# 服务注册与发现

扫码试看/订阅
《玩转 Spring 全家桶》

## 使用 Eureka 作为服务注册中心

## 认识 Eureka

**什么是 Eureka**

- Eureka 是在 AWS 上定位服务的 REST 服务

**Netflix OSS**

不在AWS上时候用发布在Netflix中。但是不在AWS上不建议用
现在2.0版本已经不再开源

- https://netflix.github.io

**Spring 对 Netflix 套件的支持**

- Spring Cloud Netflix

## 在本地启动一个简单的 Eureka 服务

**Starter**

- spring-cloud-dependencies
- spring-cloud-starter-netflix-eureka-starter

**声明** 在config类上

- @EnableEurekaServer 这是单机的，要部署一个集群来用！！！

**注意事项**

- 默认端口8761 Localhost:8761 直接打开
- Eureka 自己不要注册到 Eureka 了

# 将服务注册到 Eureka Server

**Starter** 如果俩注解都不加，clasthPath里面有这个依赖的话也行

- spring-cloud-starter-netflix-eureka-client

**声明**

- `@EnableDiscoveryClient` 这是个集成的注解
- `@EnableEurekaClient`

**一些配置项**

- `eureka.client.service-url.default-zone`
- `eureka.client.instance.prefer-ip-address` 优先以IP地址

# 关于 Bootstrap 属性

跟配置application.properties一样的,这里配置bootstrap.properties

**Bootstrap 属性**

- 启动引导阶段加载的属性
- bootstrap.properties | .yml
- `spring.cloud.bootstrap.name=bootstrap`

**常用配置**

- `spring.application.name=应用名`
- 配置中心相关

注意jaxb依赖在JDK11里去掉了，需要自己加入，jdk8还有

“Talk is cheap, show me the code.”

*Chapter 12 / eureka-server eureka-waiter-service*

## 使用 Spring Cloud LoadBalancer 访问服务

server.port = 0 ——随机选一个端口

## 如何获得服务地址

**EurekaClient**

- `getNextServerFromEureka()`

**DiscoveryClient** springCloud给的抽象，建议使用

- `getInstances()`

# Load Balancer Client

增强了RestTemplate

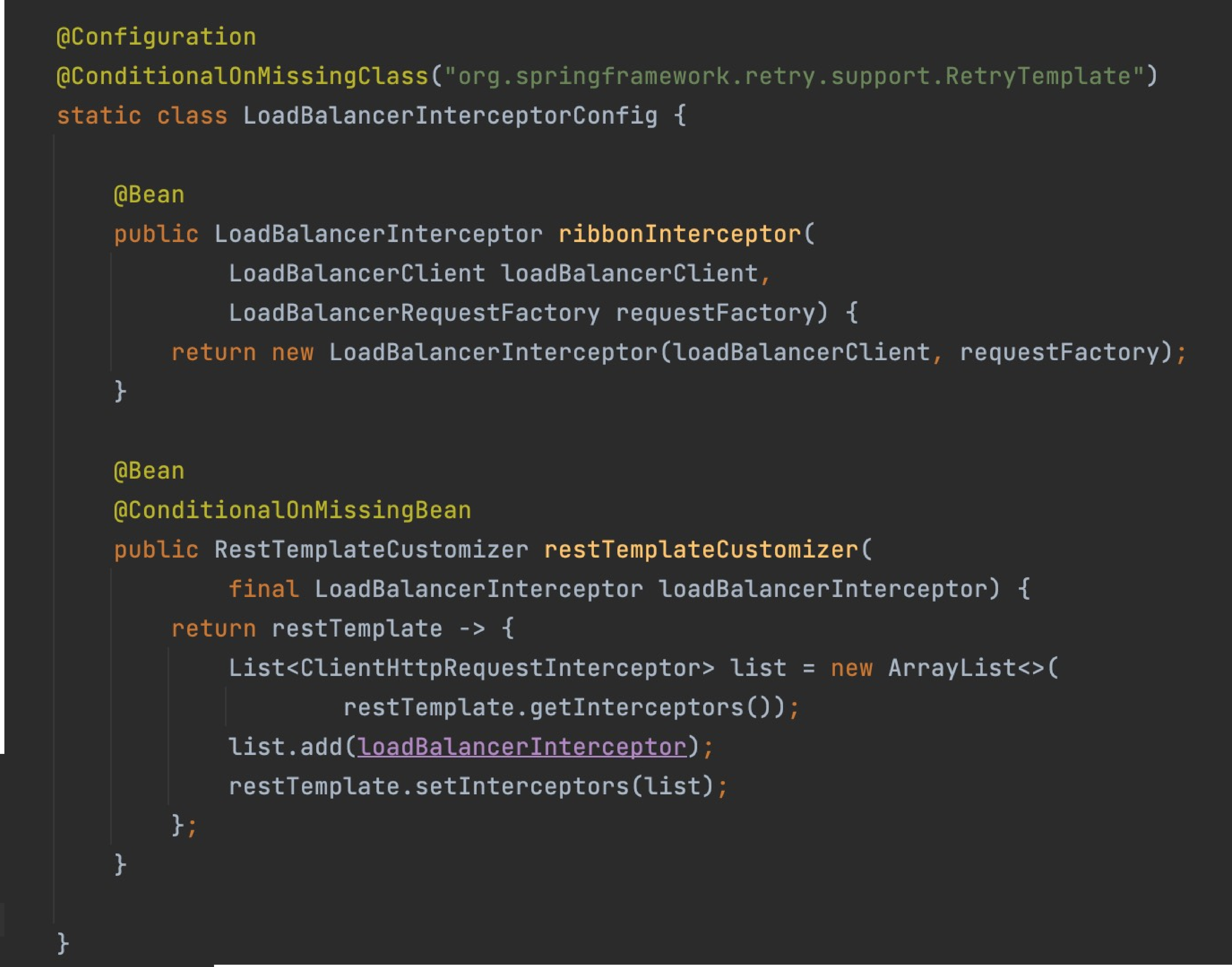
**RestTemplate 与 WebClient**

- @LoadBalaced 为restTemplate或者WebClient增加负载均衡支持
- 实际是通过 ClientHttpRequestInterceptor 实现的 接口
  - LoadBalancerInterceptor
  - LoadBalancerClient 获取目标地址
    - RibbonLoadBalancerClient

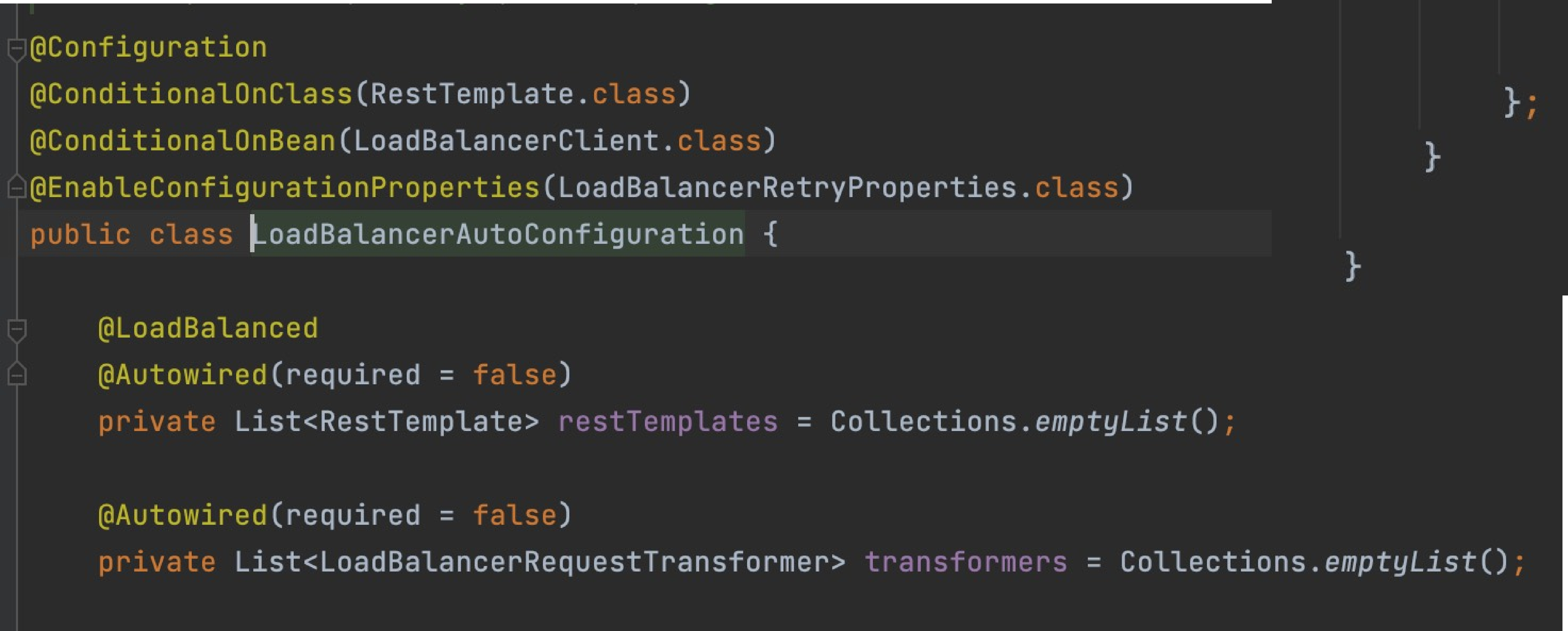
ribbon是netflix套件提供的

“Talk is cheap, show me the code.”

*Chapter 12 / ribbon-customer-service*



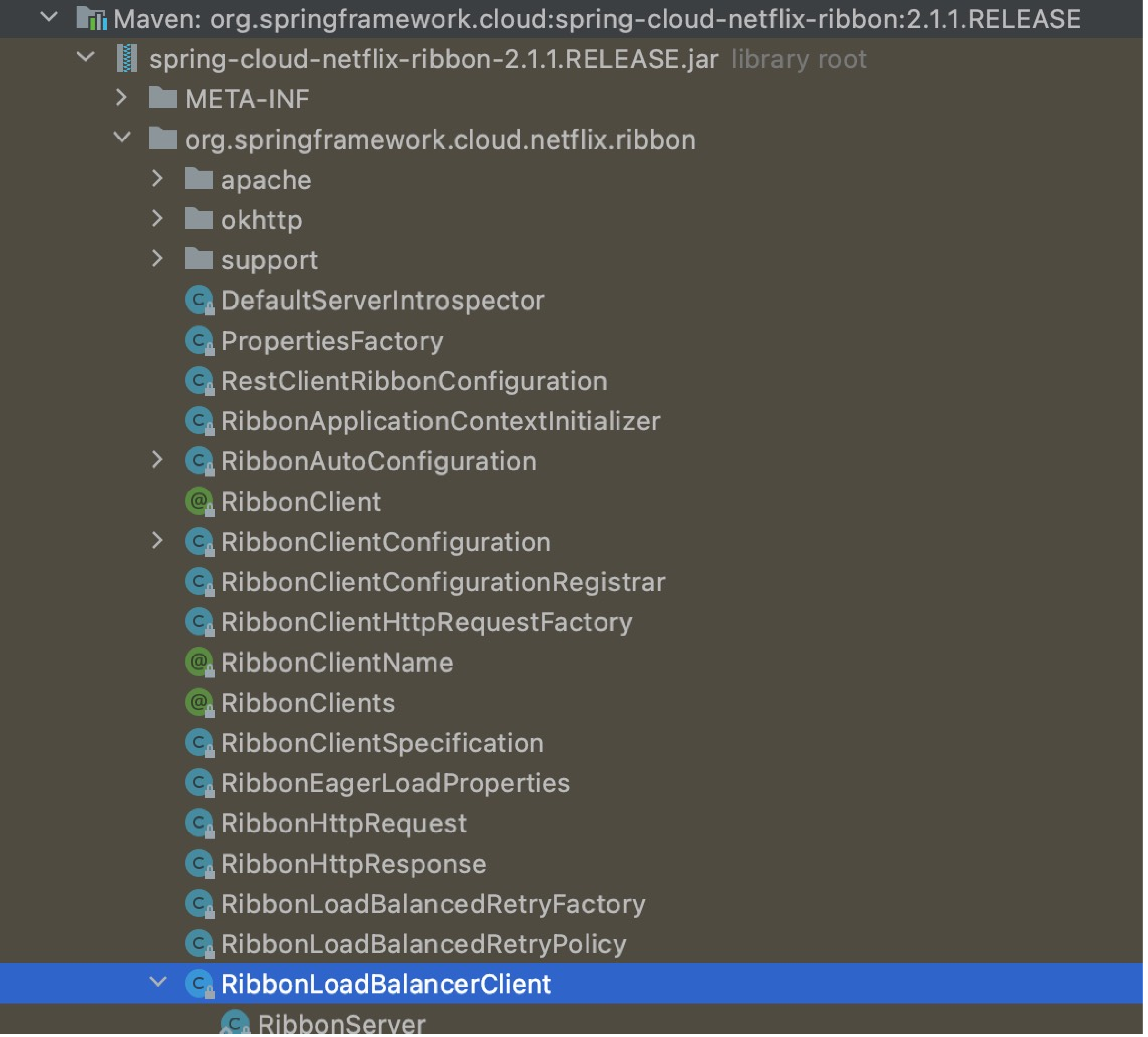
```java
@Configuration
@ConditionalOnMissingClass("org.springframework.retry.support.RetryTemplate")
static class LoadBalancerInterceptorConfig {

    @Bean
    public LoadBalancerInterceptor ribbonInterceptor(
            LoadBalancerClient loadBalancerClient,
            LoadBalancerRequestFactory requestFactory) {
        return new LoadBalancerInterceptor(loadBalancerClient, requestFactory);
    }

    @Bean
    @ConditionalOnMissingBean
    public RestTemplateCustomizer restTemplateCustomizer(
            final LoadBalancerInterceptor loadBalancerInterceptor) {
        return restTemplate -> {
            List<ClientHttpRequestInterceptor> list = new ArrayList<>(
                    restTemplate.getInterceptors());
            list.add(loadBalancerInterceptor);
            restTemplate.setInterceptors(list);
        };
    }

}
```



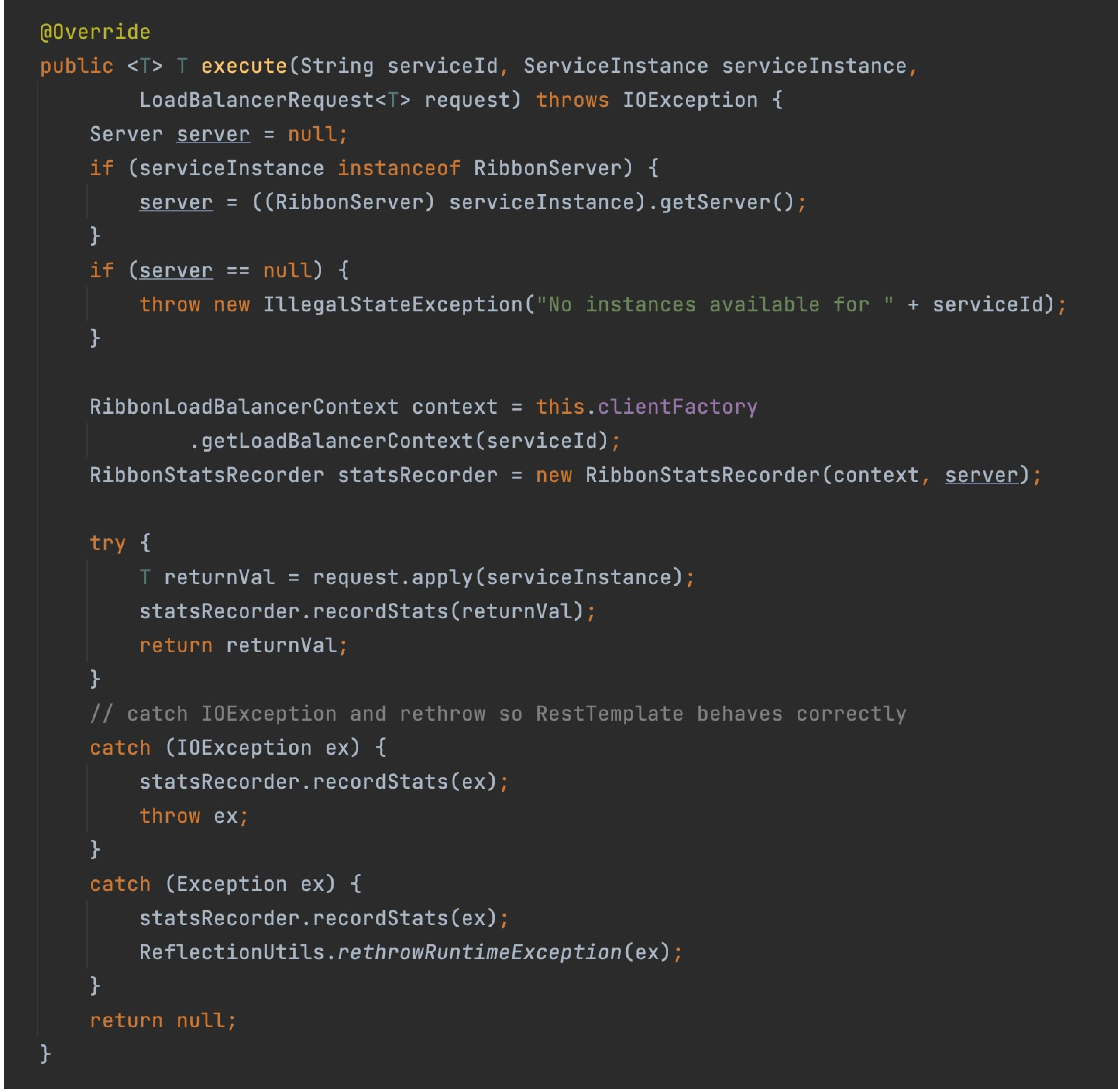
```java
@Configuration
@ConditionalOnClass(RestTemplate.class)
@ConditionalOnBean(LoadBalancerClient.class)
@EnableConfigurationProperties(LoadBalancerRetryProperties.class)
public class LoadBalancerAutoConfiguration {

    @LoadBalanced
    @Autowired(required = false)
    private List<RestTemplate> restTemplates = Collections.emptyList();

    @Autowired(required = false)
    private List<LoadBalancerRequestTransformer> transformers = Collections.emptyList();
```

```
Maven: org.springframework.cloud:spring-cloud-commons:2.1.1.REL
  spring-cloud-commons-2.1.1.RELEASE.jar library root
    META-INF
    org.springframework.cloud
      client
        actuator
        circuitbreaker
        discovery
        hypermedia
        loadbalancer
          reactive
          AsyncLoadBalancerAutoConfiguration
          AsyncLoadBalancerInterceptor
          AsyncRestTemplateCustomizer
          ClientHttpResponseStatusCodeException
          Read-only class
          LoadBalancedRecoveryCallback
          LoadBalancedRetryContext
          LoadBalancedRetryFactory
          LoadBalancedRetryPolicy
          LoadBalancerAutoConfiguration
          LoadBalancerClient
          LoadBalancerInterceptor
          LoadBalancerRequest
          LoadBalancerRequestFactory
          LoadBalancerRequestTransformer
          LoadBalancerRetryProperties
          RestTemplateCustomizer
          RetryableStatusCodeException
          RetryLoadBalancerInterceptor
          ServiceInstanceChooser
          ServiceRequestWrapper
```

```java
Author: Spencer Gibb, Dave Syer, Ryan Baxter, William Tran
public class LoadBalancerInterceptor implements ClientHttpRequestInterceptor {

    private LoadBalancerClient loadBalancer;

    private LoadBalancerRequestFactory requestFactory;

    public LoadBalancerInterceptor(LoadBalancerClient loadBalancer,
            LoadBalancerRequestFactory requestFactory) {
        this.loadBalancer = loadBalancer;
        this.requestFactory = requestFactory;
    }

    public LoadBalancerInterceptor(LoadBalancerClient loadBalancer) {
        // for backwards compatibility
        this(loadBalancer, new LoadBalancerRequestFactory(loadBalancer));
    }

    @Override
    public ClientHttpResponse intercept(final HttpRequest request, final byte[] body,
            final ClientHttpRequestExecution execution) throws IOException {
        final URI originalUri = request.getURI();
        String serviceName = originalUri.getHost();
        Assert.state( expression: serviceName != null,
                 message: "Request URI does not contain a valid hostname: " + originalUri);
        return this.loadBalancer.execute(serviceName,
                this.requestFactory.createRequest(request, body, execution));
    }

}
```



```
Maven: org.springframework.cloud:spring-cloud-netflix-ribbon:2.1.1.RELEASE
  spring-cloud-netflix-ribbon-2.1.1.RELEASE.jar library root
    META-INF
    org.springframework.cloud.netflix.ribbon
      apache
      okhttp
      support
      DefaultServerIntrospector
      PropertiesFactory
      RestClientRibbonConfiguration
      RibbonApplicationContextInitializer
      RibbonAutoConfiguration
      RibbonClient
      RibbonClientConfiguration
      RibbonClientConfigurationRegistrar
      RibbonClientHttpRequestFactory
      RibbonClientName
      RibbonClients
      RibbonClientSpecification
      RibbonEagerLoadProperties
      RibbonHttpRequest
      RibbonHttpResponse
      RibbonLoadBalancedRetryFactory
      RibbonLoadBalancedRetryPolicy
      RibbonLoadBalancerClient
        RibbonServer
```



```java
@Override
public <T> T execute(String serviceId, ServiceInstance serviceInstance,
        LoadBalancerRequest<T> request) throws IOException {
    Server server = null;
    if (serviceInstance instanceof RibbonServer) {
        server = ((RibbonServer) serviceInstance).getServer();
    }
    if (server == null) {
        throw new IllegalStateException("No instances available for " + serviceId);
    }

    RibbonLoadBalancerContext context = this.clientFactory
            .getLoadBalancerContext(serviceId);
    RibbonStatsRecorder statsRecorder = new RibbonStatsRecorder(context, server);

    try {
        T returnVal = request.apply(serviceInstance);
        statsRecorder.recordStats(returnVal);
        return returnVal;
    }
    // catch IOException and rethrow so RestTemplate behaves correctly
    catch (IOException ex) {
        statsRecorder.recordStats(ex);
        throw ex;
    }
    catch (Exception ex) {
        statsRecorder.recordStats(ex);
        ReflectionUtils.rethrowRuntimeException(ex);
    }
    return null;
}
```

## 使用 Feign 访问服务

# 认识 Feign

**Feign**

- 声明式 REST Web 服务客户端
- https://github.com/OpenFeign/feign

**Spring Cloud OpenFeign**

- spring-cloud-starter-openfeign

# Feign 的简单使用

**开启 Feign 支持**

- @EnableFeignClients 用contextId = “” 区分同一个name= “”

**定义 Feign 接口**

- @FeignClient 添加了这个注解的接口会被实例化一个Bean

**简单配置**

- FeignClientsConfiguration
- Encoder / Decoder / Logger / Contract / Client …

不要把@RequestMapping加到接口上，直接用@FeignClient(path= “” )

极客时间

## 通过配置定制 Feign

yml方式

```yaml
feign:
  client:
    config:
      feignName:
        connectTimeout: 5000
        readTimeout: 5000
        loggerLevel: full
        errorDecoder: com.example.SimpleErrorDecoder
        retryer: com.example.SimpleRetryer
        requestInterceptors:
          - com.example.FooRequestInterceptor
          - com.example.BarRequestInterceptor
        decode404: false
        encoder: com.example.SimpleEncoder
        decoder: com.example.SimpleDecoder
        contract: com.example.SimpleContract
```

## Feign 的一些其他配置

使用application.properties配置

- feign.okhttp.enabled=true
- feign.httpclient.enabled=true

支持httpClient的话就要放进去httpClient的包

- feign.compression.response.enabled=true
- feign.compression.request.enabled=true
- feign.compression.request.mime-types=
  text/xml,application/xml,application/json

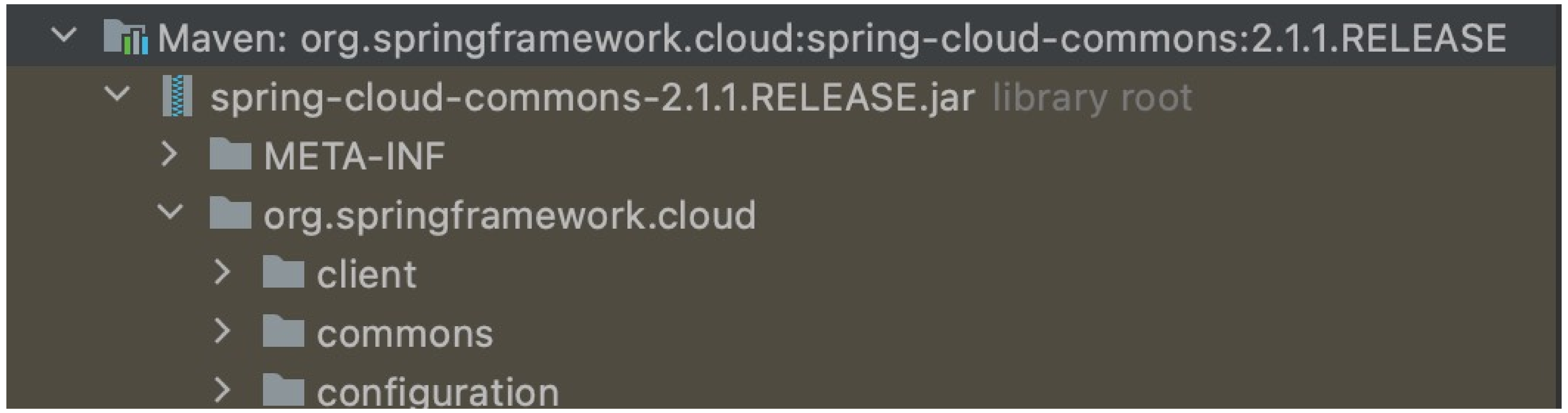
对xml和

- feign.compression.request.min-request-size=2048

“Talk is cheap, show me the code.”

*Chapter 12 / feign-customer-service*

## 深入理解 DiscoveryClient



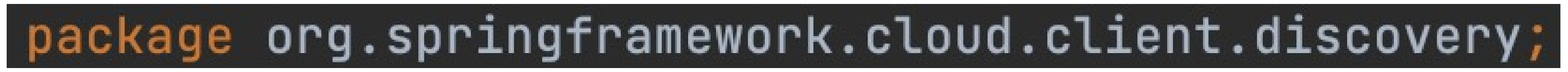
```
Maven: org.springframework.cloud:spring-cloud-commons:2.1.1.RELEASE
  spring-cloud-commons-2.1.1.RELEASE.jar library root
    META-INF
    org.springframework.cloud
      client
      commons
      configuration
```

## Spring Cloud Commons 提供的抽象

**服务注册抽象**

- 提供了 ServiceRegistry 抽象 专门负责注册

**客户发现抽象**



```
package org.springframework.cloud.client.discovery;
```

- 提供了 DiscoveryClient 抽象 不管后端具体使用了什么服务中心
  - @EnableDiscoveryClient
- 提供了 LoadBalancerClient 抽象

## 自动向 Eureka 服务端注册

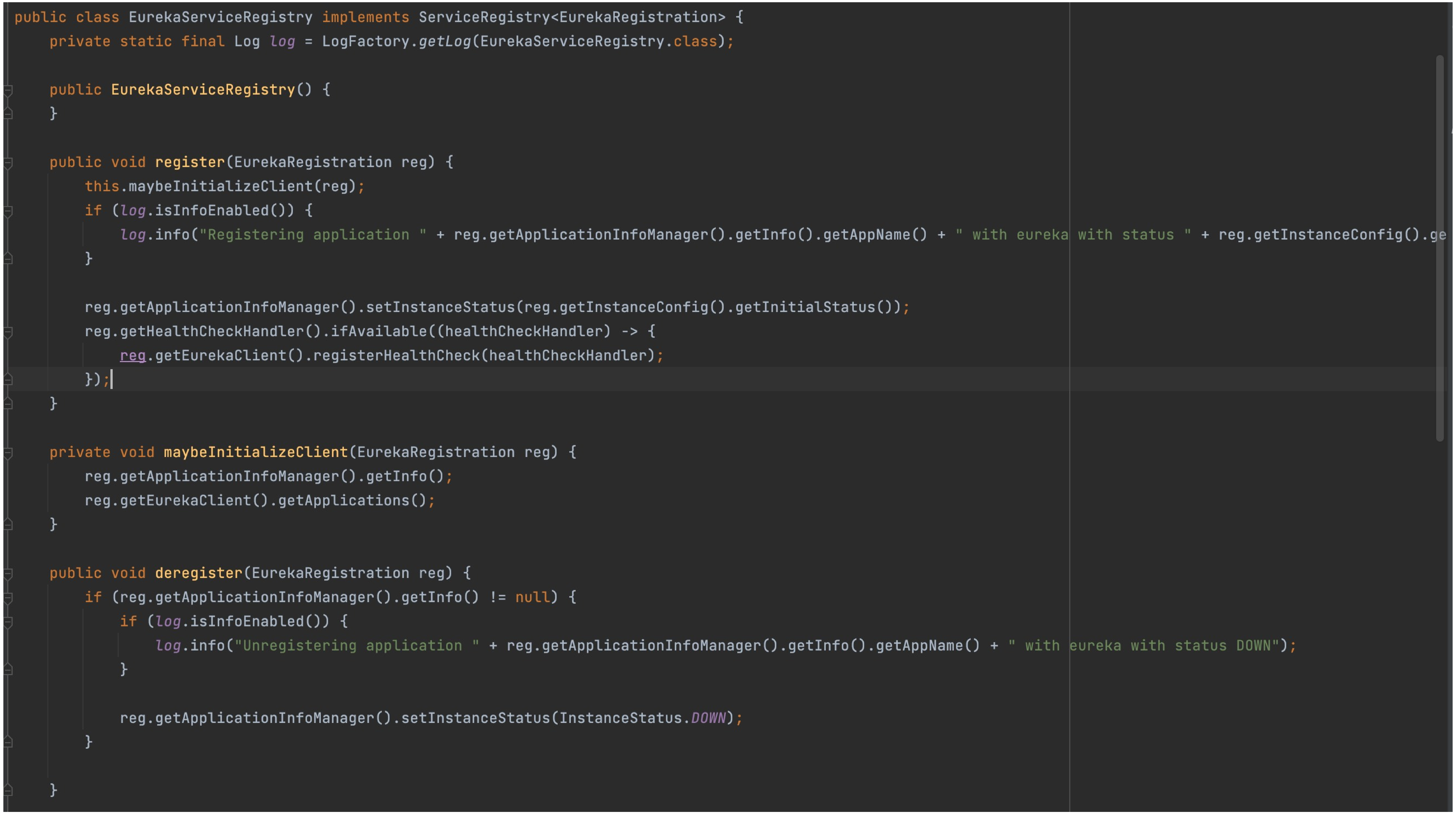
**ServiceRegistry**

- EurekaServiceRegistry  Eureka服务注册
- EurekaRegistration  注册信息放在这

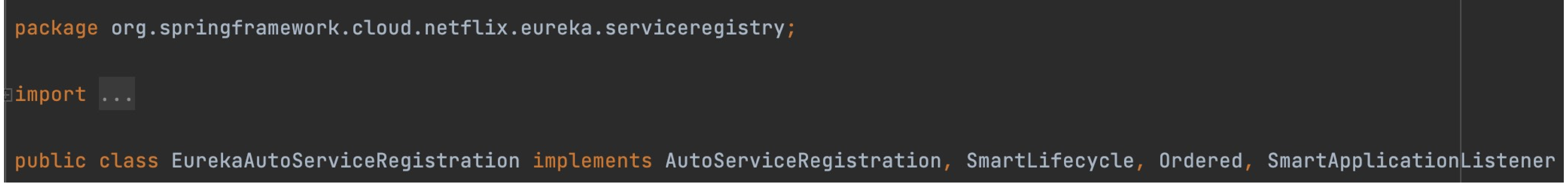
**自动配置**

- EurekaClientAutoConfiguration
- EurekaAutoServiceRegistration
  - SmartLifecycle  本质上由lifeCycle实现的

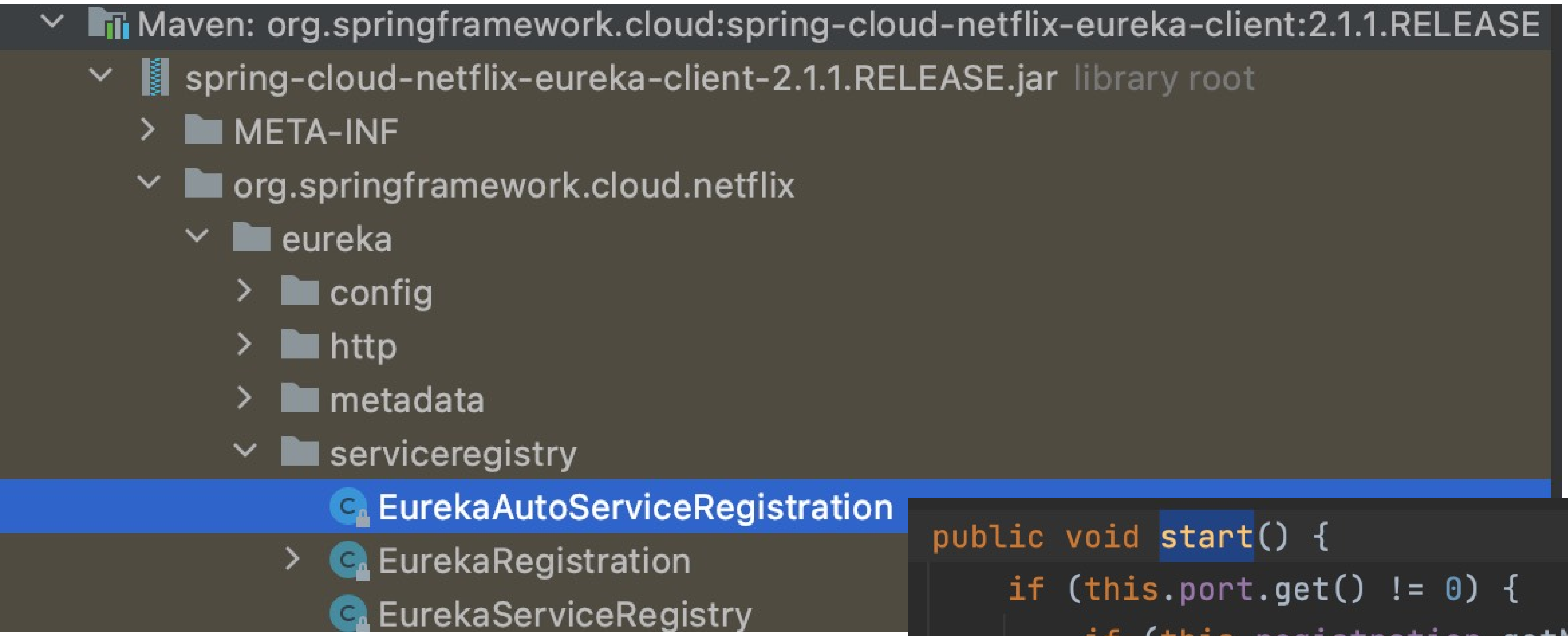
```java
package org.springframework.cloud.netflix.eureka.serviceregistry;
```



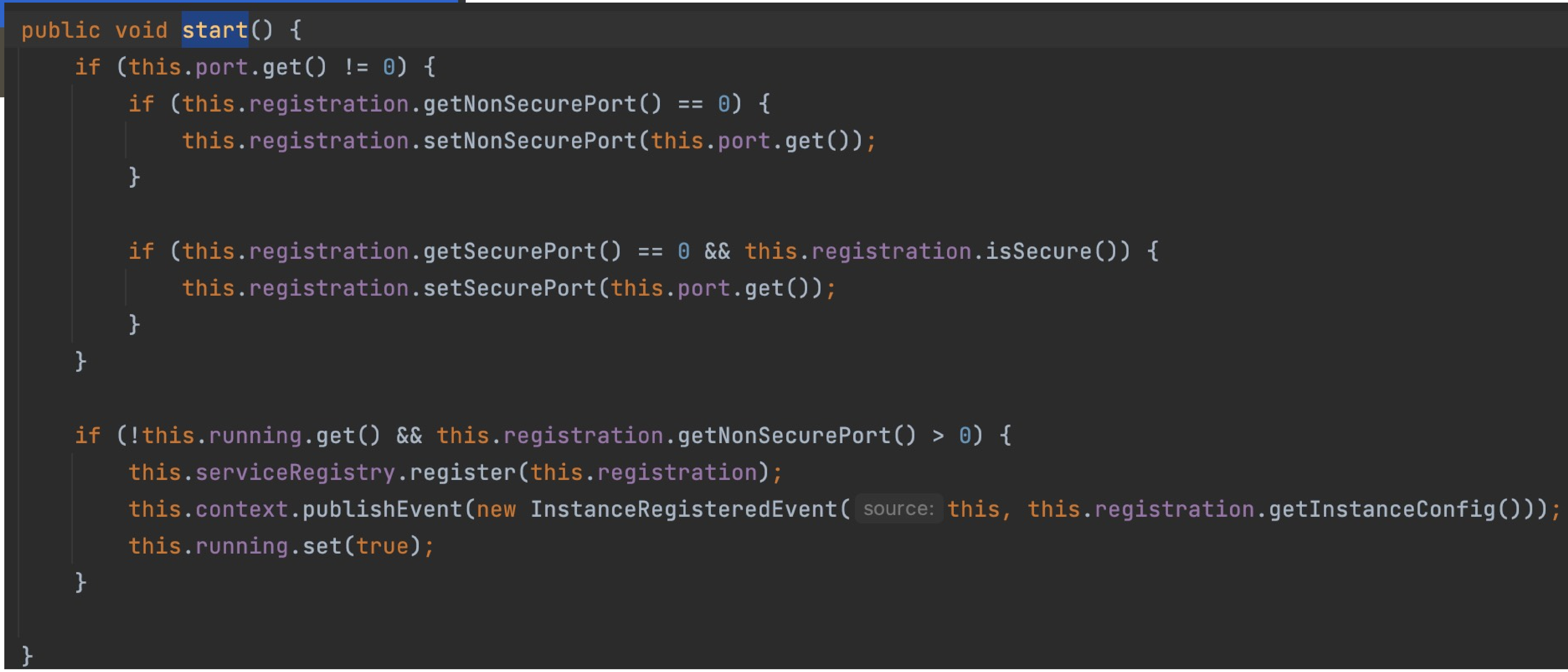
```java
public class EurekaServiceRegistry implements ServiceRegistry<EurekaRegistration> {
    private static final Log log = LogFactory.getLog(EurekaServiceRegistry.class);

    public EurekaServiceRegistry() {
    }

    public void register(EurekaRegistration reg) {
        this.maybeInitializeClient(reg);
        if (log.isInfoEnabled()) {
            log.info("Registering application " + reg.getApplicationInfoManager().getInfo().getAppName() + " with eureka with status " + reg.getInstanceConfig().ge
        }

        reg.getApplicationInfoManager().setInstanceStatus(reg.getInstanceConfig().getInitialStatus());
        reg.getHealthCheckHandler().ifAvailable((healthCheckHandler) -> {
            reg.getEurekaClient().registerHealthCheck(healthCheckHandler);
        });
    }

    private void maybeInitializeClient(EurekaRegistration reg) {
        reg.getApplicationInfoManager().getInfo();
        reg.getEurekaClient().getApplications();
    }

    public void deregister(EurekaRegistration reg) {
        if (reg.getApplicationInfoManager().getInfo() != null) {
            if (log.isInfoEnabled()) {
                log.info("Unregistering application " + reg.getApplicationInfoManager().getInfo().getAppName() + " with eureka with status DOWN");
            }

            reg.getApplicationInfoManager().setInstanceStatus(InstanceStatus.DOWN);
        }

    }
```



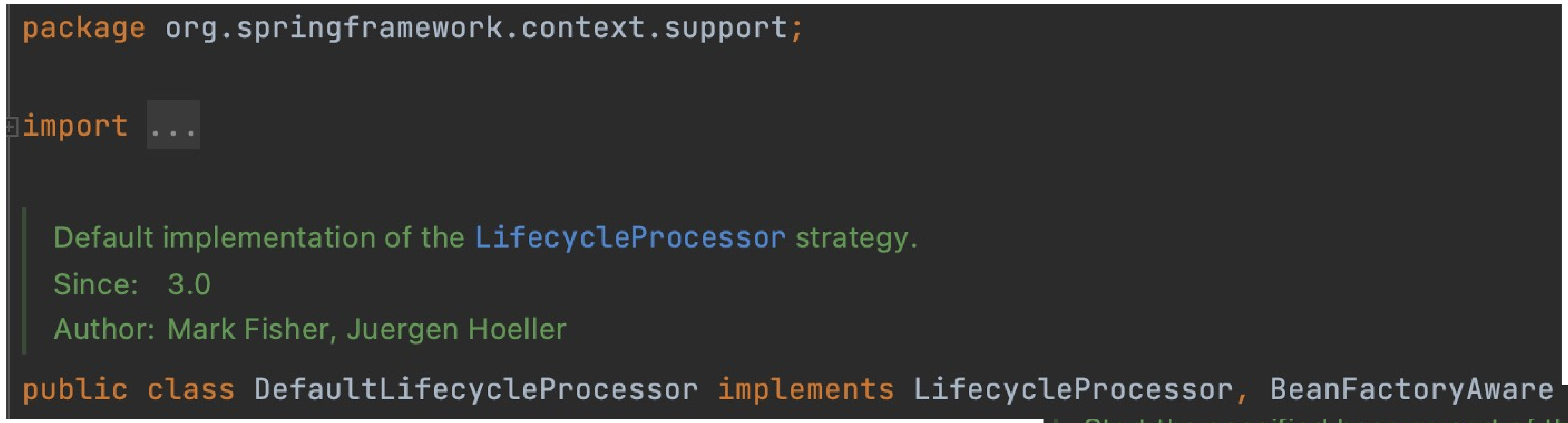
```java
package org.springframework.cloud.netflix.eureka.serviceregistry;

import ...

public class EurekaAutoServiceRegistration implements AutoServiceRegistration, SmartLifecycle, Ordered, SmartApplicationListener
```



```
Maven: org.springframework.cloud:spring-cloud-netflix-eureka-client:2.1.1.RELEASE
  spring-cloud-netflix-eureka-client-2.1.1.RELEASE.jar library root
    META-INF
    org.springframework.cloud.netflix
      eureka
        config
        http
        metadata
        serviceregistry
          EurekaAutoServiceRegistration
          EurekaRegistration
          EurekaServiceRegistry
```



```java
public void start() {
    if (this.port.get() != 0) {
        if (this.registration.getNonSecurePort() == 0) {
            this.registration.setNonSecurePort(this.port.get());
        }

        if (this.registration.getSecurePort() == 0 && this.registration.isSecure()) {
            this.registration.setSecurePort(this.port.get());
        }
    }

    if (!this.running.get() && this.registration.getNonSecurePort() > 0) {
        this.serviceRegistry.register(this.registration);
        this.context.publishEvent(new InstanceRegisteredEvent( source: this, this.registration.getInstanceConfig()));
        this.running.set(true);
    }

}
```

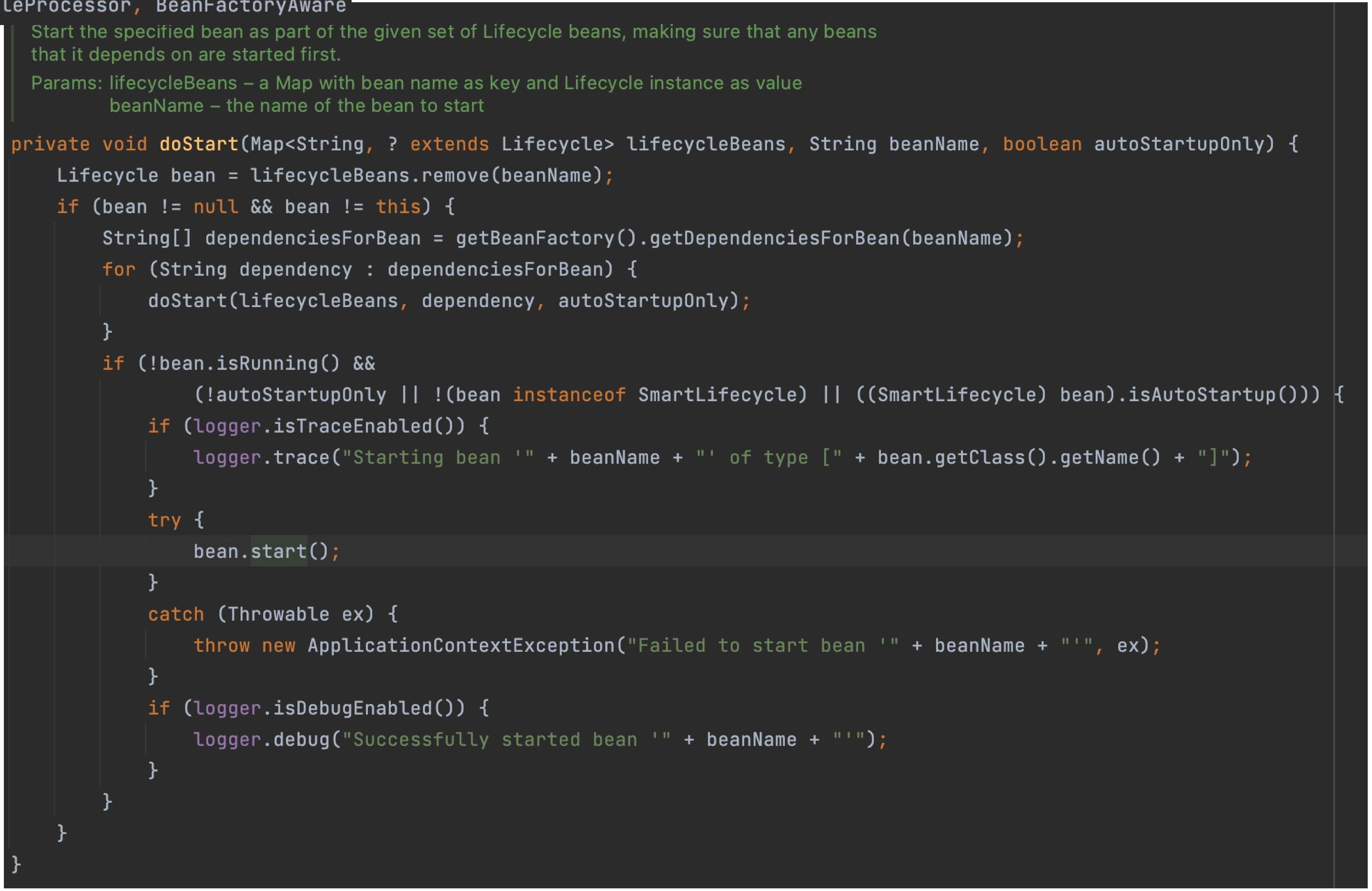


```
package org.springframework.context.support;

import ...
```

Default implementation of the LifecycleProcessor strategy.
Since: 3.0
Author: Mark Fisher, Juergen Hoeller

```
public class DefaultLifecycleProcessor implements LifecycleProcessor, BeanFactoryAware
```

调用方：



Start the specified bean as part of the given set of Lifecycle beans, making sure that any beans that it depends on are started first.
Params: lifecycleBeans – a Map with bean name as key and Lifecycle instance as value
beanName – the name of the bean to start

```
private void doStart(Map<String, ? extends Lifecycle> lifecycleBeans, String beanName, boolean autoStartupOnly) {
    Lifecycle bean = lifecycleBeans.remove(beanName);
    if (bean != null && bean != this) {
        String[] dependenciesForBean = getBeanFactory().getDependenciesForBean(beanName);
        for (String dependency : dependenciesForBean) {
            doStart(lifecycleBeans, dependency, autoStartupOnly);
        }
        if (!bean.isRunning() &&
                (!autoStartupOnly || !(bean instanceof SmartLifecycle) || ((SmartLifecycle) bean).isAutoStartup())) {
            if (logger.isTraceEnabled()) {
                logger.trace("Starting bean '" + beanName + "' of type [" + bean.getClass().getName() + "]");
            }
            try {
                bean.start();
            }
            catch (Throwable ex) {
                throw new ApplicationContextException("Failed to start bean '" + beanName + "'", ex);
            }
            if (logger.isDebugEnabled()) {
                logger.debug("Successfully started bean '" + beanName + "'");
            }
        }
    }
}
```

## 使用 Zookeeper 作为服务注册中心

# 认识 Zookeeper

**Zookeeper**

- A Distributed Coordination Service for Distributed Applications
- http://zookeeper.apache.org

**设计目标** 用起来跟文件系统一致

- 简单
- 多副本 读多写少
- 有序
- 快

# 使用 Zookeeper 作为注册中心

**Spring Cloud Zookeeper**

- spring-cloud-starter-zookeeper-discovery
- Apache Curator Spring cloud使用的zk客户端

**简单配置** 生产上使用三副本或者五副本

- `spring.cloud.zookeeper.connect-string=localhost:2181`
  application.properties

**提示**

- 注意 Zookeeper 的版本
  - 3.5.x 还是 Beta，但很多人在生产中使用它

# 使用 Zookeeper 作为注册中心的问题

**两篇文章值得阅读**

- 《阿里巴巴为什么不用 Zookeeper 做服务发现》
- 《Eureka! Why You Shouldn’t Use ZooKeeper for Service Discovery》

**核心思想**

- 在实践中，注册中心不能因为自身的任何原因破坏服务之间本身的可连通性
- 注册中心需要 AP，而 Zookeeper 是 CP
  - CAP - 一致性、可用性、分区容忍性

每个服务尽可能在一个机房里面，否则可能zk
在脑裂的时候问题比较大

# 通过 Docker 启动 Zookeeper

**官方指引**

- https://hub.docker.com/_/zookeeper

**获取镜像**

- `docker pull zookeeper:3.5`

**运行 Zookeeper 镜像**

- `docker run --name zookeeper -p 2181:2181 -d zookeeper:3.5`

“Talk is cheap, show me the code.”

*Chapter 12 / zk-waiter-service zk-customer-service*

## 使用 Consul 作为服务注册中心

推荐

"Consul is a distributed, highly available, and data center aware solution to connect and configure applications across dynamic, distributed infrastructure."

– *https://github.com/hashicorp/consul*

# 认识 HashiCorp Consul

**Consul**

- https://www.consul.io

**关键特性**

- 服务发现
- 健康检查
- KV 存储
- 多数据中心支持
- 安全的服务间通信

# 使用 Consul 提供服务发现能力

**Consul 的能力**

- Service registry, integrated health checks, and DNS and HTTP interfaces enable any service to discover and be discovered by other services

**好用的功能**

- HTTP API 服务注册与发现，健康检查也是通过HTTP
- DNS（xxx.service.consul）在这里做一些些均衡的事情，比如waiter-service.service.consul在本地解析出目标,这样在不能改动的基础设施可以通过这样域名访问，节点有变化时候域名也会感知到
- 与 Nginx 联动，比如 ngx_http_consul_backend_module

通过这module实现，后盾服务节点变化，Nginx upstream可以感知到

# 使用 Consul 作为注册中心

**Spring Cloud Consul**

- spring-cloud-starter-consul-discovery

**简单配置** server.port=0 是要启动的waiter-service这个服务的端口，0是随机

- `spring.cloud.consul.host=localhost`
- `spring.cloud.consul.port=8500`
- `spring.cloud.consul.discovery.prefer-ip-address=true`

## 通过 Docker 启动 Consul

**官方指引**

- https://hub.docker.com/_/consul

**获取镜像**

- `docker pull consul`

**运行 Consul 镜像**

还有别的，比如land,wan的gossip的端口

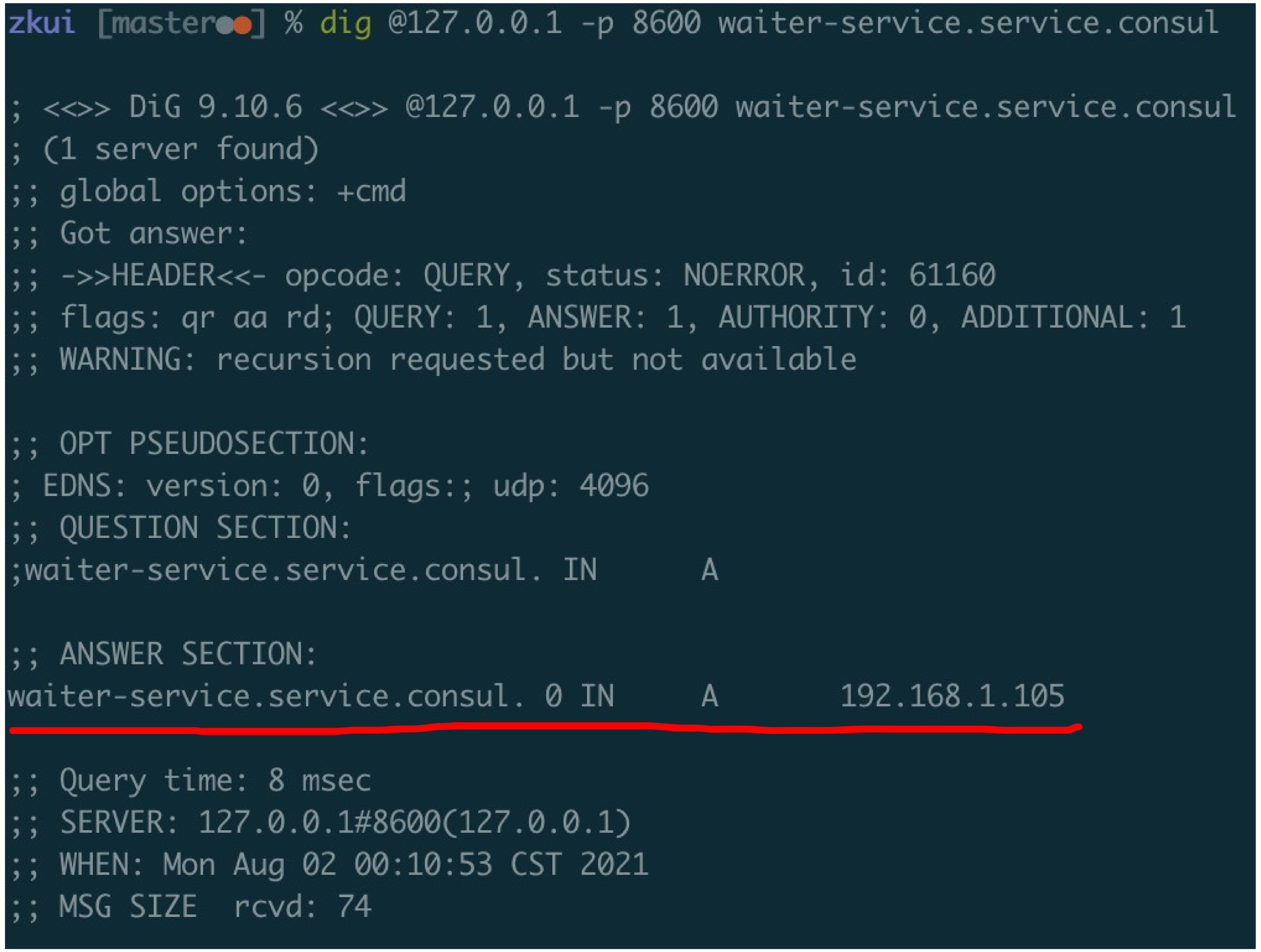
- `docker run --name consul -d -p 8500:8500 -p 8600:8600/udp consul`

8600绑定UDP,通过DNS解析服务

“Talk is cheap, show me the code.”

*Chapter 12 / consul-waiter-service consul-customer-service*

域名解析:



```
zkui [master●●] % dig @127.0.0.1 -p 8600 waiter-service.service.consul

; <<>> DiG 9.10.6 <<>> @127.0.0.1 -p 8600 waiter-service.service.consul
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 61160
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;waiter-service.service.consul. IN      A

;; ANSWER SECTION:
waiter-service.service.consul. 0 IN     A       192.168.1.105

;; Query time: 8 msec
;; SERVER: 127.0.0.1#8600(127.0.0.1)
;; WHEN: Mon Aug 02 00:10:53 CST 2021
;; MSG SIZE  rcvd: 74
```

spring-cloud-alibaba提供的

## 使用 Nacos 作为服务注册中心

配置管理+服务管理

# 认识 Nacos

**Nacos**

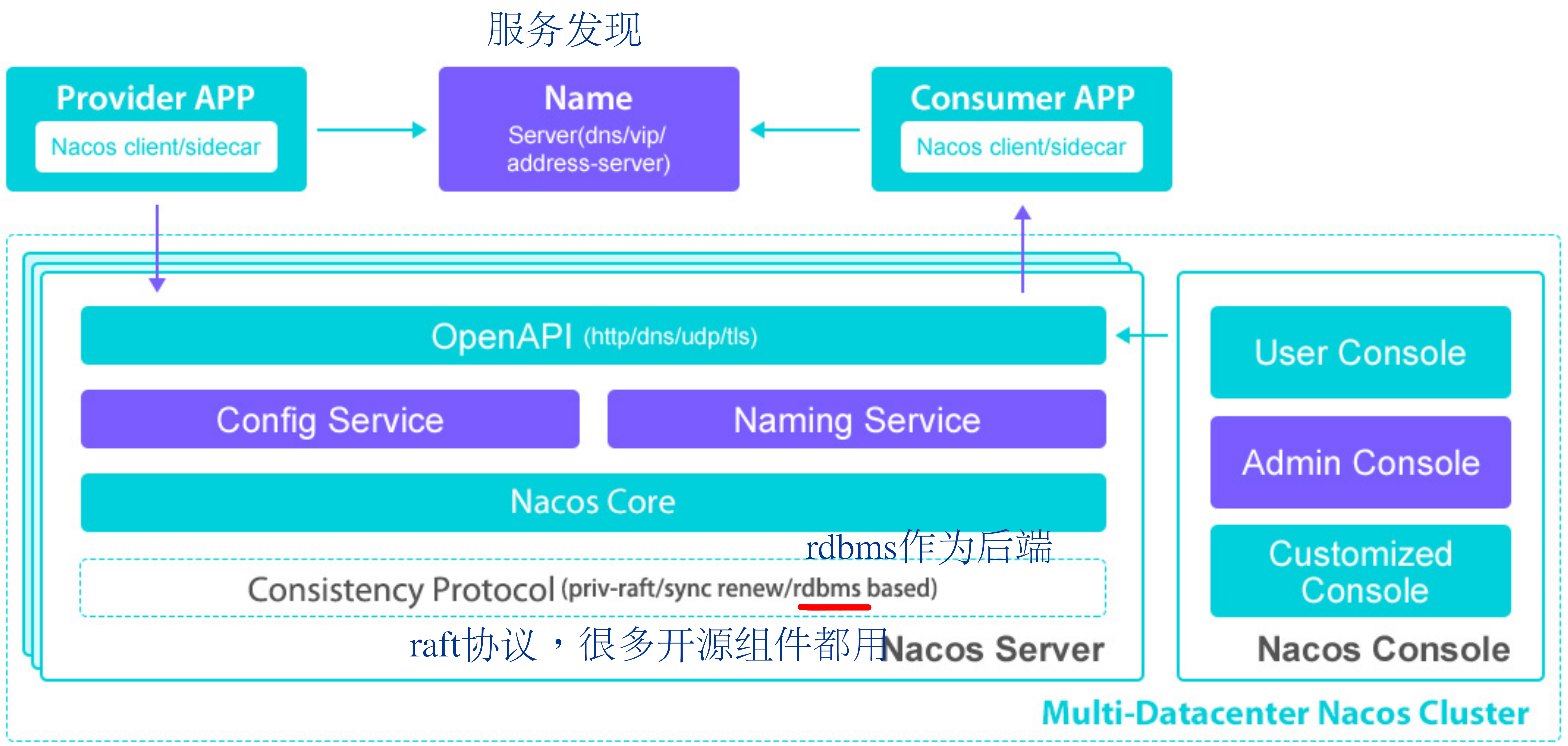
- 一个更易于构建云原生应用的动态服务发现、配置管理和服务管理平台。
- https://nacos.io/zh-cn/index.html 有完善的中文文档

**功能**

- 动态服务配置
- 服务发现和管理
- 动态 DNS 服务 不仅可以服务发现，还能做配置中心

程序猿DD的博客有相关文章

# 认识 Nacos


服务发现
Provider APP
Nacos client/sidecar
Name
Server(dns/vip/
address-server)
Consumer APP
Nacos client/sidecar
OpenAPI (http/dns/udp/tls)
Config Service
Naming Service
Nacos Core
rdbms作为后端
Consistency Protocol (priv-raft/sync renew/rdbms based)
raft协议，很多开源组件都用
Nacos Server
User Console
Admin Console
Customized Console
Nacos Console
Multi-Datacenter Nacos Cluster

# 使用 Nacos 作为注册中心

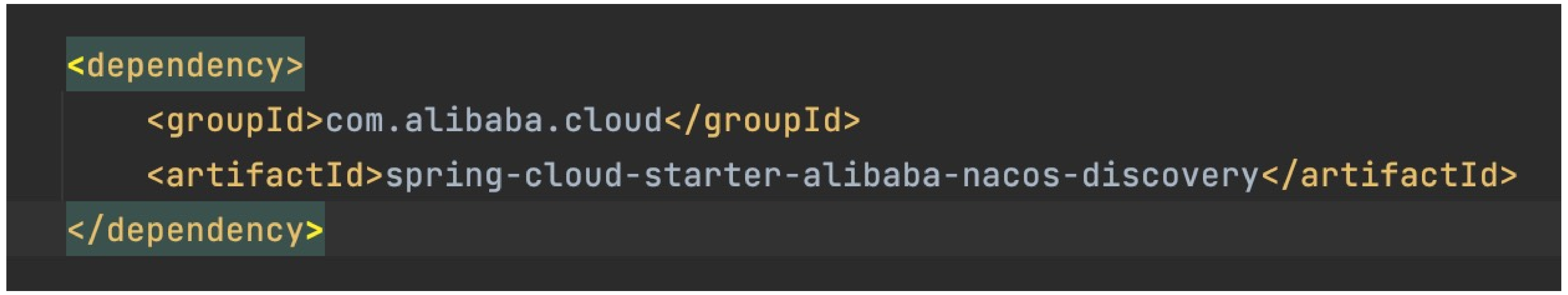
**Spring Cloud Alibaba**

- spring-cloud-alibaba-dependencies 引入这个BOM
- spring-cloud-starter-alibaba-nacos-discovery

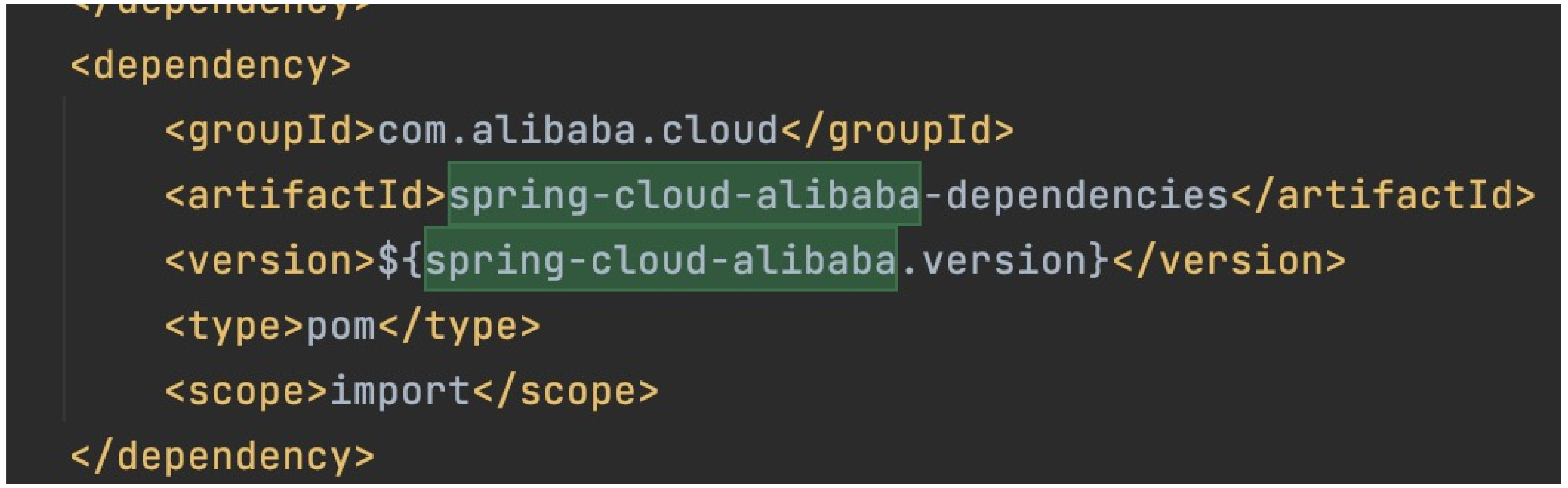
**简单配置**

- `spring.cloud.nacos.discovery.server-addr`

Nacos服务端口，默认8848



```
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
</dependency>
```



```
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-alibaba-dependencies</artifactId>
    <version>${spring-cloud-alibaba.version}</version>
    <type>pom</type>
    <scope>import</scope>
</dependency>
```

当Spring Boot是2.x版本时候，请引用：

```
<dependency>
      <groupId>com.alibaba.cloud</groupId>
      <artifactId>spring-cloud-alibaba-dependencies</artifactId>
      <version>${spring-cloud-alibaba.version}</version>
      <type>pom</type>
      <scope>import</scope>
   </dependency>
```

```
<spring-cloud-alibaba.version>2.1.2.RELEASE</spring-cloud-alibaba.version>
```

## 通过 Docker 启动 Nacos

**官方指引**

- https://hub.docker.com/r/nacos/nacos-server

**获取镜像**

- `docker pull nacos/nacos-server`

**运行 Nacos 镜像**

- `docker run --name nacos -d -p 8848:8848 -e MODE=standalone nacos/nacos-server`
- 用户名密码为 nacos

本地访问：http://localhost:8848/nacos/

“Talk is cheap, show me the code.”

*Chapter 12 / nacos-waiter-service nacos-customer-service*

## 如何定制自己的 DiscoveryClient

# 已经接触过的 Spring Cloud 类

**DiscoveryClient**

- `EurekaDiscoveryClient`
- `ZookeeperDiscoveryClient`
- `ConsulDiscoveryClient`
- `NacosDiscoveryClient`

**LoadBalancerClient**

- `RibbonLoadBalancerClient`

# 实现自己的 DiscoveryClient

**需要做的：**

- 返回该 DiscoveryClient 能提供的服务名列表
- 返回指定服务对应的 ServiceInstance 列表
- 返回 DiscoveryClient 的顺序 一般不更改顺序
- 返回 HealthIndicator 里显示的描述

接下来要实现LoadBalanceClient,由于Ribbon提供的很好用了，直接实现RibbonClient的ServerList

# 实现自己的 RibbonClient 支持

**需要做的：**

- 实现自己的 ServerList<T extends Server>
  可以使用Ribbon提供的抽象类
  - Ribbon 提供了 AbstractServerList<T extends Server>
- 提供一个配置类，声明 ServerList Bean 实例

“Talk is cheap, show me the code.”

*Chapter 12 / fixed-discovery-client-demo*

## SpringBucks 实战项目进度小结

# 本章小结

**各种服务注册中心**

- Eureka、Zookeeper、Consul、Nacos

**如何在服务间进行负载均衡** 底层都是Ribbon做负载均衡

- Ribbon、OpenFeign Feign做声明式调用会清爽不少

**Spring Cloud 的服务注册与发现机制**

- `ServiceRegistry`、`DiscoveryClient`
  注册 发现
- `LoadBalancerClient` 负载均衡

## SpringBucks 进度小结

**waiter-service**

- 使用多种服务注册中心注册服务
  - Eureka、Zookeeper、Consul、Nacos

**customer-service**

- 通过多种服务注册中心发现 waiter-service
  - Eureka、Zookeeper、Consul、Nacos