博学谷
www.boxuegu.com

第三章 Spring Cloud Feign 声明式服务调用

一样的在线教育，不一样的教学品质

目录 Contents

使用 Feign 调用服务

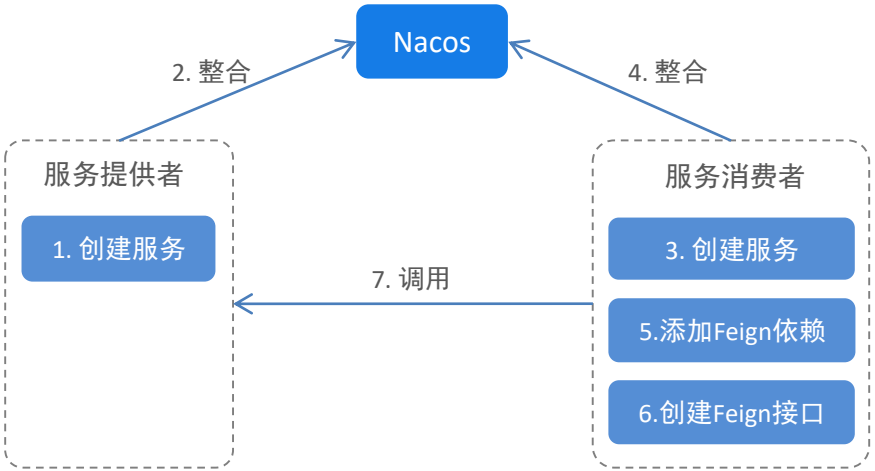
小节导学

上一章我们做服务调用时使用的是 RestTemplate，现在我们要改用 Feign 来实现服务调用。

本节目标是掌握使用 Feign 服务调用的具体开发方法。

实践步骤：

- 创建一个服务提供者，整合 Nacos
- 创建一个服务消费者，整合 Nacos
- 服务消费者添加 Feign 依赖
- 服务消费者创建 Feign 客户端接口
- 服务消费者使用 Feign 接口调用服务提供者
- 启动并测试



使用 Feign 调用服务

服务提供者

添加 Nacos 依赖：

```
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-
nacos-discovery</artifactId>
</dependency>
```

启动类添加服务发现注解

```
@EnableDiscoveryClient
```

属性配置：

```
server:
  port: 8081
spring:
  application:
    name: service-provider
  cloud:
    nacos:
      discovery:
        server-addr: localhost:8848
```

使用 Feign 调用服务

服务消费者

添加 Feign 依赖：

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

添加 Feign 注解：

```
@EnableFeignClients
```

使用 Feign 调用服务

服务消费者

创建 Feign 接口：

```
@FeignClient(name="service-provider")
public interface FeignClientDemo {
    @GetMapping("/demo")
    String demo(@RequestParam("name") String name);
}
```

服务提供者的服务名

服务提供者的接口名

服务提供者接口的参数

调用 Feign 接口：

```
@Autowired
FeignClientDemo feignClientDemo;
...
public String demo(String name){
    return "feigin consumer get : " + feignClientDemo.demo(name);
}
```

引入 Feign 接口

调用 Feign 接口

使用 Feign 调用服务-总结

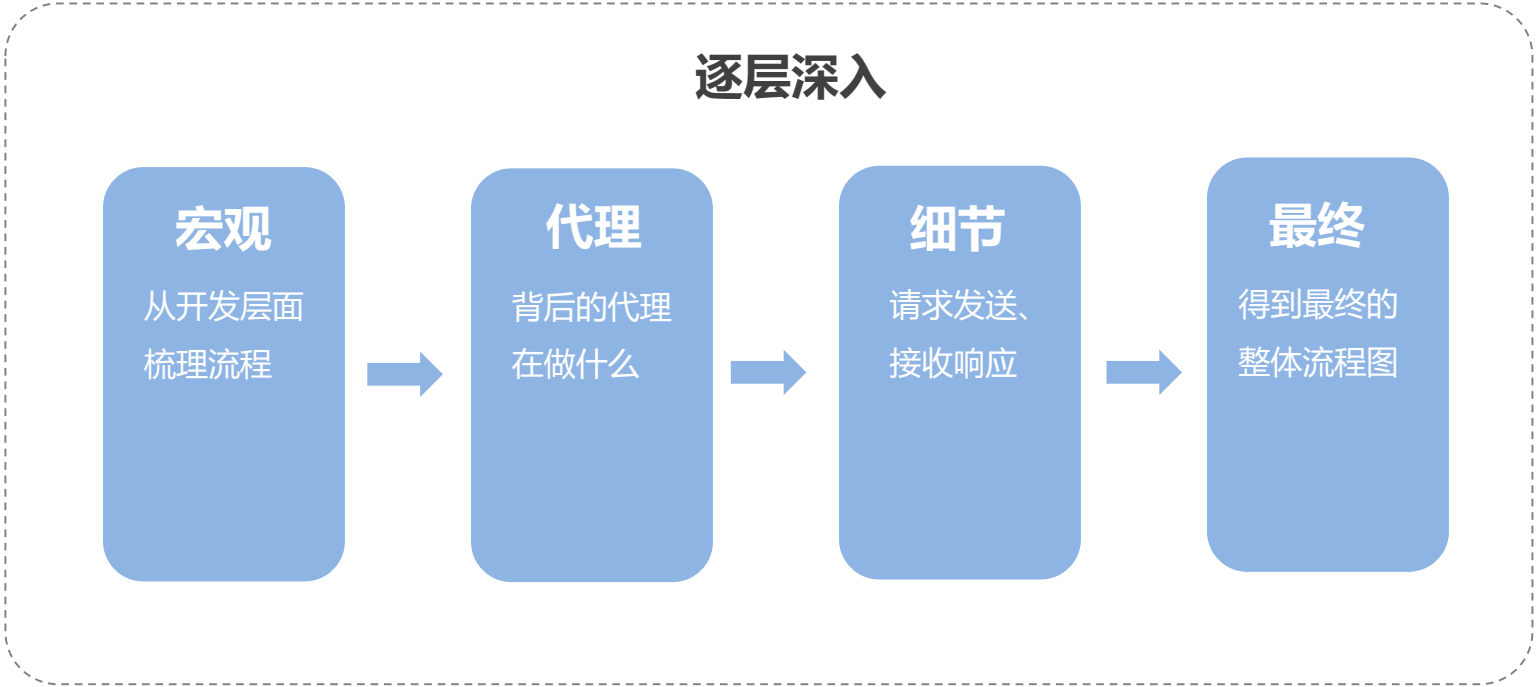
重难点

1. Feign 的概念
2. Feign 接口创建方法
3. 使用 Feign 调用服务的方法

目录 Contents

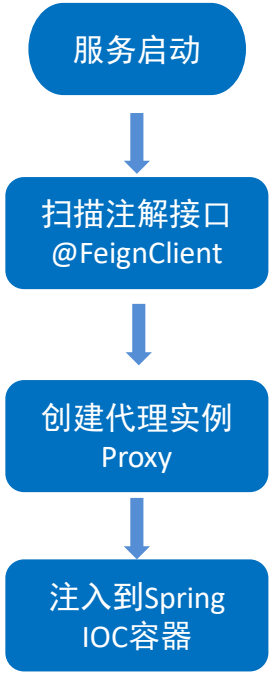
Feign 工作原理

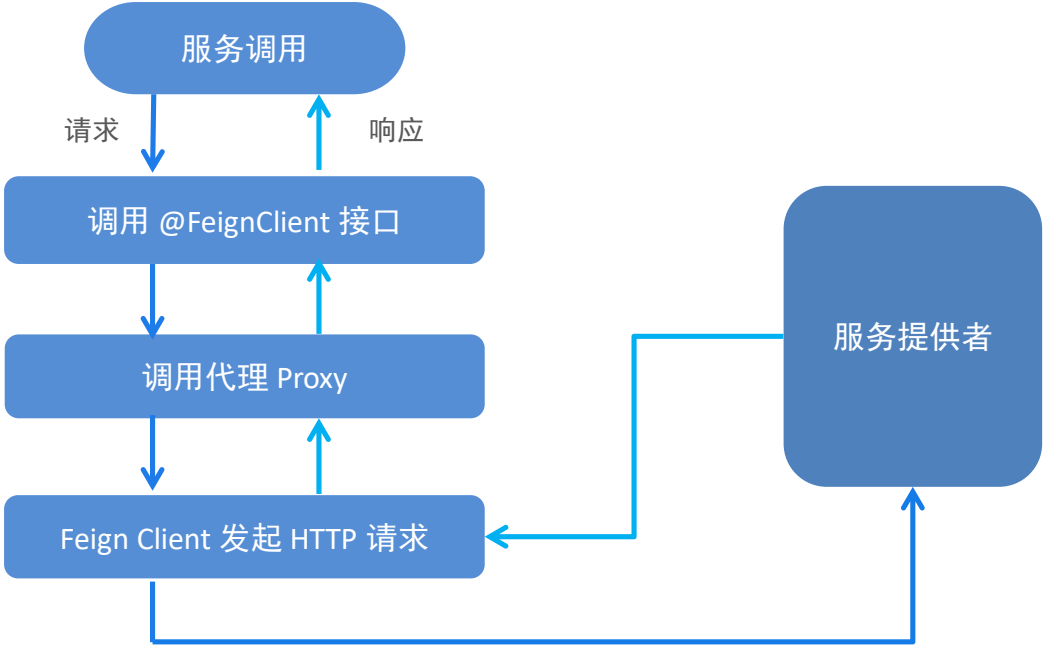
小节导学



Feign 工作原理

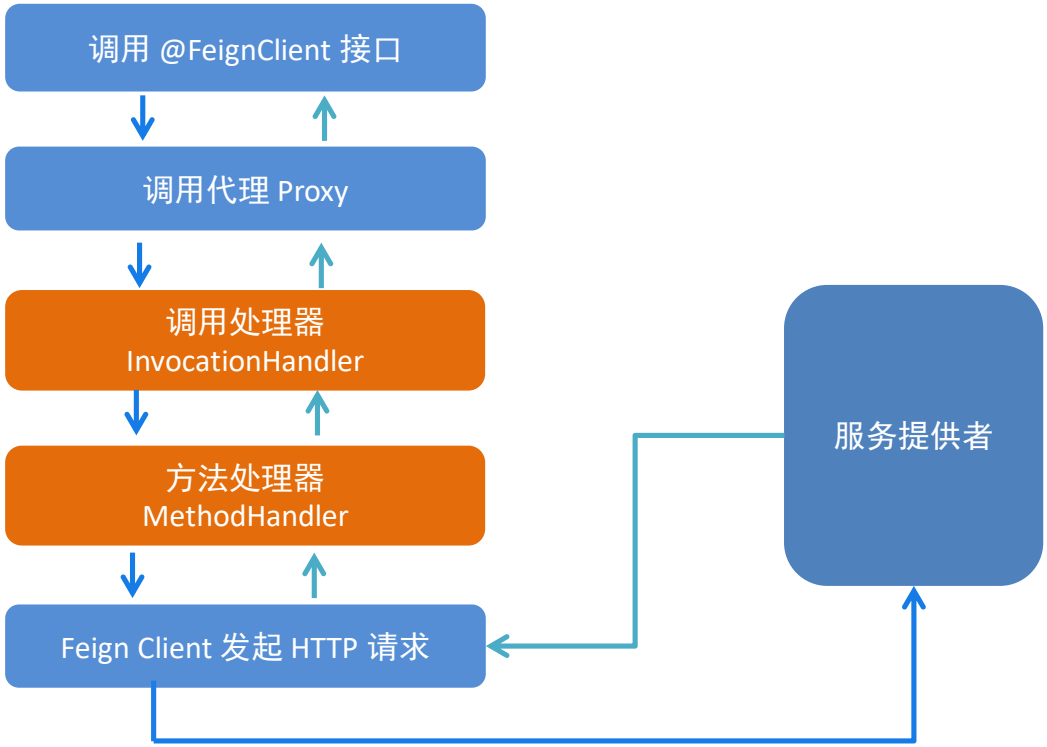
从开发层面梳理流程





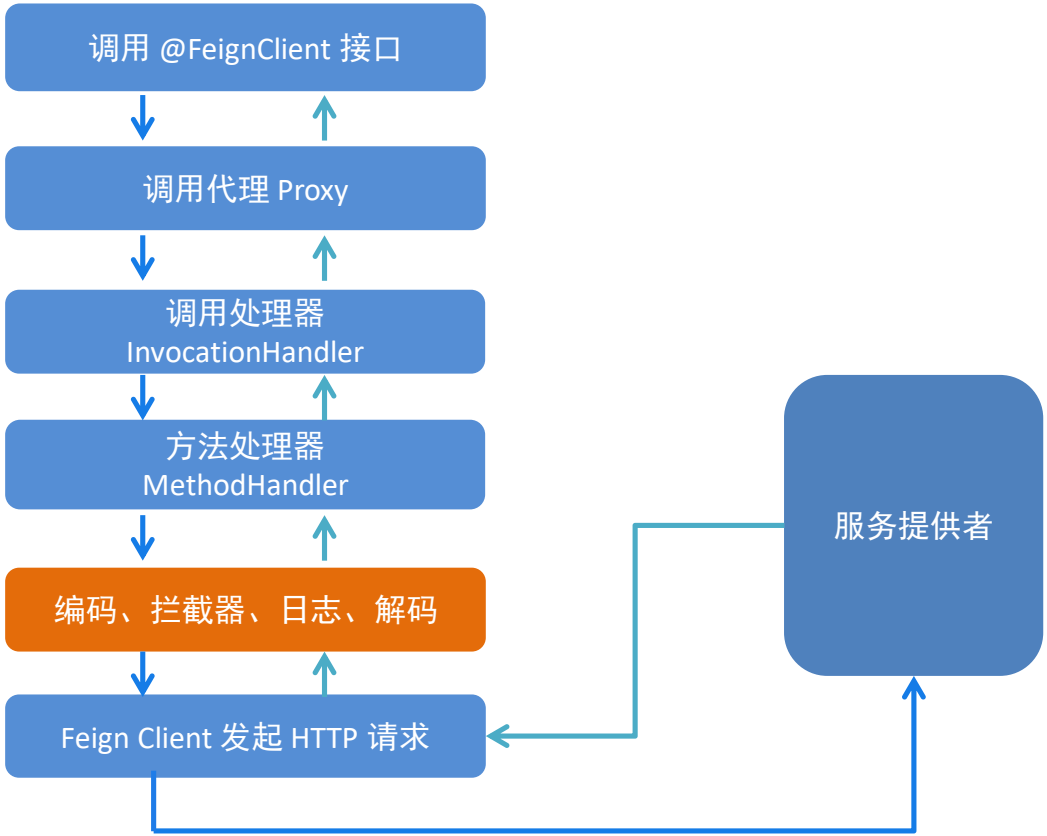
Feign 工作原理

Proxy 代理的工作



Feign 工作原理

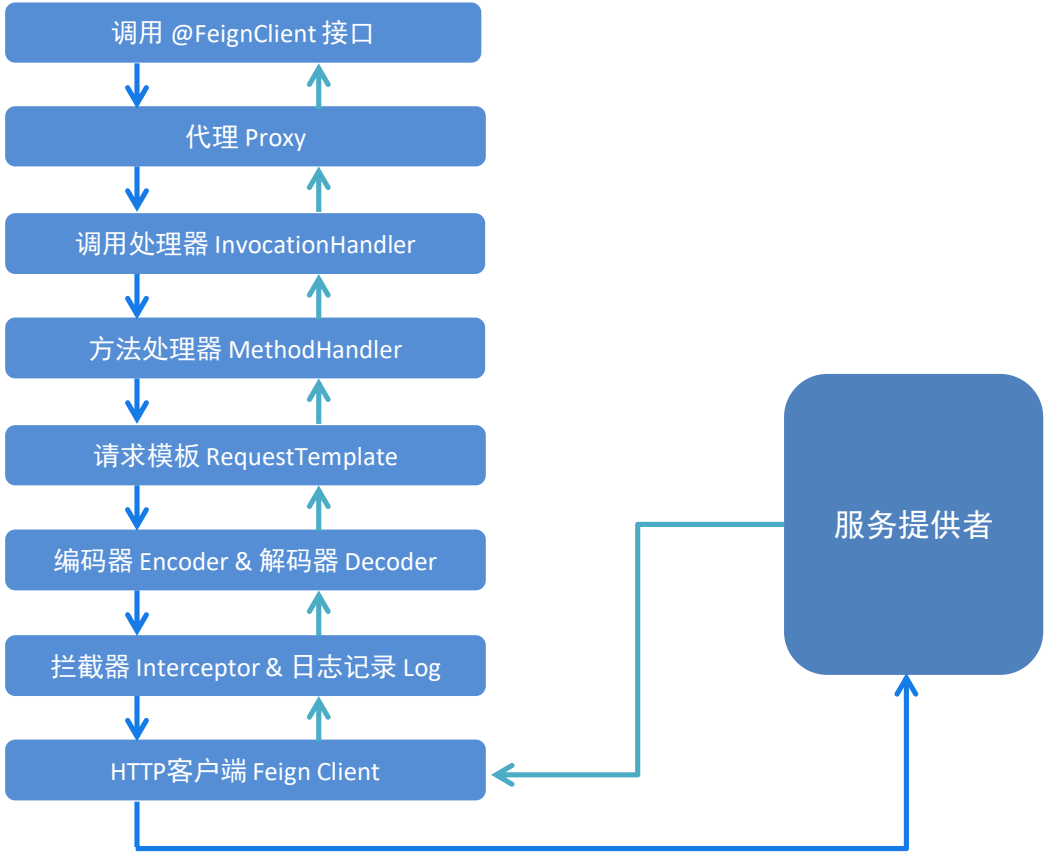
细节工作



Feign 工作原理

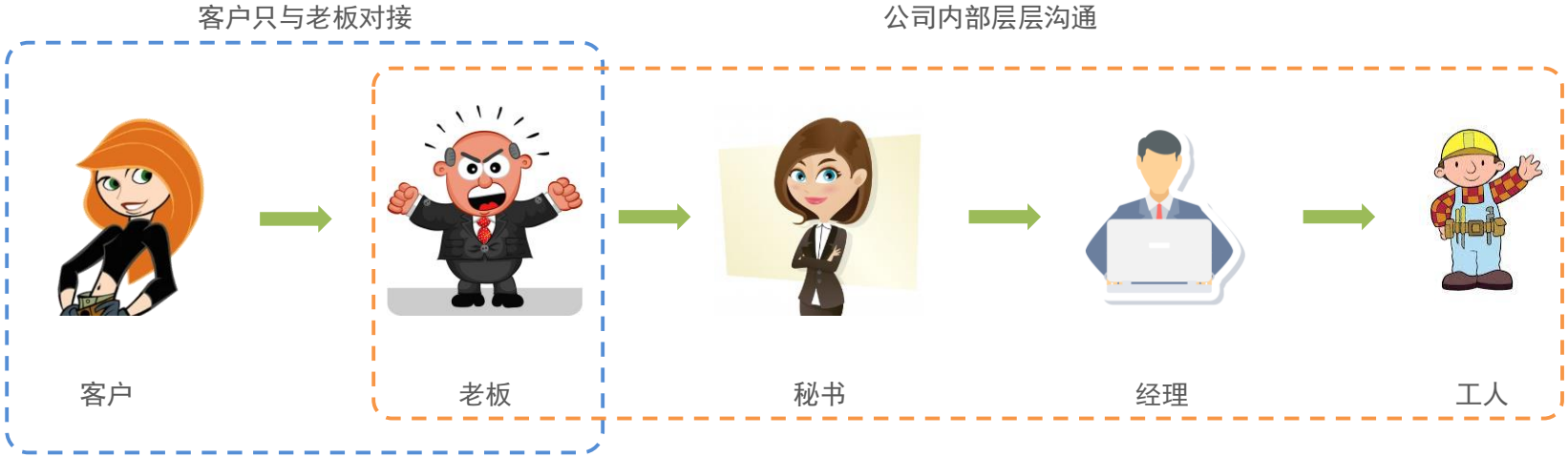
博学谷 www.boxuegu.com

最终流程



Feign 工作原理

最终流程



调用者

FeignClient

Proxy 代理

调用处理器
InvocationHandler

方法处理器
MethodHandler

Feign 工作原理-总结

重难点

1. Feign 的整体工作流程
2. Proxy 代理所做的主要工作
3. Proxy 工作细节
4. Feign 全局工作流程图

目录 Contents

Feign 日志配置

小节导学

如果 Feign 远程调用的时候出现了问题，我们应该怎么定位问题呢？如果能看到 HTTP 调用的细节信息就会方便很多。Feign 在构建服务客户端时，会为每个客户端都创建一个 feign.logger 实例，我们可以使用这个日志对象的调试模式输出请求细节。

这一节我们就学习一下如何配置 Feign 日志

- Feign 日志级别
- 局部日志配置
- 全局日志配置

```
[HiService#sayhi] ---> GET http://localhost:8002/user/hello?name=a HTTP/1.1
[HiService#sayhi] ---> END HTTP (0-byte body)
[HiService#sayhi] <--- HTTP/1.1 200 (159ms)
[HiService#sayhi] content-length: 8
[HiService#sayhi] content-type: text/plain;charset=UTF-8
[HiService#sayhi] date: Tue, 03 Mar 2020 03:17:35 GMT
[HiService#sayhi]
[HiService#sayhi] hello a!
[HiService#sayhi] <--- END HTTP (8-byte body)
```

1. Feign 日志级别

级别	描述
NONE	默认值，不输出日志
BASIC	只输出基本的日志信息，包括请求方法、URL地址、响应状态码、执行时间
HEADERS	在 BASIC 基础上，输出请求和响应的 Header 信息
FULL	输出最全的日志信息，包括请求响应的 Header、Body、元数据信息

2.局部日志配置

代码方式配置日志

(1) 配置文件中设置 Feigin client 接口的日志级别

```
logging:
  level:
    com.example.serviceconsumerfeign.feiginclient: debug
```

设置 Feign 接口 package 日志级别为 debug

(3) Feiginclient接口中引用

```
@FeignClient(name="service-provider",
    configuration = FeignClientDemoConfig.class)
public interface FeignClientDemo {
    ...
}
```

注解中指定配置类

(2) 创建日志级别配置类

```
public class FeignClientDemoConfig {
    @Bean
    public Logger.Level level(){
        return Logger.Level.FULL;
    }
}
```

设置日志级别

2.局部日志配置

属性方式配置日志

(1) 配置文件中设置 Feigin client 接口的日志级别

```
logging:
  level:
    com.example.serviceconsumerfeign.feiginclient: debug
```

(2) 设置目标调用服务的日志级别

```
feign:
  client:
    config:
      service-provider: # 想要调用的服务名称
        loggerLevel: FULL
```

3. 全局方式配置日志

局部的日志配置是针对某个 Feign 接口的，如果需要配置的接口比较多，比较适合使用全局配置。

全局配置同样可以通过代码的方式或者属性方式来配置。

（1）代码方式

```
@SpringBootApplication
// 把日志配置类定义到启动类中
@EnableFeignClients(defaultConfiguration
    = FeignClientDemoConfig.class)
public class ServiceConsumerFeignApplication {
    public static void main(String[] args) {
        SpringApplication.run(
        ServiceConsumerFeignApplication.class,
        args);
    }
}
```

(2) 属性方式

```
feign:
  client:
    config:
      default: # 全局配置
        loggerLevel: FULL
```

Feign 日志配置-总结

重难点

1. Feigin 日志级别
2. 局部日志配置（代码方式、属性方式）
3. 全局日志配置（代码方式、属性方式）

目录 Contents

多参数传递

小节导学

假设服务提供者有一个获取用户的接口地址如下：

http://service-provider/user/get?**username**=zhaosi&**city**=bj

看似很简单，但在 Feign 中是需要特别注意的，称为“**多参数传递问题**”，本节我们就具体看一下这到底是个什么问题，以及如何解决。

- 多参数传递问题描述
- 解决方法1：参数注解
- 解决方法2：独立参数

1. 多参数传递问题描述

什么是多参数传递问题?

直接描述不好理解，下面开发一个示例程序，通过报错信息就清晰明了了。

Feign 接口中的参数定义与服务提供者中接口定义一致，但实际会报错：

status 405 reading FeignClientUser#getUser(User) .....

2. 解决方法1：参数注解

SpringMVC 的 get 方法支持直接绑定 POJO，而 Feign 并未覆盖所有 SpringMVC 功能，不能直接绑定 POJO，但解决起来也很简单，只需要添加一个注解 @SpringQueryMap。

```
@FeignClient(name="service-provider")
public interface FeignClientUser {
    @GetMapping("/getUser")
    String getUser(@SpringQueryMap User user);
}
```

3. 解决方法2：独立参数

这是最为直观的方式，URL中有几个参数，Feign 接口中的方法就定义几个参数。使用 @RequestParam 注解指定请求的参数。

```
@FeignClient(name="service-provider")
public interface FeignClientUser {

    @GetMapping("/getUser")
    String getUser_param(@RequestParam("id") Long id,
            @RequestParam("name") String name,
            @RequestParam("age") int age);
}
```

多参数传递-总结

重难点

1. 理解 Feign 多参数传递问题
2. 通过 @SpringQueryMap 注解解决
3. 通过独立参数传递来解决

目录 Contents

复杂参数形式与文件上传

小节导学

假设服务提供者有一个接口形式如下：

```
public String demo_complex(@PathVariable("id") String id,
                           @RequestBody Map<String, Object> map,
                           @RequestParam String name) ;
```

比较复杂，FeignClient 接口中如何定义呢？

还有 FeignClient 如何对接服务提供者的上传接口呢？本节我们就一起解决这些问题。

- Feign 处理复杂参数形式
- Feign 文件上传

1. Feign 处理复杂参数形式

服务提供者

实际开发中，我们可能会通过多种方式进行参数传递，例如下面的形式。

```
@RestController
public class DemoController {
    @RequestMapping(path = "/demo_complex/{id}",
            method = RequestMethod.POST)
    public String demo_complex(@PathVariable("id") String id,
                           @RequestBody Map<String, Object> map,
                           @RequestParam String name) {
        Object json = JSON.toJSON(map);
        return "PathVariable id :" + id + "\nRequestParam name : " + name + "\nRequestBody
map: " + json.toString();
    }
}
```

1. Feign 处理复杂参数形式

服务消费者

```
@FeignClient(name="service-provider")
public interface FeignClientDemo {
      @RequestMapping(
              path = "/demo_complex/{id}",
              method = RequestMethod.POST
      )
      public String demo_complex(@PathVariable("id") String id,
                                 @RequestBody Map<String, Object> map,
                                 @RequestParam String name) ;
}
```

2. Feign 文件上传

服务提供者

文件上传接口：

```
@RestController
public class UploadController {
    @PostMapping(value = "/uploadFile", consumes = MediaType.MULTIPART_FORM_DATA_VALUE)
    public String fileUploadServer(MultipartFile file ) throws Exception{
        return file.getOriginalFilename();
    }
}
```

2. Feign 文件上传

服务消费者

早先 Feign 本身是没有上传文件的能力的，要想实现这一点，需要自己去编写 Encoder 去实现上传。

现在我们幸福了很多，因为Feign官方提供了子项目 feign-form，其中实现了上传所需的 Encoder 。

```
<dependency>
    <groupId>io.github.openfeign.form</groupId>
    <artifactId>feign-form</artifactId>
    <version>3.8.0</version>
</dependency>
<dependency>
    <groupId>io.github.openfeign.form</groupId>
    <artifactId>feign-form-spring</artifactId>
    <version>3.8.0</version>
</dependency>
```

2. Feign 文件上传

服务消费者

FeignClient 中定义：

```
@RequestMapping(value = "/uploadFile", method = RequestMethod.POST,
            produces = {MediaType.APPLICATION_JSON_UTF8_VALUE},
            consumes = MediaType.MULTIPART_FORM_DATA_VALUE)
String handleFileUpload(@RequestPart(value = "file") MultipartFile file);
```

Controller 中定义：

```
@PostMapping(value = "/upload")
public String imageUpload( MultipartFile file ) throws Exception{
    return feignClientDemo.handleFileUpload(file);
}
```

复杂参数形式与文件上传 -总结

重难点

1. @PathVariable、@RequestBody、@RequestParam

   多种参数形式混合情况下 Feign 的处理方式

2. 通过 feign-form 实现文件上传

目录 Contents

Feign Client 改用 HTTPClient 与 OKHTTP

小节导学

Feign 的 HTTP 客户端支持 3 种框架：

- HttpURLConnection（默认）
- HttpClient
- OKhttp

传统的 HttpURLConnection 是 JDK 自带的，并不支持连接池，效率非常低。

为了提高效率，可以通过连接池提高效率，appache httpclient 和 okhttp 都是支持链接池的。

本节我们学习如何把 FeignClient 替换为 HttpClient 和 OKhttp。

- 替换 HttpClient
- 替换 OKhttp

1. 替换 HttpClient

步骤：

（1）添加依赖

```
<dependency>
    <groupId>io.github.openfeign</groupId>
    <artifactId>feign-httpclient</artifactId>
</dependency>
```

（2）属性配置，开启 HttpClient 的支持。

```
feign:
  httpclient:
    enabled: true
    # 最大连接数
    max-connections: 200
    # 单个路由的最大连接数
    max-connections-per-route: 50
```

（3）测试验证

开启日志，验证是否输出 HttpClient 相关日志

```
logging:
  level:
    org.apache.http.wire: debug
    org.apache.http.headers: debug
```

2. 替换 OKHttp

步骤：

（1）添加依赖

```
<dependency>
    <groupId>io.github.openfeign</groupId>
    <artifactId>feign-okhttp</artifactId>
    <version>10.2.0</version>
</dependency>
<dependency>
    <groupId>com.parkingwang</groupId>
    <artifactId>okhttp3-
loginterceptor</artifactId>
    <version>0.5</version>
</dependency>
```

（2）属性配置，开启 OKHttp 的支持。

```
feign:
  client:
    config:
      default: # 全局配置
        loggerLevel: FULL
  httpclient:
    enabled: false
    max-connections: 200
    max-connections-per-route: 50
  okhttp:
    enabled: true
```

2. 替换 OKHttp

博学谷 www.boxuegu.com

步骤：

（3）OKHttp 配置类，配置连接池，以及日志拦截器

```
@Configuration
@ConditionalOnClass(Feign.class)
@AutoConfigureBefore(FeignAutoConfiguration.class)
public class FeignClientOkHttpConfiguration {
    @Bean
    public OkHttpClient okHttpClient() {
        return new OkHttpClient.Builder()
                .connectTimeout(20, TimeUnit.SECONDS).readTimeout(20, TimeUnit.SECONDS)
                .writeTimeout(20, TimeUnit.SECONDS).retryOnConnectionFailure(true)
                .connectionPool(new ConnectionPool()).addInterceptor(new LogInterceptor())
                .build();
    }
    @Bean
    public ConnectionPool pool() {return new ConnectionPool(50, 5, TimeUnit.MINUTES);}
}
```

Feign Client 改用 HTTPClient 与 OKHTTP-总结

重难点

1. 替换 HTTPClient 的方法，以及验证方式
2. 替换 OKHTTP 的方法，以及验证方式

一样的在线教育，不一样的教学品质