Spring Cloud项目开发参考指南

**文档摘要**

Spring cloud是一个分布式开发架构，基于Spring boot来实现。本文主要是结合**批发采购**项目的实际案例，介绍实际的使用规范和对项目中要做的哪些修改。

批发采购项目作为一个spring cloud项目的参考，共有四个子项目组成：公共模块fxCommon、服务的实现、服务接口和服务调用。

1）fxCommon中目前已经加入了一下公共帮助类、数据源配置类、Consul配置类（之所以把Consul的配置类放到Common中是便于后期Consul某些规则的改变方便维护，不用每个人都去自己项目修改代码，只需使用相应版本即可）、常用过滤器、公共常量、主函数的父类CommonApplication （便于需要时扩展）；

2）服务的实现包括拦截器验证、Mybatis配置（在类头上引入对应的数据源类）、Consul配置的引入、一些初始化操作、数据库操作、RestController接口的具体实现；

3）Rest接口是对RestController的抽象，通过FeignClient进行调用，接口中配置了一个Request Header的拦截器，可以设置或者从Request中获取对应的header信息传递到接口实现层；

4）接口调用，通过引入接口api想使用service一样调用接口，该层像普通springMVC项目开发一样，只是需要引入相应jar和配置对应的属性即可。

文档内容会详细介绍以上4个模块的具体使用可操作。以上模块对应源码的SVN地址如下：

|  |  |
| --- | --- |
| 项目名称 | Svn地址 |
| 公共模块fxCommon | http://svn.tomshopping.com/shanghai/postmall/ule.fenxiao/fxCommon |
| 接口实现wholesale-purchase-service | http://svn.tomshopping.com/shanghai/postmall/ule.fenxiao/microservice/wholesale-purchase-parent/wholesale-purchase-service |
| 接口wholesale-purchase-client | http://svn.tomshopping.com/shanghai/postmall/ule.fenxiao/microservice/wholesale-purchase-parent/wholesale-purchase-client |
| 接口调用fxPurchase-web | http://svn.tomshopping.com/shanghai/postmall/ule.fenxiao/purchase/fxPurchase-web |

所用技术和版本

SpringBoot Version: 1.5.4.RELEASE

SpringCloud Version: Dalston.RELEASE

Hystrix

Ribbon

Sleuth

FeignClient Http: okHttp

# 整体介绍

完整的项目结构分为：

公共模块：fxCommon

服务模块：

接口实现（wholesale-purchase-service）

接口客户端jar包（wholesale-purchase-client ）

接口调用模块：fxPurchase

## fxCommon中的常用帮助类如下表：

|  |  |
| --- | --- |
| 类名 | 描述 |
| BeanUtils | 替代了common包中的BeanUtils，为了解决字符串日期转对象时的格式问题 |
| DataUtil | 转换HttpServletRequest对象中的指定key，根据需要调用对应方法获取需要类型的数据，方法有parseString,parseLong,parseDouble,parseInt；parseInt(Object str)，boolean isNumber(String str)，汉字转换成拼音String toPinyin(String input)，String ToDBC(String input) |
| HttpClientUtil | Post请求 |
| MD5Tools | 获取系统默认分隔符String getOSFileSeparator()，获取加密后的字符串 String getMD5Password(String password) ，获取指定日期的年月日int getYear(final Date date)，int getMonth(final Date date)，int getDay(final Date date)，int getHour(final Date date)，int getMinute(final Date date)，int getSecond(final Date date) |
| ResultDTO | 所有rest接口统一返回对象 |
| LocalDeveloyEnv | 所有spring boot项目均通过以下命令来启动-Dspring.profiles.active=dev spring-boot:run  原因：在发现Common中添加了后缀为dev的配置文件，便于开发时不和beta配置冲突，而且类中会通过这个属性来判断是否为dev环境，代码中可以使用LocalDeveloyEnv.isDev |

## 数据源配置类

DataBaseMysqlConfiguration和DataBaseOracleConfiguration，根据项目使用数据库类型在项目中Mybatis配置类上导入数据源对象

## 公共常量配置

在platfrom-properties.yml文件中，通@Import({CommonConstants.class})进行初始化。目前是要求所有项目都要继承CommonApplication，所以无需自己再操作。

目前已有常量配置如下：

commom:

uploadTempDir: /data/postmall/tomcat/temp

uploadFileToDFSUrl: //static.beta.ulecdn.com

uploadFileUrl: http://upload.beta.ule.com/upload

globalStaticServer1: //i0.beta.ulecdn.com

globalStaticServer2: //i1.beta.ulecdn.com

globalStaticServer3: //i2.beta.ulecdn.com

searchDistrIpPort: cloudSearch.beta.uledns.com:9020

server-provider-host: opc.beta.ule.com

zipkin-base-url: http://172.25.200.86:9411

consul-host: client.consul.beta.uledns.com

consul-port: 8500

通过CommonConstants获取相应的值

## CommonApplication类的使用

要求所有项目的Application继承该类，继承该类后回自动实例化常量对象，还可以减少两个注解 @SpringBootApplication @EnableDiscoveryClient

## POM的GAV要求

父项目的GAV如下

<groupId>com.ule.wholesale.microservice</groupId>

<artifactId>wholesale-{模块名}-parent</artifactId>

服务项目service引入parent的groupId并指定自己的artifactId：

<artifactId>wholesale-{模块名}-service</artifactId>

服务接口client引入parent的groupId并指定自己的artifactId：

<artifactId>wholesale-{模块名}-client</artifactId>

web应用项目

<groupId>com.ule.wholesale.webui</groupId>

<artifactId>{模块名}</artifactId>

## Consul的服务名

服务注册中心显示的服务名和spring.application.name一致

spring.application.name要求

mircoservice-{模块名} -provider

我们目前服务都是通过war形式部署到Tomcat，一般不会直接到ROOT下，所以会存在上下文，上下文一般和war的名字一样，为了简便就定义为{模块名}Service

比如批销采购项目：

spring.application.name = mircoservice-purchase -provider

context-path: /purchaseService

pom中name:

name=purchaseService

打包的war名称 purchaseService-beta-1.0.0.HEAD.war

FeignClient中的serviceName和servicePath的值如下：

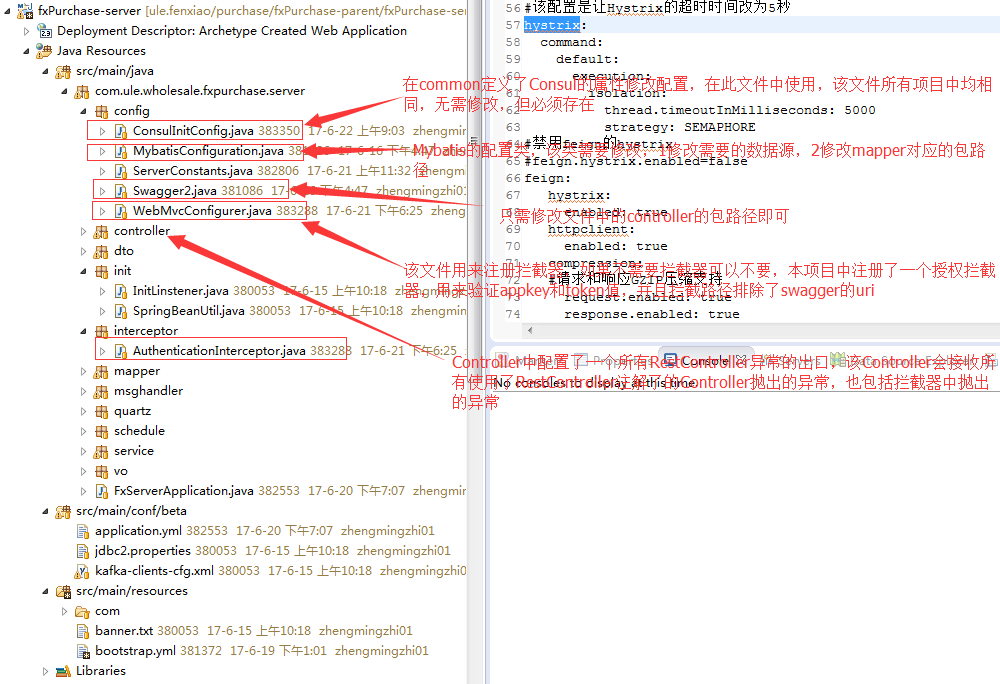
serviceName = mircoservice-purchase -provider

servicePath = purchaseService

# Springcloud服务项目开发

## 服务实现开发

### 项目结构



|  |  |  |
| --- | --- | --- |
| 包名 | 类名 | 描述 |
| com.ule.wholesale.fxpurchase.server.config | ConsulInitConfig | 初始化服务注册组件,只需加入该类，无需修改 |
| MybatisConfiguration | Mybatis配置，数据源导入，分页插件配置，需要导入对应的数据源和修改要扫描的mapper的包路径 |
| Swagger2 | Swagger配置，需要修改自己项目的controller路径 |
| WebMvcConfigurer | 拦截器管理，引用拦截器并设置拦截路径和不拦截路径，根据自己项目需要进行曾减 |
| com.ule.wholesale.fxpurchase.server.controller | RestControllerAdvice | 所有RestController异常处理，包括拦截器抛出的异常，一般不需要修改 |
| com.ule.wholesale.fxpurchase.server.interceptor | AuthenticationInterceptor | WebMvcConfigurer引入了该拦截器，验证appkey好token |
| com.ule.wholesale.fxpurchase.server.init |  |  |
| com.ule.wholesale.fxpurchase.server.mapper |  | Mybatis接口文件和对应的映射文件 |
| com.ule.wholesale.fxpurchase.server.service |  |  |
| com.ule.wholesale.fxpurchase.server.vo |  | 实体对象包 |
| 配置文件 | bootstrap.yml | context-path、spring.application.name、Mybatis配置 |
| application.yml | 自定义属性配置和覆盖common中是属性配置 |
| jdbc2.properties | 数据源配置 |

### 数据源引用和Mybatis配置

在项目中添加MybatisConfiguration类并用@Import导入对应数据源类

@Import({ DataBaseMysqlConfiguration.class })

指定Mapper文件包路径

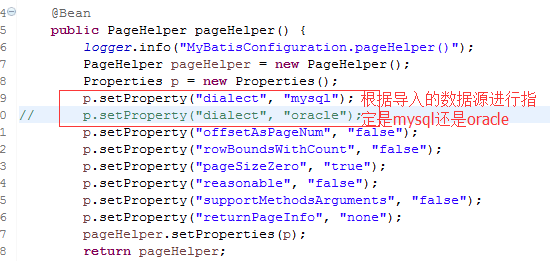
@MapperScan(basePackages = { "com.ule.wholesale.fxpurchase.mapper" })

Mysql数据配置类 DataBaseMysqlConfiguration

Oracle数据源配置类 DataBaseOracleConfiguration

数据源配置类中需要的jdbc2.properties的配置需要使用者在自己项目中添加

分页插件



Mysql的配置文件内容如下

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | jdbc.ule\_uwds\_selfsupport-master.url=jdbc:mysql://ip:3306/dbname?autoReconnect=true&autoReconnectForPools=true&useUnicode=true&characterEncoding=UTF-8&zeroDateTimeBehavior=convertToNull  jdbc.ule\_uwds\_selfsupport-master.username=username  jdbc.ule\_uwds\_selfsupport-master.password=\*\*\*  jdbc.ule\_uwds\_selfsupport-master.initialSize=5  jdbc.ule\_uwds\_selfsupport-master.maxActive=5  jdbc.ule\_uwds\_selfsupport-master.maxIdle=5  jdbc.ule\_uwds\_selfsupport-master.minIdle=1  jdbc.ule\_uwds\_selfsupport-master.maxWait=6000  jdbc.ule\_uwds\_selfsupport-master.validationQuery=SELECT 1 |

Oracle的配置文件内容如下

jdbc.postmall-master.url=jdbc:oracle:thin:@172.24.144.126:1521/ULE\_UWDS.APP

jdbc.postmall-master.username=ULEAPP\_UWDS\_PAYMENT

jdbc.postmall-master.password=ule.123

jdbc.postmall-master.initialSize=5

jdbc.postmall-master.maxActive=5

jdbc.postmall-master.maxIdle=5

jdbc.postmall-master.minIdle=1

jdbc.postmall-master.maxWait=6000

jdbc.postmall-master.validationQuery=SELECT 1 FROM DUAL

在使用中如果有配置文件中的前缀不是jdbc.postmall-master和jdbc.ule\_uwds\_selfsupport-master的请及时提出

### Consul服务注册和发现

Consul的配置已经在fxCommon中的ConsulConfig类中进行了配置，只需在项目中添加ConsulInitConfig并实现InitializingBean（保证执行的优先级高）调用initConsulConfig方法即可

### 集成EJB服务

在spring cloud中如需调用EJB的服务需引入以下jar包，并且jbossClient的版本号为5.1.2.GA，spring-boot-starter-jta-narayana 不引入eclipse中启动正常，Tomcat下会报一些class找不到

<dependency>

<groupId>jbossClient</groupId>

<artifactId>jbossClient</artifactId>

<version>5.1.2.GA</version>

<type>jar</type>

</dependency>

<dependency>

<groupId>com.ule.common</groupId>

<artifactId>ule.framework</artifactId>

<type>jar</type>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-jta-narayana</artifactId>

<exclusions>

<exclusion>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter</artifactId>

</exclusion>

<exclusion>

<groupId>org.jboss.narayana.jta</groupId>

<artifactId>jms</artifactId>

</exclusion>

<exclusion>

<groupId>org.jboss.narayana.jta</groupId>

<artifactId>jdbc</artifactId>

</exclusion>

</exclusions>

</dependency>

### 服务启动

import org.springframework.boot.SpringApplication;

import org.springframework.boot.builder.SpringApplicationBuilder;

import org.springframework.context.annotation.ComponentScan;

import com.ule.wholesale.common.CommonApplication;

@ComponentScan({"com.ule.wholesale.fxpurchase.server.init",

"com.ule.wholesale.fxpurchase.server.config",

"com.ule.wholesale.fxpurchase.server.service",

"com.ule.wholesale.fxpurchase.server.controller"})

public class FxServerApplication extends CommonApplication{

protected SpringApplicationBuilder configure(SpringApplicationBuilder builder) {

return builder.sources(FxServerApplication.class);

}

public static void main(String[] args) {

SpringApplication.run(FxServerApplication.class, args);

}

}

服务启动类 需要继承CommonApplication和指定要扫描的包，**扫描的包中不能包括springBoot定时服务，否则服务中心发现不了服务**

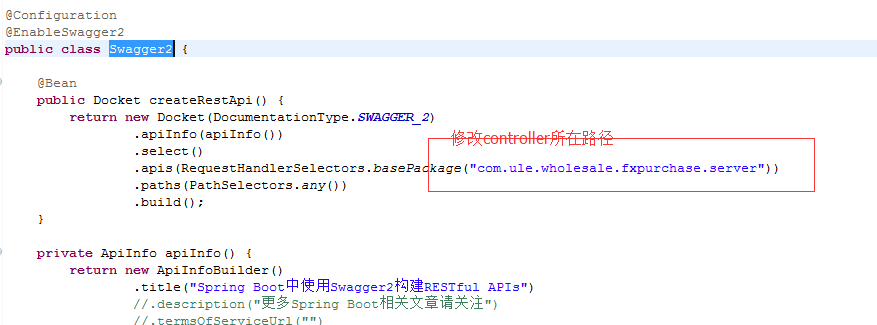
通过mvn命令进行启动，-Dspring.profiles.active=dev spring-boot:run 该命令中指定了配置文件的profile，开发环境下会使用fxcommon中的dev环境配置文件，并且会根据是否存在spring.profiles.active变量判断是否为dev环境，为此定义了一个本地环境变量 LocalDeveloyEnv.isDev的全局变量

### 拦截器配置

如需拦截器在WebMvcConfigurer配置，并且指定拦截路径和排除拦截的路径

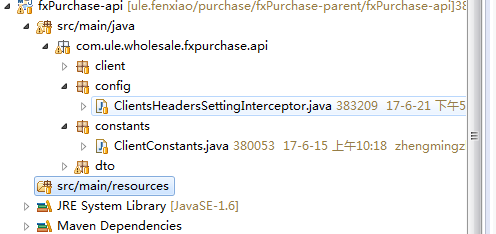


### Swagger2配置



## 服务接口开发

### 项目结构



### 服务名和路径配置

import java.util.HashMap;

import java.util.Map;

public class ClientConstants {

//接口服务中的server.context-path的值

public final static String SERVER\_PATH = "purchaseService";

//对应接口服务中的Consul的serviceName,一般就是spring.application.name

public final static String SERVICE\_NAME= "wholesale-purchase-provider";

//用来设置一下固定的head信息传递到接口服务中，使用代码参照ClientsHeadersSettingInterceptor

public static Map<String, Object> headMap = new HashMap<String, Object>();

}

### FeignClient RequestBody

所有使用了RequestBody的接口，必须指定RequestMapping的method为POST

### 拦截器的使用

代码如下：

@Configuration

public class ClientsHeadersSettingInterceptor {

//指定哪些head中的信息需要传递到接口服务中，

//如果是固定值（appke，token）可以直接在常量ClientConstants.headMap设置

//如果request head中已经存在可以指定key从request中进行获取，

//此时的值key为任意值,为了避免不必要的处理可以通过headerKeyList

//指定要从request中获取的key进行处理

private static List<String> headerKeyList = new ArrayList<String>();

static{

headerKeyList.add("appkey");

headerKeyList.add("token");

headerKeyList.add("sign");

}

@Autowired

HttpServletRequest request;

@Bean

public RequestInterceptor headerInterceptor() {

return new RequestInterceptor() {

@Override

public void apply(RequestTemplate requestTemplate) {

for(String key : headerKeyList){

String value = request.getHeader(key);

if(StringUtils.isNotBlank(value))

requestTemplate.header(key, value);

}

for(String key : ClientConstants.headMap.keySet()){

if(ClientConstants.headMap.get(key) != null && StringUtils.isNotBlank(ClientConstants.headMap.get(key).toString()))

requestTemplate.header(key,

ClientConstants.headMap.get(key).toString());

}

}

};

}

}

# 服务调用者开发

## 项目结构

|  |  |  |
| --- | --- | --- |
| 包名 | 类名 | 说明 |
| com.ule.wholesale.fxpurchase.config | InitBean | Feign的请求超时时间和重复请求设置 |
| PropertiesConfiguration | 自定义属性类 |
| com.ule.wholesale.fxpurchase.init |  | 需要在启动时初始化的数据等 |
| com.ule.wholesale.fxpurchase.controller |  | 返回页面的controller |
| com.ule.wholesale.fxpurchase.filter |  | 自己的过滤器 |
| com.ule.wholesale.fxpurchase.service |  | 自己的服务层，包括调用其他接口的实现和公共方法的实现等 |
| com.ule.wholesale.fxpurchase.util |  | 自己业务化的帮助类 |
| 配置文件 | bootstrap.yml | context-path、spring.application.name、mvc、consul discovery配置 |
| application.yml | 自定义属性，要覆盖公共配置的配置 |

## Consul配置

服务调用中和服务实现不同，调用者只需和服务中心通讯而不需要被服务中心发现，所以在项目中需要添加以下配置：

|  |  |
| --- | --- |
| 1  2  3  4  5 | z |

## FeignClient配置

一般无需改动，只需要引入该文件即可

@Component

public class InitBean {

@Value("${feignConnectTimeout:10000}")

private Integer connectTimeoutMillis;

@Value("${feignReadTimeout:60000}")

private Integer readTimeoutMillis;

// feign client默认的connectTimeout为10s，readTimeout为60.

// 单纯设置timeout，可能没法立马见效，因为默认的retry为5次

@Bean

Request.Options feignOptions() {

return new Request.Options(connectTimeoutMillis, readTimeoutMillis);

}

//禁用重发请求，对于修改操作可以防止数据被多次提交

@Bean

Retryer feignRetryer() {

return Retryer.NEVER\_RETRY;

}

}

## 服务启动

@EnableCircuitBreaker

@EnableHystrixDashboard

@EnableFeignClients("com.ule.wholesale.fxpurchase.api.\*")

//@PropertySource("classpath:selfproperties.yml")引入自定义配置文件

public class FxWebApplication extends CommonApplication{

protected SpringApplicationBuilder configure(SpringApplicationBuilder builder) {

return builder.sources(FxWebApplication.class);

}

public static void main(String[] args) {

SpringApplication.run(FxWebApplication.class, args);

}

//图片上传的servlet

@Bean

public ServletRegistrationBean initUploadImgServlet() {

ServletRegistrationBean registration = new ServletRegistrationBean(new UploadImgServlet());

registration.addUrlMappings("/uploadImgServlet");

return registration;

}

//登录验证的Filter

@Bean

public FilterRegistrationBean userLoginStatusFilter() {

FilterRegistrationBean filterRegistrationBean = new FilterRegistrationBean();

filterRegistrationBean.setFilter(new LoginFilter());

filterRegistrationBean.addUrlPatterns(LocalDeveloyEnv.isDev?"":"/\*");

filterRegistrationBean.addInitParameter("exclusions", "\*.js,\*.gif,\*.jpg,\*.png,\*.css,\*.ico,/druid/\*,/health");

return filterRegistrationBean;

}

//授权的Filter

@Bean

public FilterRegistrationBean authorizationFilter() {

FilterRegistrationBean filterRegistrationBean = new FilterRegistrationBean();

filterRegistrationBean.setFilter(new AuthorizationFilter());

filterRegistrationBean.addUrlPatterns(LocalDeveloyEnv.isDev?"":"/\*");

return filterRegistrationBean;

}

}

服务启动类继承fxCommon中的CommonApplication，并启用FeignClient，指定FeignClient的包路径，根据需要添加Filter和Servlet

通过mvn命令进行启动，-Dspring.profiles.active=dev spring-boot:run 该命令中指定了配置文件的profile，开发环境下会使用fxcommon中的dev环境配置文件，并且会根据是否存在spring.profiles.active变量判断是否为dev环境，为此定义了一个本地环境变量 LocalDeveloyEnv.isDev的全局变量

# springcloud配置文件参数说明

## 项目中属性修改和添加

### 接口实现项目配置文件

a) 引入数据源配置文件jdbc2.properties

b) 添加bootstrap.yml 并配置上下文 服务名 和Mybatis

server:

port: 8000

context-path: /purchaseService

spring:

application:

name: wholesale-purchase-provider

# MyBatis

mybatis:

typeAliasesPackage: com.ule.wholesale.fxpurchase.server.vo

mapperLocations: classpath:com/ule/wholesale/fxpurchase/mapper/\*.xml

configLocation: classpath:/mybatis-config.xml

### 服务调用者属性配置

server:

port: 8802

context-path: /fxPurchase

spring:

application:

#只有通过bootstrap配置，才能在zipkin中显示在打印的日志中

name: fxpurchase

#spring mvc

mvc:

view.prefix: /WEB-INF/view/

view.suffix: .jsp

static-path-pattern: /\*\*

cloud:

consul:

discovery:

#前端只需通过consul调用服务，不需要被发现，所以无需注册

register: false

## 自定义属性配置

添加application.yml 并在文件中配置需要的属性，如需根据环境配置放置到对应环境的文件夹中

注意：如果配置属性只有一层，即属性名: 属性值，那么在编写对应类时不要使用

@ConfigurationProperties了，如果想要通过不同类配置相应的属性，最好通过指定前缀，在对应的类上使用@ConfigurationProperties (prefix="属性的前缀")





## fxCommon中的属性覆盖

如果fxCommon中提供的默认配置无法满足自己项目需要可以进行配置覆盖，

覆盖时要覆盖的内容必须配置到application.开头的配置文件中，我们统一放置在application.yml的配置文件中，确保要覆盖的命名、层级相同

# 总结

根据前内容可以发现实际开发中只需clone批发采购项目进行清理，保留配置文件和配置类，所做修改并不多，在此把需要改动的类和配置重新列举强调一下

**fxCommon**

**无需修改，在POM中引入即可，如果common中存在的工具类希望大家不要在自行添加到自己项目中了。**

**接口服务**

MybatisConfiguration类需要修改，具体修改参照上文。

Swagger2设置controller的包路径

WebMvcConfigurer中的拦截器根据实际情况进行修改

bootstrap.yml 修改上下文，应用名，Mybatis配置

application.yml添加自定义属性

EJB使用注意事项，参考上文

**接口api**

**ClientConstants 修改**SERVER\_PATH和SERVICE\_NAME

所有使用了RequestBody的接口，必须指定RequestMapping的method为POST

**接口调用**

bootstrap.yml具体修改参照上文

入口类配置Feign的包路径修改为自己的api路径

**再强调一下启动方式：-Dspring.profiles.active=dev spring-boot:run**

使用该命令启动服务需要确保pom文件中有spring-boot-maven-plugin插件

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>