链滴

微服务——Feign

作者：strangebob

原文链接：https://ld246.com/article/1681970696539

来源网站：链滴

许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

Feign远程调用

1.简介

学完Nacos，来学Feign远程调用了

以前利用RestTemplate发起远程调用的代码，存在下面的问题：

• 代码可读性差，编程体验不统一

• 参数复杂URL难以维护

Feign是一个声明式的http客户端，官方地址：https://github.com/OpenFeign/feign

其作用就是帮助我们优雅的实现http请求的发送，解决上面提到的问题。

2.Feign替代RestTemplate

Fegin的使用步骤如下：

1）引入依赖

我们在order-service服务的pom文件中引入feign的依赖：

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

2）添加注解

在order-service的启动类添加注解开启Feign的功能：

```
@EnableFeignClients
@MapperScan("cn.itcast.order.mapper")
@SpringBootApplication
public class OrderApplication {

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

3）编写Feign的客户端

在order-service中新建一个接口，内容如下：

```
package cn.itcast.order.client;

import cn.itcast.order.pojo.User;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;

@FeignClient("userservice")
public interface UserClient {
    @GetMapping("/user/{id}")
    User findById(@PathVariable("id") Long id);
}
```

这个客户端主要是基于SpringMVC的注解来声明远程调用的信息，比如：

- 服务名称：userservice
- 请求方式：GET
- 请求路径：/user/{id}
- 请求参数：Long id
- 返回值类型：User

这样，Feign就可以帮助我们发送http请求，无需自己使用RestTemplate来发送了。

4）测试

修改order-service中的OrderService类中的queryOrderById方法，使用Feign客户端代替RestTemplte：

```
@Autowired
private UserClient userClient;

public Order queryOrderById(Long orderId) {
    // 1.查询订单
    Order order = orderMapper.findById(orderId);
    // 2.利用Feign发起http请求，查询用户
    User user = userClient.findById(order.getUserId());
    // 3.封装user到Order
    order.setUser(user);
    // 4.返回
    return order;
}
```

是不是看起来优雅多了。

5）总结

使用Feign的步骤：

① 引入依赖

② 添加@EnableFeignClients注解

③ 编写FeignClient接口

④ 使用FeignClient中定义的方法代替RestTemplate

3.自定义配置

Feign可以支持很多的自定义配置，如下表所示：

类型	作用	说明
feign.Logger.Level	修改日志级别	含四种不同的级别：NONE、BASIC、HEADERS、FULL
feign.codec.Decoder	响应结果的解析器	ttp远程调用的结果做解析，例如解析json字符串为java对象
feign.codec.Encoder	请求参数编码	请求参数编码，便于通过http请求发送
feign. Contract	支持的注解格式	认是SpringMVC的注解
feign. Retryer	失败重试机制	请失败的重试机制，默认是没有，不过会使用Ribbon的重试

一般情况下，默认值就能满足我们使用，如果要自定义时，只需要创建自定义的@Bean覆盖默认Bea即可。

下面以日志为例来演示如何自定义配置。

3.1.配置文件方式

基于配置文件修改feign的日志级别可以针对单个服务：

```
feign:
  client:
    config:
      userservice: # 针对某个微服务的配置
        loggerLevel: FULL #  日志级别
```

也可以针对所有服务：

```
feign:
  client:
    config:
      default: # 这里用default就是全局配置，如果是写服务名称，则是针对某个微服务的配置
        loggerLevel: FULL #  日志级别
```

而日志的级别分为四种：

- NONE：不记录任何日志信息，这是默认值。

- BASIC：仅记录请求的方法，URL以及响应状态码和执行时间
- HEADERS：在BASIC的基础上，额外记录了请求和响应的头信息
- FULL：记录所有请求和响应的明细，包括头信息、请求体、元数据。

3.2.Java代码方式

也可以基于Java代码来修改日志级别，先声明一个类，然后声明一个Logger.Level的对象：

```
public class DefaultFeignConfiguration  {
    @Bean
    public Logger.Level feignLogLevel(){
        return Logger.Level.BASIC; // 日志级别为BASIC
    }
}
```

如果要**全局生效**，将其放到启动类的@EnableFeignClients这个注解中：

```
@EnableFeignClients(defaultConfiguration = DefaultFeignConfiguration .class)
```

如果是**局部生效**，则把它放到对应的@FeignClient这个注解中：

```
@FeignClient(value = "userservice", configuration = DefaultFeignConfiguration .class)
```

4.Feign使用优化

Feign底层发起http请求，依赖于其它的框架。其底层客户端实现包括：

• URLConnection：默认实现，不支持连接池

• Apache HttpClient ： 支持连接池

• OKHttp：支持连接池

因此提高Feign的性能主要手段就是使用**连接池**代替默认的URLConnection。

这里我们用Apache的HttpClient来演示。

1）引入依赖

在order-service的pom文件中引入Apache的HttpClient依赖：

```
<!--httpClient的依赖 -->
<dependency>
    <groupId>io.github.openfeign</groupId>
    <artifactId>feign-httpclient</artifactId>
</dependency>
```

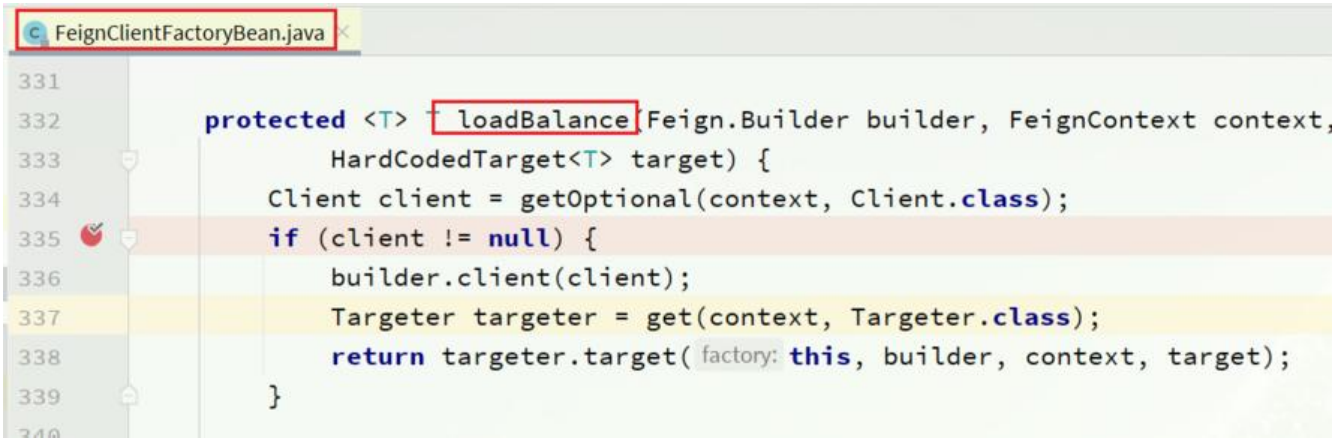
2）配置连接池

在order-service的application.yml中添加配置：

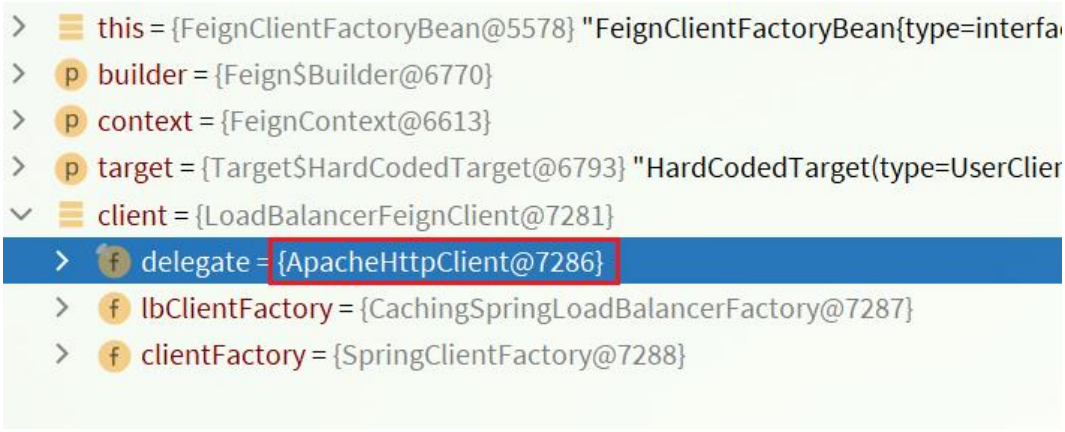
```
feign:
  client:
```

```
  config:
    default: # default全局的配置
      loggerLevel: BASIC # 日志级别，BASIC就是基本的请求和响应信息
httpclient:
  enabled: true # 开启feign对HttpClient的支持
  max-connections: 200 # 最大的连接数
  max-connections-per-route: 50 # 每个路径的最大连接数
```

接下来，在FeignClientFactoryBean中的loadBalance方法中打断点：



Debug方式启动order-service服务，可以看到这里的client，底层就是Apache HttpClient：



总结，Feign的优化：

1.日志级别尽量用basic

2.使用HttpClient或OKHttp代替URLConnection

① 引入feign-httpClient依赖

② 配置文件开启httpClient功能，设置连接池参数

5.最佳实践

所谓最近实践，就是使用过程中总结的经验，最好的一种使用方式。

自习观察可以发现，Feign的客户端与服务提供者的controller代码非常相似：

feign客户端：

```java
@FeignClient("userservice")
public interface UserClient {
    @GetMapping("/user/{id}")
    User findById(@PathVariable("id") Long id);
}
```

```java
/**
 * 路径: /user/110
 *
 * @param id 用户id
 * @return 用户
 */
@GetMapping("/user/{id}")
public User queryById(@PathVariable("id") Long id) {
    return userService.queryById(id);
}
```

UserController：

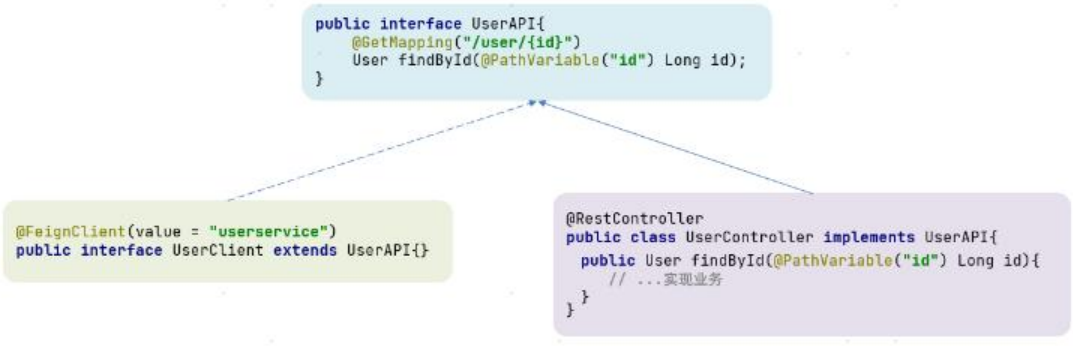
有没有一种办法简化这种重复的代码编写呢？

5.1继承方式

一样的代码可以通过继承来共享：

1）定义一个API接口，利用定义方法，并基于SpringMVC注解做声明。

2）Feign客户端和Controller都集成改接口



优点：

- 简单
- 实现了代码共享

缺点：

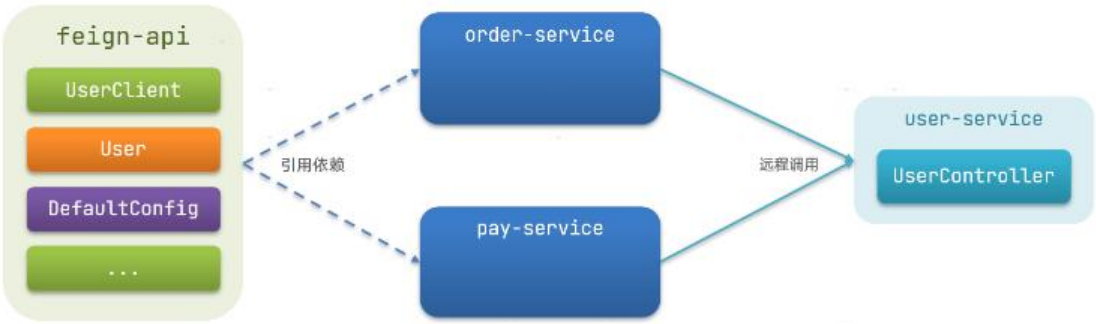
- 服务提供方、服务消费方紧耦合

- 参数列表中的注解映射并不会继承，因此Controller中必须再次声明方法、参数列表、注解

5.2抽取方式

将Feign的Client抽取为独立模块，并且把接口有关的POJO、默认的Feign配置都放到这个模块中，供给所有消费者使用。

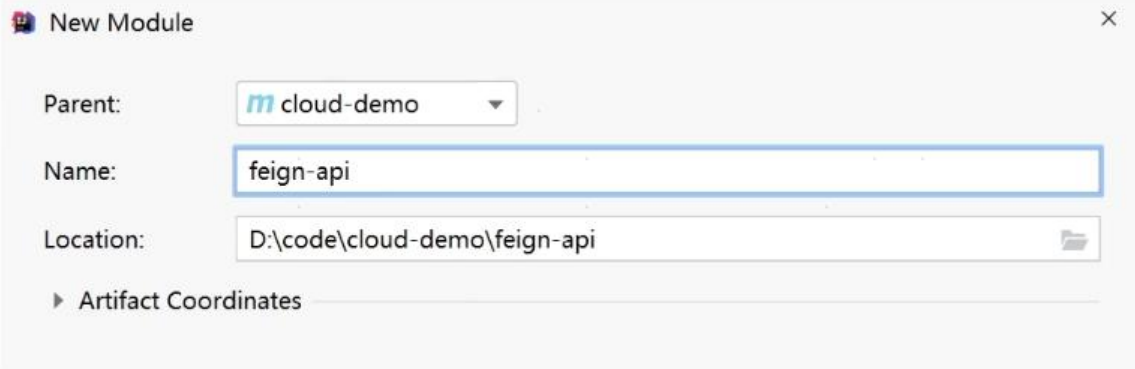
例如，将UserClient、User、Feign的默认配置都抽取到一个feign-api包中，所有微服务引用该依赖，即可直接使用。



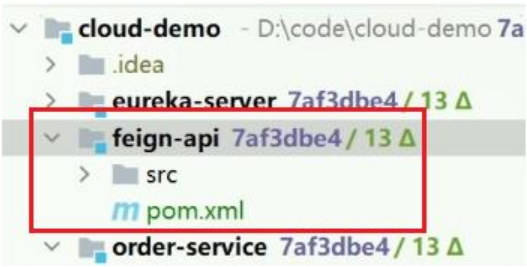
5.3实现基于抽取的最佳实践

1）抽取

首先创建一个module，命名为feign-api：



项目结构：

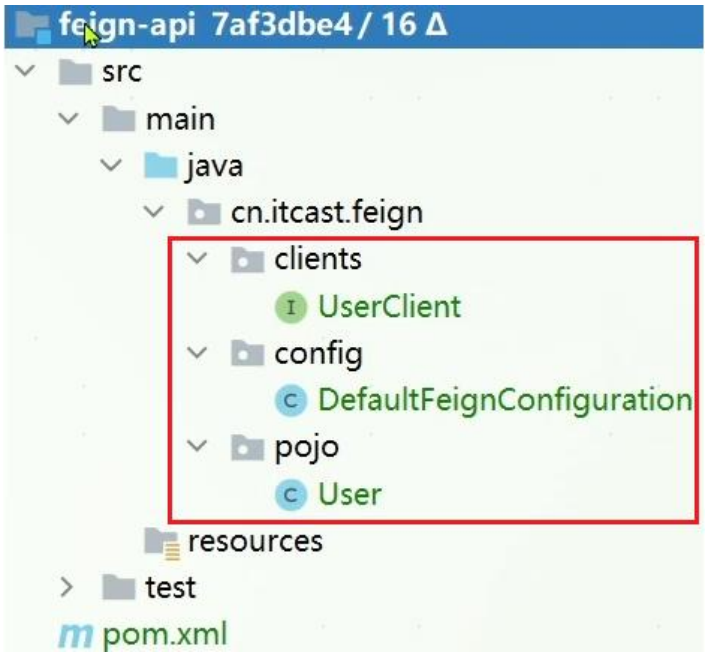


在feign-api中然后引入feign的starter依赖

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

```
</dependency>
```

然后，order-service中编写的UserClient、User、DefaultFeignConfiguration都复制到feign-ap项目中



2）在order-service中使用feign-api

首先，删除order-service中的UserClient、User、DefaultFeignConfiguration等类或接口。

在order-service的pom文件中中引入feign-api的依赖：

```
<dependency>
    <groupId>cn.itcast.demo</groupId>
    <artifactId>feign-api</artifactId>
    <version>1.0</version>
</dependency>
```

修改order-service中的所有与上述三个组件有关的导包部分，改成导入feign-api中的包

3）重启测试

重启后，发现服务报错了：

```
Field userClient in cn.itcast.order.service.OrderService required a bean of type
'cn.itcast.feign.clients.UserClient' that could not be found.

The injection point has the following annotations:
    - @org.springframework.beans.factory.annotation.Autowired(required=true)
```

这是因为UserClient现在在cn.itcast.feign.clients包下，

而order-service的 @EnableFeignClients注解是在cn.itcast.order包下，不在同一个包，无法扫描到serClient。

4）解决扫描包问题

方式一：

指定Feign应该扫描的包：

```
@EnableFeignClients(basePackages = "cn.itcast.feign.clients")
```

方式二：

指定需要加载的Client接口：

```
@EnableFeignClients(clients = {UserClient.class})
```