Spring Cloud 服务调用与熔断

核心概念

WebServices

XML 约束

- DTD
- XSD（Schema） - 强类型表达

JSON

Open API Specification

Spring Cloud Open Feign

Feign、Spring Cloud Open Feign、JAX-RS、Spring Web MVC 注解驱动特性

REST 框架	使用场景	请求映射注解	请求参数
Feign	客户端声明	@RequestLine	@Param
Spring Cloud Open Feign	客户端声明	@ReqeustMapping	@RequestParam
JAX-RS	客户端、服务端声明	@Path	@*Param
Spring Web MVC	服务端声明	@ReqeustMapping	@RequestParam

Spring Cloud Open Feign 利用 Feign 高扩展性，使用标准 Spring Web MVC 来声明客户端 Java 接口

- Feign
  - 注解扩展性
  - HTTP 请求处理
  - REST 请求元信息解析
- Spring Cloud Open Feign
  - 提供 Spring Web MVC 注解处理
  - 提供 Feign 自动装配

Spring Cloud Open Feign 是通过 Java 接口的方式来声明 REST 服务提供者的请求元信息，通过调用 Java 接口的方式来实现 HTTP/REST 通讯。

实现细节猜想

- Java 接口（以及方法）与 REST 提供者元信息如何映射
- `@FeignClient` 注解所指定的应用（服务）名称可能用到了服务发现，一个服务可以对应多个服务实例（HOST:PORT)
- `@EnableFeignClients` 注解是如何感知或加载标注 `@FeignClient` 的配置类（Bean）
- Feign 请求和响应的内容是如何序列化和反序列化对应的 POJO 的

Feign

扩展支持 (Contract 接口扩展)

- 内建 Feign 注解
- JAX-RS 1/2 注解
- JAXB
- OkHttp
- ...

> Contract 接口扩展：提供 Feign 接口方法与 REST 请求元信息契约，解析出相关的方法元信息

基本步骤

service-client -> service-provider(Spring Boot + Web MVC + 服务注册与发现)

1. 增加依赖

```
org.springframework.cloud:spring-cloud-starter-
openfeign
```

2.激活 Feign 客户端

```
@SpringBootApplication
@EnableFeignClients
public class Application {
    ...
}
```

3.定义 Feign 的接口

```
@FeignClient("stores") // "stores" 应用（服务）名称
public interface StoreClient {

    @RequestMapping(method = RequestMethod.POST, value
= "/stores/{storeId}", consumes = "application/json")
    Store update(@PathVariable("storeId") Long storeId,
Store store);
}
```

常见一些 Spring Boot/Cloud 中“坑”

@Value("server.port") 服务端口不一定靠谱，当 server.port = 0 时

`@LocalServerPort` 也不靠谱，因为在注入阶段 "local.server.port" 不一定存在

Spring Cloud + Netflix Ribbon 有一个 30 秒延迟

> DynamicServerListLoadBalancer
>
> 非 Eureka 实现 - com.netflix.loadbalancer.PollingServerListUpdater
>
> Eureka 实现 - com.netflix.niws.loadbalancer.EurekaNotificationServerListUpdater

Spring Cloud 服务调用

- 服务发现 - DiscoveryClient（Eureka、ZK、Consul、Nacos 等）
- 负载均衡 - Netflix Ribbon（唯一选择）
- Feign （唯一选择）

Spring Cloud Open Feign 实现细节

`@EnableFeignClients`

实现策略： Enable 模块驱动

具体实现：`org.springframework.cloud.openfeign.FeignClientsRegistrar`

主要工作：

- 注册默认配置
- 注册所有标注 `@FeignClient` 配置类

注册默认配置

源码位置：

org.springframework.cloud.openfeign.FeignClientsRegistrar#registerDefaultConfiguration

注册所有标注 `@FeignClient` 配置类

源码位置：

org.springframework.cloud.openfeign.FeignClientsRegistrar#registerFeignClients

通过 `ClassPathScanningCandidateComponentProvider` 扫描指定的 basePackages 集合中的类是否标注 `@FeignClient`，如果有的话，作为 `AnnotatedBeanDefinition` 返回，其中包含 `@FeignClient` 属性元数据，来自于 `AnnotationMetadata`，再重新注册 `FeignClientFactoryBean` 的 `BeanDefinition`

`@EnableFeignClient` -> `@Feignclient` 元信息 -> 标注接口定义的 `FeignClientFactoryBean` -> 形成被标注接口的代理对象

Spring Cloud Open Feign 大致流程：

1. Spring Web MVC 注解元信息解析

    - Contract
        - Feign 内建实现
            - `@RequestLine`
            - `@Headers`
        - JAX-RS 1/2 注解
            - @Path
            - @PathParam
            - @HeaderParam
        - Spring Web MVC (Spring Cloud Open Feign 扩展)
2. 通过 @FeignClient 所生成代理对象的方法调用实现 HTTP 调用
3. 通过 SpringDecoder 实现 Response 与接口返回类型的反序列化
4. 负载均衡
    - Spring Cloud 替换了 Client 实现 - `LoadBalancerFeignClient`
5. 重试重试
    - Spring Cloud 在外部包装 Feign 接口
6. 熔断
    - feign.hystrix.HystrixFeign

Spring Boot Actuator

JMX

Web Endpint

Enable 模块驱动

- `org.springframework.context.annotation.ImportBeanDefinitionRegistrar`
  - `@EnableFeignClients`
- `org.springframework.context.annotation.ImportSelector`
  - `@EnableAsync`
- `@Configuration` 类
  - `@EnableWebMvc`

Bean 注入的实现（DI 过程）

- 普通 Bean
  - 编码 BeanDefintion 生成的
  - XML 配置的
  - 注解标注的
  - 直接注册的
  - FactoryBean 生成的

FactoryBean 是如何当做一个 Bean 被注入?

- 返回 Bean 类型
  - getObjectType()
- 返回 Bean 对象
  - getObject()

BeanFactory 依赖查找

- 通过名称查找
  - getBean(String)
- 通过类型查找
  - getBean(Class)
    - 直接查找 BeanDefinition#getBeanClassName() : String
    - 间接查找 BeanFactory#getObjectType
- 通过名称 + 类型查找
  - getBean(String,Class)
- 通过注解查找
  - getBeansWithAnnotation(Annotation)