# 项目准备

## 1、创建项目feign-consumer

## 2、继续使用之前的服务提供者hello-service

# 2、简单的声明式服务调用

## 1、pom.xml

*<?*xml version="1.0" encoding="UTF-8"*?>*<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.didispace</groupId>  
 <artifactId>feign-consumer</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <packaging>jar</packaging>  
  
 <name>feign-consumer</name>  
 <description>Demo project for Spring Boot</description>  
  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>1.3.7.RELEASE</version>  
 <relativePath/> <!-- lookup parent from repository -->  
 </parent>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>  
 <java.version>1.8</java.version>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-actuator</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-eureka</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-starter-feign</artifactId>  
 </dependency>  
  
 </dependencies>  
  
 <dependencyManagement>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.cloud</groupId>  
 <artifactId>spring-cloud-dependencies</artifactId>  
 <version>Brixton.SR5</version>  
 <type>pom</type>  
 <scope>import</scope>  
 </dependency>  
 </dependencies>  
 </dependencyManagement>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
  
</project>

## 2、注解开启 spring cloud feign申明式服务调用功能

@EnableFeignClients  
@EnableDiscoveryClient  
@SpringBootApplication  
public class ConsumerApplication {  
  
  
  
 public static void main(String[] args) {  
 SpringApplication.run(ConsumerApplication.class, args);  
 }  
  
}

## 3、定义HelloService接口，通过@FeignClient 注解指定服务名称来绑定服务，然后通过springMVc 注解绑定服务提供者提供的REST接口，和服务提供者中的url是一模一样哦

@FeignClient(name="HELLO-SERVICE")  
public interface HelloService {  
  
 @RequestMapping("/hello")  
 String hello();  
  
}

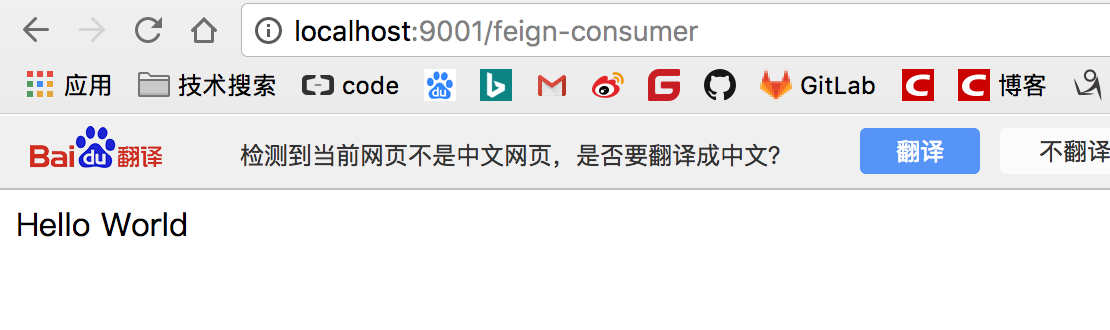
## 4、Controller 通过调用上面的接口来实现从本客户端发起对hello-service服务接口的hello的调用

//这里用来测试对feign客户端的调用  
@RestController  
public class ConsumerController {  
  
 @Autowired  
 HelloService helloService;  
  
 @RequestMapping(value = "/feign-consumer", method = RequestMethod.***GET***)  
 public String helloConsumer() {  
 return helloService.hello();  
 }  
}

## 5、同ribbon实现的服务消费一样，需要指定服务的注册中心

spring.application.name=feign-consumer  
server.port=9001  
  
eureka.client.serviceUrl.defaultZone=http://localhost:1111/eureka/

## 6、开始验证，浏览器中访问http://localhost:9001/feign-consumer



# 3、参数绑定

## 1、为了验证参数绑定，再服务提供者中再添加一些方法

@RestController  
public class HelloController {  
  
 private final Logger logger = Logger.getLogger(getClass());  
  
 @Autowired  
 private DiscoveryClient client;  
  
 @RequestMapping(value = "/hello", method = RequestMethod.***GET***)  
 public String hello() {  
  
 ServiceInstance instance = client.getLocalServiceInstance();  
 logger.info("/hello, host:" + instance.getHost() + ", service\_id:" + instance.getServiceId());  
 return "Hello World";  
 }  
  
  
  
 @RequestMapping(value = "/hello1", method = RequestMethod.***GET***)  
 String hello(@RequestParam("name") String name) {  
  
 return "hello" +name;  
 }  
  
 @RequestMapping(value = "/hello2", method = RequestMethod.***GET***)  
 User hello(@RequestHeader("name") String name, @RequestHeader("age") Integer age){  
 return new User(name,age);  
 }  
  
   
  
 @RequestMapping(value = "/hello3", method = RequestMethod.***POST***)  
 String hello(@RequestBody User user){  
 return "hello"+user.getName()+","+user.getAge();  
 }  
  
  
  
}

## 2、再声明式服务调用的service接口中添加如下，同时要创建和上面一模一样的User类

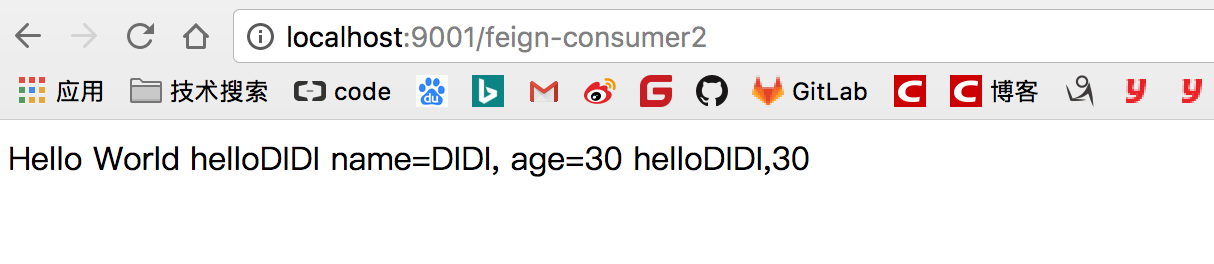
### 1、注意点：下面注解中的value绝对不仅能为空，这是因为它和feign合作了

//绑定服务提供者，并使用spirngMvc 注解绑定具体REST接口  
@FeignClient(name="HELLO-SERVICE")  
public interface HelloService {  
  
 @RequestMapping("/hello")  
 String hello();  
  
  
 @RequestMapping(value = "/hello1", method = RequestMethod.***GET***)  
 String hello(@RequestParam("name") String name) ;  
  
 @RequestMapping(value = "/hello2", method = RequestMethod.***GET***)  
 User hello(@RequestHeader("name") String name, @RequestHeader("age") Integer age);  
  
 @RequestMapping(value = "/hello3", method = RequestMethod.***POST***)  
 String hello(@RequestBody User user);  
  
  
}

## 3、Controller 调用声明式服务传参开始吧

/\*\*  
 \* 2、测试传参  
 \*/  
  
@RequestMapping(value = "/feign-consumer2", method = RequestMethod.***GET***)  
public String helloConsumer2() {  
 StringBuilder sb = new StringBuilder();  
 sb.append(helloService.hello()).append("\n");  
 sb.append(helloService.hello("DIDI")).append("\n");  
 sb.append(helloService.hello("DIDI", 30)).append("\n");  
 sb.append(helloService.hello(new User("DIDI", 30))).append("\n");  
 return sb.toString();  
}

## 4、开始验证，浏览器中访问



# 4、继承特性

## 解释：

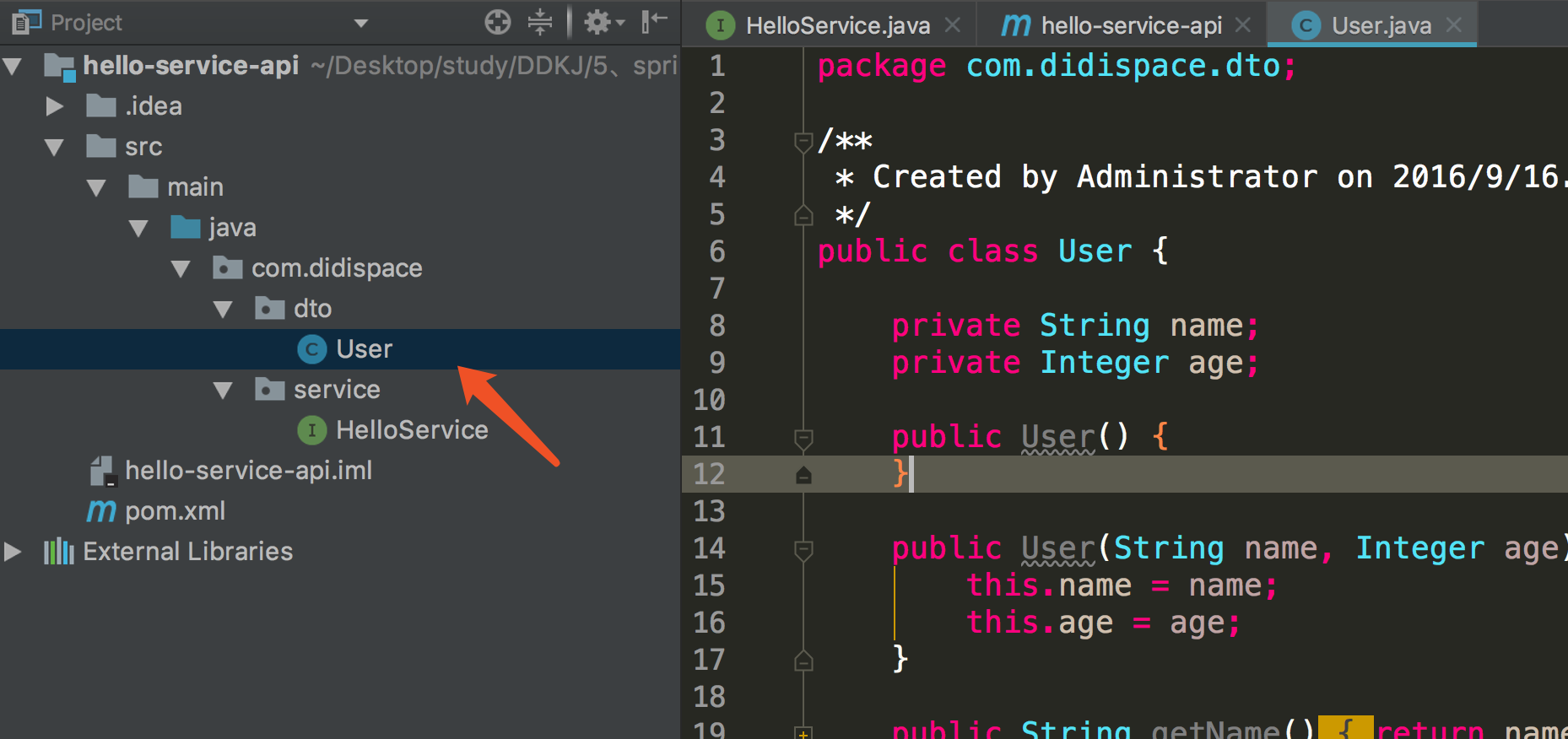
### 1、这个时候我们会发现，在声明式服务中的User和服务提供者中的User的内容一抹一样，但是位置却不是一样的，所以这样很不利于维护，因为修改了一处，还要继续去修改另外的一处。

### 2、在声明式@FeginClient中提供的接口其实服务提供者也可以使用。所以，请看下面

## 1、创建一个maven项目 hello-service-api,pom.xml如下

*<?*xml version="1.0" encoding="UTF-8"*?>*<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.didispace</groupId>  
 <artifactId>hello-service-api</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <packaging>jar</packaging>  
  
 <name>hello-service-api</name>  
  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>1.3.7.RELEASE</version>  
 <relativePath/> <!-- lookup parent from repository -->  
 </parent>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>  
 <java.version>1.8</java.version>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
 </dependencies>  
  
</project>

## 2、将3中的User复制过来



## 3、将声明式服务调用中的接口类HelloService 复制过来，还要做一番修改哦

### 1、将头部前缀修改为refactor

### 2、url路径也修改更名为hello4 5 6

@RequestMapping("/refactor")  
public interface HelloService {  
  
 @RequestMapping(value = "/hello4", method = RequestMethod.***GET***)  
 String hello(@RequestParam("name") String name) ;  
  
 @RequestMapping(value = "/hello5", method = RequestMethod.***GET***)  
 User hello(@RequestHeader("name") String name, @RequestHeader("age") Integer age);  
  
 @RequestMapping(value = "/hello6", method = RequestMethod.***POST***)  
 String hello(@RequestBody User user);  
  
}

## 4。下面开始对服务提供者进行重构，

### 1.、在hellop-service项目pom.xml中引入hello-service-api的依赖,一定要记得对它进行mvn install

<dependency>  
 <groupId>com.didispace</groupId>  
 <artifactId>hello-service-api</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
</dependency>

### 2、重点来了，创建controller类基础上面的HelloService,没有必要在加@RequestMapping，但是其他的是一定要加的

@RestController  
public class RefactorHelloController implements HelloService{  
  
  
 public String hello(@RequestParam("name") String name) {  
  
 return "hello" +name;  
 }  
  
 public User hello(@RequestHeader("name") String name, @RequestHeader("age") Integer age){  
 return new User(name,age);  
 }  
   
  
 public String hello(@RequestBody User user){  
 return "hello"+user.getName()+","+user.getAge();  
 }  
  
}

## 5、下面开始对声明式服务调用开始改造

### 1、引入pom依赖

<dependency>  
 <groupId>com.didispace</groupId>  
 <artifactId>hello-service-api</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
</dependency>

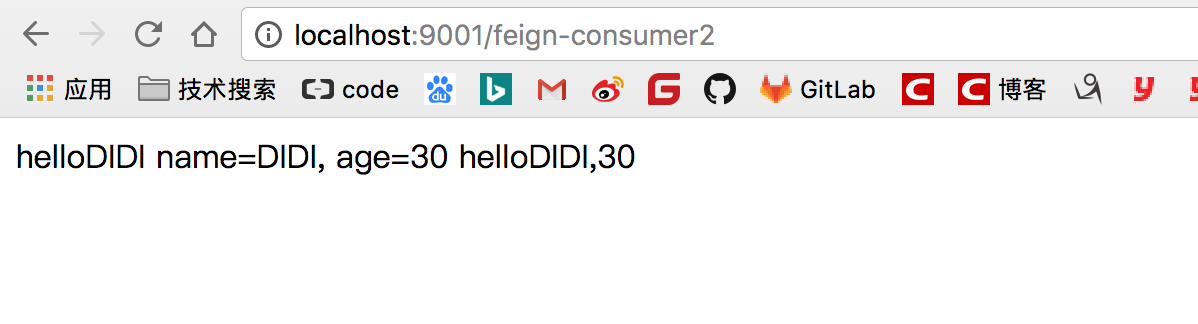
### 2、绑定服务提供者，接口让他来继承HelloService，这样就保证了和服务提供者中的R EST是一致的了

//绑定服务提供者  
@FeignClient(name="HELLO-SERVICE")  
public interface RefactorHelloService extends HelloService {  
  
  
}

### 3、Controller开测

//这里用来测试对feign客户端的调用  
@RestController  
public class ConsumerController {  
  
 @Autowired  
 RefactorHelloService refactorHelloService;  
  
 /\*\*  
 \* 2、测试传参  
 \*/  
  
 @RequestMapping(value = "/feign-consumer2", method = RequestMethod.***GET***)  
 public String helloConsumer2() {  
 StringBuilder sb = new StringBuilder();  
 sb.append(refactorHelloService.hello("DIDI")).append("\n");  
 sb.append(refactorHelloService.hello("DIDI", 30)).append("\n");  
 sb.append(refactorHelloService.hello(new User("DIDI", 30))).append("\n");  
 return sb.toString();  
 }  
}

## 6、开始测试，浏览器访问 http://localhost:9001/feign-consumer2 成功



# 5、服务降级配置，类似于断路器(使用3中的项目开发)

## 1、声明式服务调用项目中添加service HelloServiceFallback，使它实现HelloService 用来作为当调用服务提供者失败后返回的内容

@Component  
public class HelloServiceFallback implements HelloService {  
  
  
 @Override  
 public String hello(@RequestParam("name") String name) {  
 return "error";  
 }  
  
 @Override  
 public User hello(@RequestHeader("name") String name, @RequestHeader("age") Integer age) {  
 return new User("未知", 0);  
 }  
  
 @Override  
 public String hello(@RequestBody User user) {  
 return "error";  
 }  
  
}

## 2.、@Feign 绑定服务的时候就开始，添加降级配置

//绑定服务提供者，并使用spirngMvc 注解绑定具体REST接口  
@FeignClient(name="HELLO-SERVICE" ,fallback =HelloServiceFallBack.class )  
public interface HelloService {  
  
 @RequestMapping("/hello")  
 String hello();  
  
 @RequestMapping(value = "/hello1", method = RequestMethod.***GET***)  
 String hello(@RequestParam("name") String name) ;  
  
 @RequestMapping(value = "/hello2", method = RequestMethod.***GET***)  
 User hello(@RequestHeader("name") String name, @RequestHeader("age") Integer age);  
  
 @RequestMapping(value = "/hello3", method = RequestMethod.***POST***)  
 String hello(@RequestBody User user);  
  
  
}

## 3、浏览器测试（不开启服务提供者） http://localhost:9001/feign-consumer2

