Feign

> feign 的出现是为了简化restTamplate的使用，在使用restTamplate的过程中，我们需要设置url，封装请求参数，封装返回结果，能不能像调用rest接口一样实现远程调用 -> Feign简化服务调用
>
> **Ribbon 请查看Nacos**

1，什么是Feign

> Feign是模板化的HTTP客服端，不在使用，而是用OpenFeign
>
> OpenFeign对Feign进行了增强，使其支持SpringMVC，另外整合了Ribbon和Nacos，从而使得Feign的使用更加方便。
>
> `OpenFeign集成了Ribbon集成RestTamplate集成了负载均衡器，底层就是RestTamplate`
>
> **注意：** Feign是客服端的远程掉组件，声明在服务消费放即调用方

1.1 优势

> Feign可以做到使用HTTP请求远程服务时就像调用本地方法一样。

2，Spring Cloud Alibaba 整合Feign

2.1，使用

1. 添加依赖

> 如果添加了spring cloud的版本管理器就不需要指定版本了

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

2. 添加注解开启Feign功能

> @EnableFeignClients

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

3. 添加Rest请求接口

> Feign的请求接口一般声明与远程的controller接口一样，以便于区分
>
> @FeignClient属性：
>
> - name: 指定调用rest接口所对应的服务名
> - path：指定调用rest接口所在的服务controller层基地址即类上@RequestMapping("/path")地址

> - configuration: 这顶该feign接口的配置类
>
> **注意：** Feign只是一个HTTP的伪客服端，在使用接口进行服务调用时对SpringMVC的注解要求更加严格，注解及其注解的属性不能少如：
>
> - @PathVariable
> - @RequestBody
> - ...

```
@FeignClient(name = "stock-service", path = "/stock")
public interface StockFeignService {

    @GetMapping("/sub")
    String subStock();
}
```

4. 属性注入进行调用

```
@Autowired
private StockFeignService stockFeignService;

@GetMapping
public String order() {
    String s1 = stockFeignService.subStock();
    return "调用订单服务" + "\n" + s1;
}
```

2.2 Feign 自定义配置及使用

> Feign提供了很多扩展机制，让用户可以更加灵活的使用

2.2.1 日志配置

> spring boot 默认的日志级别时info，feign的默认日志级别时debug，在测试时要正对feign进行日志级别设置

设置日志级别

```
logging:
  level: # 如果在这里设置级别如debug是针对所有包和类的
    com.cloud.order.feign: debug # 包或类路径，针对某个进行日志级别设置
```

全局配置：配置类

> 全局配置：当使用@Configuration 会将配置作用所有的服务提供方
>
> 局部配置：如果只想针对一个或几个服务进行配置，就不要加@Configuration

```
/**
 * 全局配置：当使用@Configuration 会将配置作用所有的服务提供方
 * 局部配置：如果只想针对一个或几个服务进行配置，就不要加@Configuration
 */
@Configuration
public class FeignConfig {

    @Bean
    public Logger.Level feignLoggerLevel() {
        return Logger.Level.FULL;
    }
}
```

局部配置: 配置类针对某个服务

> 配置类：不加spring的@Configuration注解

```
public class FeignConfig {
    @Bean
    public Logger.Level feignLoggerLevel() {
        return Logger.Level.FULL;
    }
}
```

Feign 接口指定配置

```
@FeignClient(name = "stock-service", path = "/stock", configuration = {FeignConfig.class})
public interface StockFeignService {

    @GetMapping("/sub")
    String subStock();
}
```

2.2.2 超时时间配置

> 在使用RestTemplate时，可以通过RestTemplate设置超时时间，在使用Feign将如何设置超时时间呢?

```
@Bean
public RestTemplate restTemplate(RestTemplateBuilder builder) {
    return builder
        //.basicAuthentication("username", "password")
        .setConnectTimeout(Duration.ofSeconds(3))// 连接时间
        .setReadTimeout(Duration.ofSeconds(5)) // 超时时间
        // .rootUri("http://api.test.com/")
        .build();
}
```

> 但是Feign中没有直接设置RestTemplate（底层还是存在的），将如何设置呢

全局配置

> 通过Options 可以设置连接超时时间和读取超时时间，Options的第一个参数就是连接的超时时间，默认2s；第二个参数是处理的超时时间（单位ms），默认5s

```
@Bean
public Request.Options options() {
    // 该构造器已经过时，请使用重载的构造器
    return new Request.Options(5000, 1000);
}
```

局部配置：yaml配置

> 连接超时时间: 假设A 请求B ， A请求B网络连接的时间
>
> 请求处理超时时间：假设A 请求B，B处理请求进行响应的时间

```
feign:
  client:
    config:
      stock-servic: # 远程服务名
        # 连接超时时间，默认2s
        connectTimeout: 5000
        # 请求处理超时时间，默认5s
        readTimeout: 10000
```

2.3 Feign 自定义拦截器

> Feign可以自定义拦截器以实现自己的逻辑，注意不是SpringMVC的拦截器，类似于Axios的请求响应拦截器或者前端路由守卫
>
> 作用：
>
> - 加公共参数
> - 认证授权
> - ...

1. 实现拦截器接口

> 将其实现类起作用
>
> 1. 使用配置类方式，@Bean
> 2. 使用配置文件方式
> 3.

```
public class MyFeignInterceptor implements RequestInterceptor {
    @Override
    public void apply(RequestTemplate requestTemplate) {
        // 设置请求头
        requestTemplate.header("myheader", "hv");
        // 修改请求参数
        requestTemplate.query("qk", "111");
        // 更改uri
        requestTemplate.uri("/9");
    }
}
```

```
@Bean
public RequestInterceptor requestInterceptor() {
    return new MyFeignInterceptor();
}
```

application.yml

```
feign:
  client:
    config:
      stock-servic: # 远程服务名
        requestInterceptors[0]: # 是数组
          com.cloud.order.feign.MyFeignInterceptor
```