



下载APP



14 | OpenFeign 实战：OpenFeign 组件有哪些高级玩法？

2022-01-12 姚秋辰

《Spring Cloud 微服务项目实战》

课程介绍 >



讲述：姚秋辰

时长 17:57 大小 16.45M



你好，我是姚秋辰。

在上一讲中，我们已经将 OpenFeign 组件集成到了实战项目中。今天我们来进一步深入 OpenFeign 的功能特性，学习几个 OpenFeign 的进阶使用技巧：异常信息排查、超时判定和服务降级。

异常信息排查是我们开发人员每天都要面对的事情。如果你正在开发一个大型微服务应用，你经常需要集成一些由其他团队开发的 API，这就免不了要参与各种联调和问题排查。如果你是一个经验丰富的老码农，那你一定经常说这样一句话：“你的 Request 是什么？”这句台词在我们平时的 API 联调和线上异常排查中出镜率很高，因为**服务请求的入参和出参是分析和排查问题的重要线索。**



为了获得服务请求的参数和返回值，我们经常使用的一个做法就是**打印日志**。你可以在程序中使用 `log.info` 或者 `log.debug` 方法将服务请求的入参和返回值一一打印出来。但是，对一些复杂的业务场景来说就没有那么轻松了。

假如你在开发的是一个下单服务，执行一次下单流程前前后后要调用十多个微服务。你需要在请求发送的前后分别打印 Request 和 Response，不仅麻烦不说，我们还未必能把包括 Header 在内的完整请求信息打印出来。

那我们如何才能引入一个既简单又不需要硬编码的日志打印功能，让它自动打印所有远程方法的 Request 和 Response，方便我们做异常信息排查呢？接下来，我就来给你介绍一个 OpenFeign 的小功能，轻松实现**远程调用参数的日志打印**。

日志信息打印

为了让 OpenFeign 可以主动将请求参数打印到日志中，我们需要做两个代码层面的改动。

首先，你需要在配置文件中**指定 FeignClient 接口的日志级别为 Debug**。这样做是因为 OpenFeign 组件默认将日志信息以 debug 模式输出，而默认情况下 Spring Boot 的日志级别是 Info，因此我们必须将应用日志的打印级别改为 debug 后才能看到 OpenFeign 的日志。


我们打开 coupon-customer-impl 模块的 application.yml 配置文件，在其中加上以下几行 logging 配置项。

```
1 logging:
2   level:
3     com.geekbang.coupon.customer.feign.TemplateService: debug
4     com.geekbang.coupon.customer.feign.CalculationService: debug
```

[复制代码](#)

在上面的配置项中，我指定了 TemplateService 和 CalculationService 的日志级别为 debug，而其它类的日志级别不变，仍然是默认的 Info 级别。

接下来，你还需要在应用的上下文中使用代码的方式**声明 Feign 组件的日志级别**。这里的日志级别并不是我们传统意义上的 Log Level，它是 OpenFeign 组件自定义的一种日志级别，用来控制 OpenFeign 组件向日志中写入什么内容。你可以打开 coupon-customer-impl 模块的 Configuration 配置类，在其中添加这样一段代码。

 复制代码

```
1 @Bean
2 Logger.Level feignLogger() {
3     return Logger.Level.FULL;
4 }
```

在上面这段代码中，我指定了 OpenFeign 的日志级别为 Full，在这个级别下所输出的日志文件将会包含最详细的服务调用信息。OpenFeign 总共有四种不同的日志级别，我来带你了解一下这四种级别下 OpenFeign 向日志中写入的内容。


NONE：不记录任何信息，这是 OpenFeign 默认的日志级别；

BASIC：只记录服务请求的 URL、HTTP Method、响应状态码（如 200、404 等）和服务调用的执行时间；

HEADERS：在 BASIC 的基础上，还记录了请求和响应中的 HTTP Headers；

FULL：在 HEADERS 级别的基础上，还记录了服务请求和服务响应中的 Body 和 metadata，FULL 级别记录了最完整的调用信息。

我们将 Feign 的日志级别指定为 Full，并启动项目发起一个远程调用，你就可以在日志中看到整个调用请求的信息，包括请求路径、Header 参数、Request Payload 和 Response Body。我拿了一个调用日志作为示例，你可以参考一下。

 复制代码

```
1 ---> POST http://coupon-calculation-serv/calculator/simulate HTTP/1.1
2 Content-Length: 458
3 Content-Type: application/json
4
5 {"products":[{"productId":null,"price":3000, xxxx省略请求参数
6 ---> END HTTP (458-byte body)
7 <--- HTTP/1.1 200 (29ms)
8 connection: keep-alive
9 content-type: application/json
10 date: Sat, 27 Nov 2021 15:11:26 GMT
```

```
11 keep-alive: timeout=60
12 transfer-encoding: chunked
13
14 {"bestCouponId":26,"couponToOrderPrice":{"26":15000}}
15 <--- FND HTTP (52-hvta body)
```

有了这些详细的日志信息，你在开发联调阶段排查异常问题就易如反掌了。

到这里，我们就详细了解了 OpenFeign 的日志级别设置。接下来，我带你了解如何在 OpenFeign 中配置超时判定条件。

OpenFeign 超时判定

超时判定是一种保障可用性的手段。如果你要调用的目标服务的 RT (Response Time) 值非常高，那么你的调用请求也会处于一个长时间挂起的状态，这是造成服务雪崩的一个重要因素。为了隔离下游接口调用超时所带来的影响，我们可以在程序中设置一个**超时的阈值**，一旦下游接口的响应时间超过了这个阈值，那么程序会自动取消此次调用并返回一个异常。

我们以 coupon-customer-serv 为例，customer 服务依赖 template 服务来读取优惠券模板的信息，如果你想要对 template 的远程服务调用添加超时判定配置，那么我们可以在 coupon-customer-impl 模块下的 application.yml 文件中添加下面的配置项。

```
1 feign:
2   client:
3     config:
4       # 全局超时配置
5       default:
6         # 网络连接阶段1秒超时
7         connectTimeout: 1000
8         # 服务请求响应阶段5秒超时
9         readTimeout: 5000
10      # 针对某个特定服务的超时配置
11      coupon-template-serv:
12        connectTimeout: 1000
13        readTimeout: 2000
```

[复制代码](#)

从上面这段代码中可以看出，所有超时配置都放在 feign.client.config 路径之下，我在这个路径下面声明了两个节点：default 和 coupon-template-serv。


default 节点配置了全局层面的超时判定规则，它的生效范围是所有 OpenFeign 发起的远程调用。

coupon-template-serv 下面配置的超时规则只针对向 template 服务发起的远程调用。如果你想要对某个特定服务配置单独的超时判定规则，那么可以用同样的方法，在 feign.client.config 下添加目标服务名称和超时判定规则。

这里需要你注意的一点是，如果你同时配置了全局超时规则和针对某个特定服务的超时规则，那么后者的配置会覆盖全局配置，并且优先生效。

在超时判定的规则中我定义了两个属性：connectTimeout 和 readTimeout。其中，connectTimeout 的超时判定作用于“建立网络连接”的阶段；而 readTimeout 的超时判定则作用于“服务请求响应”的阶段（在网络连接建立之后）。我们常说的 RT（即服务响应时间）受后者影响比较大。另外，这两个属性对应的超时**时间单位都是毫秒**。

配置好超时规则之后，我们可以验证一下。你可以在 template 服务中使用 Thread.sleep 方法强行让线程挂起几秒钟，制造一个超时场景。这时如果你通过 customer 服务调用了 template 服务，那么在日志中可以看到下面的报错信息，提示你服务请求超时。

 复制代码

```
1 [TemplateService#getTemplate] <--- ERROR SocketTimeoutException: Read timed ou
2 [TemplateService#getTemplate] java.net.SocketTimeoutException: Read timed out
```

到这里，相信你已经清楚如何通过 OpenFeign 的配置项来设置超时判定规则了。接下来，我带你了解一下 OpenFeign 是如何通过降级来处理服务异常的。


OpenFeign 降级

降级逻辑是在远程服务调用发生超时或者异常（比如 400、500 Error Code）的时候，自动执行的一段业务逻辑。你可以根据具体的业务需要编写降级逻辑，比如执行一段兜底逻辑将服务请求从失败状态中恢复，或者发送一个失败通知到相关团队提醒它们来线上排查问题。

在后面课程中，我将会使用 Spring Cloud Alibaba 的组件 Sentinel 跟你讲解如何搭建中心化的服务容错控制逻辑，这是一种重量级的服务容错手段。

但在这节课中，我采用了一种完全不同的服务容错手段，那就是借助 OpenFeign 实现 Client 端的服务降级。尽管它的功能远不如 Sentinel 强大，但它相比于 Sentinel 而言**更加轻量级且容易实现**，足以满足一些简单的服务降级业务需求。


OpenFeign 对服务降级的支持是借助 Hystrix 组件实现的，由于 Hystrix 已经从 Spring Cloud 组件库中被移除，所以我们需要在 coupon-customer-impl 子模块的 pom 文件中手动添加 hystrix 项目的依赖。

 复制代码

```
1 <!-- hystrix组件，专门用来演示OpenFeign降级 -->
2 <dependency>
3     <groupId>org.springframework.cloud</groupId>
4     <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
5     <version>2.2.10.RELEASE</version>
6     <exclusions>
7         <!-- 移除Ribbon负载均衡器，避免冲突 -->
8         <exclusion>
9             <groupId>org.springframework.cloud</groupId>
10            <artifactId>spring-cloud-netflix-ribbon</artifactId>
11        </exclusion>
12    </exclusions>
13 </dependency>
```

添加好依赖项之后，我们就可以编写 OpenFeign 的降级类了。OpenFeign 支持两种不同的方式来指定降级逻辑，一种是定义 fallback 类，另一种是定义 fallback 工厂。

通过 fallback 类实现降级是最为简单的一种途径，如果你想要为 TemplateService 这个 FeignClient 接口指定一段降级流程，那么我们可以定义一个降级类并实现 TemplateService 接口。我写了一个 TemplateServiceFallback 类，你可以参考一下。

 复制代码

```
1 @Slf4j
2 @Component
3 public class TemplateServiceFallback implements TemplateService {
4
5     @Override
6     public CouponTemplateInfo getTemplate(Long id) {
7         log.info("fallback getTemplate");
8         return null;
9     }
10 }
```



```
11     @Override
12     public Map<Long, CouponTemplateInfo> getTemplateInBatch(Collection<Long> i
13         log.info("fallback getTemplateInBatch");
14         return null;
15     }
16 }
```

在上面的代码中，我们可以看出 TemplateServiceFallback 实现了 TemplateService 中的所有方法。

我们以其中的 getTemplate 方法为例，如果在实际的方法调用过程中，OpenFeign 接口的 getTemplate 远程调用发生了异常或者超时的情况，那么 OpenFeign 会主动执行对应的降级方法，也就是 TemplateServiceFallback 类中的 getTemplate 方法。

你可以根据具体的业务场景，编写合适的降级逻辑。

降级类定义好之后，你还需要在 TemplateService 接口中将 TemplateServiceFallback 类指定为降级类，这里你可以借助 FeignClient 接口的 fallback 属性来配置，你可以参考下面的代码。

[复制代码](#)

```
1 @FeignClient(value = "coupon-template-serv", path = "/template",
2     // 通过fallback指定降级逻辑
3     fallback = TemplateServiceFallback.class)
4 public interface TemplateService {
5     // ... 省略方法定义
6 }
```

如果你想要在降级方法中获取到**异常的具体原因**，那么你就要借助 **fallback 工厂**的方式来指定降级逻辑了。按照 OpenFeign 的规范，自定义的 fallback 工厂需要实现 FallbackFactory 接口，我写了一个 TemplateServiceFallbackFactory 类，你可以参考一下。

[复制代码](#)

```
1 @Slf4j
2 @Component
3 public class TemplateServiceFallbackFactory implements FallbackFactory<Templat
4
5     @Override
```

```
6     public TemplateService create(Throwable cause) {
7         // 使用这种方法你可以捕捉到具体的异常cause
8         return new TemplateService() {
9
10            @Override
11            public CouponTemplateInfo getTemplate(Long id) {
12                log.info("fallback factory method test");
13                return null;
14            }
15
16            @Override
17            public Map<Long, CouponTemplateInfo> getTemplateInBatch(Collection
18                log.info("fallback factory method test");
19                return Maps.newHashMap();
20            }
21        };
22    }
23 }
```

从上面的代码中，你可以看出，抽象工厂 create 方法的入参是一个 Throwable 对象。这样一来，我们在降级方法中就可以获取到原始请求的具体报错异常信息了。

当然了，你还需要将这个工厂类添加到 TemplateService 注解中，这个过程和指定 fallback 类的过程有一点不一样，你需要借助 FeignClient 注解的 fallbackFactory 属性来完成。你可以参考下面的代码。

[复制代码](#)

```
1 @FeignClient(value = "coupon-template-serv", path = "/template",
2             // 通过抽象工厂来定义降级逻辑
3             fallbackFactory = TemplateServiceFallbackFactory.class)
4 public interface TemplateService {
5     // ... 省略方法定义
6 }
```

到这里，我们就完成了 OpenFeign 进阶功能的学习。针对这里面的某些功能，我想从日志打印和超时判定这两个方面给你一些实践层面的建议。

在日志打印方面，OpenFeign 的日志信息是测试开发联调过程中的好帮手，但是在生产环境中你是用不上的，因为几乎所有公司的生产环境都不会使用 Debug 级别的日志，最多是 Info 级别。

在超时判定方面，有时候我们在线上会使用多维度的超时判定，比如 OpenFeign + 网关层超时判定 + Sentinel 等等判定。它们可以互相作为兜底方案，一旦某个环节突然发生故障，另一个可以顶上去。但这就形成了一个木桶理论，也就是几种判定规则中最严格的那个规则会优先生效。

总结

今天我们了解了 OpenFeign 的三个进阶小技巧。首先，你使用 OpenFeign 的日志模块打印了完整的远程服务调用信息，我们可以利用这个功能大幅提高线下联调测试的效率。然后，我带你了解了 OpenFeign 组件如何设置超时判定规则，通过全局配置 + 局部配置的方式对远程接口进行超时判定，这是一种有效的防止服务雪崩的可用性保障手段。最后，我们动手搭建了 OpenFeign 的降级业务，通过 fallback 类和 fallback 工厂两种方式实现了服务降级。

关于**服务降级的方案选型**，我想分享一些自己的见解。很多开发人员过于追求功能强大的新技术，但我们**做技术选型的时候也要考虑开发成本和维护成本**。

比如像 Sentinel 这类中心化的服务容错控制台，它的功能固然强大，各种花式玩法它都考虑到了。但相对应地，如果你要在项目中引入 Sentinel，在运维层面你要多维护一个 Sentinel 服务集群，并且在代码中接入 Sentinel 也是一个成本项。如果你只需要一些简单的降级功能，那 OpenFeign+Hystrix 的 Client 端降级方案就完全可以满足你的要求，我认为没必要拿大炮打苍蝇，过于追求一步到位的高大上方案。

到这里，我们 OpenFeign 组件的课程就结束了，下一节课程我将带你学习如何使用 Nacos 实现配置管理。

思考题

结合这节课的 OpenFeign 超时判定功能，你知道有哪些超时判定的算法吗？它们的底层原理是什么？欢迎在留言区写下自己的思考，与我一起讨论。

好啦，这节课就结束啦。欢迎你把这节课分享给更多对 Spring Cloud 感兴趣的朋友。我是姚秋辰，我们下节课再见！

分享给需要的人，Ta订阅本课程，你将得 20 元

生成海报并分享

赞 1 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 13 | OpenFeign 实战：如何实现服务间调用功能？

下一篇 15 | 配置中心在微服务中发挥着怎样的作用？

精选留言 (8)

写留言



gallifrey

2022-01-12

hystrix使用2.2.10.RELEASE的版本时，貌似需要在配置文件里面加上feign.circuitbreaker.enabled: true才行

作者回复: 是的，源码里带上了这行配置



1



alex_lai

2022-01-13

Openfeign client 不是non block的？如果我的框架基于reactive 风格写的是不是没有必要introduce openfeign了，我可以自己写wrap加future在client side。社区未来会提供支持么？openfeign的业界地位是什么样的, nice to have？

作者回复: feign本质是spring mvc模式的封装，如果项目需要大量使用non blocking功能建议用webflux之类的方案。也有民间热心群众开源的ReactiveFeign版本可以作为一个选择



1



逝影落枫

2022-01-12

是先有熔断，才有降级吗？熔断条件如何配置？

作者回复: 熔断器的设置会在sentinel相关章节讲到, 熔断是指在一段时间内, 服务调用直接走降级逻辑



被圣光照黑了

2022-01-12

我在coupon-customer-serv的启动类上加了@EnableHystrix, yml里加了feign:hystrix:enabled: true, coupon-template-serv里有个自定义异常, 调用报错了怎么不触发熔断啊

作者回复: 在专栏里使用的这个spring cloud版本中已经不推荐使用hystrix作为熔断器了, hystrix依赖项已经从SC项目中全面剔除了, 同学可以等后面介绍到sentinel的时候学习更强大的降级熔断组件。

这里介绍的feign降级是一个简化版的降级方案, 配置文件里添加feign.circuitbreaker.enabled=true试试



so long

2022-01-12

OpenFeign+spring-cloud-starter-alibaba-sentinel 的 Client 端降级方案也可以吧

作者回复: sentinel的降级对象可以是任何资源, 可以把openfeign接口服务作为一个“资源”托管给sentinel实现降级



小仙

2022-01-12

超时判定

```
res = future.get(timeout, TimeUnit.MILLISECONDS);
```

信号量

```
semaphore = new Semaphore(semaphoreValue);
```



kimoti

2022-01-12

好像是滑动窗口算法

作者回复: Bingo



peter
2022-01-12

请教老师3个问题：

Q1 容错时用Hystrix，是因为OpenFeign在基于Feign而Feign本来就能和Hystrix集成吗？除了搭配Hystrix，OpenFeign能搭配Resilience4j吗？

Q2 "06"篇中，思考题提到“3个模块分别部署到不同的集群上”，如果能分别部署，就不是单体应用了啊，而是像微服务了啊。单体应用就是难以分开部署，不是吗？...

展开 ∨

作者回复: Q1：没错，完全可以搭配Resilience4j，后面还会介绍一个中心化容错组件sentinel

Q2：06篇作为到微服务的过渡，我们设想这三个服务都变成了微服务之后，是如何发起调用的

Q3：后面会介绍Sentinel服务容错组件做限流+服务熔断，提供了一定的大盘监控能力。但这个不是专业的仪表监控系统，同学如果感兴趣的话可以了解下grafana，很多公司用这个监控k8s集群

Q4：CICD部分没有包含在专栏里面