采用feign继承特性创建

注：以下方式在服务方提供fegin 启动没问题 打包出现了问题  可以将feign调用写在消费者解决

## 创建父pom 进行版本控制

尽量不要使用开发版本

```
  <properties>
       <spring-cloud.version>Greenwich.RELEASE</spring-cloud.version>
   </properties>

  <dependencyManagement>
       <dependencies>
           <dependency>
               <groupId>org.springframework.cloud</groupId>
               <artifactId>spring-cloud-dependencies</artifactId>
               <version>${spring-cloud.version}</version>
               <type>pom</type>
               <scope>import</scope>
           </dependency>
       </dependencies>

   </dependencyManagement>
```

### 所有的项目继承父pom 如下（修改你建的项目）

```
<parent>
       <groupId>com.example</groupId>
       <artifactId>demo-boot</artifactId>
       <version>0.0.1-SNAPSHOT</version>
 </parent>
```

# 首先创建注册中心

pom文件如下 继承父pom

```
 使用eureka注册中心
<dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
        </dependency>
    </dependencies>

```

### yml配置

```
server:
  port: 1000  端口号
eureka:
  client:
    service-url:
      defaultZone: http://localhost:1000/eureka  注册中心地址
    fetch-registry: false                        非集群不注册到注册中心上
    register-with-eureka: false                  同上
spring:
  application:
    name: center                                 应用名
```

### 启动类配置

```
@SpringBootApplication
@EnableEurekaServer      添加注册中心注解
public class DemoBootCenterApplication {

    public static void main(String[] args) {
        SpringApplication.run(DemoBootCenterApplication.class, args);
    }
```

```
8
9 }
```

服务提供者、服务消费者使用feign的继承功能 将接口路径进行提取到公共项目

Feign 中继承， 我们整体可以分两步来实现：

1. 在 **provider** 中实现公共接口；

2. 在 **feign-consumer** 中去调用接口。

## 创建common项目（项目是maven项目去掉启动类跟test文件夹）

pom文件 继承父pom

```
1     <dependencies>
2         <dependency>
3             <groupId>org.springframework.boot</groupId>
4             <artifactId>spring-boot-starter-web</artifactId>
5         </dependency>
6         <!--feign配置-->
7         <dependency>
8             <groupId>org.springframework.cloud</groupId>
9             <artifactId>spring-cloud-starter-openfeign</artifactId>
10        </dependency>
11    </dependencies>
12
```

创建统一接口

表明生产者

```
1 @FeignClient("provide")
2 public interface HelloClient {
3
4     @GetMapping
5     String hello(@RequestParam String name);
6 }
```

```
7
8
```

## 创建服务提供者provide

### pom配置 继承父pom 继承commons写的统一调用路径

```
    <dependencies>
        <!--必须有的web依赖-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <!--作为eureka客户端注册到注册中心-->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
        </dependency>
        <!--feign配置-->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-openfeign</artifactId>
            <version>2.0.2.RELEASE</version>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <!--继承common 使用配置的统一接口路径-->
        <dependency>
            <groupId>com.example</groupId>
            <artifactId>demo-boot-commons</artifactId>
        </dependency>
    </dependencies>
```

### yml配置

```yaml
server:
  port: 1001
eureka:
  client:
    service-url:
      defaultZone: http://localhost:1000/eureka
spring:
  application:
    name: provide
```

### 启动类配置

```java
@SpringBootApplication
@EnableEurekaClient    添加eureka客户端注册配置
public class DemoBootProvideApplication {

    public static void main(String[] args) {
        SpringApplication.run(DemoBootProvideApplication.class, args);
    }

}
```

### 服务提供方采用实现方式接受消费方的调用

```java
@RestController
public class HelloRest implements HelloClient {


    @Override
    public String hello(String name) {
        System.out.println("1001");
        return "hello: " + name + "!";
```

```
    }
}

```

回顾一下common的统一配置

```
@FeignClient("provide")
public interface HelloClient {

    @GetMapping
    String hello(@RequestParam String name);
}
```

## 优缺点分析

通过上面案例的搭建， 相信大家对 Feign 的继承特性已经有了一个大致的了解， 那么这种写法和上篇文章我们的写

法各自有什么优缺点呢？ 我们来分析下：

1. 使用**继承特性**， 代码简洁明了， 不易出错， 不必担心接口返回值是否写对， 接口地址是否写对。 如果接口地址有

变化， 也不用 provider 和 feign-consumer 大动干戈， 只需要修改 commons 模块即可，provider 和 feignconsumer 就自然变了；

2. 前面提到的在 feign-consumer 中绑定接口时， 如果是 key/value 形式的参数或者放在 header 中的参数， 就必须

要使用 @RequestParam 注解或者 @RequestHeader 注解， 这个规则在这里一样适用。即在 commons 中定义

接口时， 如果涉及到相关参数， 该加的@RequestParam 注解或者 @RequestHeader 注解一个都不能少；

3. 当然， 使用了继承特性也不是没有缺点。 继承的方式将 provider 和 feign-consumer 绑定在一起， 代码耦合度变

高， 一变俱变， 此时就需要严格的设计规范， 否则会牵一发而动全身， 增加项目维护的难度。

# 创建服务消费者

## pom配置 继承父pom

```
<dependencies>
        <!--web基础配置-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <!--客户端基础配置-->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <!--common基础配置-->
        <dependency>
            <groupId>com.example</groupId>
            <artifactId>demo-boot-commons</artifactId>
        </dependency>


    </dependencies>
```

## yml配置

```
server:
  port: 1002
eureka:
  client:
```

```
    service-url:
      defaultZone: http://localhost:1000/eureka
spring:
  application:
    name: customer
```

### 启动类配置

配置feign 必须配置**EnableFeignClients**扫描包不然会启动项目失败 找不到服务提供者写的调用接口

```
@SpringBootApplication
@EnableEurekaClient
@EnableFeignClients(basePackages ="com.example.demo.boot.*")
public class DemoBootCustomerApplication {

    public static void main(String[] args) {
        SpringApplication.run(DemoBootCustomerApplication.class, args);
    }

}
```

### 消费者接口

```
@RestController
public class CustomerRest {


    @Autowired
    HelloClient helloClient;

    @GetMapping("hello")
    public String hello(String name) {
        return helloClient.hello(name);
    }
}

```

再次赋值一个服务提供者 修改端口号 启动后模拟集群  调用接口就会看到循环输出两个服务提供者提供的服务

调用 http://localhost:1002/hello?name=aaa 获取你想要的结果吧!

## 使用spring cloud gateway 搭建简单的网关

pom 依赖父pom

```
    <dependencies>
        <!--gateway相关-->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-gateway</artifactId>
        </dependency>
        <!--客户端相关-->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
        </dependency>
    </dependencies>
```

yml配置

```
server:
  port: 1020
eureka:
  client:
    service-url:
      defaultZone: http://localhost:1000/eureka
spring:
  application:
    name: gateway
  cloud:
     gateway:
        discovery:
```

```
          locator:
             enabled: true              # 是否开启通过注册中心进行路由转发的功能，
                                        #  serviceId 转发到服务， 默认为 false。
             lowerCaseServiceId: true   # 指定服务调用使用小写 默认大写
logging:
  level:
    org.springframework.cloud.gateway: debug
```

启动类配置 添加客户端注解 注册到注册中心

```
@SpringBootApplication
@EnableEurekaClient
public class DemoBootGatewayApplication {

    public static void main(String[] args) {
        SpringApplication.run(DemoBootGatewayApplication.class, args);
    }

}

```

采用下面格式就可以使用gateway进行转发了

```
http://网关地址：  端口/服务中心注册 serviceId/具体的 url
```