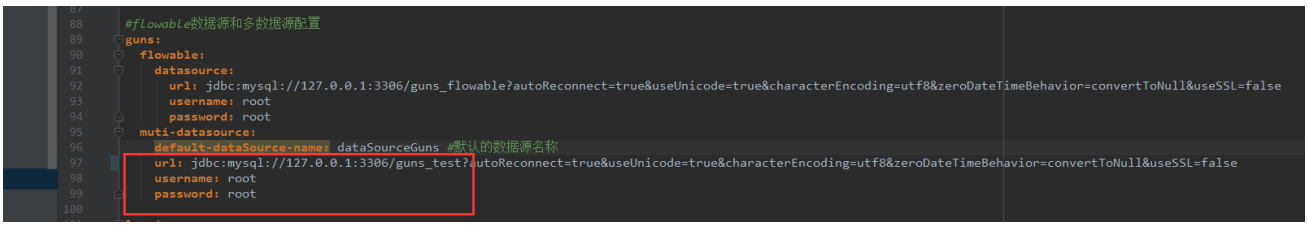
1. 对表进行代码生成，方便测试两个数据源



2. 打开application.yml中的多数据源开关



3. 配置application.yml中的多数据源的连接信息



4. 编写测试多数据源的代码，注意观察`@DataSource注解`

```java
package com.stylefeng.guns.modular.order.service;

/**
 * 测试多数据源的服务
 *
 * @author fengshuonan
 * @date 2018年1月25日21:48:34
 */
public interface ITestService {

    /**
     * 测试第二个数据源
     */
    void testBiz();

    /**
     * 测试guns本身的数据源
     */
    void testGuns();
}
```

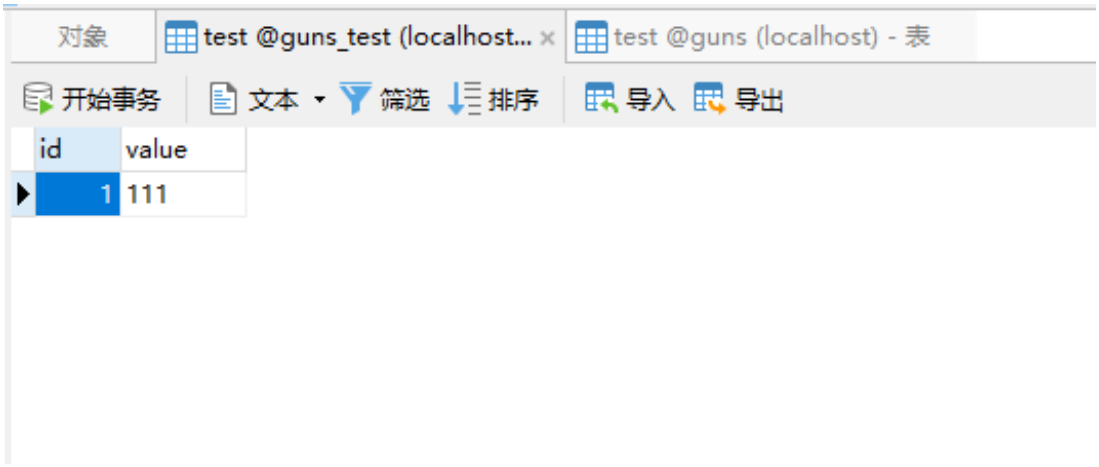
```java
package com.stylefeng.guns.modular.order.service.impl;

import com.stylefeng.guns.common.constant.DatasourceEnum;
import com.stylefeng.guns.common.persistence.dao.TestMapper;
import com.stylefeng.guns.common.persistence.model.Test;
import com.stylefeng.guns.core.mutidatasource.annotion.DataSource;
import com.stylefeng.guns.modular.order.service.ITestService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
```

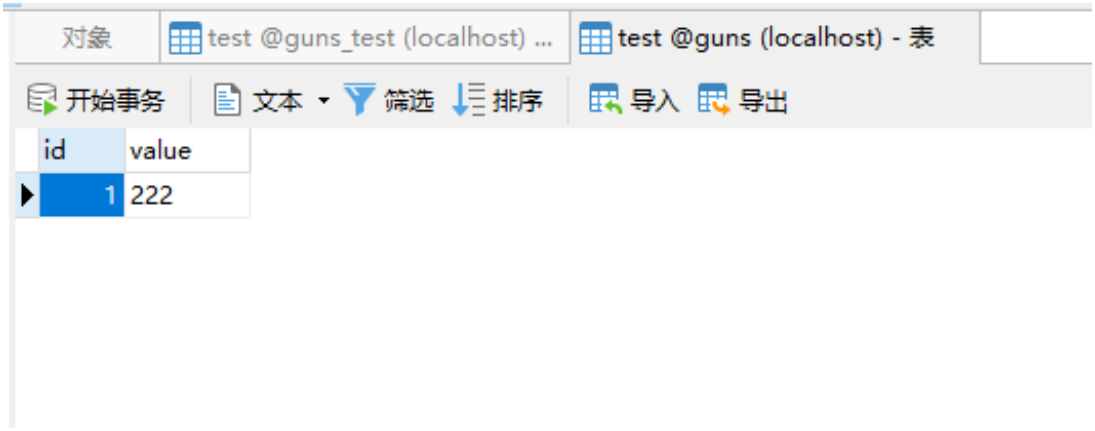
```
10.
11.  /**
12.   * 测试服务
13.   *
14.   * @author fengshuonan
15.   * @date 2018年1月25日21:48:39
16.   */
17.  @Service
18.  public class TestServiceImpl implements ITestService {
19.
20.      @Autowired
21.      TestMapper testMapper;
22.
23.      @Override
24.      @DataSource(name = DatasourceEnum.DATA_SOURCE_BIZ)
25.      public void testBiz() {
26.          Test test1 = new Test();
27.          test1.setValue("111");
28.          testMapper.insert(test1);
29.      }
30.
31.
32.      @Override
33.      @DataSource(name = DatasourceEnum.DATA_SOURCE_GUNS)
34.      public void testGuns() {
35.          Test test1 = new Test();
36.          test1.setValue("222");
37.          testMapper.insert(test1);
38.      }
39.  }
```

```
1.  package com.stylefeng.guns.system;
2.
3.  import com.stylefeng.guns.base.BaseJunit;
4.  import com.stylefeng.guns.modular.order.service.ITestService;
5.  import org.junit.Test;
6.  import org.springframework.beans.factory.annotation.Autowired;
7.
8.  /**
9.   * 业务测试
10.  *
11.  * @author fengshuonan
12.  * @date 2018年1月25日21:50:23
13.  */
```

```java
public class BizTest extends BaseJunit {

    @Autowired
    ITestService testService;

    @Test
    public void test() {
        testService.testGuns();

        testService.testBiz();
    }
}
```

1. 执行`BizTest`这个测试类，可以看出，两条数据同时插入了不同的数据库中的两张表中





多数据源的原理就是一个项目同时配置了两个`DataSource`，并把这两个`DataSource`放到`DynamicDataSource`绑定，使用AOP进行动态切换当前操作的数据源。

若想深入了解多数据源的配置和原理可参考`MybatisPlusConfig类`和`MultiSourceExAop类`，

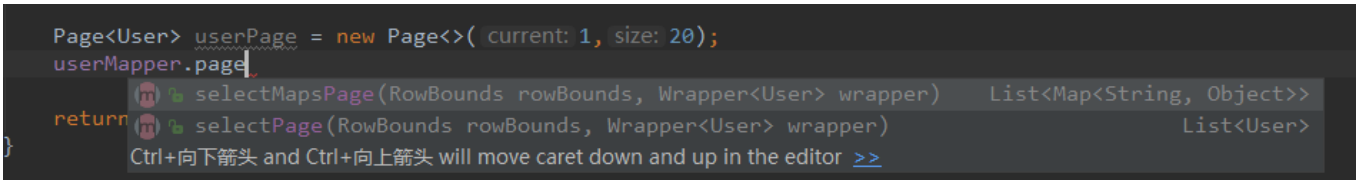
也可参考视频教程第`7节 多数据源配置和使用`，内有详细的讲解

3.5 如何分页

Guns的分页是通过mybatis-plus的分页插件实现的，大体分如下两种情况

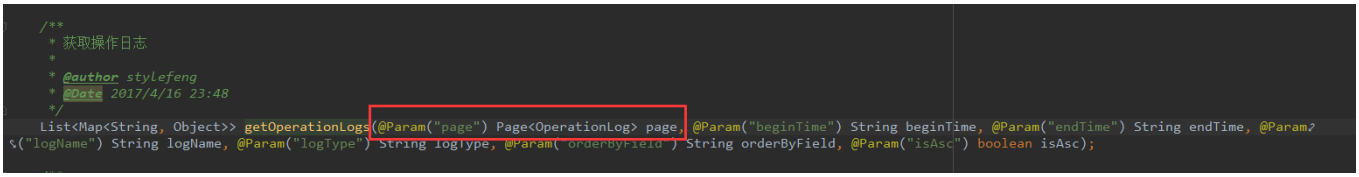
3.5.1 简单查询的分页

如果查询结果为单表查询，例如查询用户列表，则可以调用mybatis plus的自动生成的mapper中的`selectPage()`或者`selectMapsPage()`方法，`Page`类的构造函数中第一个参数为当前查询第几页，第二个参数为每页的记录数。



3.5.2 复杂查询的分页

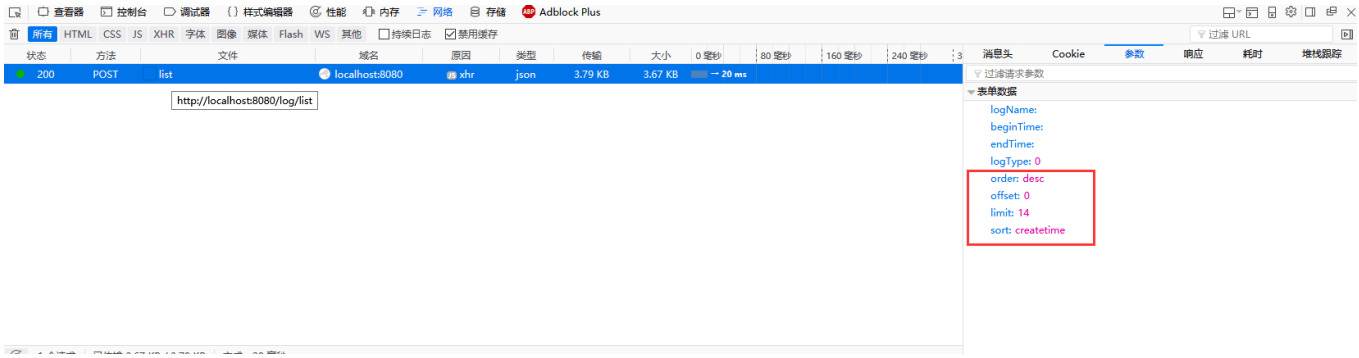
若查询结果是关联多个表的操作，则需要用到自定义的mapper，此时的分页操作也很简单，只需要给mapper的第一个参数设置为`Page`对象即可，例如Guns中`LogController`中的查询`操作日志列表`，用的就是复杂查询的分页，我们可以看到在mybatis接口的第一个参数中，传递了`Page`对象，如下



当mybatis执行此方法的时候，会被mybatis-plus的分页插件自动拦截到，并且把分页查询的结果返回到这个`Page`对象中!

3.5.3 获取前端表格插件传值

Guns中前端表格用的Bootstrap Table插件，在前端执行查询时，插件会自动往后台传递分页参数，并且默认的格式如下，



`Bootstrap Table`会传递`order(升序或者降序)`，`offset(每页偏移量)`，`limit(每页条数)`，`sort(排序的字段)`这四个参数，与之对应，后台封装了一个通用的接受分页参数的类`PageFactory`，从而不用每次都来`request.getParameter()`这样接收参数，如下所示，

```
1.  public class PageFactory<T> {
2.
3.      public Page<T> defaultPage() {
4.          HttpServletRequest request = HttpKit.getRequest();
5.          int limit = Integer.valueOf(request.getParameter("limit"));
    //每页多少条数据
6.          int offset = Integer.valueOf(request.getParameter("offset"));
    //每页的偏移量(本页当前有多少条)
7.          String sort = request.getParameter("sort");          //排序字段名
    称
8.          String order = request.getParameter("order");        //asc或
    desc(升序或降序)
9.          if (ToolUtil.isEmpty(sort)) {
10.             Page<T> page = new Page<>((offset / limit + 1), limit);
11.             page.setOpenSort(false);
12.             return page;
13.         } else {
14.             Page<T> page = new Page<>((offset / limit + 1), limit, sort
    );
15.             if (Order.ASC.getDes().equals(order)) {
16.                 page.setAsc(true);
17.             } else {
18.                 page.setAsc(false);
19.             }
20.             return page;
21.         }
22.     }
23. }
```

在后台代码中如需接收参数，构建分页Page对象的时候，只需如下这样一调用即可构建分页对象

```
Page<OperationLog> page = new PageFactory<OperationLog>().defaultPage();
```

3.6 数据范围

3.6.1 介绍

Guns的数据范围是指当前部门的用户可以看到当前部门和子部门的数据，子部门的数据不可以看到上级部门的数据，但`超级管理员`例外，例如，`userA`和`userB`两个用户都有查看用户列表的权限，但是`userA`在总公司部门，`userB`在运营部，他们有如下部门关系

那么`userA`在查看用户列表的时候能看到`公司所有人`的数据，`userB`只能看到`运营部`的数据，这就是数据范围!

3.6.2 如何使用

使用时，只需要`new`一个`DataScope`，并在构造方法中传递给当前用户用后的部门权限(一般我们用封装好的`ShiroKit.getDeptDataScope()`方法即可获取到当前用户的部门权限集合)，之后，传递给mybatis的dao方法的第一个参数即可，例子如下

```
DataScope dataScope = new DataScope(ShiroKit.getDeptDataScope());
List<Map<String, Object>> users = managerDao.selectUsers(dataScope, name, beginTime, endTime,  deptid);
```

注意: 在使用过程中，原mybatis的dao方法的查询结果中必须包含`deptid字段(默认情况)`，若部门id不叫deptid也可也初始化`DateScope`对象的时候，修改该对象的`scopeName`属性，改为自定义的部门id字段名即可

3.6.3 原理

数据范围的原理是利用了`mybatis拦截器`，类似于mybatis-plus的分页插件，在原查询结果之上包装了一层`select筛选查询`，如下

```
1.  select (原语句字段) from (原语句) where deptid in (DataScope对象中包含的部门id列表)
```

若想深入了解数据范围的编写过程和原理可参考视频教程第`15节 数据范围使用和原理`，内有详细的讲解

3.7 guns-rest模块的使用

3.7.1 关于jwt鉴权

在了解guns-rest模块的使用之前，需要了解一下jwt鉴权机制，下面给出一些参考资料

> - 什么是JWT-JSON WEB TOKEN -> https://www.jianshu.com/p/576dbf44b2ae

说白了就是如果想请求服务器资源，需要先走服务器的auth接口，用账号和密码换取token，之后每个接口的请求都需要带着token去访问，否则就是鉴权失败.

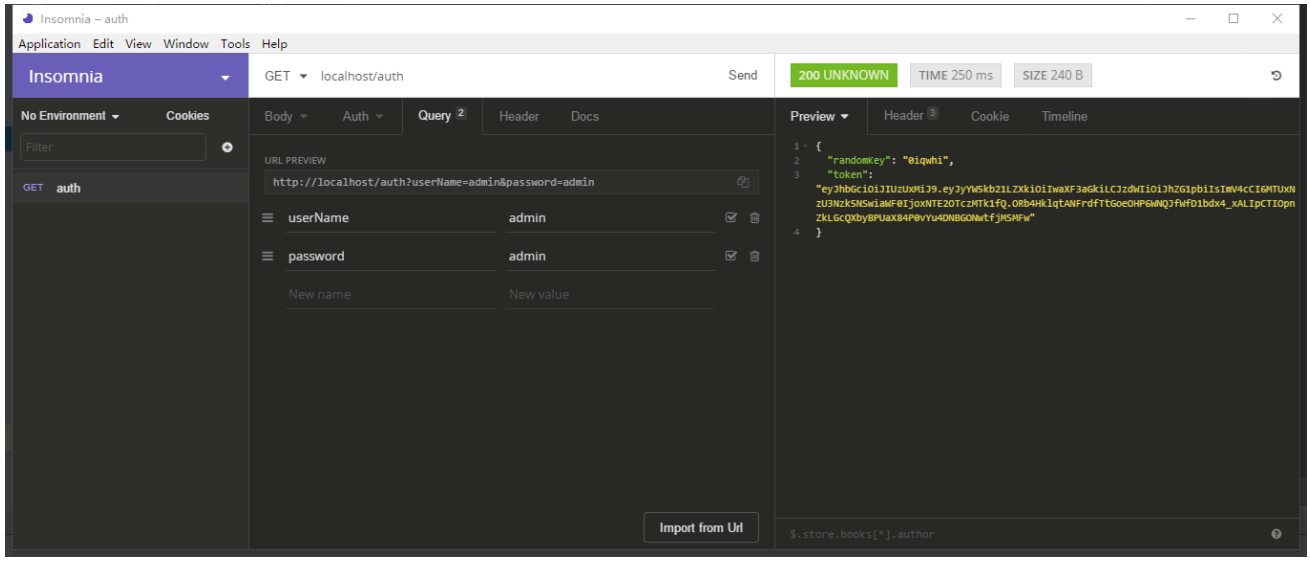
3.7.2 关于传输数据的签名

签名机制是指客户端向服务端传输数据中，对传输数据进行md5加密，并且加密过程中利用Auth接口返回的随机字符串进行混淆加密，并把md5值同时附带给服务端，服务端通获取数据之后对数据再进行一次md5加密，若加密结果和客户端传来的数据一致，则认定客户端请求的数据是没有被篡改的，若不一致，则认为被加密的数据是被篡改的.

3.7.3 guns-rest模块的运行流程

1. 执行`guns-rest`模块下的db文件夹的sql初始化脚本`guns_rest.sql`
2. 启动`guns-rest`模块

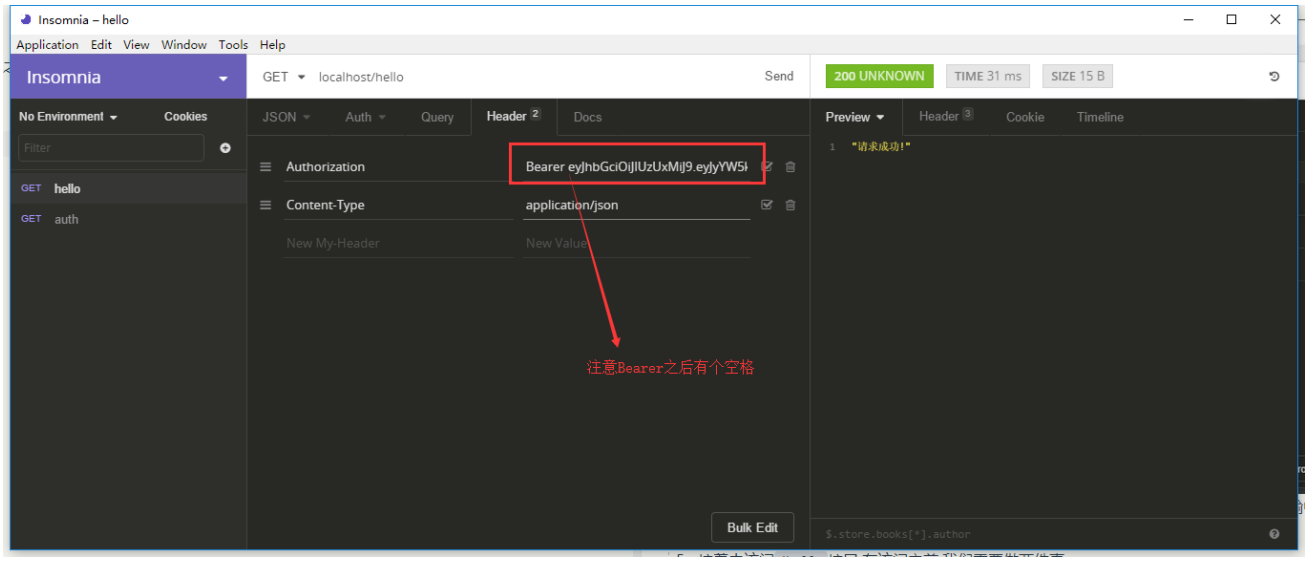
3. 下载postman接口测试工具或者insomnia接口测试工具，下面以insomnia接口测试工具为例，演示rest模块资源访问流程
4. 访问/auth接口，传递给接口账号密码获取访问接口用的token，如下



接口请求成功，auth接口返回给两个属性的json，`randomKey`的作用是在之后接口的数据传输中对数据做MD5混淆加密用的，`token`的作用是在之后访问资源的过程中，携带到请求的header中，证明我们是有权限访问资源的

5. 接着去访问`/hello`接口，在访问之前，我们需要做两件事：

第一 把请求hello接口的请求头Header中带一个`Authorization`属性，属性的值为`Bearer`和`token`值，注意中间用空格隔开



第二 `/hello`接口的所需要一个`@RequestBody`类型的数据，所以我们还需要传给这个接口一个json数据

```java
@RequestMapping("")
public ResponseEntity hello(@RequestBody SimpleObject simpleObject) {
    System.out.println(simpleObject.getUser());
    return ResponseEntity.ok("请求成功!");
}
```

注意 json数据不能直接为如下的形式

```
{"name":"ffff","user":"stylefeng","age":12,"tips":"code"}
```

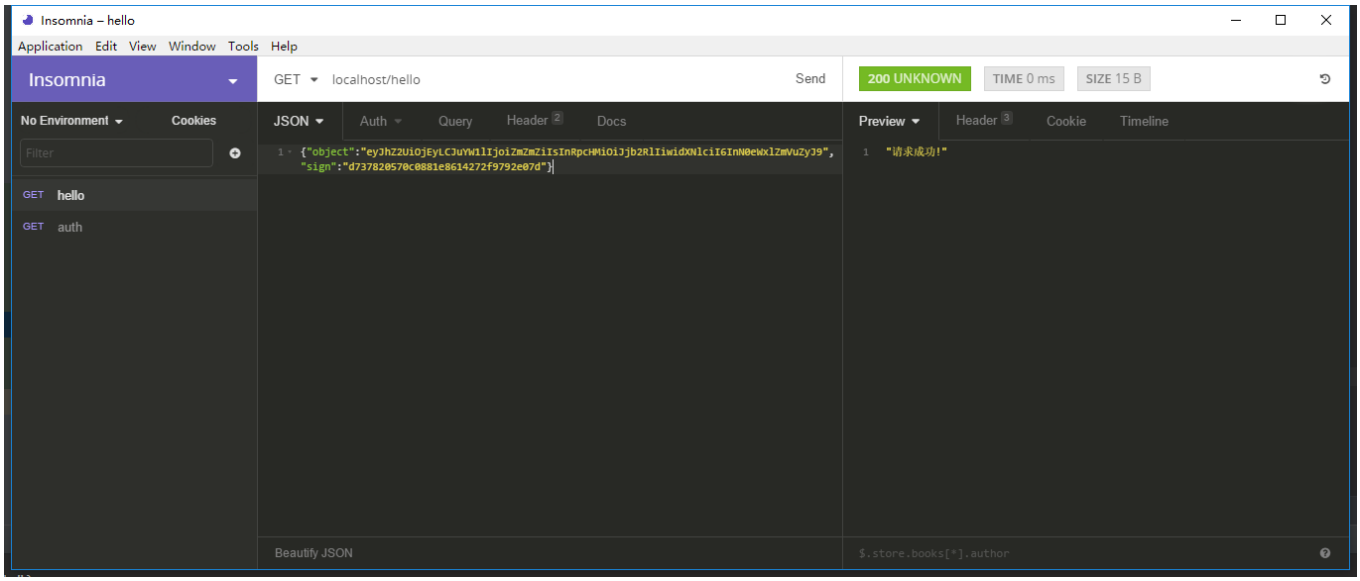
为了保证传输的数据的安全性，Guns做了对传输数据的签名，所以传输过程中需要对数据进行签名，我们可以直接运行`DecryptTest`这个测试类，直接生成签名好的json数据，如下

```java
public class DecryptTest {

    public static void main(String[] args) {

        String salt = "0iqwhi";    /auth接口获取的randomKey

        SimpleObject simpleObject = new SimpleObject();
        simpleObject.setUser("stylefeng");
        simpleObject.setAge(12);
        simpleObject.setName("ffff");
        simpleObject.setTips("code");

        String jsonString = JSON.toJSONString(simpleObject);
        String encode = new Base64SecurityAction().doAction(jsonString);
        String md5 = MD5Util.encrypt( source: encode + salt);

        BaseTransferEntity baseTransferEntity = new BaseTransferEntity();
        baseTransferEntity.setObject(encode);
        baseTransferEntity.setSign(md5);

        System.out.println(JSON.toJSONString(baseTransferEntity));
    }
```

这里注意填写md5的加密盐为刚才/auth接口生成的randomKey，运行后生成如下json

```
{"object":"eyJhZ2UiOjEyLCJuYW1lIjoiZmZmZiIsInRpcHMiOiJjb2RlIiwidXNlciI6InN0eWxlZmVuZyJ9","sign":"d737820570c0881e8614272f9792e07d"}
```

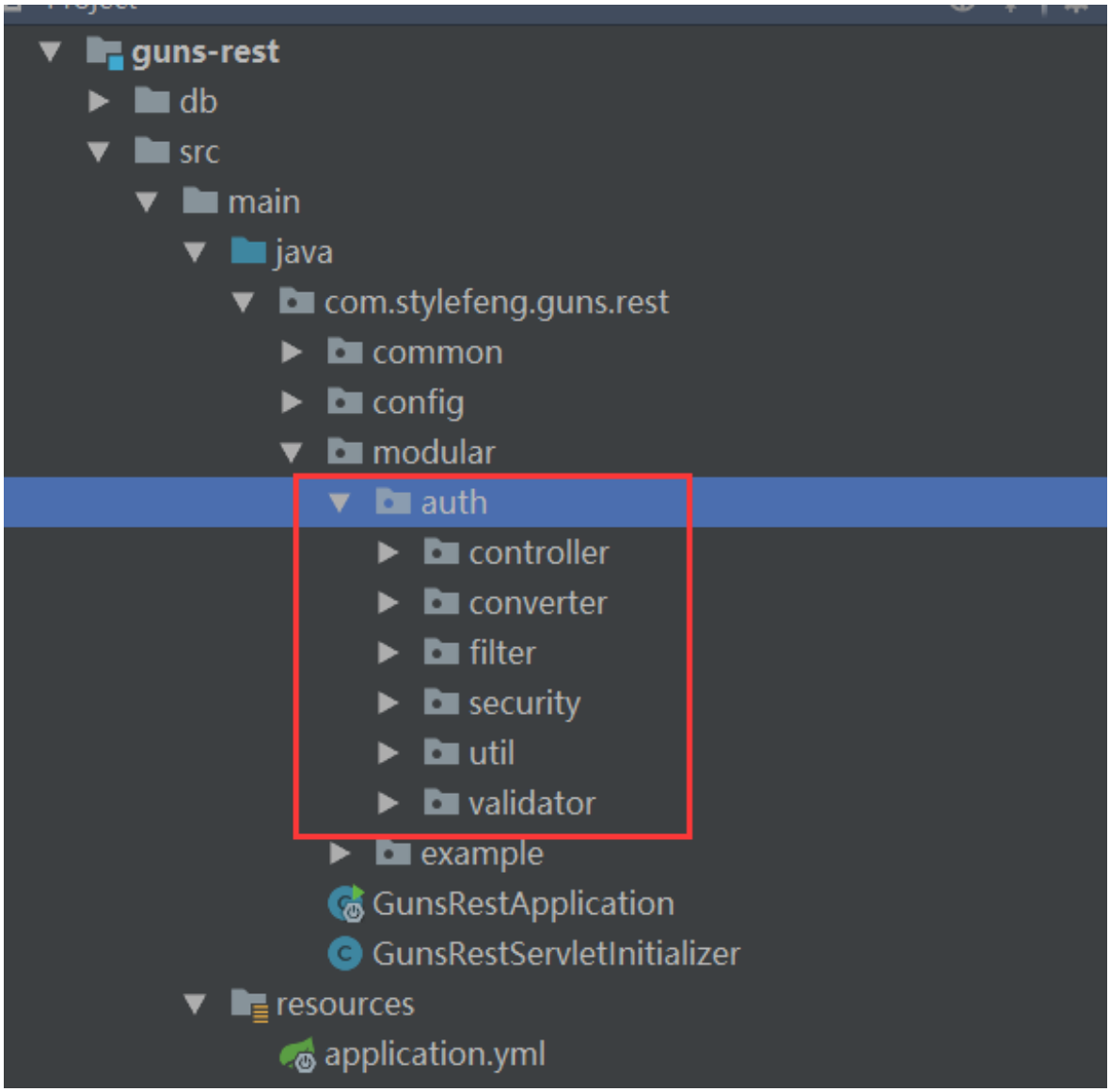
我们填入到接口的`请求体`里，并点击`Send`



接口访问成功!

3.7.4 运行原理

关于rest模块鉴权运行原理，其实就是一个简单的过滤器`AuthFilter类`实现的，若想了解运行机制可以查看下`auth包`下的类的代码(几十行)



3.8 工作流

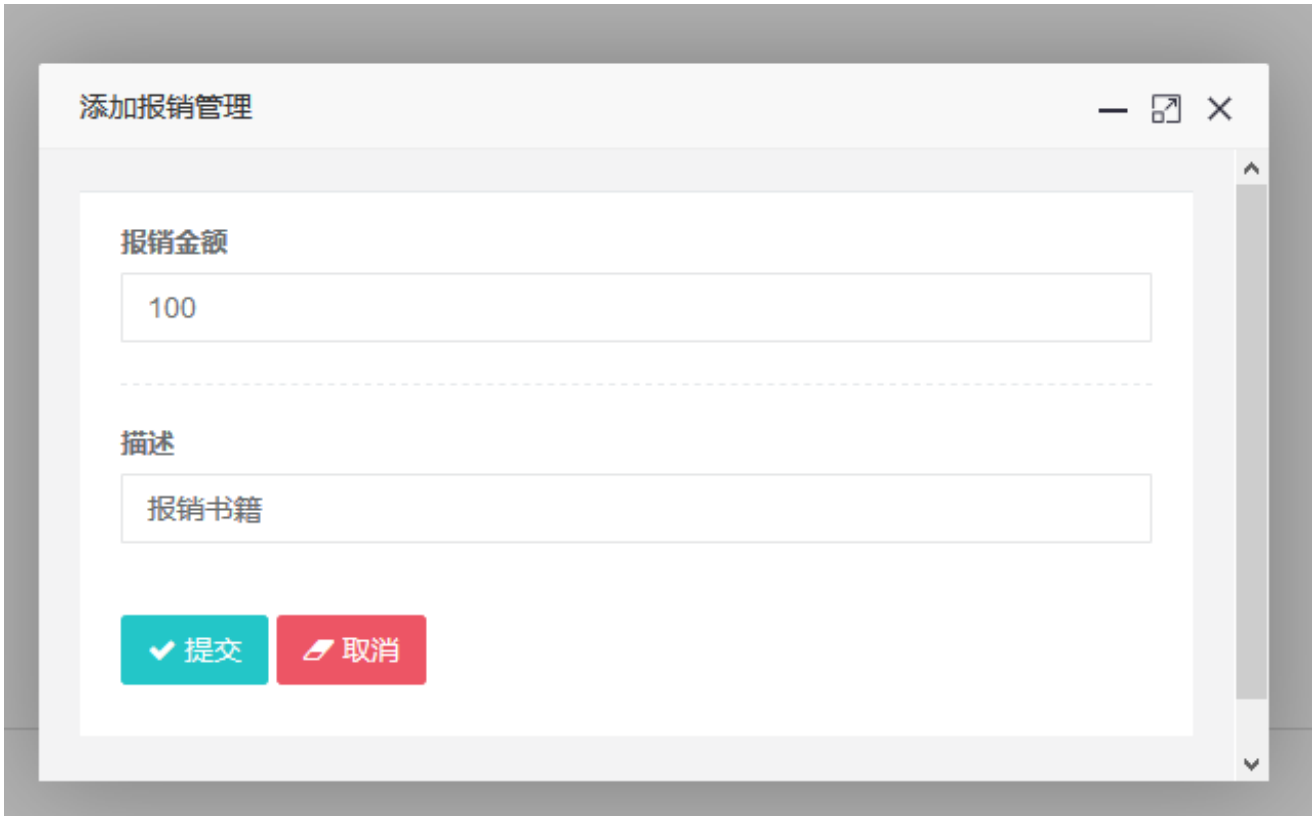
最新的Guns 3.1版本引入了工作流框架flowable 6.2.0，并自带一个报销流程供大家参考，下面做一下介绍

为了不和guns的数据库混淆，guns新建了一个数据库`guns_flowable`，并配置了一个`单独的数据源`来连接该数据库，在application.yml中的配置如下

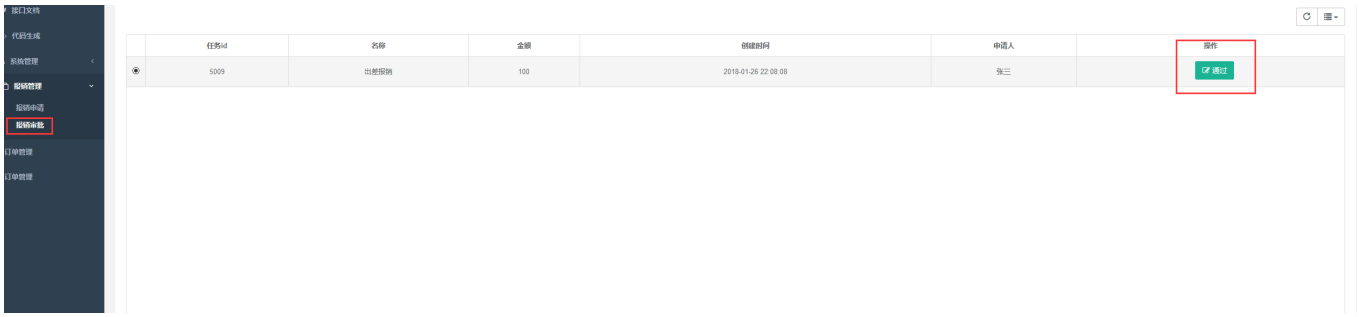
```
#flowable数据源和多数据源配置
guns:
  flowable:
    datasource:
      url: jdbc:mysql://127.0.0.1:3306/guns_flowable?autoReconnect=true&useUnicode=true&characterEncoding=utf8&zeroDateTimeBehavior=convertToNull&useSSL=false
      username: root
      password: root
  muti-datasource:
    default-dataSource-name: dataSourceGuns #默认的数据源名称
    url: jdbc:mysql://127.0.0.1:3306/guns_flowable?autoReconnect=true&useUnicode=true&characterEncoding=utf8&zeroDateTimeBehavior=convertToNull&useSSL=false
```

在guns启动过程中，若`guns_flowable`数据库没有表，flowable引擎会自动初始化工作流需要的表

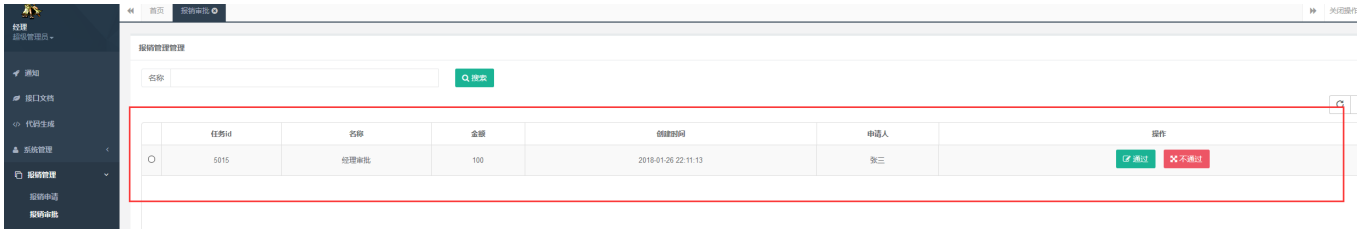
在报销管理业务中，一共有三个角色，`申请人`(账号:admin)，`经理`(账号:manager)，`老板`(账号:boss)，他们的密码都是`111111`，首先申请人填写报销单，



填写之后需要在`报销审批`菜单中，提交下自己的申请



如果报销金额小于500则是`经理(manager)`审批，我们登录经理的号，可以看到申请记录



这里点击`通过`，则该流程结束，如果点`不通过`则还需要申请人重新提交申请

关于工作流的开发，可以参考flowable官方文档

3.9 日志记录

在我们日常开发中，对于某些关键业务，我们通常需要记录该操作的内容，例如修改了什么数据，修改的内容是什么，删除了哪些数据等等，在Guns中有一整套完善的解决方案来完成此项功能

3.9.1 业务日志

我们通过`@BussinessLog注解`来记录日志，该注解源码如下，

```
/**
 * 标记需要做业务日志的方法
 *
 * @author fengshuonan
 * @date 2017-03-31 12:46
 */
@Inherited
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.METHOD})
public @interface BussinessLog {

    /**
     * 业务的名称,例如:"修改菜单"
     */
    String value() default "";

    /**
     * 被修改的实体的唯一标识,例如:菜单实体的唯一标识为"id"
     */
    String key() default "id";

    /**
     * 字典(用于查找key的中文名称和字段的中文名称)
     */
    Class<? extends AbstractDictMap> dict() default SystemDict.class;
}
```

其中，`value`为需要记录日志的业务名称，`key`为修改或删除内容的唯一标识，通过这个唯一标识可以知道具体的修改的哪条记录，删除的哪条记录等等，`dict`为对修改字段的中文翻译

字典，因为程序记录的都是英文的字段名称，这里通过字典，把英文字段和中文名称对应起来，那么日志信息记录到数据库中就可以变为中文的记录

以`UserDict`为例，

```
/**
 * 用户的字典
 *
 * @author fengshuonan
 * @date 2017-05-06 15:01
 */
public class UserDict extends AbstractDictMap {

    @Override
    public void init() {
        put("userId","账号");
        put("avatar","头像");
        put("account","账号");
        put("name","名字");
        put("birthday","生日");
        put("sex","性别");
        put("email","电子邮件");
        put("phone","电话");
        put("roleid","角色名称");
        put("deptid","部门名称");
        put("roleIds","角色名称集合");
    }

    @Override
    protected void initBeWrapped() {
        putFieldWrapperMethodName("sex","getSexName");
        putFieldWrapperMethodName("deptid","getDeptName");
        putFieldWrapperMethodName("roleid","getSingleRoleName");
        putFieldWrapperMethodName("userId","getUserAccountById");
        putFieldWrapperMethodName("roleIds","getRoleName");
    }
}
```

翻译字典类中包含两个方法`init()`和`initBeWrapped()`，其中`init()`为存放英文字段和中文字段的匹配，`initBeWrapped()`操作的是把某些字段的数字值翻译为中文直观名称的过程，例如当修改用户信息时，用户修改了一个人性别信息(数据库中1是男，2是女)，由1变为了2，程序记录的是数据库中1变为2，但是这句话给业务人员看到他是不知道1和2是什么东西

的，所以这里做了一个`值的包装`，把`1`包装为对应的中文名称`男`，`2`包装为对应的中文名称`女`，这样，记录到数据库中，信息就变为了，xxx用户操作了`修改用户`功能，值由`男`变为了`女`.
在initBeWrapped()方法中`putFieldWrapperMethodName()`这个方法的第一参数是被包装的字段名，第二个参数是`ConstantFactory`中的方法名，因为默认会调用`ConstantFactory`来包装值属性

下面介绍业务日志记录的具体步骤:

> - 1.在需要被记录日志的接口上添加@BussinessLog注解，并根据需要填写三个属性(value，key，dict)
> - 2.若是添加或者修改业务，往往需要去编写Dict字典类
> - 3.若是修改业务，例如修改用户信息，因为点击更新用户的时候不会提交修改之前的数据，所以在更新用户信息之前需要保存一下用户的旧的信息才可以记录用户修改的内容，这个缓存用户临时信息的地方一般添加在跳转到用户详情接口，用`LogObjectHolder.me().set(user);`这行代码来缓存用户的旧的信息，具体用法可以参考`UserMgrController`类中的`userEdit()`和`edit()`

3.9.2 异常日志

由于Guns有统一的异常拦截器，一般程序的报错，不管是业务异常还是未知的RuntimeException都会拦截并记录到数据库，若是您有自己的异常日志需要记录到数据库或者日志文件，推荐如下做法

1. 如果记录到数据库，调用Guns的日志记录工具类，如下

```
1.  LogManager.me().executeLog();
```

该方法为异步记录日志的方法，executeLog()方法中需要传递一个`TimerTask`对象，TimerTask对象可以用`LogTaskFactory`类创建，在`LogTaskFactory`类中，有5个方法，可以分别记录不用的日志，有`登录日志`，`退出日志`，`业务日志`，`异常日志`等等，可以自行选择调用
2. 若需要记录日志到文件中，可以采用slf4j的`org.slf4j.Logger`类记录，具体方法如下

```
//首先在类中初始化
private Logger log = LoggerFactory.getLogger(this.getClass());

//再在方法中调用
log.error("业务异常:", e);
```

3.10 如何使用缓存

在Guns中使用缓存的地方不多，主要在ConstantFactory的查询中用了缓存，在ConstantFactory有高频调用的查询，所以在这些方法上加了缓存，搜索加上缓存后还要注意在修改了相关数据的时候要删除缓存，否则可能导致数据的不一致，在Guns中默认用的是Ehcache缓存，并配合了spring cache使用，用spring cache的好处就是，spring cache是缓存的抽象，如果想换为redis缓存，则不用修改代码，改一下配置即可实现，下面介绍两种操作缓存的方法

3.10.1 用工具类操作

在guns-core中封装了一些常用的操作Ehcache缓存的工具类`CacheKit`，此类采用静态方法调用的方式，可以添加，获取，删除缓存，用法非常简单

```
//添加缓存，第一个参数为缓存的名称，是ehcache.xml中<cache>节点的NAME，key为添加缓存的键值，value为缓存的值
public static void put(String cacheName, Object key, Object value);

//获取某个缓存名称中的某个键值对应的缓存
public static <T> T get(String cacheName, Object key);

//获取某个缓存的所有key
public static List getKeys(String cacheName);

//删除某个key对应的缓存
public static void remove(String cacheName, Object key);

//删除某个缓存名称下的所有缓存
public static void removeAll(String cacheName);
```

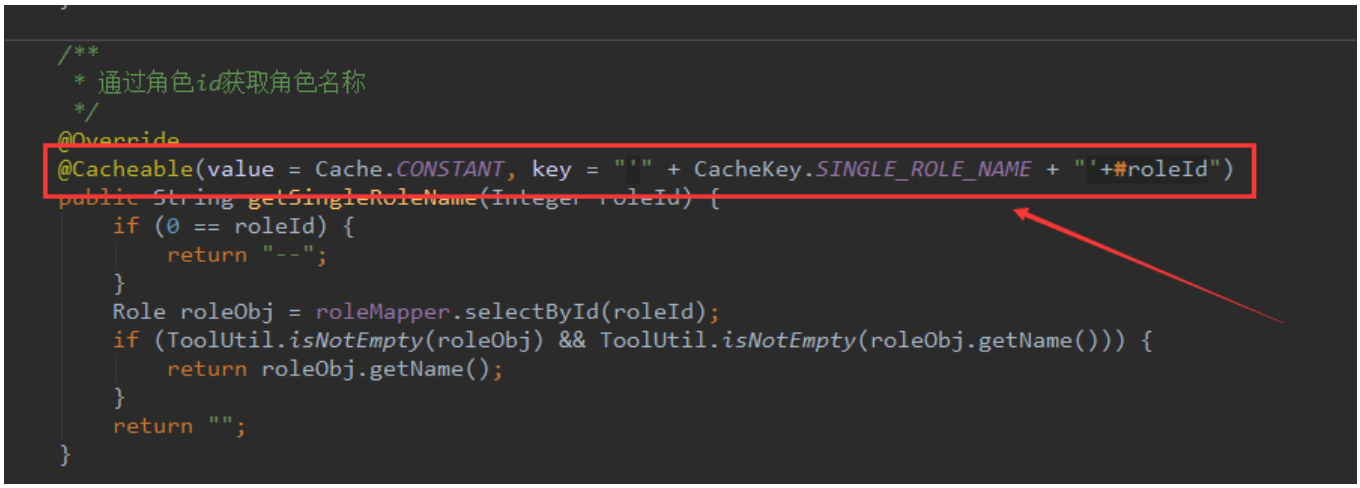
3.10.2 用spring cache操作缓存

利用spring cache来操作缓存，可以很方便的在redis和ehcache之间切换缓存实现，利用spring cache 的缓存注解，加到方法之上可以很方便的缓存方法的结果，如果参数对应的键

值存在了缓存，则下一次走这个方法则会直接返回缓存的结果，spring cache提供了4个注解来操作缓存.

> - 1.@Cacheable表明在调用方法之前，首先应该在缓存中查找方法的返回值，如果这个值能够找到，则会返回缓存的值，否则执行该方法，并将返回值放到缓存中，一般在数据库查询(`select`)之后调用这个注解
> - 2.@CachePut表明在方法调用前不会检查缓存，方法始终都会被调用，调用之后把结果放到缓存中，一般在数据库操作插入数据(`save`)的时候调用
> - 3.@CacheEvict表明spring会清除一个或者多个缓存，一般在数据库更新或者删除数据的时候调用(`update`或者`delete`)
> - 4.@Caching分组的注解，可以同时应用多个其他缓存注解，可以相同类型或者不同类型

一般在用这些注解的时候，我们需要填写两个参数，一个是`value`代表缓存的名称，一个是`key`代表缓存的键值



如上图所示，键值`key`一般包含两部分组成，一部分是`键的标识`例如上图中的`CacheKey.SINGLE_ROLE_NAME`，一部分是`参数`(一般是参数的值)例如上图中的`#roleId`

3.11 使用枚举

在Guns中，枚举一般分两类，一种是状态枚举，一种是异常枚举，`状态枚举`的作用是枚举状态，列出状态的所有值，例如

```
1.  /**
```

```
 * 菜单的状态
 *
 * @author fengshuonan
 * @Date 2017年1月22日 下午12:14:59
 */
public enum MenuStatus {

    ENABLE(1, "启用"),
    DISABLE(0, "禁用");

    int code;
    String message;

    MenuStatus(int code, String message) {
        this.code = code;
        this.message = message;
    }

    ...
}
```

异常枚举的作用是枚举所有出现的业务异常，例如，

```
/**
 * 所有业务异常的枚举
 *
 * @author fengshuonan
 * @date 2016年11月12日 下午5:04:51
 */
public enum BizExceptionEnum implements ServiceExceptionEnum{

    /**
     * 错误的请求
     */
    SESSION_TIMEOUT(400, "会话超时"),
    SERVER_ERROR(500, "服务器异常");

    BizExceptionEnum(int code, String message) {
        this.code = code;
        this.message = message;
    }

    private Integer code;

```

```
22.     private String message;
23.
24.     ...
25. }
```

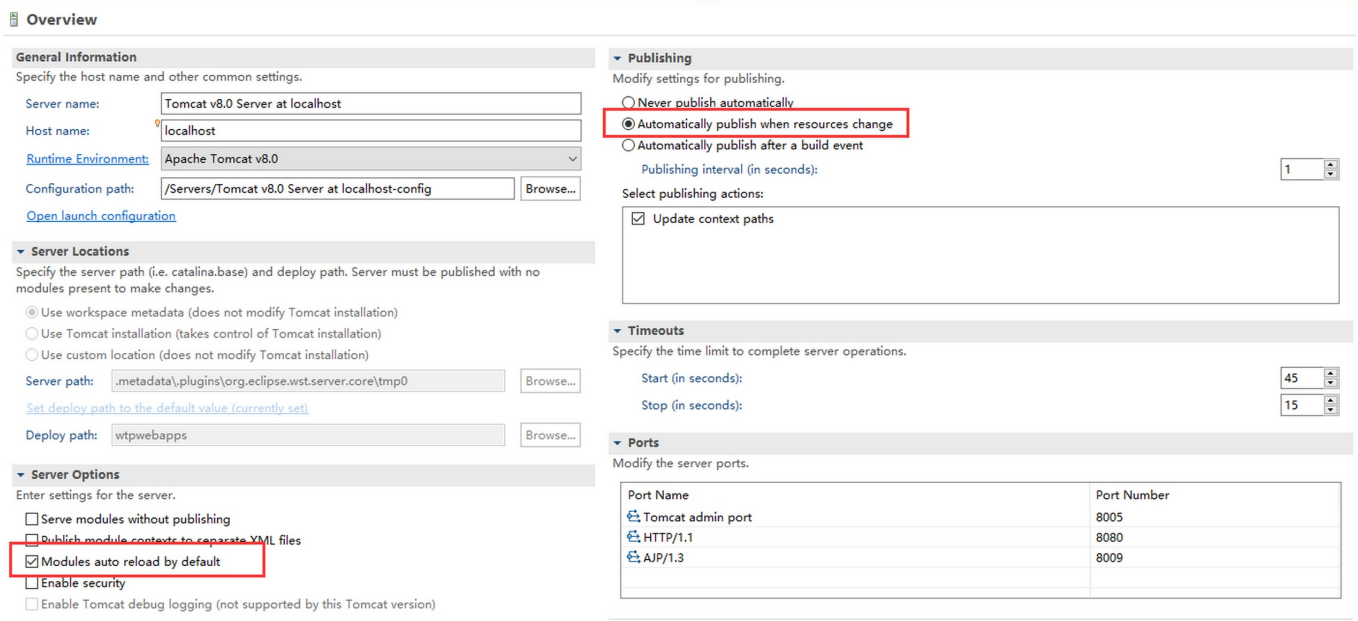
使用枚举可以方便维护一些状态的值和管理所有的业务异常，所以在有`状态`或者新的`业务异常`的时候推荐写到枚举里

3.12 spring boot热部署

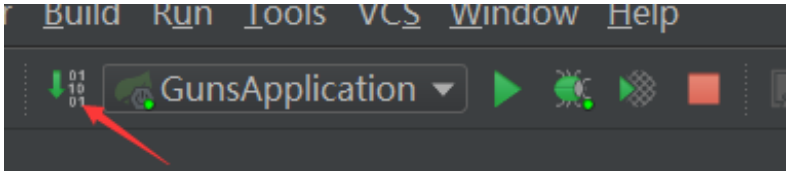
热部署的两种情况(适用于main方法启动)

3.12.1 重新加载html

如果是eclipse修改html保存后可以自动替换，如果不能请检查server配置



如果是IDEA，可以修改html之后点击这个按钮，或者按快捷键CTRL+F9，即可更新



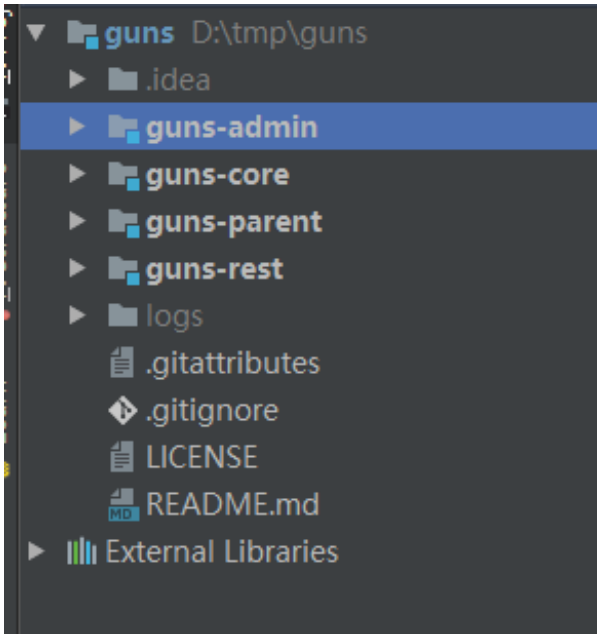
3.12.2 重新加载java类

如果是eclipse，在application.yml中找到配置spring.devtools.restart.enabled改为true即可

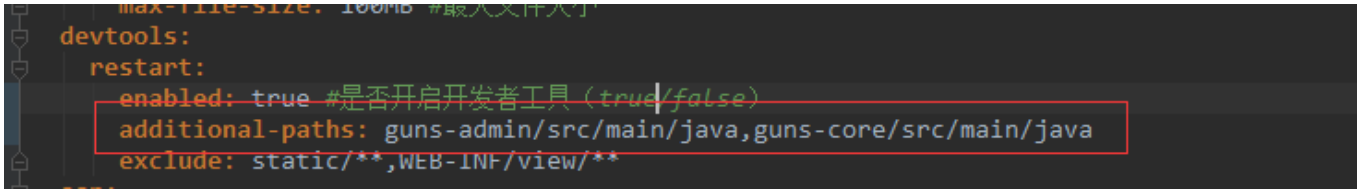
如果是在IDEA中:

第一步 请先修改spring.devtools.restart.enabled=true

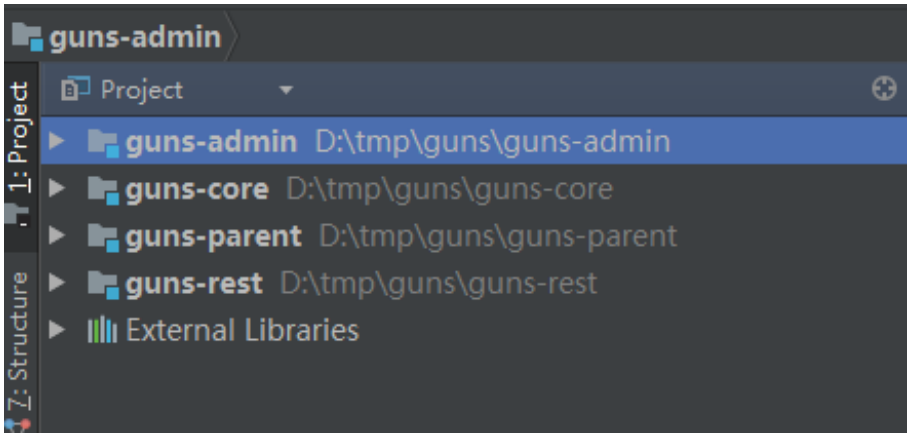
第二步 如果项目以guns根目录形式导入，例如下面所示情况



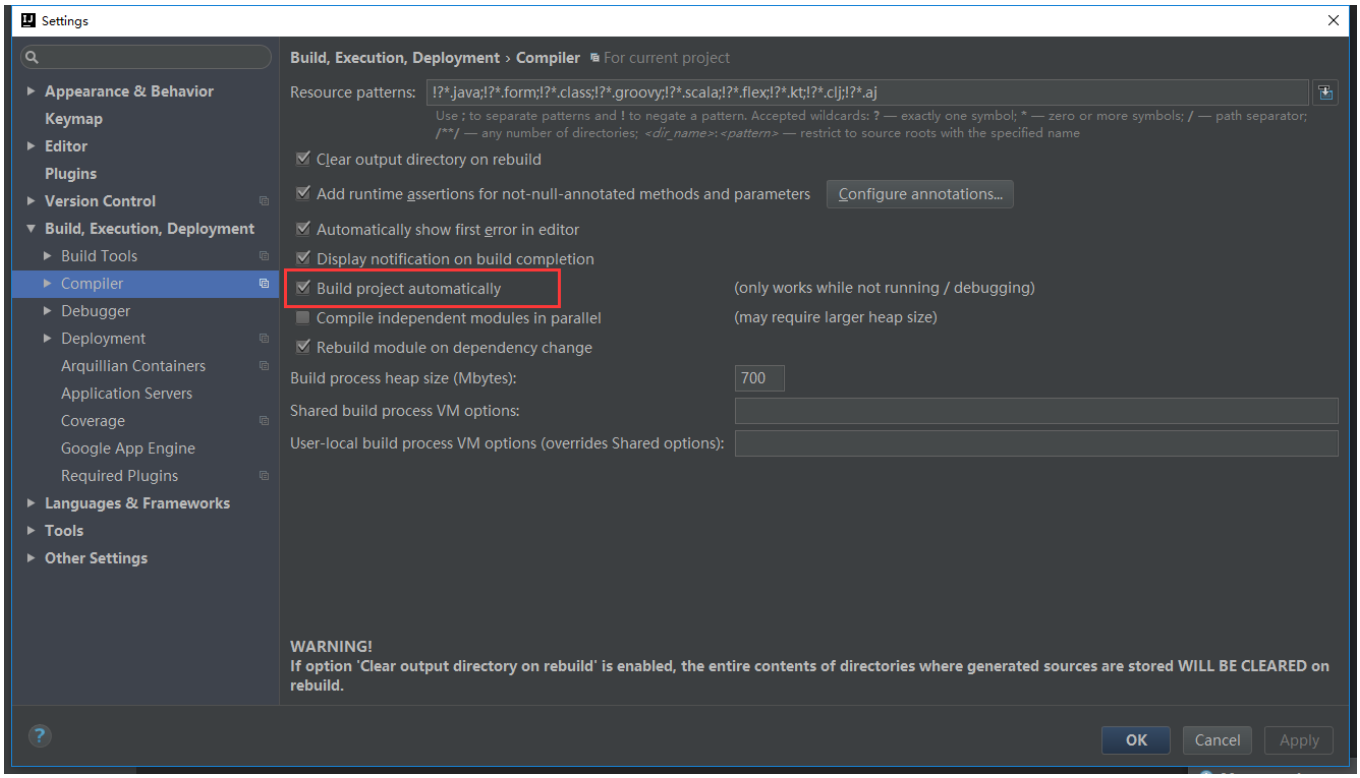
需要修改application.yml中spring.devtools.restart.additional-paths添加上相应的项目名称，如下所示:



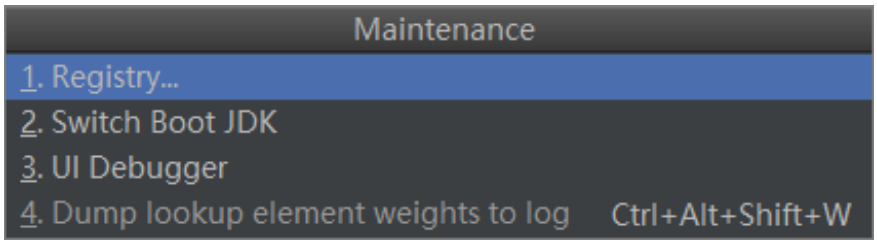
如果项目以guns-parent形式导入，例如下面所示情况，则不需要修改additional-paths



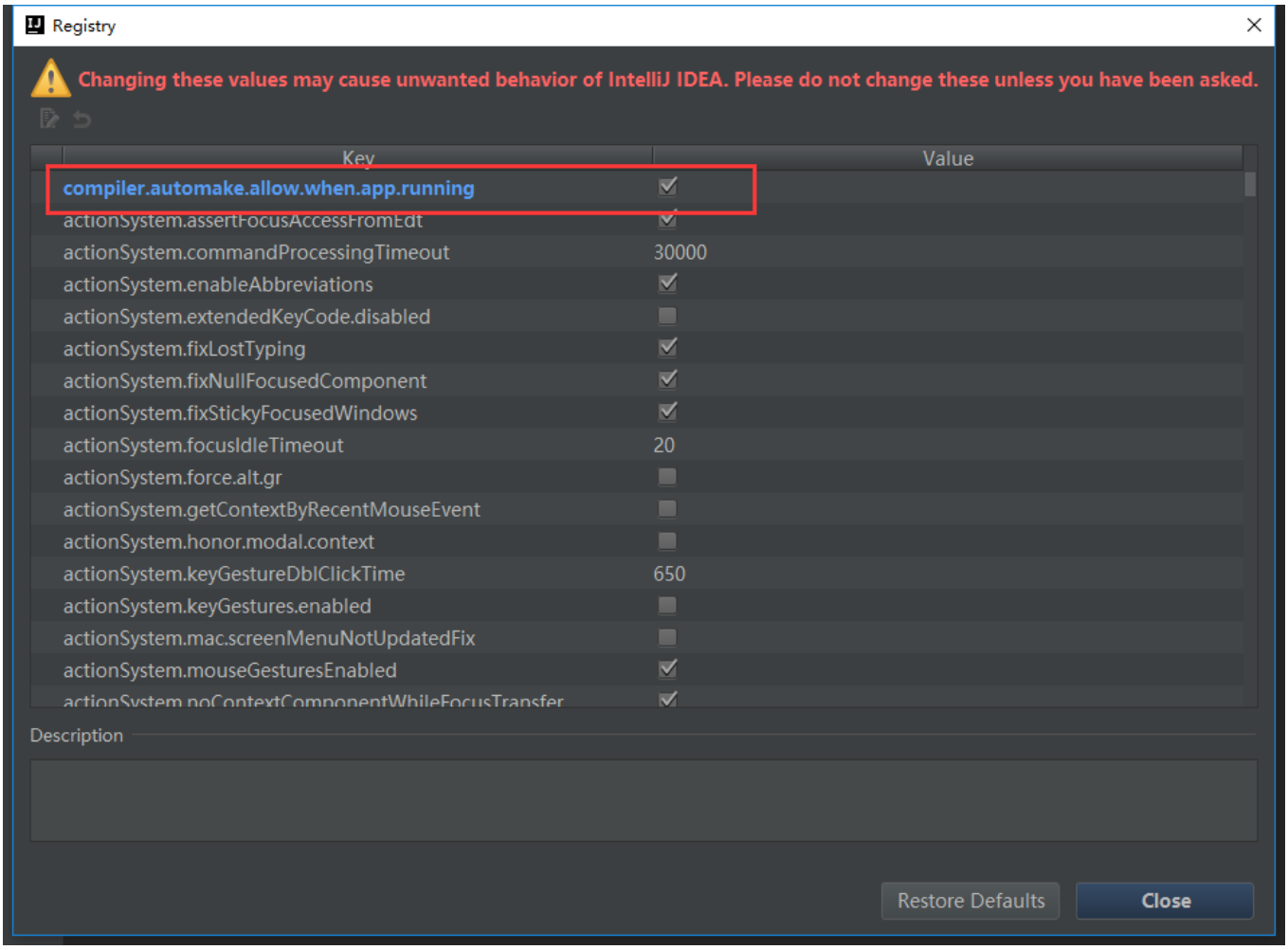
第三步 如下idea配置，打上对勾



第四步 按下 Shift+Ctrl+Alt+/，选择Registry



进去之后，找到如下图所示的选项，打勾

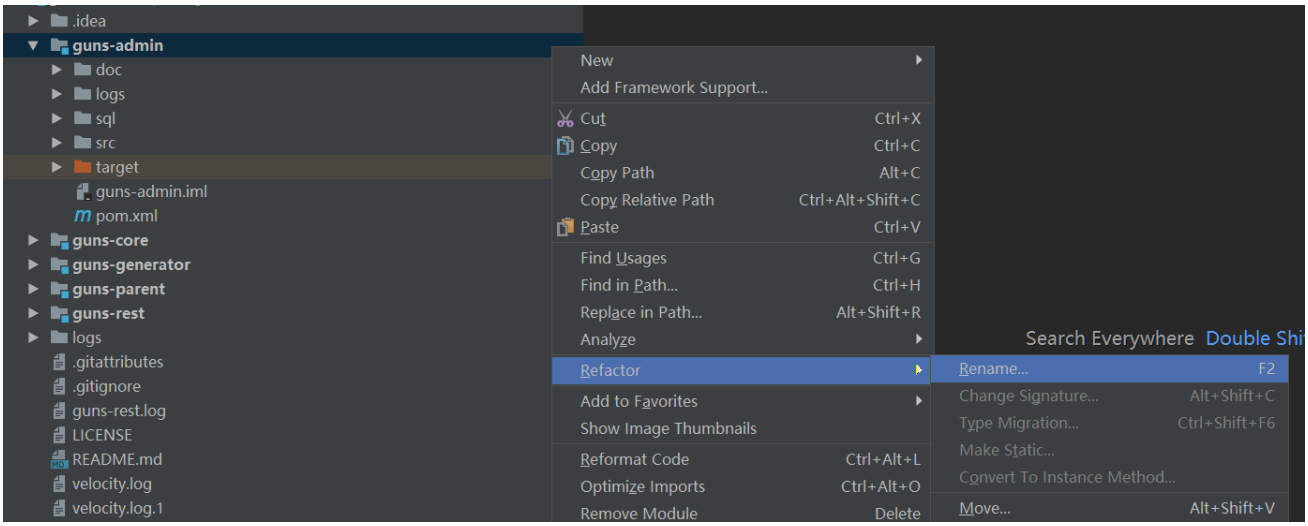


4. 扩展与高级配置

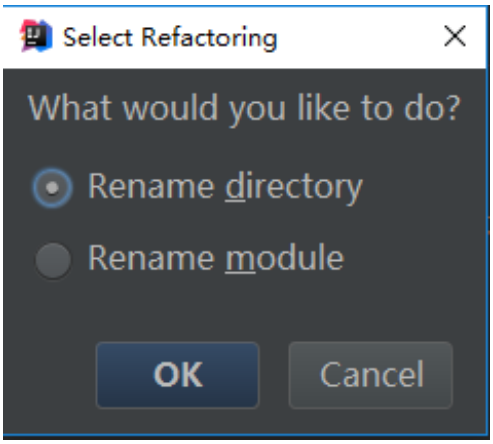
4.1 修改项目名和包名

4.1.1 修改项目名

1. 以guns-admin在idea环境下为例，右击项目，点`refactor->Rename`



2. 选`Rename directory`



3. 修改`guns-parent\pom.xml`改为`guns-myadmin`

```
<modules>
    <module>../guns-myadmin</module>
    <module>../guns-core</module>
    <module>../guns-rest</module>
    <module>../guns-generator</module>
</modules>
```

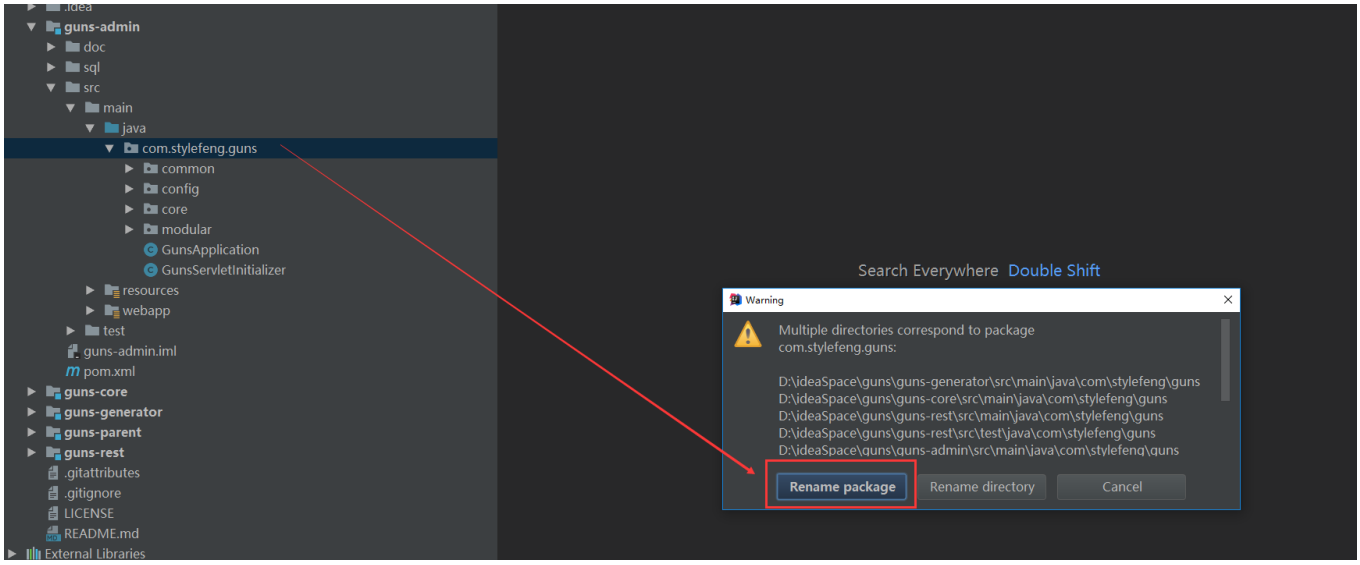
4. 修改`guns-myadmin\pom`的artifactId改为`guns-myadmin`

```
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
<modelVersion>4.0.0</modelVersion>
<parent>
    <groupId>com.stylefeng</groupId>
    <artifactId>guns-parent</artifactId>
    <version>1.0.0-SNAPSHOT</version>
    <relativePath>../guns-parent/pom.xml</relativePath>
</parent>

<artifactId>guns-myadmin</artifactId>
<version>1.0.0-SNAPSHOT</version>

<packaging>jar</packaging>

<name>guns-admin</name>
<description>guns 的spring boot版本</description>
```

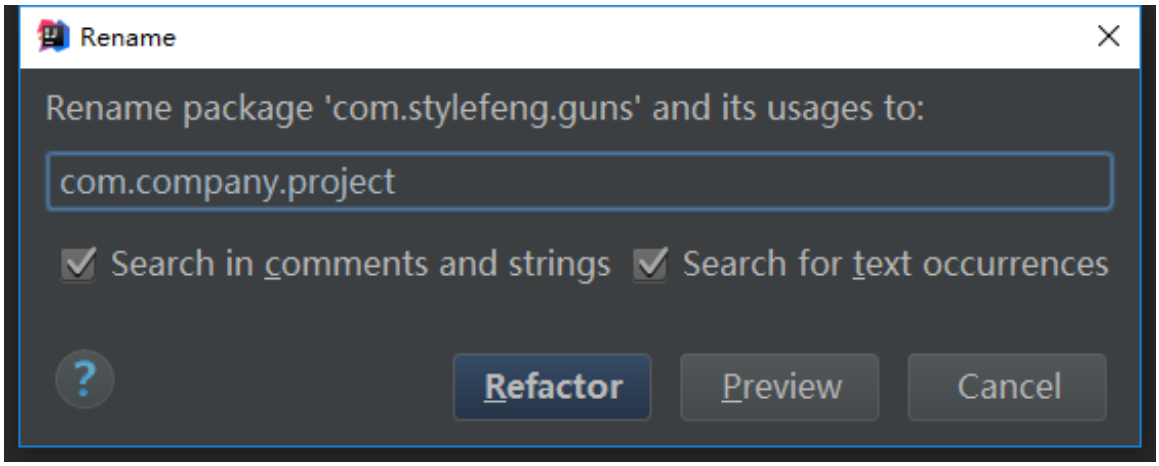
4.1.2 修改包名

下面以把`com.stylefeng.guns`改为`com.company.project`为例

1. 选择`com.stylefeng.guns`包，仍然为右键`refactor->Rename`
2. 弹出对话框选择，有两个选项，一个是`Rename Package`也就是重命名所有模块中`com.stylefeng.guns`的包，一个是`Rename Directory`只重命名当前模块的`com.stylefeng.guns`包，我们选`Rename Package`



3. 输入`com.company.project`点Refactor，之后稍等几分钟



4. 把所有包扫描的地方改为`com.company.project`，首先改`application.yml`中的mybatis-plus的配置

```
    ###################  mybatis-plus配置  ###################
  mybatis-plus:
    mapper-locations: classpath*:com/company/project/**/mapping/*.xml
    typeAliasesPackage: com.company.project.common.persistence.model
    typeEnumsPackage: com.company.project.common.constant.enums
    global-config:
```

5. 修改`MybatisPlusConfig`类中的扫描注解

```
 */
@Configuration
@EnableTransactionManagement(order = 2)//由于引入多数据源，所以让spring事务的aop要在多数据源切换aop的后面
@MapperScan(basePackages = {"com.company.project.modular.*.dao", "com.company.project.common.persistence.dao"})
public class MybatisPlusConfig {
```

6. 修改`MultiSourceExAop`类中AOP的扫描

```
    @Pointcut(value = "@annotation(com.company.project.core.mutidatasource.annotion.DataSource)")
    private void cut() {

    }
```

7. 修改`LogAop`类中aop的扫描

```
@Pointcut(value = "@annotation(com.company.project.common.annotion.BussinessLog)")
public void cutService() {
}
```

8. 修改`PermissionAop`中aop的扫描

```
@Component
public class PermissionAop {

    @Pointcut(value = "@annotation(com.company.project.common.annotion.Permission)")
    private void cutPermission() {

    }
```

9. 修改所有mybatis的mapping.xml中的类配置，例如如下配置中的`namespace`和`type`

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.company.project.common.persistence.dao.UserMapper">

    <!-- 通用查询映射结果 -->
    <resultMap id="BaseResultMap" type="com.company.project.common.persistence.model.User">
        <id column="id" property="id" />
        <result column="avatar" property="avatar" />
        <result column="account" property="account" />
        <result column="password" property="password" />
        <result column="salt" property="salt" />
        <result column="name" property="name" />
        <result column="birthday" property="birthday" />
        <result column="sex" property="sex" />
        <result column="email" property="email" />
        <result column="phone" property="phone" />
        <result column="roleid" property="roleid" />
        <result column="deptid" property="deptid" />
        <result column="status" property="status" />
        <result column="createtime" property="createtime" />
        <result column="version" property="version" />
    </resultMap>

</mapper>
```

4.2 放过接口权限验证

在日常开发中，我们可能需要放过某个接口的权限验证，即用户不用登录就可以访问接口

1. 首先我们在BlackboardController这个类中，增加一个接口

```java
/**
 * 总览信息
 *
 * @author fengshuonan
 * @Date 2017年3月4日23:05:54
 */
@Controller
@RequestMapping("/blackboard")
public class BlackboardController extends BaseController {

    @Autowired
    NoticeDao noticeDao;

    /**
     * 跳转到黑板
     */
    @RequestMapping("")
    public String blackboard(Model model) {
        List<Map<String, Object>> notices = noticeDao.list( condition: null);
        model.addAttribute( s: "noticeList",notices);
        return "/blackboard.html";
    }

    /**
     * 测试放过登录
     */
    @RequestMapping("/test")
    @ResponseBody
    public String test(Model model) {
        return "你看,这个接口不需要登录哦!!!";
    }
}
```

2. 在`ShiroConfig`类中，找到`shiroFilter()`这个方法，配置上这个接口，注意加到最上面，这个Map是有顺序的，可以用通配符

```java
    shiroFilter.setUnauthorizedUrl("/global/error");
    /**
     * 配置shiro拦截器链
     *
     * anon  不需要认证
     * authc 需要认证
     * user  验证通过或RememberMe登录的都可以
     *
     * 当应用开启了rememberMe时,用户下次访问时可以是一个user,但不会是authc,因为authc是需要重新认证的
     *
     * 顺序从上到下,优先级依次降低
     *
     */
    Map<String, String> hashMap = new LinkedHashMap<>();
    hashMap.put("/blackboard/test", "anon");
    hashMap.put("/static/**", "anon");
    hashMap.put("/login", "anon");
    hashMap.put("/global/sessionError", "anon");
    hashMap.put("/kaptcha", "anon");
    hashMap.put("/**", "user");
    shiroFilter.setFilterChainDefinitionMap(hashMap);
    return shiroFilter;
}
```

3. 启动应用，并且不登录系统，我们访问`http://localhost:8080/blackboard/test`即可看

到，这个接口不需要登录也可以访问到

4.3 静态资源和模板位置的变更

由于spring boot默认是把静态资源文件css，js等放到`resources/static`目录的，默认把前端模板文件放到`resources/templates`目录，笔者认为前端面页面还是按maven的思想放到`webapp`目录比较分层清晰，所以做了一个变动，主要变动如下:

yml配置中增加了两个配置

```
#####################  spring配置  ####################
spring:
  redis:
    host: localhost
    port: 6379
    password:
  profiles:
    active: dev
  mvc:
    static-path-pattern: /static/**
    view:
      prefix: /WEB-INF/view
  http:
```

若想变动资源和模板的位置修改这两个配置即可

4.4 三个或更多数据源如何配置

1. 新建类似于`MutiDataSourceProperties`这样的类，用于接收第三个数据源

```java
/**
 * 默认多数据源配置
 *
 * @author fengshuonan
 * @date 2017-08-16 10:02
 */
@Component
@ConfigurationProperties(prefix = "guns.muti-datasource")
public class MutiDataSourceProperties {

    private String defaultDataSourceName = "dataSourceGuns";

    private String url = "jdbc:mysql://127.0.0.1:3306/biz?autoReconnect=true&useUnicode=true&characterEncoding=utf8&zeroDateTimeBehavior=convertToNull";

    private String username = "root";

    private String password = "root";

    private String driverClassName = "com.mysql.jdbc.Driver";

    private String validationQuery = "SELECT 'x'";

    public void config(DruidDataSource dataSource) {
        dataSource.setUrl(url);
        dataSource.setUsername(username);
        dataSource.setPassword(password);
        dataSource.setDriverClassName(driverClassName);
        dataSource.setValidationQuery(validationQuery);
    }
```

2. 在`MybatisPlusConfig`类中，配置类似于如下代码的方法

```java
/**
 * 另一个数据源
 */
private DruidDataSource bizDataSource() {
    DruidDataSource dataSource = new DruidDataSource();
    druidProperties.config(dataSource);
    mutiDataSourceProperties.config(dataSource);
    return dataSource;
}
```

3. 在`DynamicDataSource`配置中，增加第二步新加的数据源

```java
/**
 * 多数据源连接池配置
 */
@Bean
@ConditionalOnProperty(prefix = "guns", name = "muti-datasource-open", havingValue = "true")
public DynamicDataSource mutiDataSource() {

    DruidDataSource dataSourceGuns = dataSourceGuns();
    DruidDataSource bizDataSource = bizDataSource();

    try {
        dataSourceGuns.init();
        bizDataSource.init();
    }catch (SQLException sql){
        sql.printStackTrace();
    }

    DynamicDataSource dynamicDataSource = new DynamicDataSource();
    HashMap<Object, Object> hashMap = new HashMap();
    hashMap.put(DatasourceEnum.DATA_SOURCE_GUNS, dataSourceGuns);
    hashMap.put(DatasourceEnum.DATA_SOURCE_BIZ, bizDataSource);
    dynamicDataSource.setTargetDataSources(hashMap);
    dynamicDataSource.setDefaultTargetDataSource(dataSourceGuns);
    return dynamicDataSource;
}
```

在这之下初始化第三个数据源

在这个map中加入第三个数据源

4. 同时在`DatasourceEnum`类中，增加第三个数据源名称

```java
/**
 *
 * 多数据源的枚举
 *
 * @author fengshuonan
 * @date 2017年3月5日 上午10:15:02
 */
public interface DatasourceEnum {

    String DATA_SOURCE_GUNS = "dataSourceGuns";          //guns数据源

    String DATA_SOURCE_BIZ = "dataSourceBiz";            //其他业务的数据源
}
```

5. 使用方法同第二个数据源使用方法相同

4.5 添加登录验证码

Guns系统中内置了登录输入验证码的功能，因为开发方便调试，所以默认是关闭的，若需要开启该功能，只需要在application.yml中配置开启即可，如下

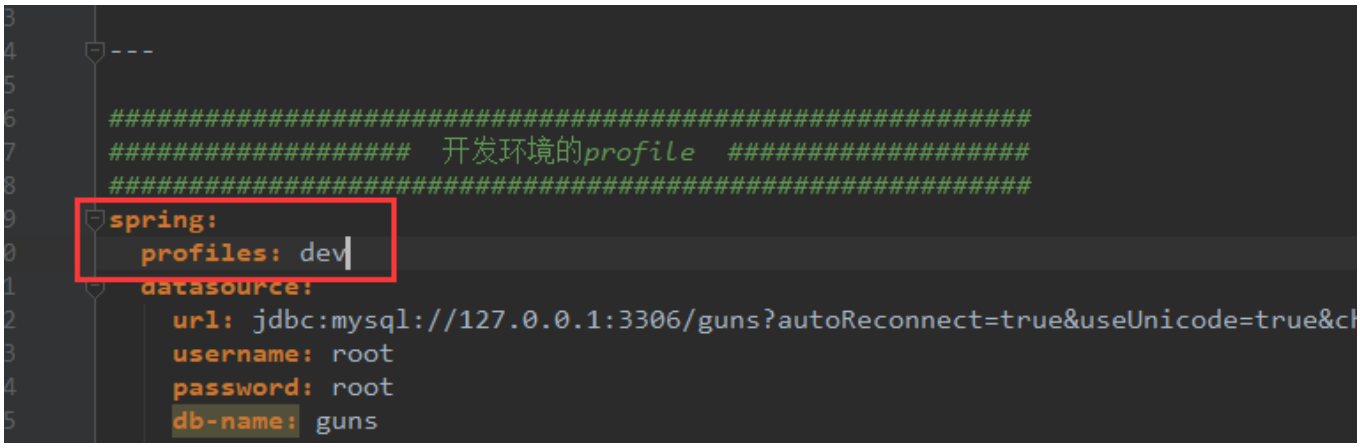
```yaml
###################  guns配置  ###################
guns:
  swagger-open: true #是否开启swagger (true/false)
  kaptcha-open: false #是否开启登录时验证码 (true/false)
  session-open: false #是否开启session超时验证(会影响的类SessionTimeoutInterceptor) (true/false)
  #file-upload-path: d:/tmp  #文件上传目录(不配置的话为java.io.tmpdir目录)
  muti-datasource-open: false #是否开启多数据源(true/false)
```

4.6 spring profile

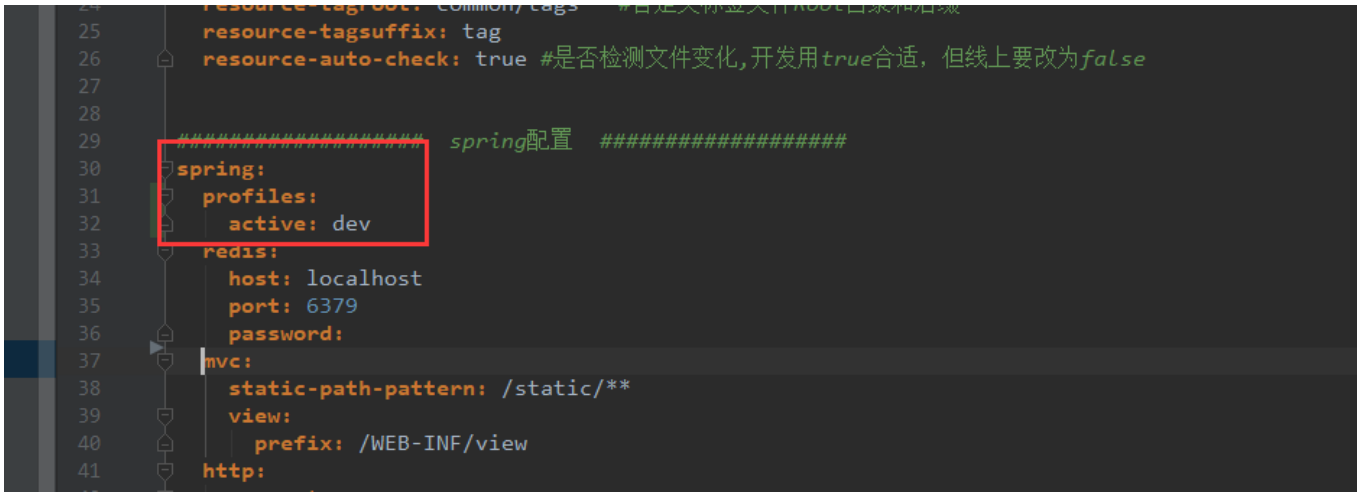
在实际的生产环境中，往往存在多个环境，例如开发环境(dev)，测试环境(test)，生产环境(prod)，并且不同环境的数据库和日志记录等配置的都不相同，为了每次发布不同环境的包时，不来回的修改这些配置，特引入了spring profile，引入之后，我们只需要把所有环境的配置都预先列出来，在每次发布不同环境的包的时候，只需要选择当前激活的是哪个环境的配置即可快速切换配置，关于spring profile的详细描述可参考这篇博文https://www.jianshu.com/p/948c303b2253

在yml配置中，我们用`---`来切分不同profile的配置，如下

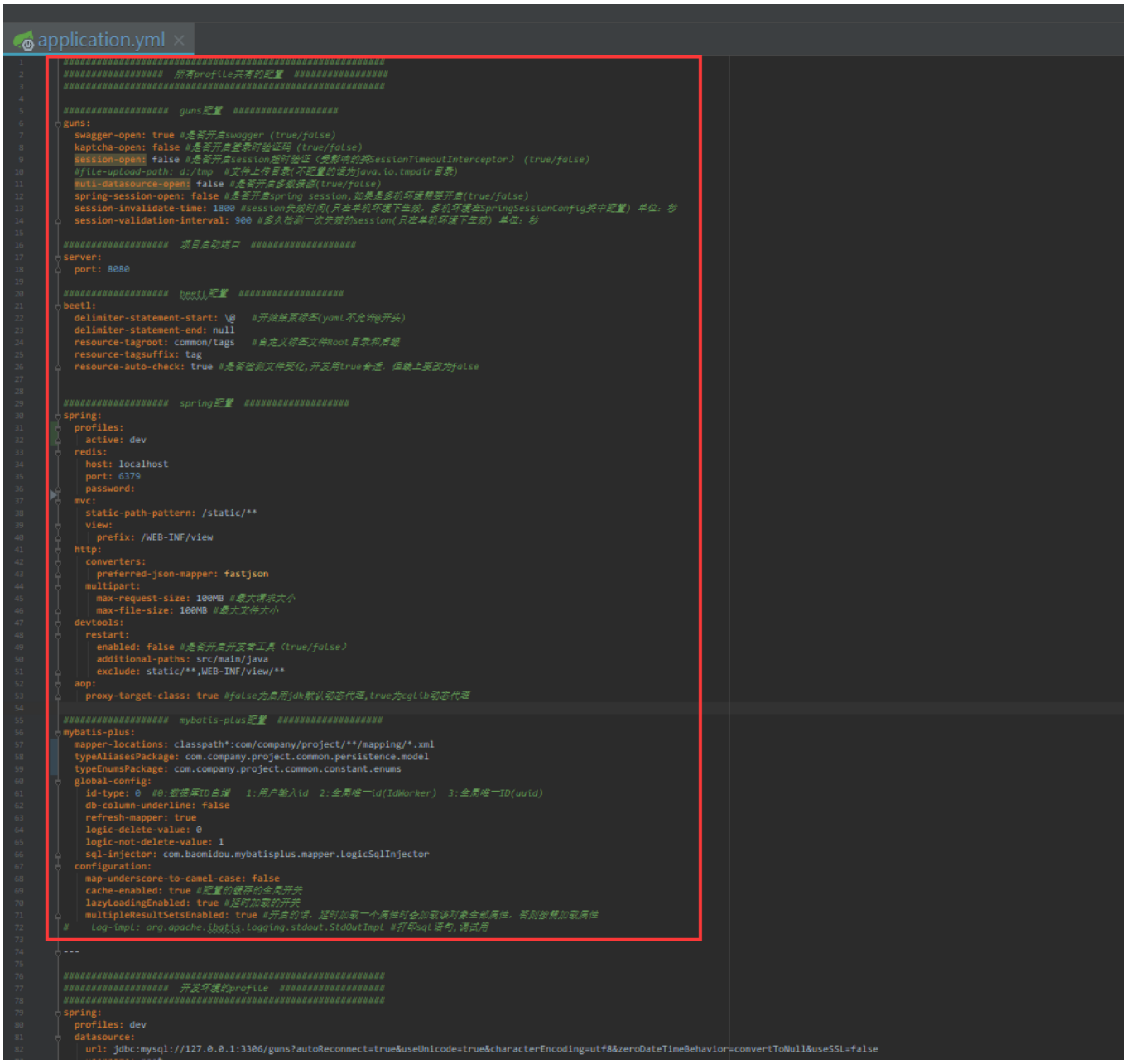
在分割线的下边我们就可以配置不同环境的配置了，`profile`可以配置多个，只需要用`spring.profiles`来标记当前节段的`profile`的名字即可



并用`spring.profiles.active`来激活当前的`profile`配置即可



`---`把配置切分成了多个节段，其中第一节是所有profile共有的配置，例如guns的配置中的这一大段



第一节段`---`下方的配置则是不同的profile的配置

4.7 多机器部署开启spring session

多机环境把session托管给redis存储，所以要部署和配置redis，另外需要注意的是开启相关配置

1.单机环境下不需要依赖spring-session，所以需要把相关依赖的注释打开

```
<dependency>
    <groupId>org.springframework.session</groupId>
```

```
        <artifactId>spring-session</artifactId>
        <scope>compile</scope><!-- 需要分布式session的话需要改为compile -->
    </dependency>
```

2.修改application.yml中guns.spring-session-open配置，改为true，打开spring-session

```
guns.spring-session-open=true
```

3.配置application.yml中，spring.redis.host，spring.redis.port，spring.redis.password

```
spring.redis.host=xxx
spring.redis.port=xxx
spring.redis.password=xxx
```

4.需要把SpringSessionConfig类中的注释打开

```
@EnableRedisHttpSession(maxInactiveIntervalInSeconds = 1800)
```

5.如需配置session失效时间，请在SpringSessionConfig类中修改maxInactiveIntervalInSeconds属性值

4.8 使用Redis

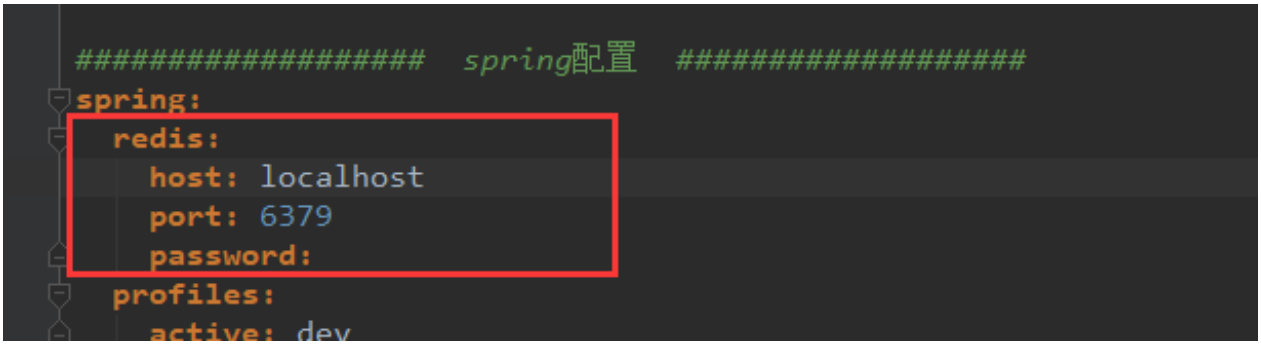
默认Guns在部署分布式的环境中使用了Redis作为分布式session的存储，如果想在项目中用redis做缓存或者存储，建议使用RedisTemplate来进行操作

1.首先下载Redis，可以在Guns的qq群里找到redis的可执行包，或者去redis官网下载
2.在项目的config包下建立redis的配置类

```
/**
 * Redis配置
 *
 * @author fengshuonan
 * @date 2018年1月28日 11:49:12
 */
@Configuration
@EnableCaching
```

```
public class RedisConfig {

    @Bean
    public RedisCacheManager cacheManager(RedisTemplate<String, Object> redisTemplate) {
        RedisCacheManager redisCacheManager = new RedisCacheManager(redisTemplate);
        redisCacheManager.setDefaultExpiration(30 * 60);
        return redisCacheManager;
    }

    @Bean
    public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory factory) {
        RedisTemplate<String, Object> template = new RedisTemplate<>();
        template.setConnectionFactory(factory);
        template.afterPropertiesSet();
        return template;
    }
}
```

3.在`application.yml`中配置redis的连接属性



4.在`GunsApplication`类中,编写`CommandLineRunner`来测试一下Redis的连接

```
@Bean
CommandLineRunner commandLineRunner() {
    return new CommandLineRunner() {
        @Override
        public void run(String... strings) throws Exception {
            BoundValueOperations<String, Object> test = redisTemplate.boundValueOps("test");
            test.set("test value");

            Object o = test.get();
            System.out.println(o);
```

```
            }
        };
    }
```

4.9 XSS过滤器

4.9.1 介绍

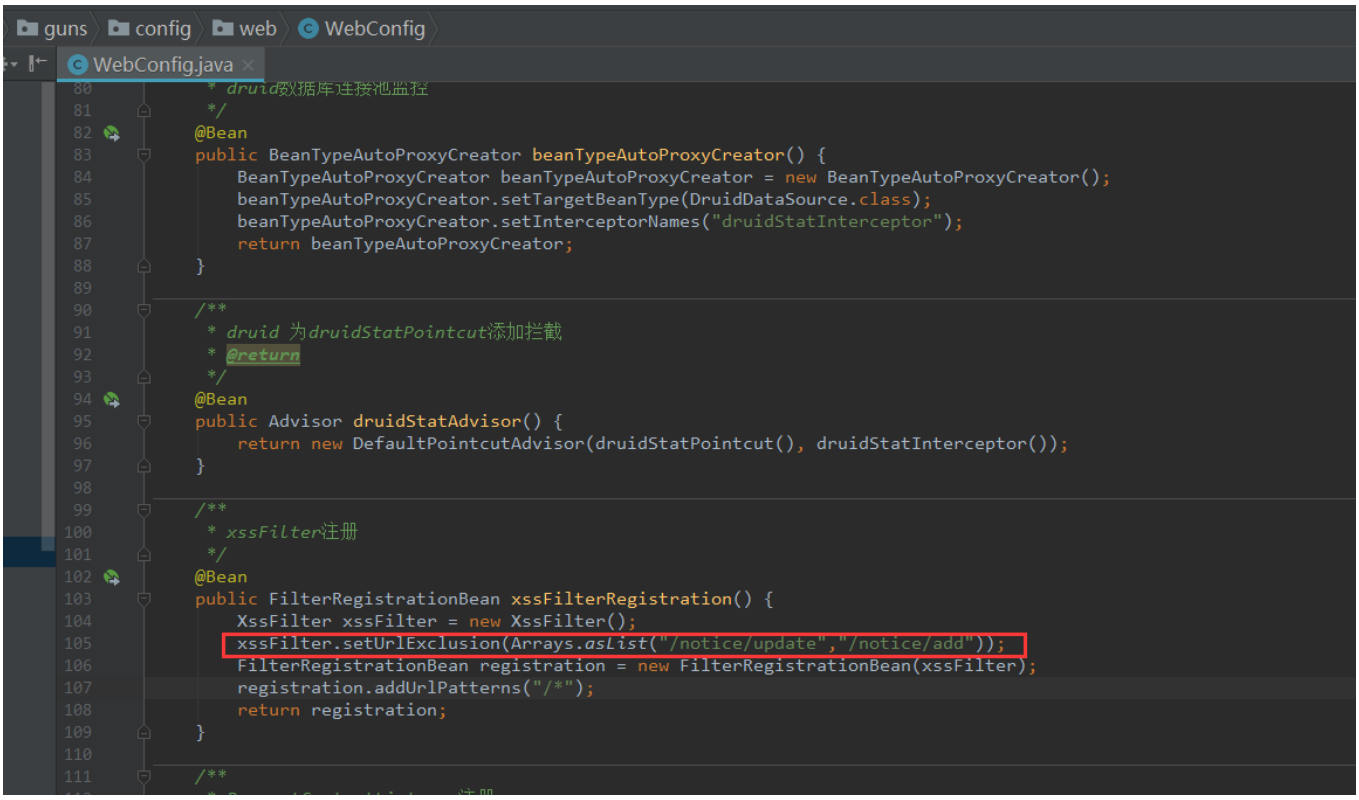
为了抵御XSS攻击，不让用户在录入数据的同时插入恶意js代码，Guns对所有传入数据中带有`html标签`和`<script>`标签的内容进行转义，转义后的内容并不是乱码，只是为了传入的html片段不会在渲染页面时让浏览器当成真正的脚本去执行。

4.9.2 原理

防止XSS攻击的原理其实就是一个过滤器，对所有请求传来的参数进行正则校验，利用`replaceAll`把所有的请求中带有html标签的`<`和`>`等这种标识过滤为了`& lt;`和`& gt;`这种特殊符号.

4.9.3 放过过滤

在`WebConfig`配置类中，找到`XssFilter`配置的地方，在`setUrlExclusion()`这里加上被放过过滤的列表即可



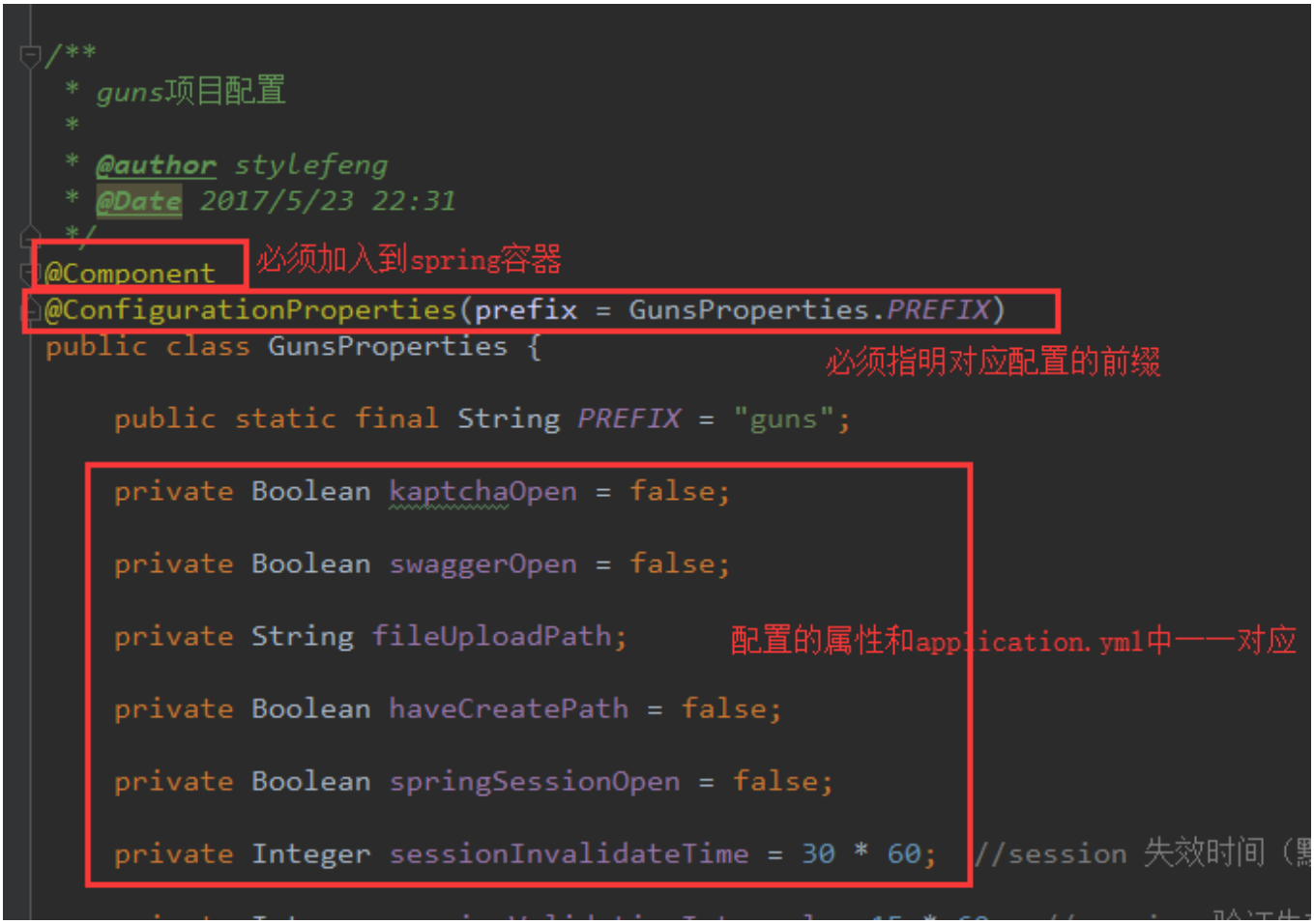
5. 核心思想

5.1 分包

在日常开发中，业务模块的包结构划分一般划分为三个`config`、`core`、`modular`或者四个`common`、`config`、`core`、`modular`

其中`common`为模块内通用的注解、常量、枚举、异常和持久化的实体等，若`common`不单独划分一个包，则可以把common包放到core包下面

`config`包存放整个模块的配置类，因为项目基于spring boot开发，大部分的spring配置都换成了java bean方式的配置，所以单独分一个包来存放配置，`config`包中除了存放配置类，还有一些以`Properties`结尾的类，这些类的作用是启动应用的时候把`application.yml`中的配置映射到类的属性上，使用时需要注意以下几点



`modular`存放按业务划分的业务代码，若本模块中包含多个模块业务，则在`modular`中建立多个业务包，在具体的业务包下再建立`controller`、`dao`、`service`、`transfer`、`warpper`这几个包，其中`transfer`为前后端传输数据所用的属性封装，`warpper`为对返回结果的包装器(下面会介绍到)，如果当前模块中只存在一类业务，那么没有必要在`modular`包下再建立多个业务模块，可直接在`modular`模块建立`controller`、`dao`、`service`、`transfer`、`warpper`

`core`包存放当前模块所运行的一些`核心机制`，例如全局的异常拦截器，日志AOP，权限的AOP，项目初始化后的监听器，工具类等，还可以存放一些对某些框架的`扩展`，例如对beetl模板的扩展配置和工具类，对flowable的扩展类，Shiro的一些拓展类等等

这样拆分的好处在于把业务，配置和运行机制清晰的拆分开，提高项目的可维护性，加快项目的开发效率!

5.2 统一异常拦截

5.2.1 介绍

统一异常拦截指对程序抛出的异常利用`@ControllerAdvice`在统一的一个类中做`catch`处理，在Guns中，我们在`GlobalExceptionHandler`类中做统一异常拦截处理，`GlobalExceptionHandler`类中可以拦截所有控制器执行过程中抛出的异常，若需要拦截其他包下的异常可以参考`SessionInterceptor`这个类中AOP的写法，来拦截其他特定包的异常。统一异常拦截的写法注意一下几点

5.2.2 优点

对异常进行统一处理，不需要再在业务代码中进行`try catch`操作，尽情写业务，有异常也会被自动拦截到，并且自动处理返回给前端提示

5.2.3 关于性能

有人可能会认为利用异常拦截这种机制，把业务逻辑的错误都用业务异常抛出进入aop的执行器，对性能会有所影响，经过笔者的调研和测试，频繁的抛出异常和`try catch`不会有性能损耗，主要的性能损耗在`catch`方法内部，并且在`catch`内，记录日志比较占用大部分的时间

所以，如果是系统特别注重性能等问题，可以把业务异常分为两类，一类是`较为频繁抛出`的业务异常，一类是`较少出现次数`的业务异常，第一类异常可以再@ExceptionHandler中不做日志记录，只进行简单的返回操作，第二类可以着重做异常处理，并做结果返回

5.3 结果包装器

我们在进行`列表查询`或`详情查询`的过程中，查到的结果中，有些值可能在数据库中存的是一些列数字(一般为状态值等)，但是我们要返回给前端，业务人员看的时候不希望直接返回给他们这些不直观的值(例如1，2，3，4)，我们更希望返回给前端中文名称(例如启用，冻结，已删除)，所以我们应该对这些数值做一下包装，把他们包装成文字描述

5.3.1 如何使用

以查询用户列表的接口为例，不包装的情况下默认的查询结果为这些字段

```
<sql id="Base_Column_List_With_Pwd">
    id, account, name, birthday,password, sex, email, avatar,
    phone, roleid,salt,
    deptid, status,
    createtime, version
</sql>
```

其中性别，角色，部门，状态都是数值或者id类型，我们需要把他们包装成文字形式返回给前端

1.首先建立`UserWarpper`类继承`BaseControllerWarpper`类

```
/**
 * 用户管理的包装类
 *
 * @author fengshuonan
 * @date 2017年2月13日 下午10:47:03
 */
public class UserWarpper extends BaseControllerWarpper {

    public UserWarpper(List<Map<String, Object>> list) {
        super(list);
    }

    @Override
    public void warpTheMap(Map<String, Object> map) {
        map.put("sexName", ConstantFactory.me().getSexName((Integer) map.get("sex")));
        map.put("roleName", ConstantFactory.me().getRoleName((String) map.get("roleid")));
        map.put("deptName", ConstantFactory.me().getDeptName((Integer) map.get("deptid")));
        map.put("statusName", ConstantFactory.me().getStatusName((Integer) map.get("status")));
    }
}
```

通过查看`BaseControllerWarpper`类可了解到被包装的参数必须为Map或者List类型

```
/**
 * 控制器查询结果的包装类基类
```

```
 *
 * @author fengshuonan
 * @date 2017年2月13日 下午10:49:36
 */
public abstract class BaseControllerWarpper {

    public Object obj = null;

    public BaseControllerWarpper(Object obj) {
        this.obj = obj;
    }

    @SuppressWarnings("unchecked")
    public Object warp() {
        if (this.obj instanceof List) {
            List<Map<String, Object>> list = (List<Map<String, Object>>
) this.obj;
            for (Map<String, Object> map : list) {
                warpTheMap(map);
            }
            return list;
        } else if (this.obj instanceof Map) {
            Map<String, Object> map = (Map<String, Object>) this.obj;
            warpTheMap(map);
            return map;
        } else {
            return this.obj;
        }
    }

    protected abstract void warpTheMap(Map<String, Object> map);
}
```

我们继承`BaseControllerWarpper`类主要是为了实现`warpTheMap()`方法，也就是具体的包装过程，`warpTheMap()`方法的参数map就是被包装的原始数据的每个条目，我们可以在这每个条目中增加一些字段也就是被包装字段的中文名称，如下

```
map.put("sexName", ConstantFactory.me().getSexName((Integer) map.get("sex")));
map.put("roleName", ConstantFactory.me().getRoleName((String) map.get("roleid")));
map.put("deptName", ConstantFactory.me().getDeptName((Integer) map.get("deptid")));
map.put("statusName", ConstantFactory.me().getStatusName((Integer) map.get("status")));
```

5.3.2 ConstantFactory

在包装过程中，我们经常会用到`ConstantFactory`这个类，这个类是连接数据库和包装类的

桥梁，我们可以在`ConstantFactory`中封装一些编辑的查询方法，这些方法通常会被多个包装类多次调用，并且在调用这些方法的时候`ConstantFactory.me()`的形式静态调用，可以快速的`包装一些状态和id`，非常方便，在ConstantFactory中我们可以利用spring cache的`@Cacheable`注解来缓存一些数据，把这些频繁的查询缓存起来

5.4 前端思想

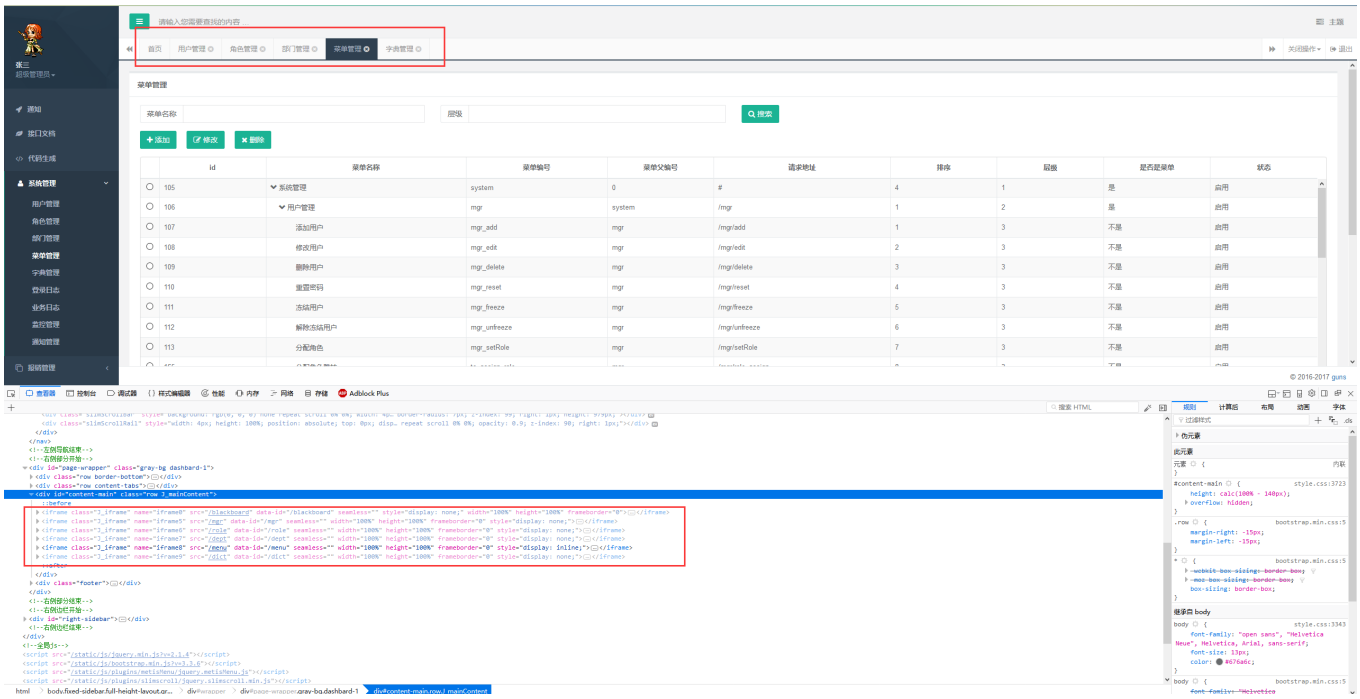
Guns前端采用了beetl模板引擎，beetl包含语法简洁，速度快，文档全，社区活跃等众多优点，所有的beetl语法都以`@`开头

5.4.1 布局

在用户登录页面后进入的是`index.html`页面，这个页面加载了整个后台管理系统的框架，我们可以看到`index.html`源代码中把整个页面分为了三部分，左侧菜单栏，右侧页面和右侧主题栏部分，其实就是利用beetl的`@include`把整个大的复杂的页面细化了，这样好维护

```
<div id="wrapper">

    <!--左侧导航开始-->
        @include("/common/_tab.html"){}
    <!--左侧导航结束-->

    <!--右侧部分开始-->
        @include("/common/_right.html"){}
    <!--右侧部分结束-->

    <!--右侧边栏开始-->
        @include("/common/_theme.html"){}
    <!--右侧边栏结束-->

</div>
```

左侧菜单和右侧主题栏部分在用户登录后会一直不变，除非刷新浏览器页面，动态变化的是页面右侧这部分，我们打开6个标签页，并打开浏览器F12调试



新建和切换标签，页面的地址不会变化，变化的是页面右侧的`iframe`这部分

下面我们分析一下右侧页面的组成，我们打开菜单管理页面，查看他的代码

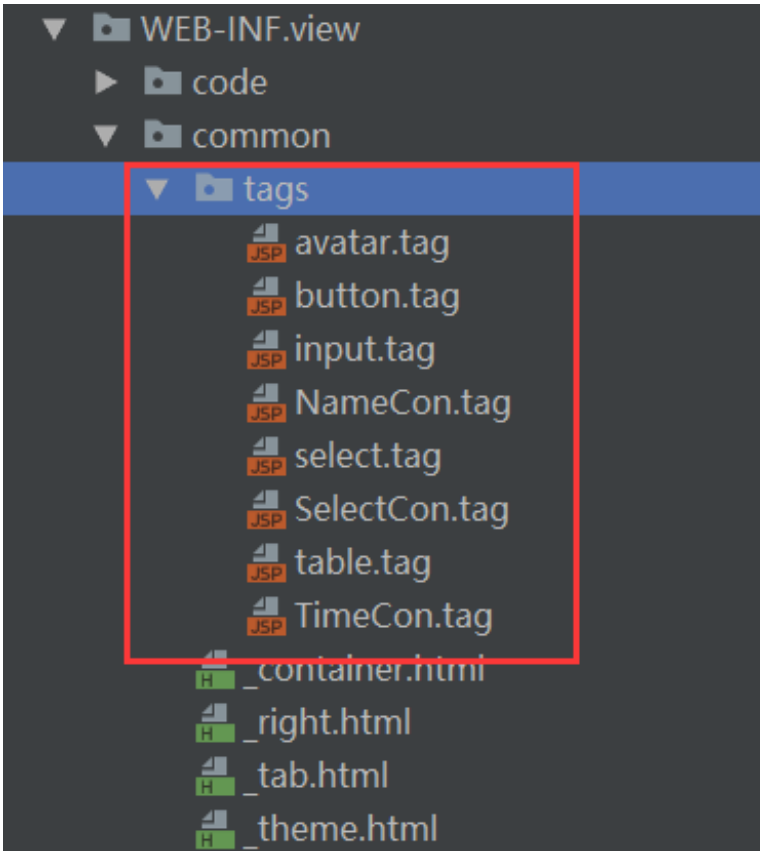
```
@layout("/common/_container.html"){
<div class="row">
    XXXX等html代码...
</div>
<script src="${ctxPath}/static/modular/system/menu/menu.js"></script>
@}
```

整个页面被`@layout`所包围，`@layout`是beetl的引用布局(具体用法文档可以查看beetl的官方文档)，Guns中内置了`/common/_container.html`这样一个布局，可以把`/common/_container.html`理解为一个html的抽象封装，我们每个页面都继承自这个模板，默认包含了一系列通用的js css引用等，这样写即简化了我们的开发和维护，又使我们的代码简洁有序，在`/common/_container.html`中的`${layoutContent}`就代表我们每个页面不同的html

5.4.2 标签

为了把一些重复性的html封装起来，我们使用了beetl的标签，这些标签的`本质是把重复性的html代码用一行html标签替代`，从而方便使用，易于维护，这些标签都放

在`common/tags`这个文件夹



标签中的一些属性例如`${name}` `${id}`等属性均为掉钱被调用时，从调用体的属性传来`<#xxxTag name="xxx" id="xxx">`

---

6. 常见问题答疑

6.1 默认的系统登录账号和密码是多少

账号是`admin` 密码是`111111`

6.2 权限异常

6.3 为何分页是前端实现

部分页面因为数据量比较少，就直接用客户端分页了，日志页面的分页是采用服务端分页的，

如果其他业务有特别需要，可以手动设置一下

6.4 关于`${ctxPath}`

这个变量在哪里定义的?这个是beetl自带的具体请看beetl文档

6.5 放过某些url的权限验证

在ShiroConfig类下的shiroFilter方法里配置，参考**4.2节**

6.6 主页的搜索功能

主页的搜索功能目前没有写实际业务，只是装饰作用

6.7 运行sql报错

在初始化guns.sql过程中，可能会出现

```
1.  [Err] 1067 - Invalid default value for 'createtime'
```

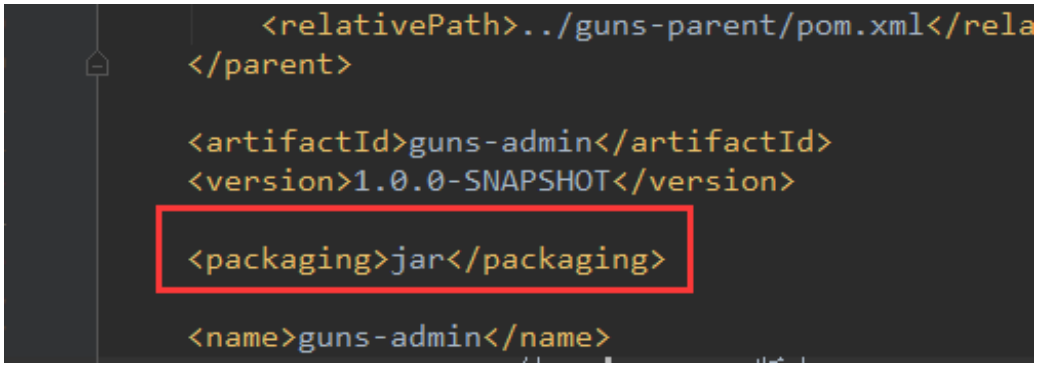
这样的报错，Guns目前支持mysql 5.7的运行环境，若您的mysql低于此版本，请把`sys_expense`表的`DEFAULT CURRENT_TIMESTAMP`这部分语句去掉即可

```
76
77  -- ----------------------------
78  -- Table structure for sys_expense
79  -- ----------------------------
80  DROP TABLE IF EXISTS `sys_expense`;
81  CREATE TABLE `sys_expense` (
82    `id` int(11) NOT NULL AUTO_INCREMENT,
83    `money` decimal(20,2) DEFAULT NULL COMMENT '报销金额',
84    `desc` varchar(255) DEFAULT '' COMMENT '描述',
85    `createtime` datetime DEFAULT CURRENT_TIMESTAMP,
86    `state` int(11) DEFAULT NULL COMMENT '状态: 1:待提交  2:待审核  3.审核通过 4:驳回',
87    `userid` int(11) DEFAULT NULL COMMENT '用户id',
88    `processId` varchar(255) DEFAULT NULL COMMENT '流程定义id',
89    PRIMARY KEY (`id`)
90  ) ENGINE=InnoDB AUTO_INCREMENT=23 DEFAULT CHARSET=utf8 COMMENT='报销表';
91
92  -- ----------------------------
```

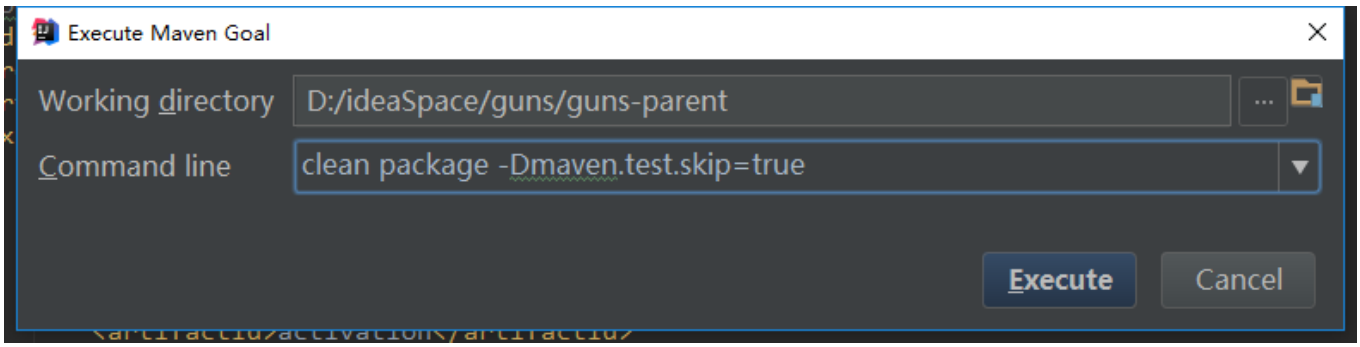
6.8 关于打包

Guns现在是多模块组成，各个模块之间有依赖关系，打包时，先修改guns-admin模块的

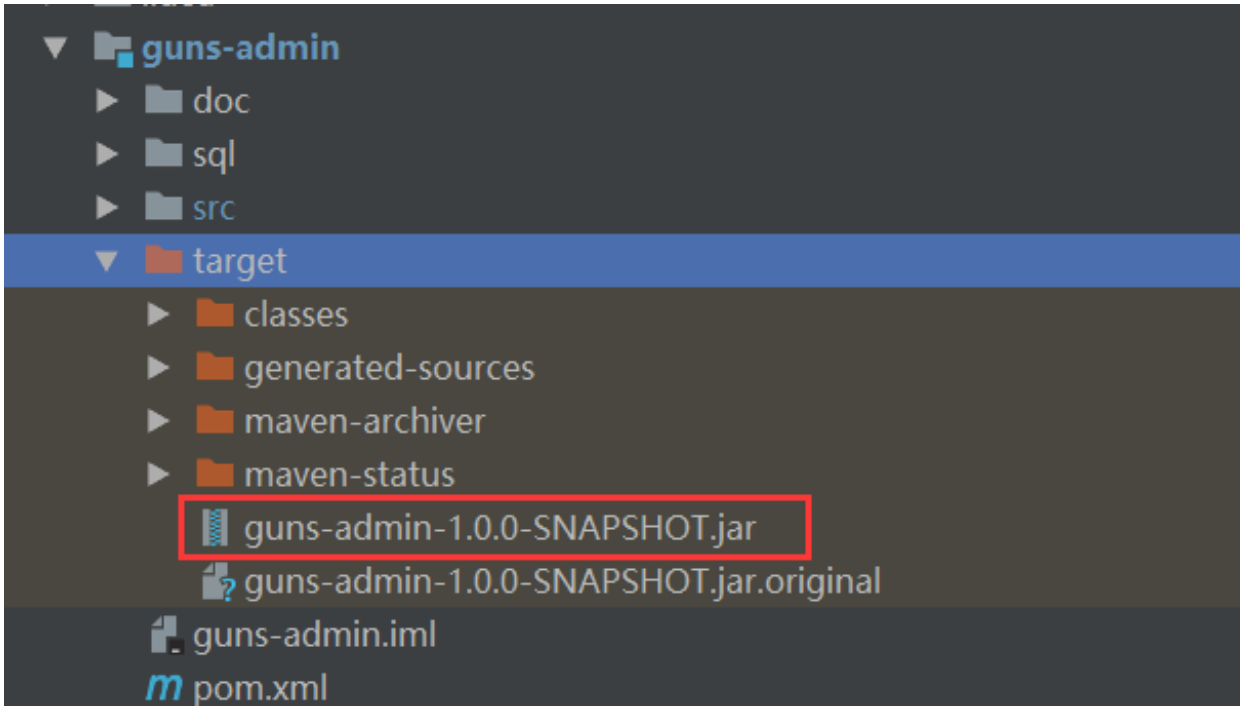
pom的`<packaging>`节点，改为jar或者war



再在`guns-parent`目录下输入`clean package -Dmaven.test.skip=true`来打出所有模块的包



执行成功后，在guns-admin目录下即可看到打好的包



6.9 查询结果的驼峰转化问题

直接参考mp的文档

6.10 为何使用beetl

beetl具有语法简介，性能超高，文档全，社区活跃等特点，所以建议用beetl模板引擎

6.11 为何有的业务没有service层

部分业务比较简单，所以就没写service层，写service是为了让复杂业务更有条理，更清晰。(此项仅供参考)

6.12 为何既有dao，又有mapper

mapper是mybatis-plus自动生成的，里边有许多mybatis-plus增强的方法，dao是自己写的业务，mybatis-plus自动生成代码时会覆盖mapper，所以就把自己写的dao分开了，生成代码的时候不影响。(此项仅供参考)