声明式客户端 OpenFeign 与负载均衡

课程讲义

主讲：Reythor 雷

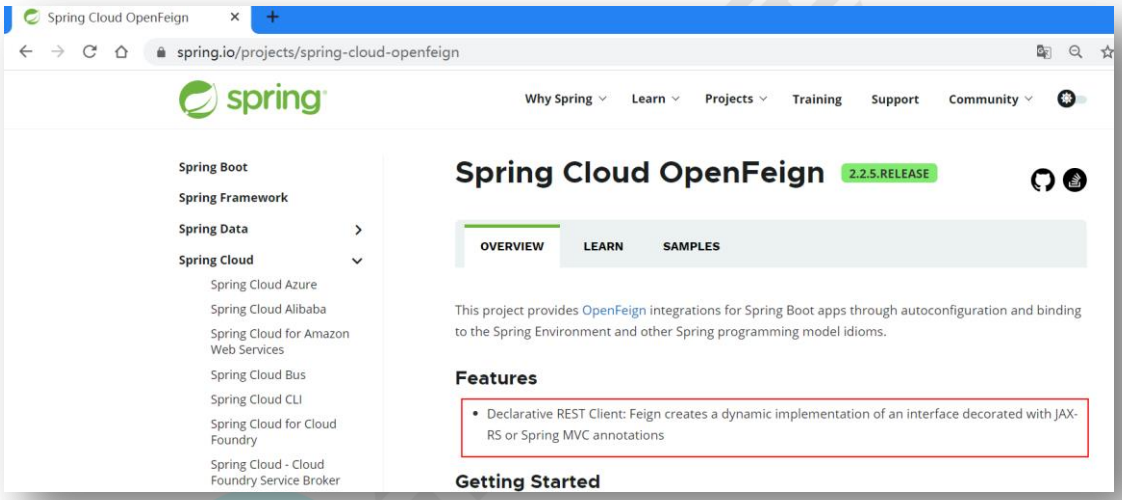
2021

声明式客户端 OpenFeign 与负载均衡

1.1 概述

1.1.1 OpenFeign 简介

（1） 官网简介



声明式 REST 客户端：Feign 通过使用 JAX-RS 或 SpringMVC 注解的装饰方式，生成接口的动态实现。

（2） 综合说明

前面官网的简介是什么意思呢？

Feign，假的，伪装的

OpenFeign 可以将提供者提供的 Restful 服务伪装为接口进行消费，消费者只需使用“feign 接口 + 注解”的方式即可直接调用提供者提供的 Restful 服务，而无需再使用 RestTemplate。

注意，OpenFeign 只有消费者有关，与提供者没有任务关系。

1.1.2 Ribbon 与 OpenFeign

说到 OpenFeign，不得不提的就是 Ribbon。OpenFeign 默认 Ribbon 作为负载均衡组件。OpenFeign 直接内置了 Ribbon。即在导入 OpenFeign 依赖后，无需再专门导入 Ribbon 依赖了。

OpenFeign 也是运行在消费者端的，使用 Ribbon 进行负载均衡，所以 OpenFeign 直接内置了 Ribbon。即在导入 OpenFeign 依赖后，无需再专门导入 Ribbon 依赖了。

1.1.3 消费者客户端技术选型

无论使用 Spring Cloud 还是使用 Spring Cloud Alibaba，消费者调用提供者服务都是需要使用客户端来提交消费请求的。常见的消费者客户端有如下三种，各有优劣。

（1） RestTemplate

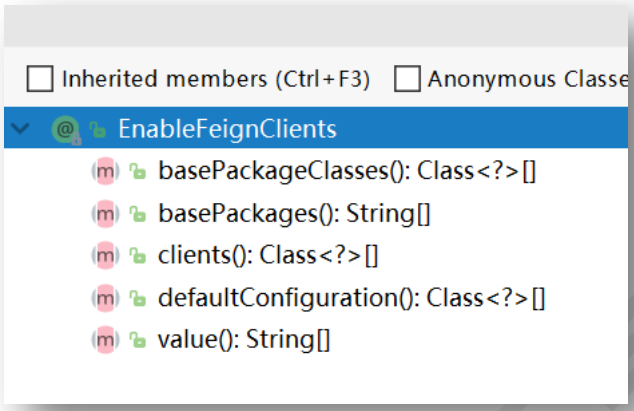
（2） OpenFeign

（3） Dubbo Spring Cloud

1.2 OpenFeign 源码解析

1.2.1 重要类与接口解析

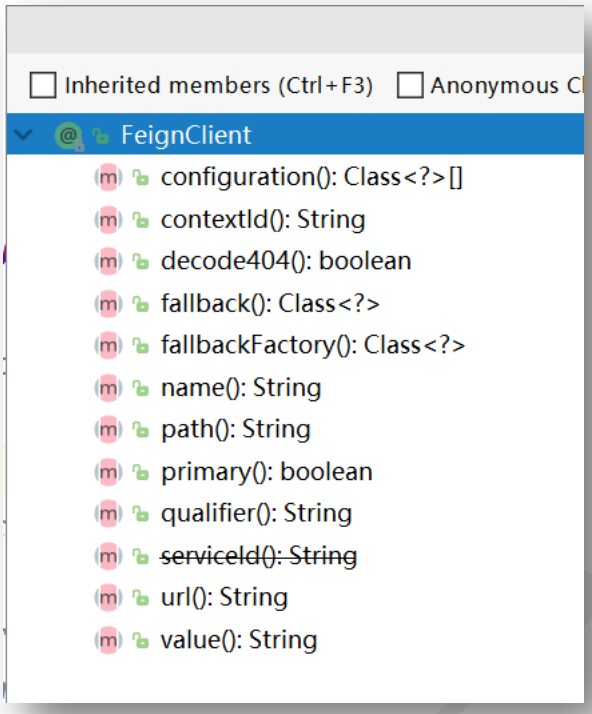
（1） @EnableFeignClients



在 SpringBoot 中存在大量的@EnableXxx 这种注解。它们的作用是，开启某项功能。其实它们本质上是为了导入某个类来完成某项功能。所以这个注解一般会组合一个@Import 注解用于导入类。导入的类一般有三种：

- 配置类：一般以 Configuration 结尾，完成自动配置
- 选择器：一般以 Selector 结尾，完成自动选择
- 注册器：一般以 Registrar 结尾，完成自动注册

（2） @FeignClient



（3） FeignClientSpecification 类

FeignClientSpecification 是一个 Feign Client 的生成规范。

```
/**
 * @author Dave Syer
 * @author Gregor Zurowski
 */
class FeignClientSpecification implements NamedContextFactory.Specification {

    private String name;

    private Class<?>[] configuration;
```

（4） BeanDefinition 接口

BeanDefinition 是一个 Bean 定义器。

```
 * @see org.springframework.beans.factory.support.ChildBeanDefinition
 */
public interface BeanDefinition extends AttributeAccessor, BeanMetadataElement {

    /**
     * Scope identifier for the standard singleton scope: "singleton".
     * <p>Note that extended bean factories might support further scopes.
     * @see #setScope
     */
    String SCOPE_SINGLETON = ConfigurableBeanFactory.SCOPE_SINGLETON;

    /**
     * Scope identifier for the standard prototype scope: "prototype".
```

（5） BeanDefinitionRegistry 接口

BeanDefinitionRegistry 是一个 BeanDefinition 注册器。

```
 */
public interface BeanDefinitionRegistry extends AliasRegistry {

    /**
     * Register a new bean definition with this registry.
     * Must support RootBeanDefinition and ChildBeanDefinition.
     * @param beanName the name of the bean instance to register
     * @param beanDefinition definition of the bean instance to regist
     * @throws BeanDefinitionStoreException if the BeanDefinition is i
     * @throws BeanDefinitionOverrideException if there is already a B
```

（6） FeignContext 类

FeignContext 是一个为 Feign Client 创建所准备的上下文对象。

```java
/**
 * A factory that creates instances of feign classes. It creates a Spring
 * ApplicationContext per client name, and extracts the beans that it needs from there.
 *
 * @author Spencer Gibb
 * @author Dave Syer
 */
public class FeignContext extends NamedContextFactory<FeignClientSpecification> {

    public FeignContext() {
        super(FeignClientsConfiguration.class, "feign", "feign.client.name");
    }

}
```

其父类中包含两个很重要的集合：

```java
// TODO: add javadoc
public abstract class NamedContextFactory<C> extends NamedContextFactory.Specification>
        implements DisposableBean, ApplicationContextAware {

    private final String propertySourceName;

    private final String propertyName;

    private Map<String, AnnotationConfigApplicationContext> contexts = new ConcurrentHashMap<>();

    private Map<String, C> configurations = new ConcurrentHashMap<>();
```

```java
private Map<String, AnnotationConfigApplicationContext> contexts = new ConcurrentHashMap<>();
```

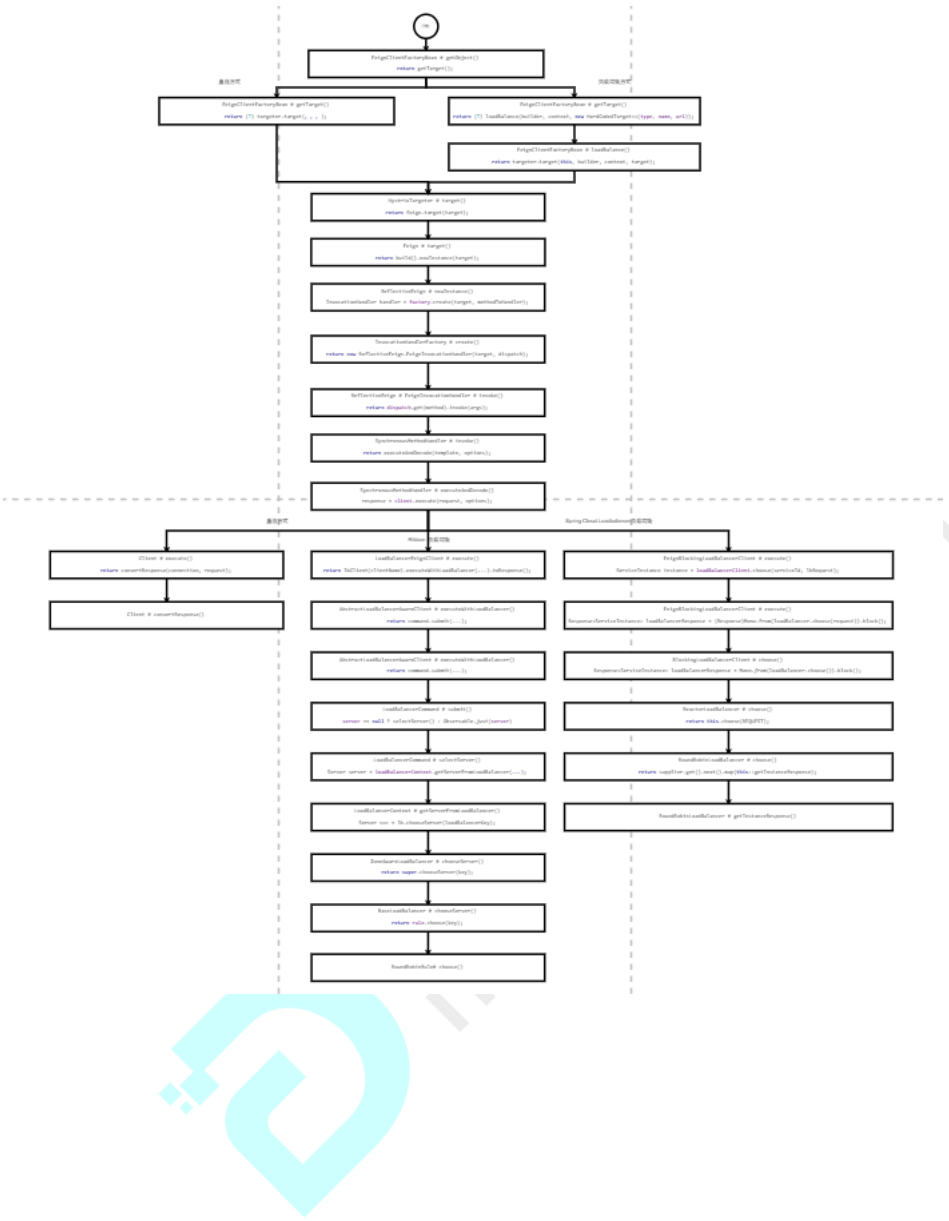
该 map 的 key 为 FeignClient 的名称（其所要调用的微服务名称），value 是组装这个 FeignClient 所必须的组件所在的 Spring 子容器

```java
private Map<String, C> configurations = new ConcurrentHashMap<>();
```

这个 map 中存放的是@EnableFeignClients 与@FeignClient 两个注解中的 configuration 属性值。这个属性值只有两类：

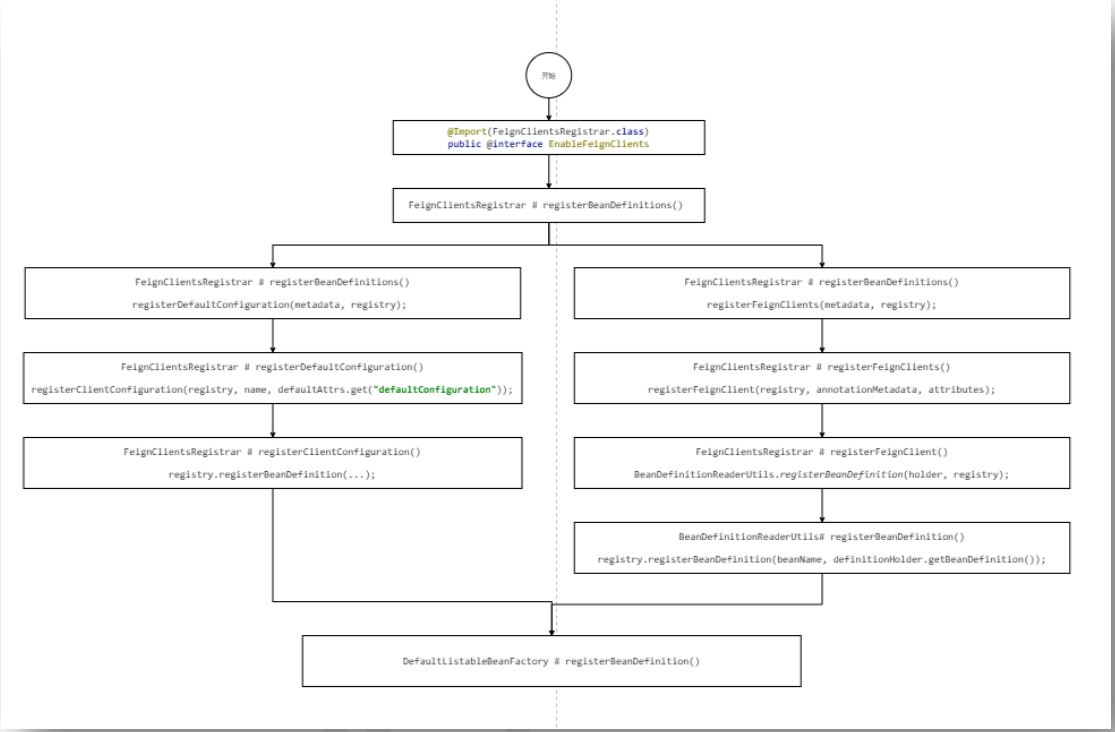
- 第一类只有一个，其 key 为字符串　default + 当前启动类的全限定性类名 ，例如：default.com.abc.ConsumerFeign8080，value 为@EnableFeignClients 的 defaultConfiguration 属性值
- 第二类有多个，其 key 为当前@FeignClient 的名称，value 为这个注解的 configuration 属性值

1.2.2 OpenFeign 的完整解析图



1.2.3 Feign Client 的创建

（1） 完成配置注册



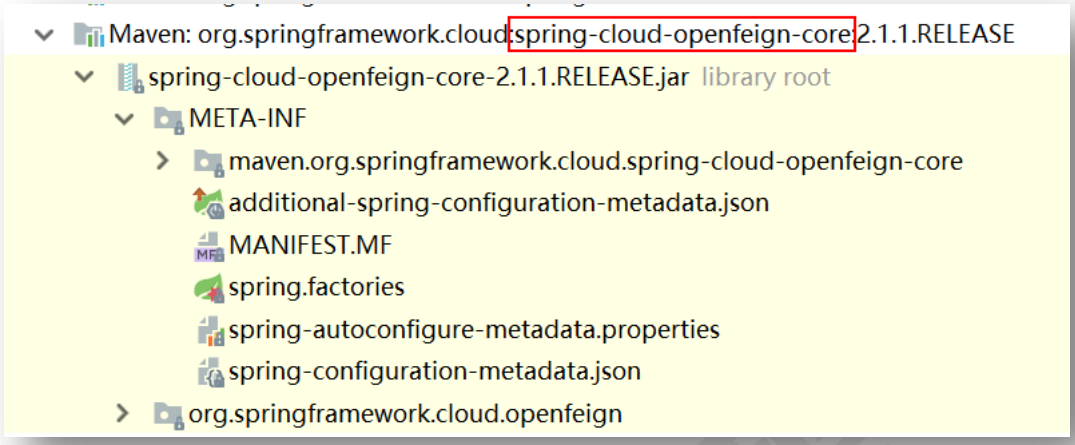
volatile 与 synchronized 的区别与联系：

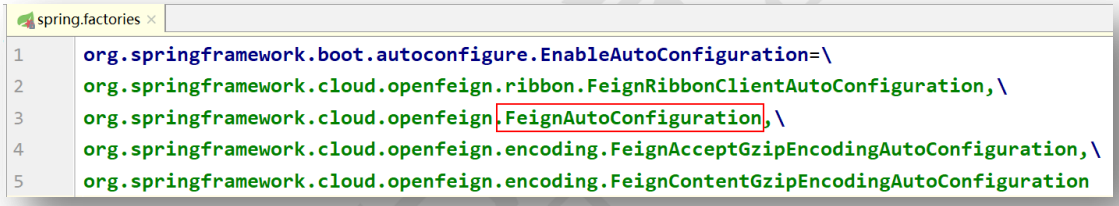
volatile 保证了　可见性

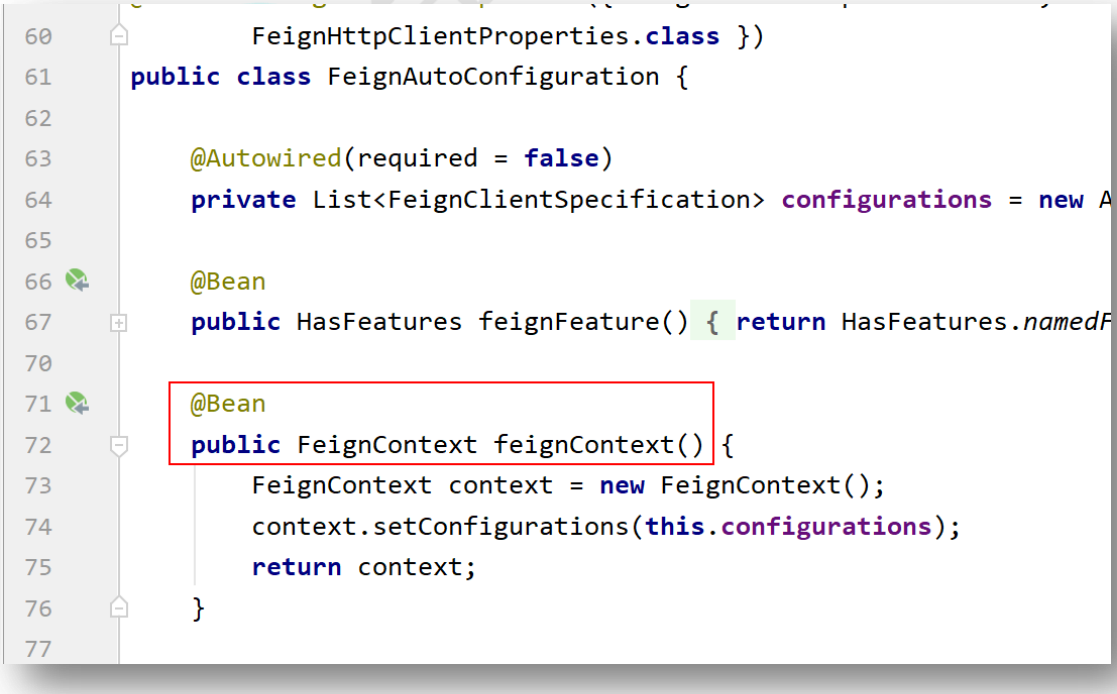
synchronized 保证了　有序性

工作内存：线程
主内存：进程

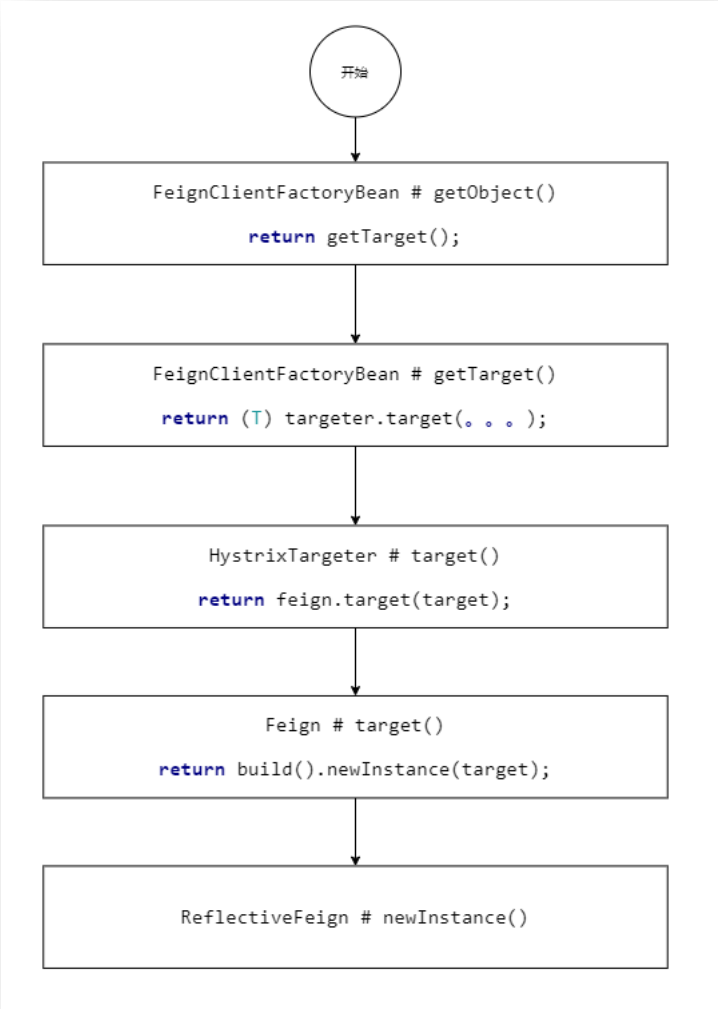
（2） 完成自动配置


Maven: org.springframework.cloud:spring-cloud-openfeign-core:2.1.1.RELEASE
spring-cloud-openfeign-core-2.1.1.RELEASE.jar library root
META-INF
maven.org.springframework.cloud.spring-cloud-openfeign-core
additional-spring-configuration-metadata.json
MANIFEST.MF
spring.factories
spring-autoconfigure-metadata.properties
spring-configuration-metadata.json
org.springframework.cloud.openfeign

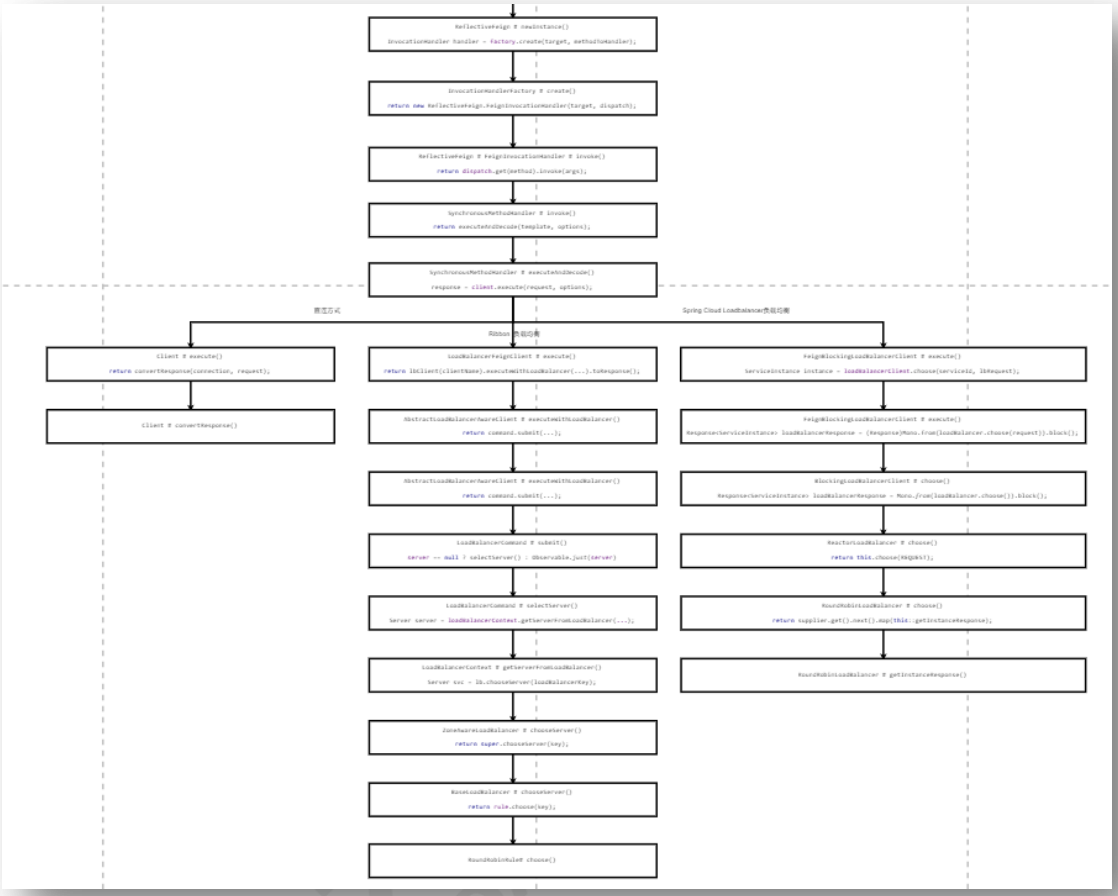

spring.factories
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
org.springframework.cloud.openfeign.ribbon.FeignRibbonClientAutoConfiguration,\
org.springframework.cloud.openfeign.FeignAutoConfiguration,\
org.springframework.cloud.openfeign.encoding.FeignAcceptGzipEncodingAutoConfiguration,\
org.springframework.cloud.openfeign.encoding.FeignContentGzipEncodingAutoConfiguration


FeignHttpClientProperties.class })
public class FeignAutoConfiguration {
@Autowired(required = false)
private List<FeignClientSpecification> configurations = new A
@Bean
public HasFeatures feignFeature() { return HasFeatures.namedF
@Bean
public FeignContext feignContext() {
FeignContext context = new FeignContext();
context.setConfigurations(this.configurations);
return context;
}

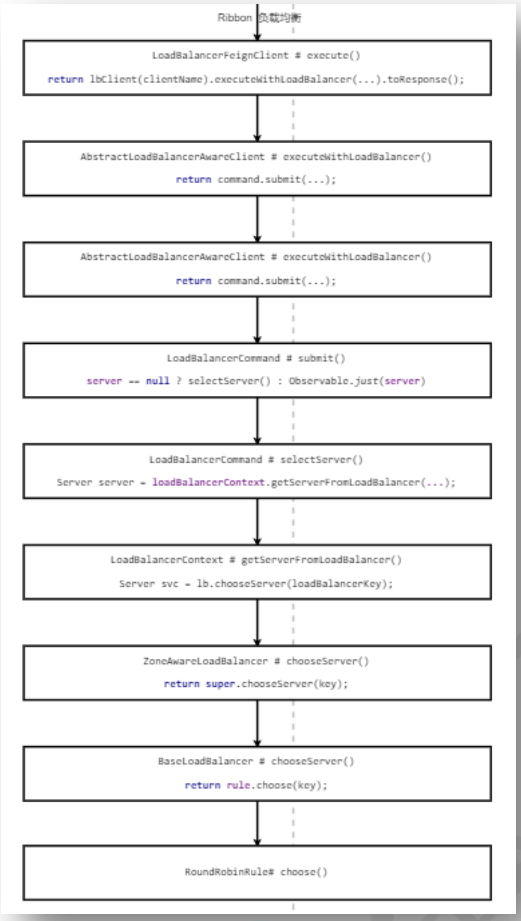
（3） 生成 Feign Client



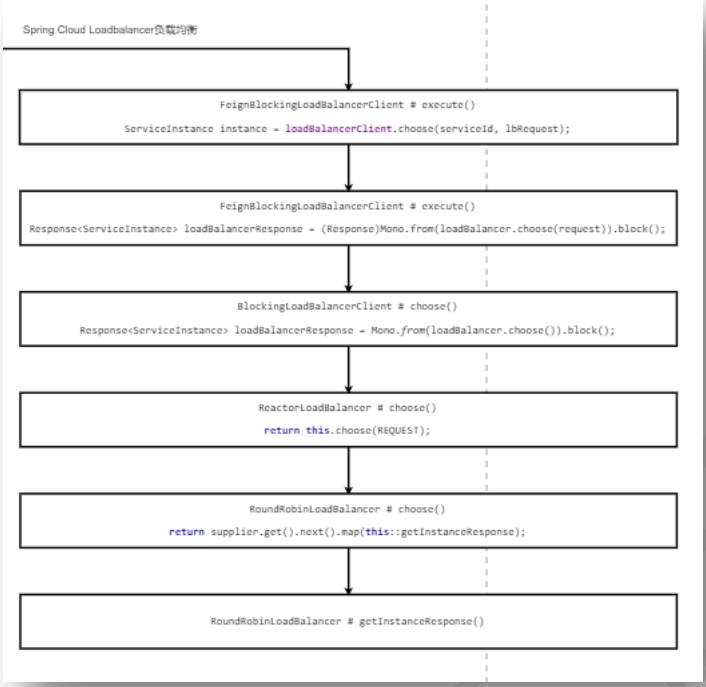
1.2.4 发出网络请求



1.2.5 Ribbon 负载均衡



1.2.6 Spring Cloud LoadBalancer 负载均衡



1.3 Ribbon 内置负载均衡算法

1.3.1 RoundRobinRule

轮询策略。Ribbon 默认采用的策略。若经过一轮轮询没有找到可用的 provider，其最多轮询 10 轮（代码中写死的，不能修改）。若还未找到，则返回 null。

1.3.2 RandomRule

随机策略，从所有可用的 provider 中随机选择一个。

1.3.3 RetryRule

重试策略。先按照 RoundRobinRule 策略获取 server，若获取失败，则在指定的时限内重试。默认的时限为 500 毫秒。

1.3.4 BestAvailableRule

最可用策略。选择并发量最小的 provider，即连接的消费者数量最少的 provider。其会遍历服务列表中的每一个 provider，选择当前连接数量 minimalConcurrentConnections 最小的 provider。

1.3.5 AvailabilityFilteringRule

可用过滤算法。该算法规则是：过滤掉处于熔断状态的 server 与已经超过连接极限的 server，对剩余 server 采用轮询策略。