

Feign

6.1 Feign 简介使用

6.1.1 简介

Feign是一个声明式的Web服务客户端,要想使用 Feign 只需要定义一个接口并且添加响应的注解即可,支持包括Feign注解和JAX-RS注解。Feign还支持可插拔编码器和解码器。Spring Cloud增加了对Spring MVC注解的支持,使Feign可以使用SpringMVC的注解来进行响应的操作,并使用Spring Web中默认使用的 **HttpMessageConverters**。Spring Cloud集成Ribbon和Eureka以在使用Feign时提供负载均衡的http客户端

6.1.2 基本使用

参考 10consumer-eureka-feign,此项目是以05consumer-eureka为基础修改的

04provider-eureka 作为提供者

05consumer-eureka 注册中心

6.1.2.1 导入依赖

要想使用feign,需要导入依赖包,如果是老版本的cloud,导入的依赖包是netflix的,F版本后更换为openfeign,为了从eureka中拉取数据,应该还需要导入eureka客户端,同时保证项目是个web项目,还需要导入starter-web

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

6.1.2.2 创建Feign客户端接口

Feign使用接口的方式来定义发出的请求,其实可以认为就是对我们之前代码的封装,因为之前的代码我们都是固定写法,请求某个服务下的某个地址,参数是什么,返回值是什么,既然有规律我们就可以定义规则进行抽取

Feign就是定义规则进行抽取,规则就是接口上面的FeignClient注解中的参数代表要请求的服务
注解内部方法的返回值代表请求服务后返回的值

方法上面的GetMapping等注解代表的是要请求的服务的地址

方法参数代表请求服务时候传递的参数

通过以上规则可以在外面调用feign的方法的时候转换成对应的代码,发起请求,请求的代表参考我们使用ribbon项目的代码

和我们之前controller不同,controller是将GetMapping对应的地址映射到与之对应的方法,而Feign上的注解是将对应的方法映射到与之对应的地址,刚好是反过来的,一个是请求地址到方法,一个是请求方法到地址

```
import com.qianfeng.microservice.consumer.pojo.User;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;

/**
 * Created by jackiechan on 19-1-9/上午10:27
 *
 * @Author jackiechan
 */
@FeignClient("04provider-eureka")//声明当前接口是一个feignclient,参数为想访问哪个服务,写的是在eureka上面的服务名字,并且Feign内置了Ribbon,此处可以直接实现负载均衡,Spring扫描到这个注解后会自动创建代理对象
public interface FeignClient01 {
    @GetMapping("/user/info/{id}")
    User getUserById(@PathVariable("id") Long id);//注意在controller里面PathVariable的路径参数和方法的形参名字一样的时候可以忽略里面的名字.但是在feign里面必须写

    @PostMapping("/user/save")//这个方法上面只要有参数 就会被发出去,如果传递的是复杂对象,不管这里是不是GET都会以post方式发出去,出错还是不出错,取决于提供者那边的限定,如果提供者那边限定了是get就会报错405,如果提供者那边没有限定或者是post就不会出错,普通参数则保持双方一直即可
    String save(User user);
}
```

6.1.2.3 修改Controller

因为我们开始使用feign,那么之前的代码就不用,全部修改为通过feign来访问

```
import com.qianfeng.microservice.consumer.feign.FeignClient01;
import com.qianfeng.microservice.consumer.pojo.User;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
```

```

/**
 * Created by jackiechan on 19-1-8/下午12:00
 *
 * @Author jackiechan
 */
@RestController
@RequestMapping("/userconsumer")
public class UserController {
    @Autowired //此处不需要我们手动创建这个对象,spring会自动帮我们创建,因为这个接口上面有注解,会创建代理对象
    private FeignClient01 feignClient01;

    @GetMapping("/info/{id}")
    public User getUserById(@PathVariable Long id) throws Exception {
        return feignClient01.getUserById(id); //调用此方法会按照规则,通过代理对象内部生成我们之前使用的代码,来通过resttemplate来进行访问
    }
    @PostMapping("/save")
    public String addUser(@RequestBody User user) {
        String result = feignClient01.save(user);
        return result;
    }
}

```

6.1.2.4 修改启动程序

```

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
import org.springframework.cloud.openfeign.EnableFeignClients;

/**
 * Created by jackiechan on 19-1-8/下午12:04
 *
 * @Author jackiechan
 */
@SpringBootApplication
@EnableEurekaClient
@EnableFeignClients //开启对feign的支持,开启此注解后,我们的Feign客户端接口才会有效
public class ConsumerStarter {

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

}

```

6.1.2.5 application.yml

此文件中没有变化

```
server:
  port: 9000
eureka: #注册中心的地址
  client:
    service-url:
      defaultZone: http://zhangsan:abc@localhost:10000/eureka #curl风格
spring:
  application:
    name: 10consumer-eureka-feign
```

6.1.2.6 启动测试

启动我们的eureka和提供者,再启动我们当前这个消费者,通过访问可以拿到提供者的数据

6.1.3 自定义Feign配置

参考项目11consumer-eureka-feign-config,是在10的基础上修改的

04provider-eureka 作为提供者

05consumer-eureka 注册中心

6.2.1 配置介绍

Spring Cloud 中心支持为每个 Feign 通过 @FeignClient注解 的name 属性进行命名(name 其实是 value 属性的别。Spring Cloud根据需要通过使用 `FeignClientsConfiguration` 为每个已命名的客户端创建一个新的 `ApplicationContext` 集合(通过查看FeignClient注解的configuration属性可以得知 `FeignClientsConfiguration`是默认属性类)。也就是FeignClient注解会创建一个 Spring 的子容器.这包含 (除其他外) `feign.Decoder` , `feign.Encoder` 和 `feign.Contract` 。



注意和 Ribbon 一样,官方不建议我们将配置文件放到 `@ComponentScan` or `@SpringBootApplication` 的扫描目录中,处理方式和Ribbon一样

`FooConfiguration` does not need to be annotated with `@Configuration` . However, if it is, then take care to exclude it from any `@ComponentScan` that would otherwise include this configuration as it will become the default source for `feign.Decoder` , `feign.Encoder` , `feign.Contract` , etc., when specified. This can be avoided by putting it in a separate, non-overlapping package from any `@ComponentScan` or `@SpringBootApplication` , or it can be explicitly excluded in `@ComponentScan` .



`serviceId`已经过时了,现在使用 `name`

The `serviceId` attribute is now deprecated in favor of the `name` attribute.



在通过url来指定feign访问的服务器的时候,之前是不需要name属性的,注意 name 属性现在是必须的了

Previously, using the `url` attribute, did not require the `name` attribute. Using `name` is now required.

可以修改的属性

Spring Cloud Netflix provides the following beans by default for feign (`BeanType` beanName:

`ClassName`):

- `Decoder` feignDecoder: `ResponseEntityDecoder` (which wraps a `SpringDecoder`) 解码
- `Encoder` feignEncoder: `SpringEncoder` 编码
- `Logger` feignLogger: `Slf4jLogger` 日志
- `Contract` feignContract: `SpringMvcContract` 契约格式,默认是springmvc契约,就是注解
- `Feign.Builder` feignBuilder: `HystrixFeign.Builder` 熔断机制
- `Client` feignClient: if Ribbon is enabled it is a `LoadBalancerFeignClient` , otherwise the default feign client is used.

6.2.2 配置Feign的契约(注解)

Feign默认使用的是SpringMVC的契约格式,也就是SpringMVC的注解,我们可以修改为Feign原生的格式

6.2.2.1 创建配置类

注解配置类不要被ComponentScan扫描到,此我们使用把文件挪到高层包的方式解决,如果此配置是默认配置,则可以放到正常目录下

```
import feign.Contract;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
 * Created by jackiechan on 19-1-9/上午11:16
 *
 * @Author jackiechan
 */
@Configuration //一定要加上这个注解
@Component
public class FeignClient01Config {
    @Bean //不要忘记这个注解
    public Contract contract() {
        return new feign.Contract.Default(); //修改feign的契约为原生默认,是feign的原生默认,
        不是我springcloud包装后默认(springmvc),修改后将无法使用springmvc的注解
    }
}
```

6.2.2.2 修改Feign客户端接口

修改方法上面的注解为Feign的,如果仍旧使用mvc的会出错

```
import com.qianfeng.microservice.FeignClient01Config;
import com.qianfeng.microservice.consumer.pojo.User;
import feign.Param;
import feign.RequestLine;
import org.springframework.cloud.openfeign.FeignClient;

/**
 * Created by jackiechan on 19-1-9/上午10:27
 *
 * @Author jackiechan
 */
@FeignClient(value = "04provider-eureka", configuration = FeignClient01Config.class) //
//声明当前接口是一个feignclient,参数为想访问哪个服务,写的是在eureka上面的服务的名字,并且指定配置文件
//为我们
public interface FeignClient01 {
    //GetMapping("/user/info/{id}")
    //User getUserById(@PathVariable("id") Long id); //注意在controller里面PathVariable的路径参数和方法的形参名字一样的时候可以忽略里面的名字.但是在feign里面必须写
    @RequestLine("GET /user/info/{id}") //因为我们现在设置的契约模式feign自带的默认值,不在支持springmvc的注解,所以使用requestline,语法格式"请求方式 空格 请求地址"
    User getUserById(@Param("id") Long id); //注意 这个参数的注解需要换为@Param注解,feign中的

    //PostMapping("/user/save") //这个方法上面只要有参数 就会被发出去,如果传递的是复杂对象,不管
    //这里是不是GET都会以post方式发出去,出错还是不出错,取决于提供者那边的限定
    @RequestLine("POST /user/save")
    String save(User user);
}
```

6.2.2.3 Controller

```
import com.qianfeng.microservice.consumer.feign.FeignClient01;
import com.qianfeng.microservice.consumer.feign.FeignClient02;
import com.qianfeng.microservice.consumer.pojo.User;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

/**
 * Created by jackiechan on 19-1-8/下午12:00
 *
 * @Author jackiechan
 */
@RestController
@RequestMapping("/userconsumer")
public class UserController {
    @Autowired //此处不需要我们手动创建这个对象,spring会自动帮我们创建,因为这个接口上面有注解
    private FeignClient01 feignClient01;
```

```

    @GetMapping("/info/{id}")
    public User getUserById(@PathVariable Long id) throws Exception {
        return feignClient01.getUserById(id);
    }
    @PostMapping("/save")
    public String addUser(@RequestBody User user) {
        String result = feignClient01.save(user);
        return result;
    }
}

```

6.2.24 启动测试

其他代码和基本使用的代码没有区别,启动注册中心,提供者,和本程序测试

6.2.3 使用URL方式

有时候我们要访问的服务并不是Java写的,也没有注册到我们的Eureka中,所以无法使用服务名字的方式来进行访问,所以通过url地址来指定服务的地址来进行访问

本例子中通过访问eureka server地址获取数据的方式来测试URL访问,因为我们给Eureka设置了账号密码,所以需要通过配置类来设置账号和密码

6.2.3.1 创建Feign Client

```

import com.qianfeng.microservice.FeignClient02Config;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;

/**
 * Created by jackiechan on 19-1-9/上午10:27
 *
 * @Author jackiechan
 */
@FeignClient(name = "dasdasdasdasdas", url = "http://localhost:10000", configuration =
FeignClient02Config.class) //声明当前接口是一个feignclient,参数为想访问哪个网址,因为10000这个地
址需要账号密码,所以我们还需要写一个配置类来设置账号密码
public interface FeignClient02 {
    @GetMapping("/eureka/apps/{serviceid}") //这个地址是eureka给我们提供的查看服务信息的地址
    String getServiceinfoById(@PathVariable("serviceid") String serviceid); //注意在
controller里面PathVariable的路径参数和方法的形参名字一样的时候可以忽略里面的名字.但是在feign里面
必须写

```

```
}
```

6.2.3.2 配置类

```
import feign.auth.BasicAuthRequestInterceptor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
 * Created by jackiechan on 19-1-9/上午11:16
 *
 * @Author jackiechan
 */
@Configuration
public class FeignClient02Config {

    //    @Bean //注意设置用户名和密码的拦截器不是这个
    //    public BasicAuthorizationInterceptor authorizationInterceptor() {
    //        return new BasicAuthorizationInterceptor("zhangsan", "abc");
    //    }
    //设置账号和密码
    @Bean
    public BasicAuthRequestInterceptor authRequestInterceptor() {
        return new BasicAuthRequestInterceptor("zhangsan", "abc");
    }
}
```

6.2.3.3 修改Controller

```
import com.qianfeng.microservice.consumer.feign.FeignClient01;
import com.qianfeng.microservice.consumer.feign.FeignClient02;
import com.qianfeng.microservice.consumer.pojo.User;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

/**
 * Created by jackiechan on 19-1-8/下午12:00
 *
 * @Author jackiechan
 */
@RestController
@RequestMapping("/userconsumer")
public class UserController {

    @Autowired //此处不需要我们手动创建这个对象, spring会自动帮我们创建, 因为这个接口上面有注解
    private FeignClient01 feignClient01;

    @Autowired //此处不需要我们手动创建这个对象, spring会自动帮我们创建, 因为这个接口上面有注解
    private FeignClient02 feignClient02;

    @GetMapping("/info/{id}")
    public User getUserById(@PathVariable Long id) throws Exception {
```



```
        return feignClient01.getUserById(id);
    }

    @PostMapping("/save")
    public String addUser(@RequestBody User user) {
        String result = feignClient01.save(user);
        return result;
    }

    @GetMapping("/serviceinfo/{id}")
    public String getServiceinfoById(@PathVariable String id) throws Exception {
        return feignClient02.getServiceinfoById(id);
    }
}
```

6.2.3.4 启动测试

启动我们的eureka和当前服务,通过访问服务下的/serviceinfo/{id} 来获取eureka上面的服务信息