【Spring Cloud Eureka】

1.Spring Cloud Eureka 简介

注册发现中心

Eureka 来源于古希腊词汇，意为“发现了”。在软件领域， Eureka 是 **Netflix** 在线影片公司开源的一个**服务注册与发现的组件**，和其他 Netflix 公司的服务组件（例如负载均衡、熔断器、网关等） 一起，被 Spring Cloud 社区整合为 Spring Cloud Netflix 模块。 Eureka 是 Netflix 贡献给 Spring Cloud 的一个框架！Netflix 给 Spring Cloud 贡献了很多框架，后面我们会学习到！

2.Spring Cloud Eureka 和 Zookeeper 的区别

2.1 什么是 CAP 原则（面试）

在分布式 微服务里面 CAP 定理

问：为什么 zookeeper 不适合做注册中心？

CAP 原则又称 CAP 定理，指的是在一个分布式系统中，

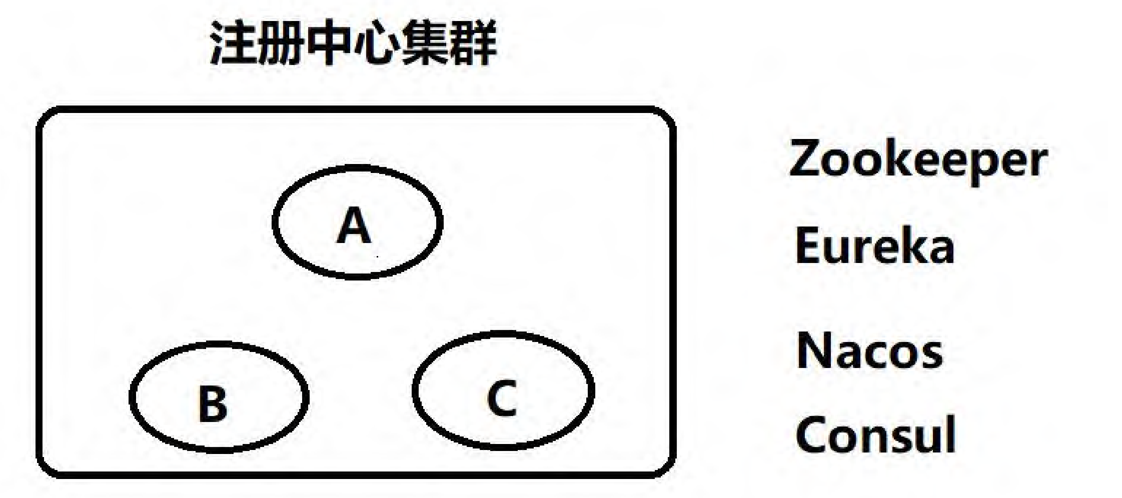
一致性（Consistency）

可用性（Availability）

分区容错性（Partition tolerance） （这个特性是不可避免的）

CAP 原则指的是，这三个要素最多只能同时实现两点，不可能三者兼顾。

2.2 分布式特征



C ：数据的一致性 （A,B,C 里面的数据是一致的）

Zk 注重数据的一致性。

Eureka 不是很注重数据的一致性!

A: 服务的可用性（若 zk 集群里面的 master 挂了怎么办）Paxos（多数派）

在 zk 里面，若主机挂了，则 zk 集群整体不对外提供服务了，需要选一个新的出来（120s 左右）才能继续对外提供服务!

Eureka 注重服务的可用性，当 Eureka 集群只有一台活着，它就能对外提供服务

P: 分区的容错性（在集群里面的机器，因为网络原因，机房的原因，可能导致数据不会里面同步），它在分布式必须需要实现的特性!

Zookeeper 注重数据的一致性，CP zk(注册中心，配置文件中心，协调中心）

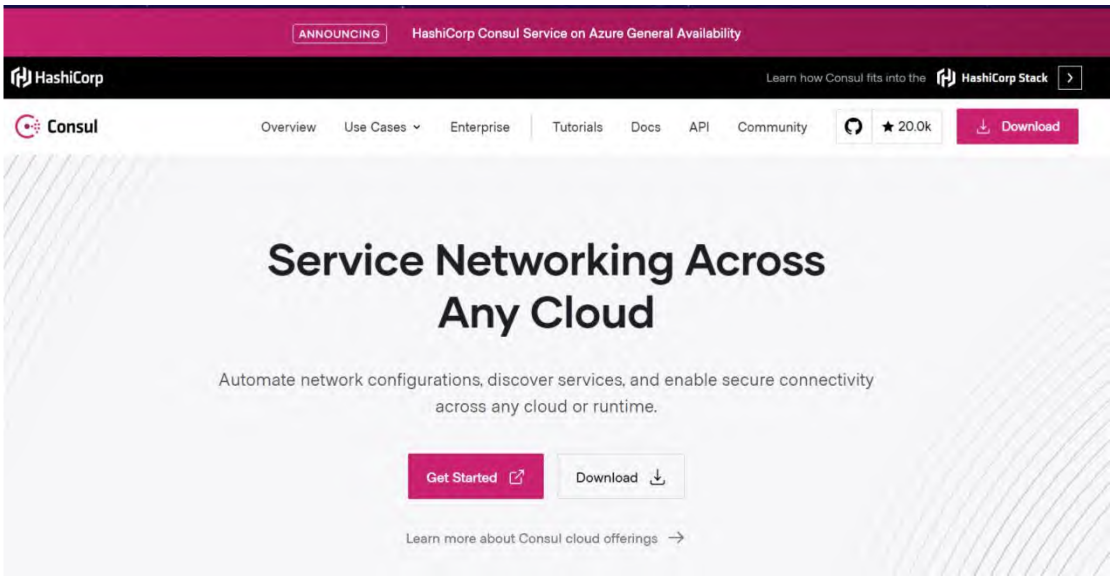
Eureka 注重服务的可用性 AP eureka （注册中心）

3.Spring Cloud 其他注册中心

Spring Cloud 还有别的注册中心 Consul ，阿里巴巴提供 Nacos 都能作为注册中心，我们的选择还是很多。

3.1 Consul

https://spring.io/projects/spring-cloud-consulConsul


ANNOUNCING HashiCorp Consul Service on Azure General Availability

Service Networking Across Any Cloud

Automate network configurations, discover services, and enable secure connectivity across any cloud or runtime.

3.2 Nacos

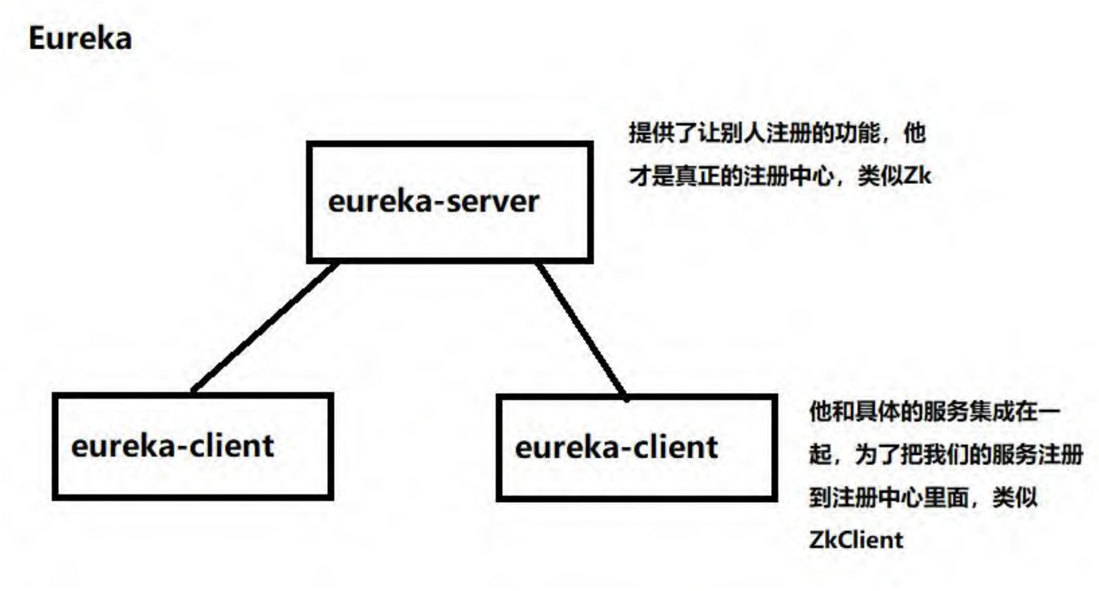
https://nacos.io/zh-cn/

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。

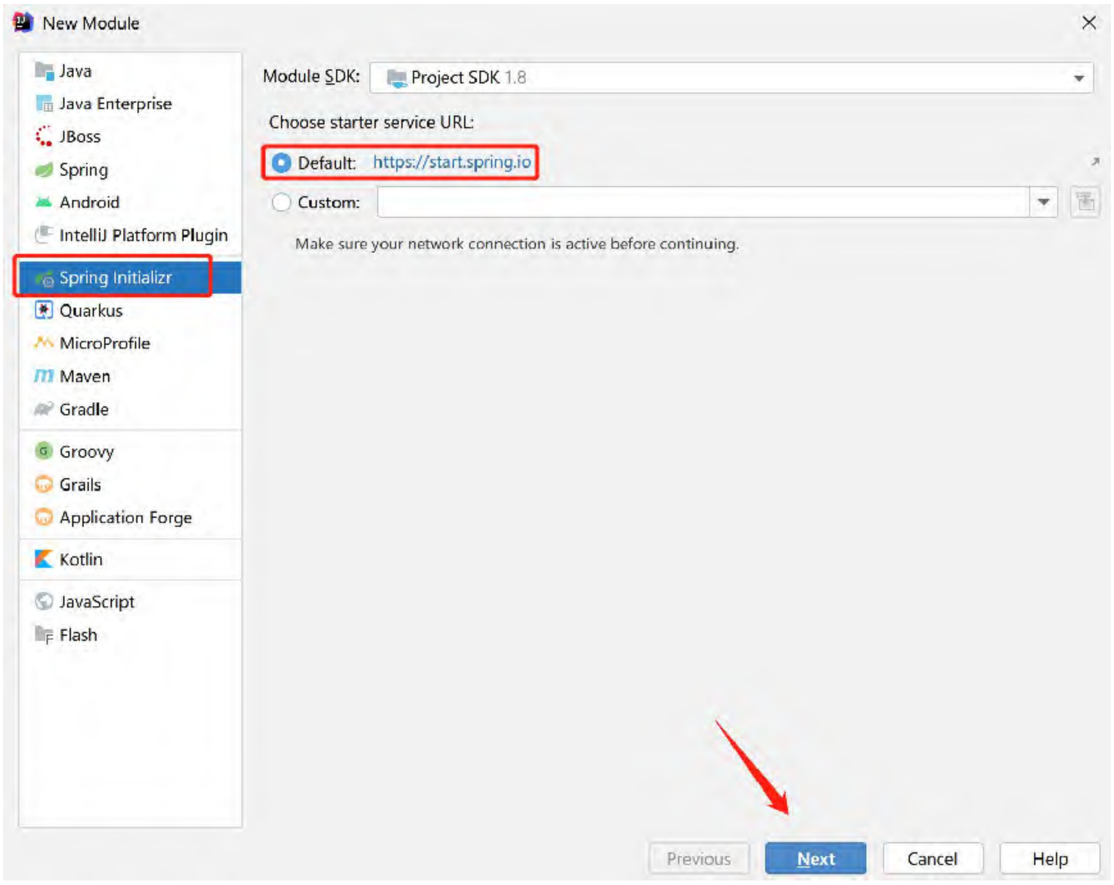
但是我们学习还是选择 Eureka ， 因为它的成熟度很高。面试时候问的也是它，不是别人！
eureka nacos

4.Spring Cloud Eureka 快速入门


Eureka
eureka-server
提供了让别人注册的功能，他才是真正的注册中心，类似Zk
eureka-client
eureka-client
他和具体的服务集成在一起，为了把我们的服务注册到注册中心里面，类似ZkClient

4.1 搭建 Eureka-server

4.1.1 创建项目



Module SDK: Project SDK 1.8

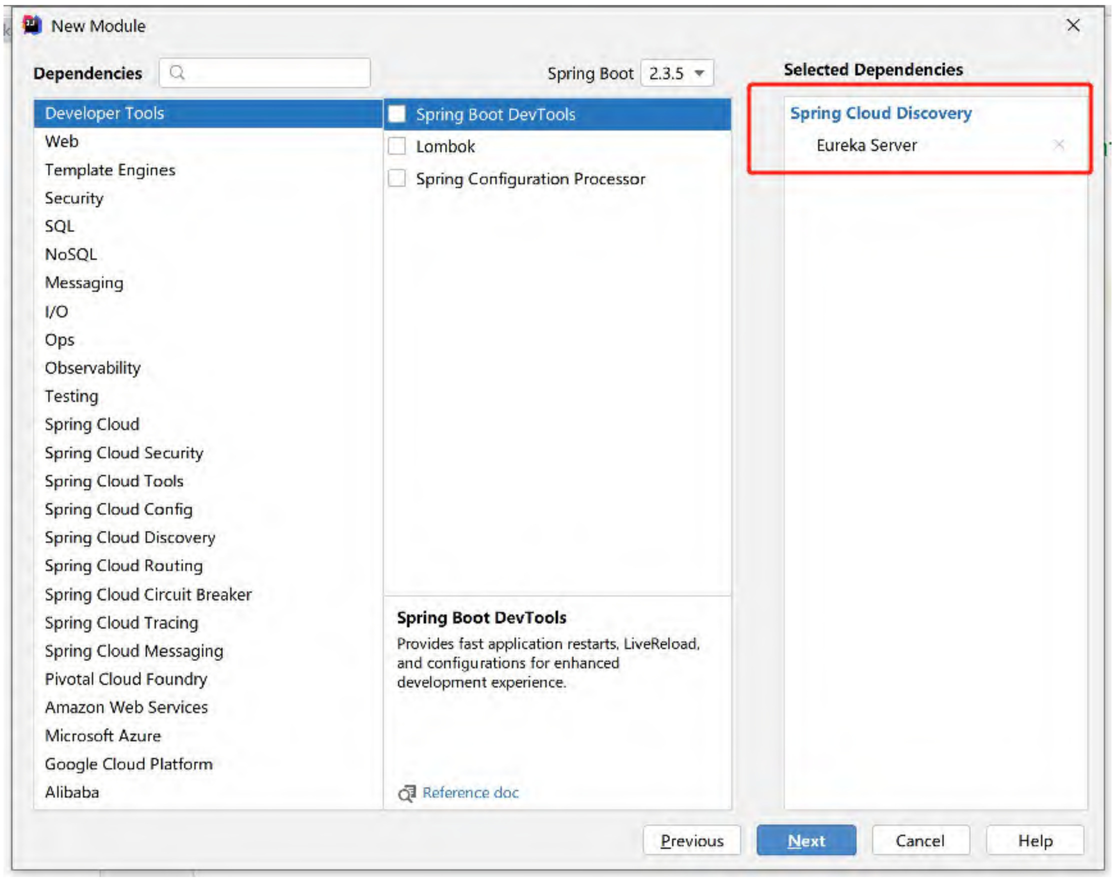
Choose starter service URL:

[✓ Default: https://start.spring.io]

Custom:

Make sure your network connection is active before continuing.

4.1.2 选择依赖


New Module

Dependencies　Spring Boot 2.3.5

Developer Tools
Web
Template Engines
Security
SQL
NoSQL
Messaging
I/O
Ops
Observability
Testing
Spring Cloud
Spring Cloud Security
Spring Cloud Tools
Spring Cloud Config
Spring Cloud Discovery
Spring Cloud Routing
Spring Cloud Circuit Breaker
Spring Cloud Tracing
Spring Cloud Messaging
Pivotal Cloud Foundry
Amazon Web Services
Microsoft Azure
Google Cloud Platform
Alibaba

- [ ] Spring Boot DevTools
- [ ] Lombok
- [ ] Spring Configuration Processor

Spring Boot DevTools
Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

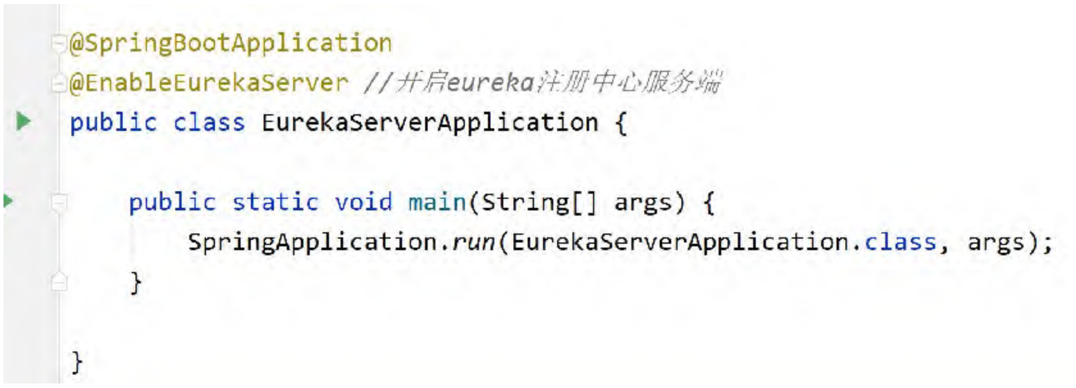
Reference doc

Selected Dependencies
Spring Cloud Discovery
Eureka Server

4.1.3 分析 pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <!-- 实质还是 springboot 项目-->
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.3.12.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.bjpowernode</groupId>
    <artifactId>eureka-server</artifactId>
    <version>1.0</version>
    <name>eureka-server</name>
    <description>Demo project for Eureka-Server</description>

    <properties>
        <java.version>1.8</java.version>
        <!--  这里控制了 springcloud 的版本-->
        <spring-cloud.version>Hoxton.SR12</spring-cloud.version>
    </properties>

    <dependencies>
        <!-- eureka 注册中心的服务端-->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
            <exclusions>
                <exclusion>
                    <groupId>org.junit.vintage</groupId>
                    <artifactId>junit-vintage-engine</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
    </dependencies>
    <!-- 依赖管理, cloud 的依赖-->
    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-dependencies</artifactId>
                <version>${spring-cloud.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

4.1.4 修改启动类



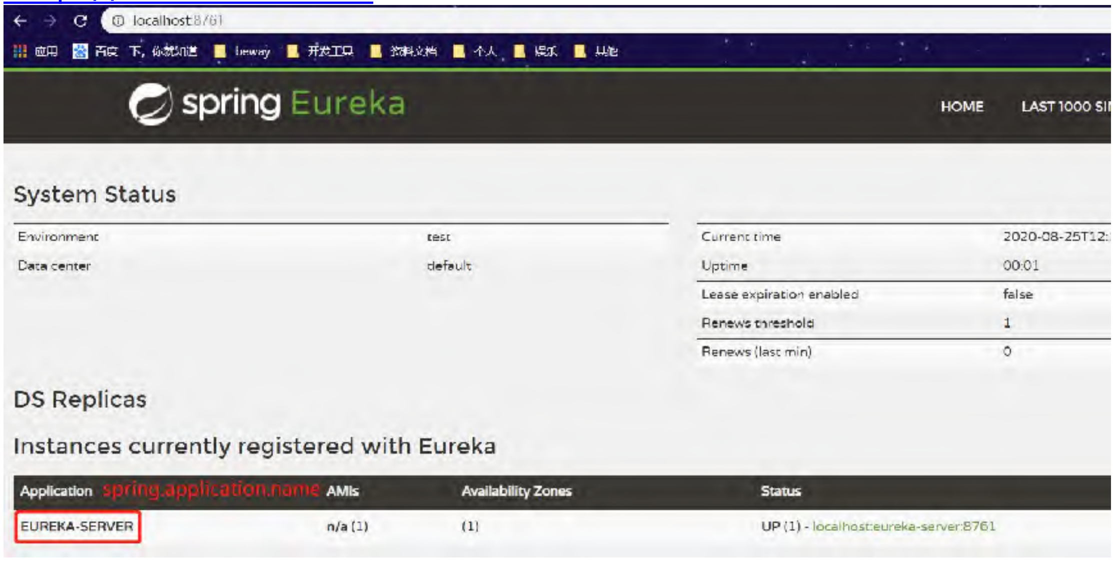
```java
@SpringBootApplication
@EnableEurekaServer //开启eureka注册中心服务端
public class EurekaServerApplication {

    public static void main(String[] args) {
        SpringApplication.run(EurekaServerApplication.class, args);
    }

}
```

4.1.5 修改配置文件

```yaml
server:
    port: 8761  #为什么是8761，其他端口就报错
spring:
    application:
        name: eureka-server #服务名称
```

4.1.6 访问测试

http://localhost:8761



localhost:8761

spring Eureka HOME LAST 1000 SI

System Status

Environment	test
Data center	default

Current time	2020-08-25T12:
Uptime	00:01
Lease expiration enabled	false
Renews threshold	1
Renews (last min)	0

DS Replicas

Instances currently registered with Eureka

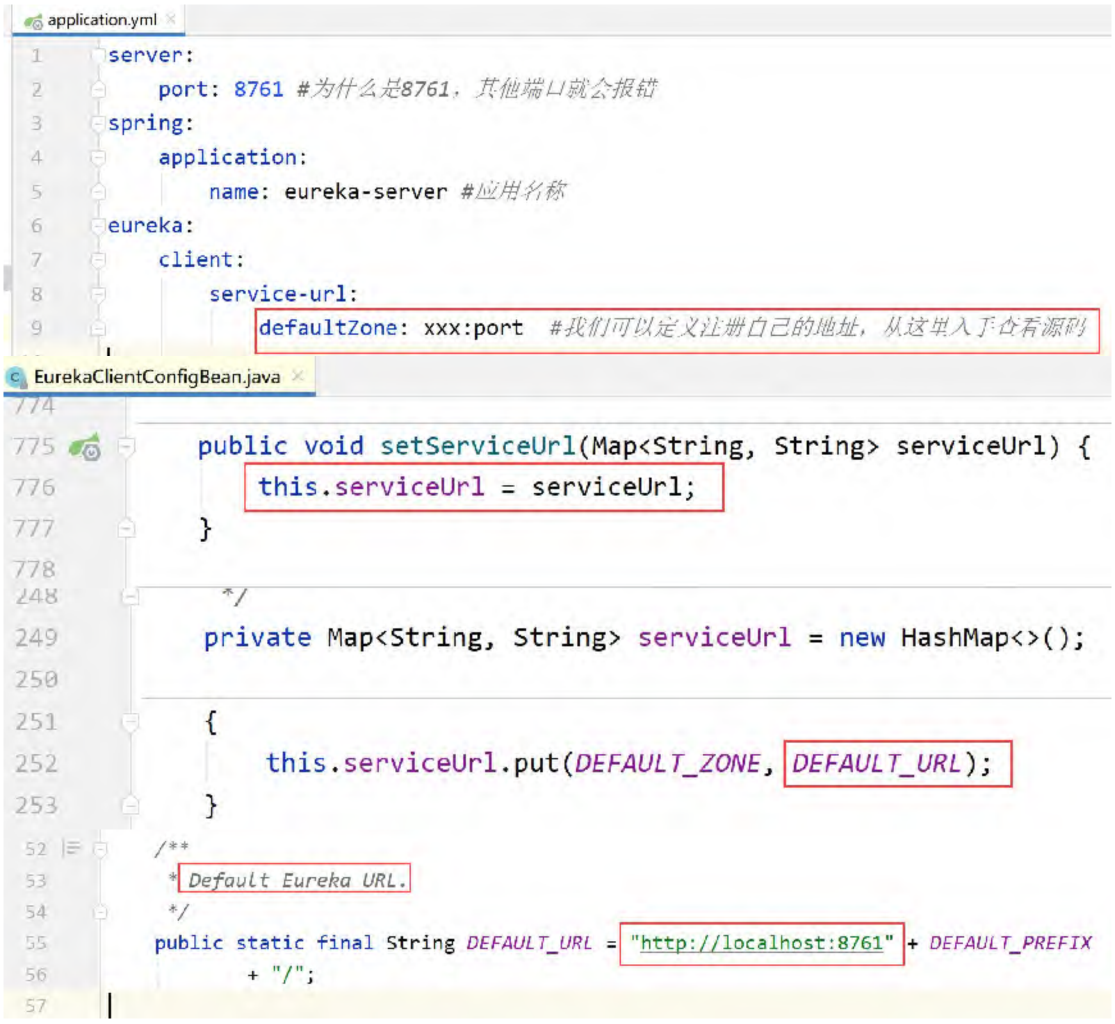
Application spring.application.name	AMIs	Availability Zones	Status
EUREKA-SERVER	n/a (1)	(1)	UP (1) - localhost:eureka-server:8761

4.1.7 分析端口 8761

Eureka-Server 不仅提供让别人注册的功能，它也能注册到别人里面，自己注册自己

所以，在启动项目时，默认会注册自己，我们也可以关掉这个功能。

那么往哪个地址注册自己呢？我们看一下源码



application.yml

```yaml
server:
    port: 8761 #为什么是8761，其他端口就会报错
spring:
    application:
        name: eureka-server #应用名称
eureka:
    client:
        service-url:
            defaultZone: xxx:port  #我们可以定义注册自己的地址，从这里入手查看源码
```

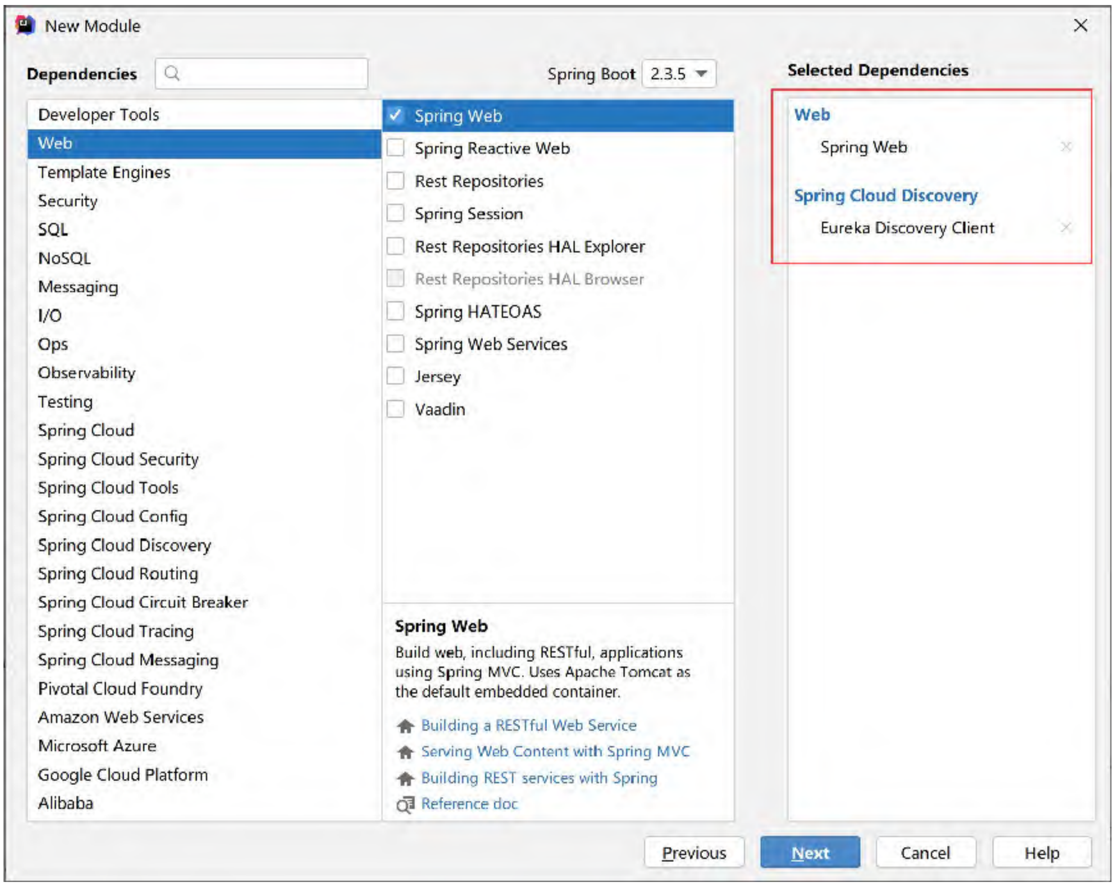
EurekaClientConfigBean.java

```java
    public void setServiceUrl(Map<String, String> serviceUrl) {
        this.serviceUrl = serviceUrl;
    }

     */
    private Map<String, String> serviceUrl = new HashMap<>();

    {
        this.serviceUrl.put(DEFAULT_ZONE, DEFAULT_URL);
    }
```

```java
    /**
     * Default Eureka URL.
     */
    public static final String DEFAULT_URL = "http://localhost:8761" + DEFAULT_PREFIX
            + "/";
```

4.2 搭建 Eureka-client

4.2.1 创建项目 client-a 选择 依赖


New Module

Dependencies | Spring Boot 2.3.5

Developer Tools
Web
Template Engines
Security
SQL
NoSQL
Messaging
I/O
Ops
Observability
Testing
Spring Cloud
Spring Cloud Security
Spring Cloud Tools
Spring Cloud Config
Spring Cloud Discovery
Spring Cloud Routing
Spring Cloud Circuit Breaker
Spring Cloud Tracing
Spring Cloud Messaging
Pivotal Cloud Foundry
Amazon Web Services
Microsoft Azure
Google Cloud Platform
Alibaba

- [x] Spring Web
- [ ] Spring Reactive Web
- [ ] Rest Repositories
- [ ] Spring Session
- [ ] Rest Repositories HAL Explorer
- [ ] Rest Repositories HAL Browser
- [ ] Spring HATEOAS
- [ ] Spring Web Services
- [ ] Jersey
- [ ] Vaadin

Spring Web
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
Building a RESTful Web Service
Serving Web Content with Spring MVC
Building REST services with Spring
Reference doc

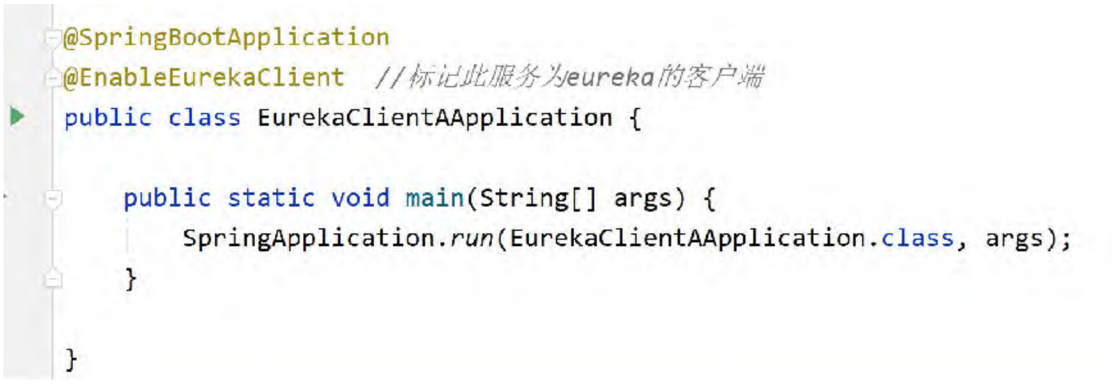
Selected Dependencies
Web
Spring Web
Spring Cloud Discovery
Eureka Discovery Client

4.2.2 分析 pom.xml

```xml
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

```xml
<!--  eureka-client eureka的客户端的依赖-->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

4.2.3 修改启动类



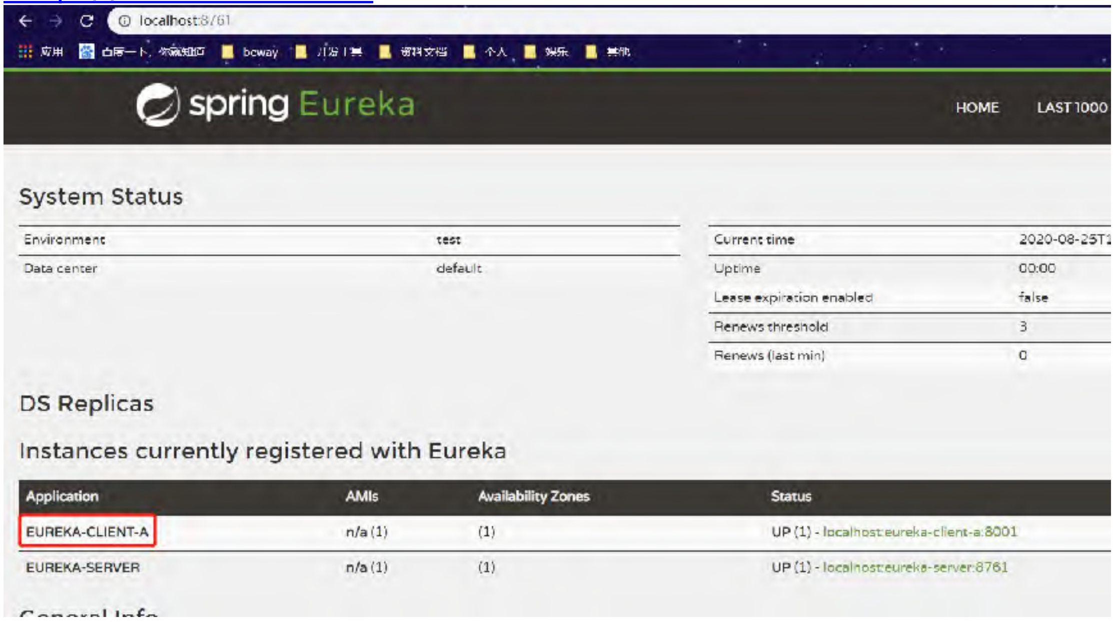
```java
@SpringBootApplication
@EnableEurekaClient   //标记此服务为eureka的客户端
public class EurekaClientAApplication {

    public static void main(String[] args) {
        SpringApplication.run(EurekaClientAApplication.class, args);
    }

}
```

4.2.4 修改配置文件

```yaml
server:
    port: 8001
spring:
    application:
        name: eureka-client-a
eureka:
    client:
        service-url:    #eureka 服务端和客户端的交互地址
            defaultZone: http://localhost:8761/eureka/
```

4.2.5 访问测试

http://localhost:8761



System Status

Environment	test
Data center	default

Current time	2020-08-25T1
Uptime	00:00
Lease expiration enabled	false
Renews threshold	3
Renews (last min)	0

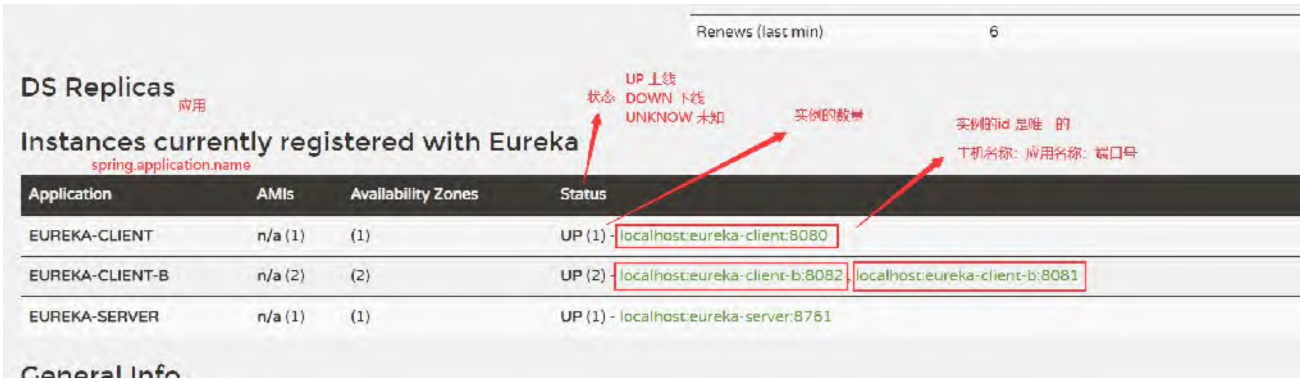
DS Replicas

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
EUREKA-CLIENT-A	n/a (1)	(1)	UP (1) - localhost:eureka-client-a:8001
EUREKA-SERVER	n/a (1)	(1)	UP (1) - localhost:eureka-server:8761

4.2.6 再创建项目 client-b

如 client-a 一样，这里就不贴多余截图了，**注意端口和服务名以及启动类上的注解**，在测试查看是否注册上去，在 eureka 里面是通过 spring.application.name 来区分服务的



Renews (last min) 6

DS Replicas

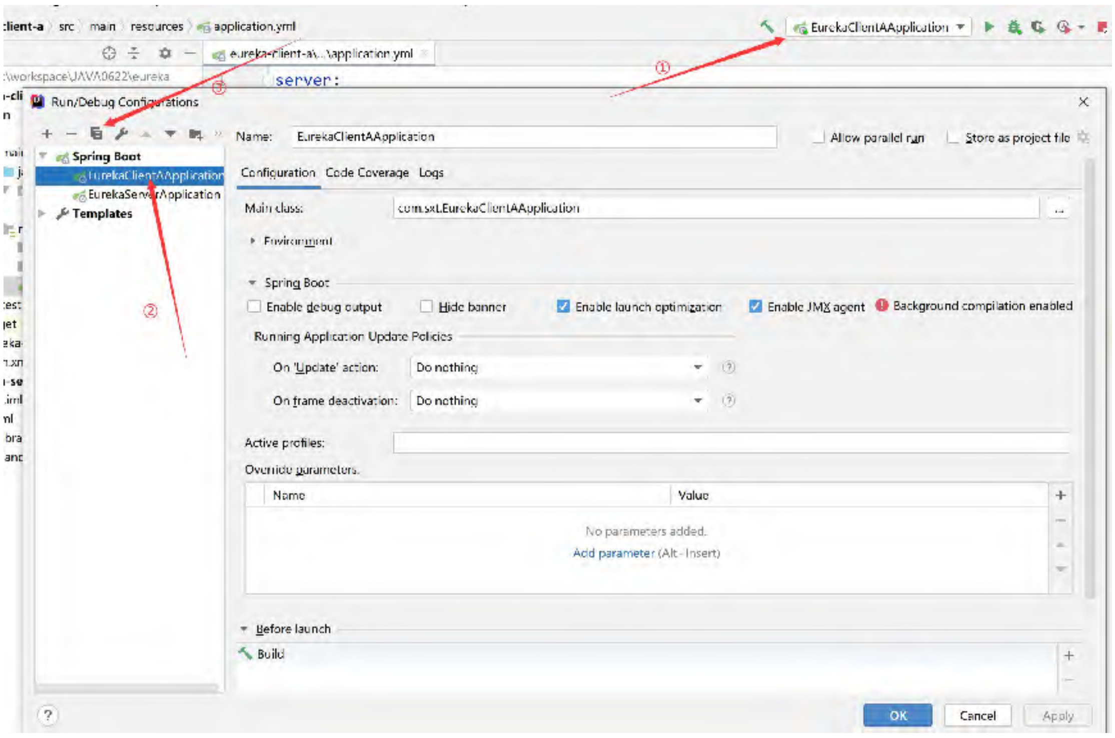
Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
EUREKA-CLIENT	n/a (1)	(1)	UP (1) - localhost:eureka-client:8080
EUREKA-CLIENT-B	n/a (2)	(2)	UP (2) - localhost:eureka-client-b:8082 , localhost:eureka-client-b:8081
EUREKA-SERVER	n/a (1)	(1)	UP (1) - localhost:eureka-server:8761

General Info

4.3 同一个服务（客户端）启动多台

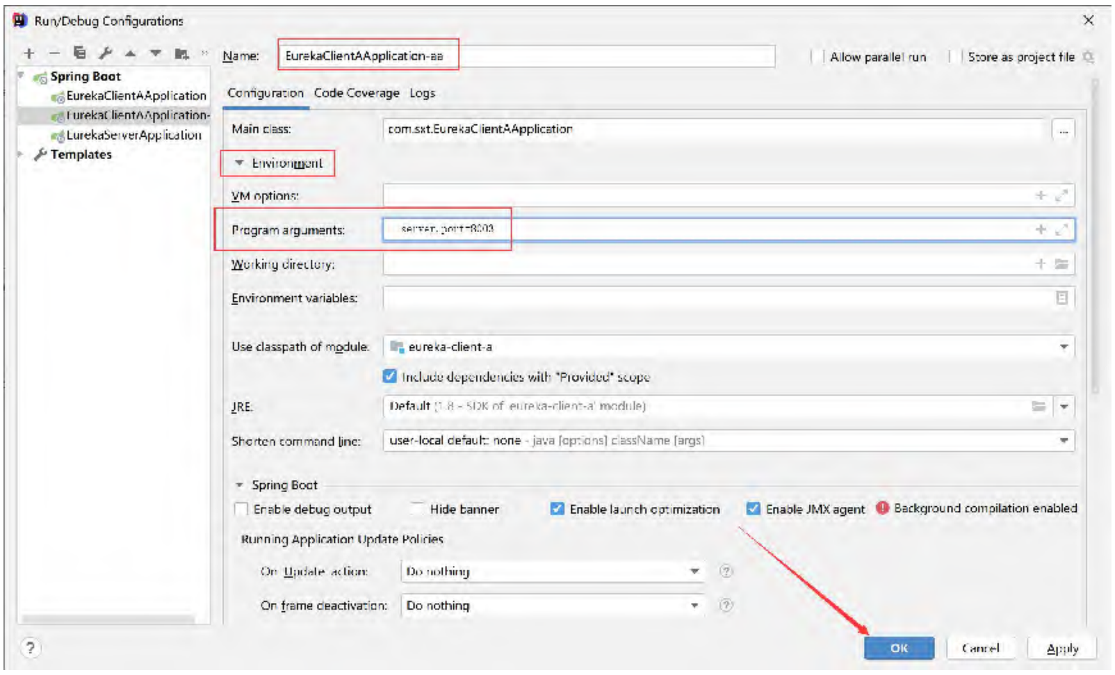
4.3.1 IDEA 启动多台服务操作



Name: EurekaClientAApplication
Allow parallel run
Store as project file
Main class: com.sxt.EurekaClientAApplication
Environment
Spring Boot
Enable debug output
Hide banner
✓ Enable launch optimization
✓ Enable JMX agent
Background compilation enabled
Running Application Update Policies
On 'Update' action: Do nothing
On frame deactivation: Do nothing
Active profiles:
Override parameters:

Name	Value
No parameters added. Add parameter (Alt+Insert)	

Before launch
Build



Run/Debug Configurations

- Spring Boot
  - EurekaClientAApplication
  - EurekaClientAApplication-
  - EurekaServerApplication
- Templates

Name: EurekaClientAApplication-aa　　Allow parallel run　　Store as project file

Configuration　Code Coverage　Logs

Main class: com.sxt.EurekaClientAApplication

Environment

VM options:

Program arguments: --server.port=8003

Working directory:

Environment variables:

Use classpath of module: eureka-client-a

[✓] Include dependencies with "Provided" scope

JRE: Default (1.8 - SDK of 'eureka-client-a' module)

Shorten command line: user-local default: none - java [options] className [args]

Spring Boot

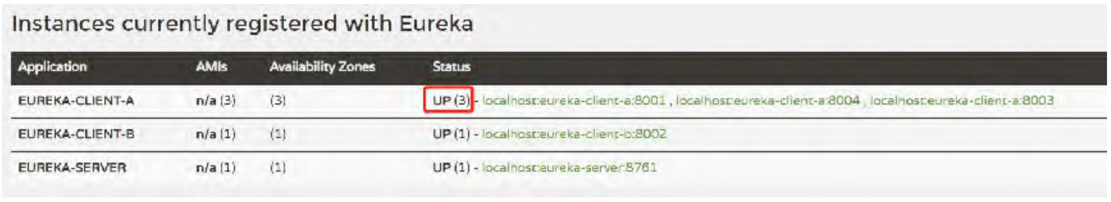
[ ] Enable debug output　[ ] Hide banner　[✓] Enable launch optimization　[✓] Enable JMX agent　Background compilation enabled

Running Application Update Policies

On 'Update' action: Do nothing

On frame deactivation: Do nothing

4.3.2 访问查看



Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
EUREKA-CLIENT-A	n/a (3)	(3)	UP (3) - localhost:eureka-client-a:8001 , localhost:eureka-client-a:8004 , localhost:eureka-client-a:8003
EUREKA-CLIENT-B	n/a (1)	(1)	UP (1) - localhost:eureka-client-b:8002
EUREKA-SERVER	n/a (1)	(1)	UP (1) - localhost:eureka-server:8761

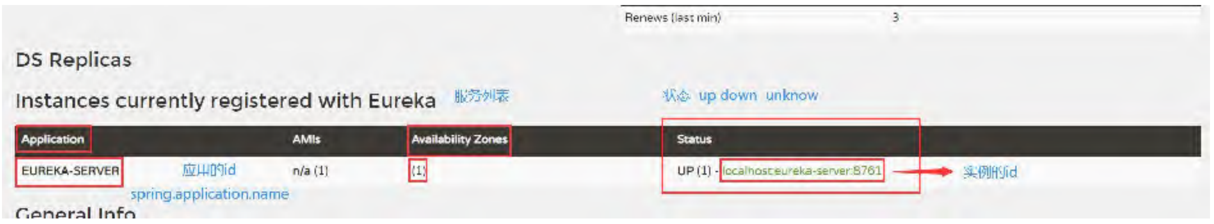
4.4 注册中心的状态认识

UP：服务是上线的，括号里面是具体服务实例的个数，提供服务的最小单元

DOWN：服务是下线的

UN_KONW：服务的状态未知

4.4.1 服务的实例名称



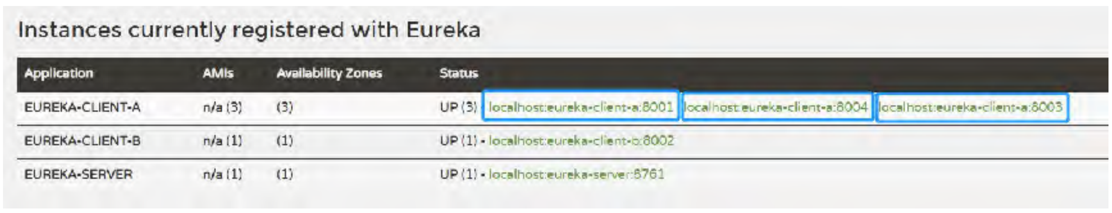
Renews (last min)　3

DS Replicas

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
EUREKA-SERVER	n/a (1)	(1)	UP (1) - localhost:eureka-server:8761

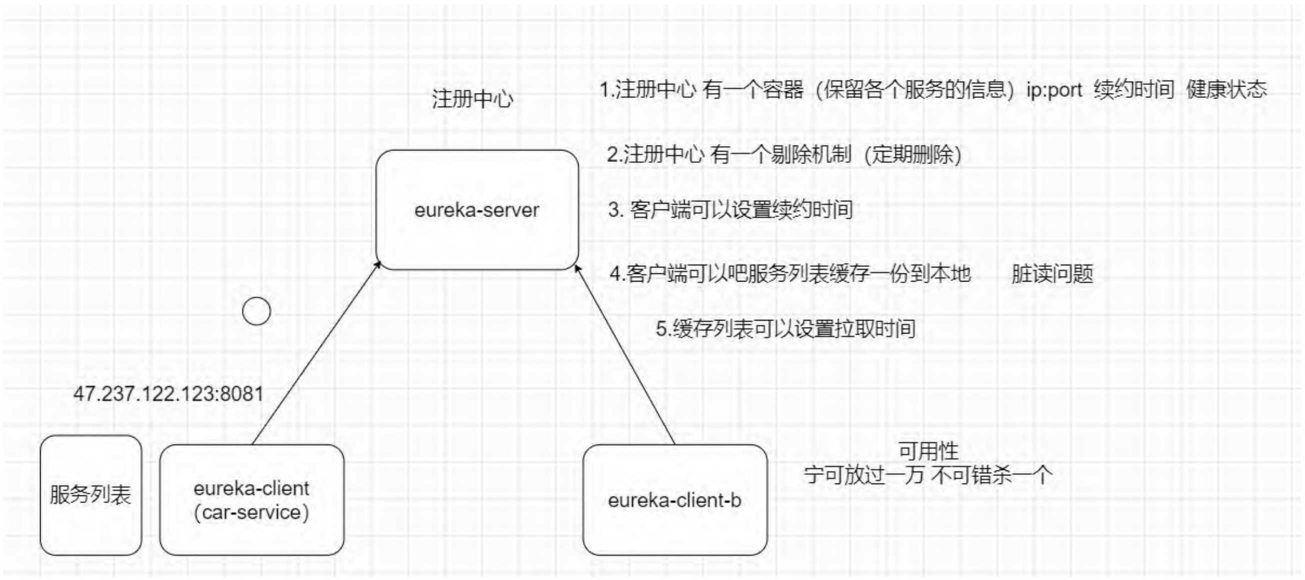
General Info



Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
EUREKA-CLIENT-A	n/a (3)	(3)	UP (3) - localhost:eureka-client-a:8001 , localhost:eureka-client-a:8004 , localhost:eureka-client-a:8003
EUREKA-CLIENT-B	n/a (1)	(1)	UP (1) - localhost:eureka-client-b:8002
EUREKA-SERVER	n/a (1)	(1)	UP (1) - localhost:eureka-server:8761

4.5 常用配置文件设置


注册中心
1.注册中心 有一个容器（保留各个服务的信息）ip:port 续约时间 健康状态
2.注册中心 有一个剔除机制（定期删除）
eureka-server
3. 客户端可以设置续约时间
4.客户端可以吧服务列表缓存一份到本地　　脏读问题
5.缓存列表可以设置拉取时间
47.237.122.123:8081
服务列表
eureka-client
(car-service)
eureka-client-b
可用性
宁可放过一万 不可错杀一个

4.5.1 server 中常用的配置

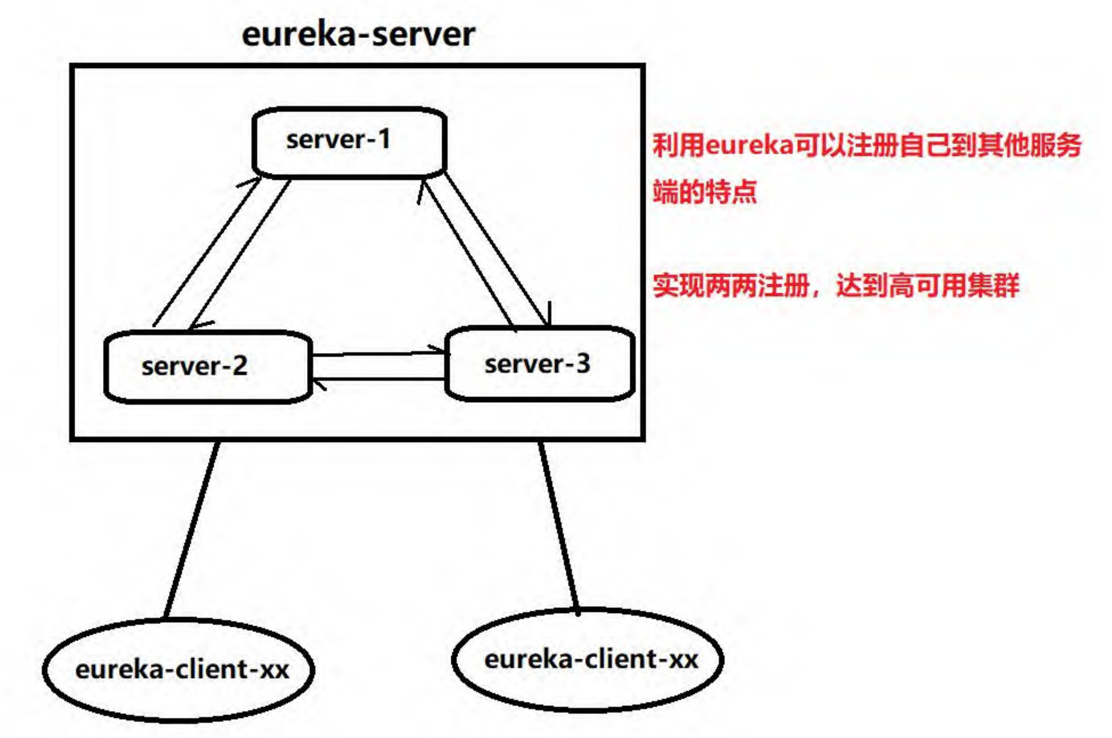
```
server:
  port: 8761
spring:
  application:
    name: eureka-server
eureka:
  client:
    service-url:   #eureka 服务端和客户端的交互地址,集群用,隔开
      defaultZone: http://localhost:8761/eureka
    fetch-registry: true   #是否拉取服务列表
    register-with-eureka: true    #是否注册自己（单机 eureka 一般关闭注册自己,集群注意打开）
  server:
    eviction-interval-timer-in-ms: 30000    #清除无效节点的频率(毫秒)--定期删除
    enable-self-preservation: true     #server 的自我保护机制，避免因为网络原因造成误剔除,生产环境建议打开
    renewal-percent-threshold: 0.85    #85%，如果在一个机房的client 端，15 分钟内有85%的client 没有续约，那么则可能是
```

```
网络原因，认为服务实例没有问题，不会剔除他们，宁可放过一万，不可错杀一个，确保高可用
  instance:
    hostname: localhost # 服务主机名称
    instance-id: ${eureka.instance.hostname}:${spring.application.name}:${server.port}  # 实例id
    prefer-ip-address: true  # 服务列表以 ip 的形式展示
    lease-renewal-interval-in-seconds: 10  # 表示eureka client 发送心跳给server 端的频率
    lease-expiration-duration-in-seconds: 20  #表示eureka server 至上一次收到client 的心跳之后，等待下一次心跳的超时时
间，在这个时间内若没收到下一次心跳，则将移除该实例
```

4.5.2 client 中常用的配置

```
server:
  port: 8080
spring:
  application:
    name: eureka-client
eureka:
  client:
    service-url:    #eureka 服务端和客户端的交互地址,集群用,隔开
      defaultZone: http://localhost:8761/eureka
    register-with-eureka: true  #注册自己
    fetch-registry: true    #拉取服务列表
    registry-fetch-interval-seconds: 5 # 表示eureka-client 间隔多久去拉取服务注册信息
  instance:
    hostname: localhost # 服务主机名称
    instance-id: ${eureka.instance.hostname}:${spring.application.name}:${server.port}  # 实例id
    prefer-ip-address: true  # 服务列表以 ip 的形式展示
    lease-renewal-interval-in-seconds: 10  # 表示eureka client 发送心跳给server 端的频率
    lease-expiration-duration-in-seconds: 20  #表示eureka server 至上一次收到client 的心跳之后，等待
下一次心跳的超时时间，在这个时间内若没收到下一次心跳，则将移除该实例
```

5.构建高可用的 Eureka-Server 集群



5.1 对刚才的 eureka-server 修改配置文件

5.1.1 server-1

```
server:
    port: 8761  #为什么是 8761，其他端口就报错
spring:
    application:
        name: eureka-server #服务名称
eureka:
    client:
        fetch-registry: true   #是否拉取服务列表
        register-with-eureka: true     #是否注册自己（集群需要注册自己和拉取服务）
        service-url:
            defaultZone: http://localhost:8762/eureka/,http://localhost:8763/eureka/
    server:
        eviction-interval-timer-in-ms: 90000     #清除无效节点的评率(毫秒)
    instance:
        lease-expiration-duration-in-seconds: 90     #server 在等待下一个客户端发送的心跳
时间，若在指定时间不能收到客户端心跳，则剔除此实例并且禁止流量
```

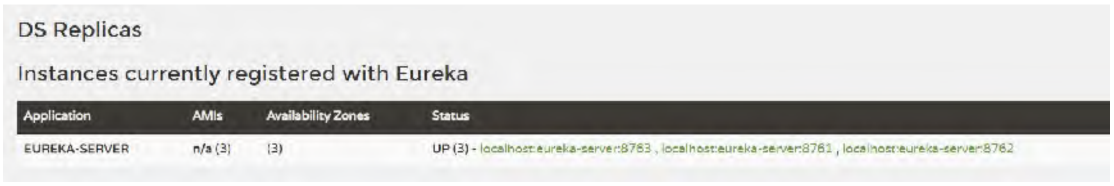
5.1.2 server-2

```
server:
    port: 8762
spring:
    application:
        name: eureka-server #服务名称
eureka:
    client:
        fetch-registry: true   #是否拉取服务列表
        register-with-eureka: true     #是否注册自己（集群需要注册自己和拉取服务）
        service-url:
            defaultZone: http://localhost:8761/eureka/,http://localhost:8763/eureka/
    server:
        eviction-interval-timer-in-ms: 90000     #清除无效节点的评率(毫秒)
    instance:
        lease-expiration-duration-in-seconds: 90     #server 在等待下一个客户端发送的心跳
时间，若在指定时间不能收到客户端心跳，则剔除此实例并且禁止流量
```

5.1.3 server-3

```
server:
    port: 8763
spring:
    application:
        name: eureka-server #服务名称
eureka:
    client:
        fetch-registry: true   #是否拉取服务列表
        register-with-eureka: true     #是否注册自己（集群需要注册自己和拉取服务）
        service-url:
            defaultZone: http://localhost:8761/eureka/,http://localhost:8762/eureka/
    server:
        eviction-interval-timer-in-ms: 90000     #清除无效节点的评率(毫秒)
    instance:
        lease-expiration-duration-in-seconds: 90     #server 在等待下一个客户端发送的心跳
时间，若在指定时间不能收到客户端心跳，则剔除此实例并且禁止流量
```

5.1.4 测试访问查看



DS Replicas

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
EUREKA-SERVER	n/a (3)	(3)	UP (3) - localhost:eureka-server:8763 , localhost:eureka-server:8761 , localhost:eureka-server:8762

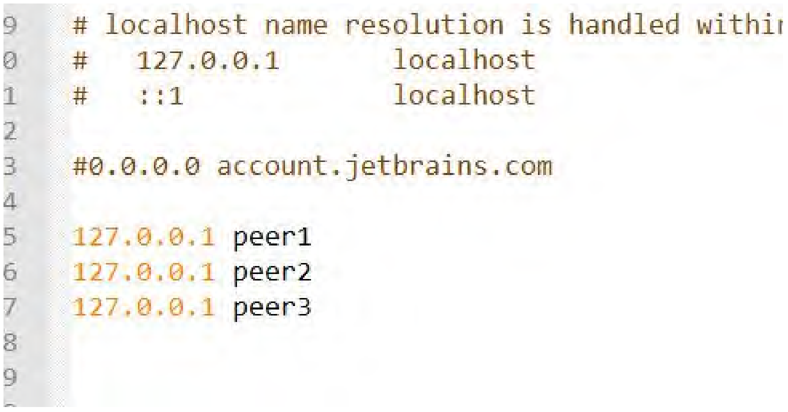
发现并没有出现集群信息，只是同一个服务 server 启动了多台 没有数据交互　不是真正意义上的集群

原因是因为：

http://localhost:8761/eureka/,http://localhost:8762/eureka/
这样写，eureka 认为只有一个机器，就是 localhost
所以这里面不能写成一样
修改 hosts 文件：   C:\Windows\System32\drivers\etc

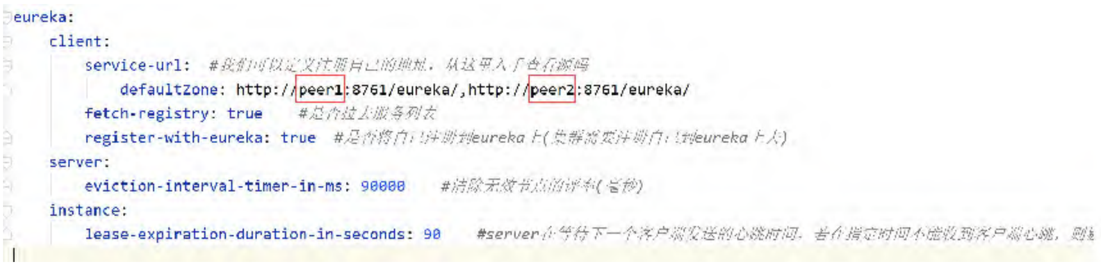
如果你修改了 hosts 文件 发现没有生效 记得在 cmd 里面刷新一下

ipconfig /flushdns



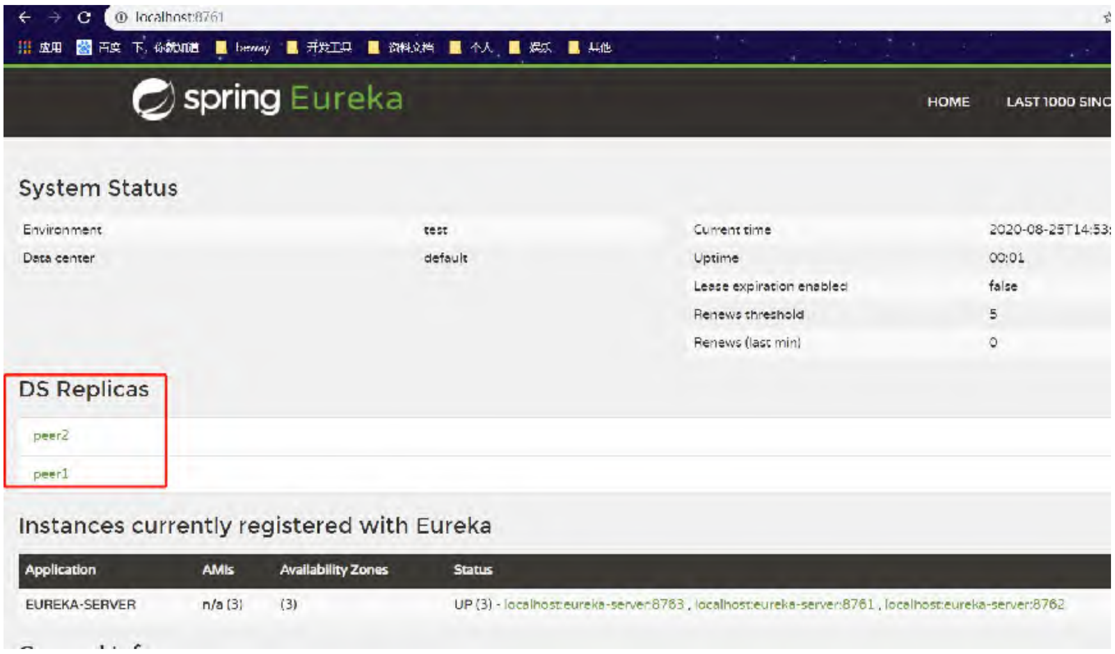
```
# localhost name resolution is handled withi
#    127.0.0.1       localhost
#    ::1             localhost

#0.0.0.0 account.jetbrains.com

127.0.0.1 peer1
127.0.0.1 peer2
127.0.0.1 peer3
```

5.1.5 重新修改配置文件



```
eureka:
    client:
        service-url:  #我们可以定义注册自己的地址，从这里入广告在源码
            defaultZone: http://peer1:8761/eureka/,http://peer2:8761/eureka/
        fetch-registry: true    #是否拉取服务列表
        register-with-eureka: true  #是否将自己注册到eureka上(集群需要注册自己到eureka上)
    server:
        eviction-interval-timer-in-ms: 90000    #清除无效节点的评率(毫秒)
    instance:
        lease-expiration-duration-in-seconds: 90    #server在等待下一个客户端发送的心跳时间，若在指定时间不能收到客户端心跳，则
```

5.1.6 测试查看集群信息



System Status

Environment	test	Current time	2020-08-25T14:53:
Data center	default	Uptime	00:01
		Lease expiration enabled	false
		Renews threshold	5
		Renews (last min)	0

DS Replicas

peer2

peer1

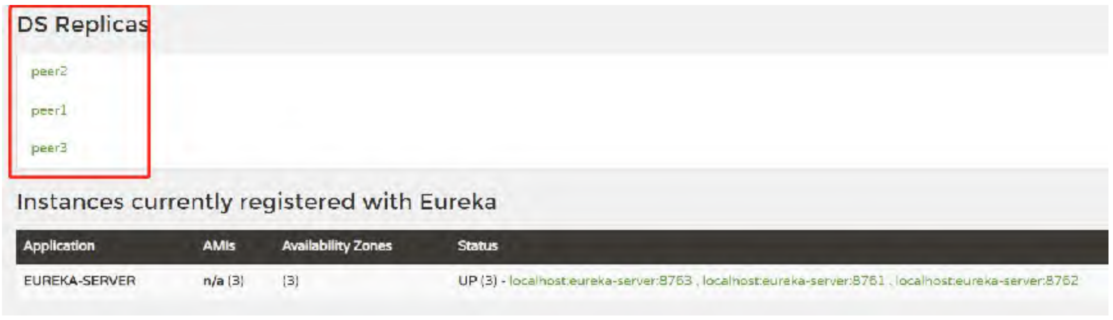
Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
EUREKA-SERVER	n/a (3)	(3)	UP (3) - localhost:eureka-server:8763 , localhost:eureka-server:8761 , localhost:eureka-server:8762

5.1.7 最终优化配置文件

```
server:
    port: 8761  #不需要修改 defaultZone 了,修改端口起三个服务
spring:
    application:
        name: eureka-server #服务名称
eureka:
    client:
        fetch-registry: true   #是否拉取服务列表
        register-with-eureka: true    #是否注册自己(集群需要注册自己和拉取服务)
        service-url:
            defaultZone: http://peer1:8761/eureka/,http://peer2:8762/eureka/,http://peer3:8763/eureka/
    server:
        eviction-interval-timer-in-ms: 90000   #清除无效节点的评率(毫秒)
    instance:
        lease-expiration-duration-in-seconds: 90    #server 在等待下一个客户端发送的心跳时间, 若在指定时间
不能收到客户端心跳, 则剔除此实例并且禁止流量
```

5.1.8 最终的集群信息



DS Replicas

peer2

peer1

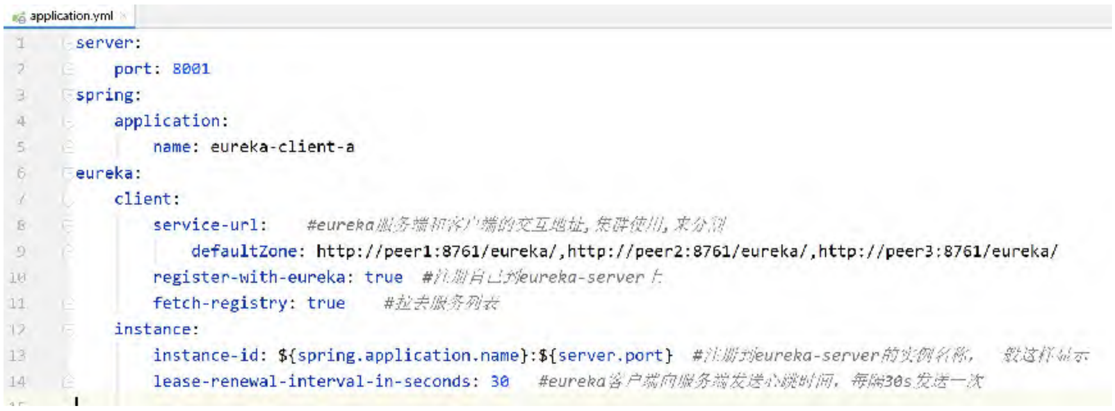
peer3

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
EUREKA-SERVER	n/a (3)	(3)	UP (3) - localhost:eureka-server:8763 , localhost:eureka-server:8761 , localhost:eureka-server:8762

5.2 集群的使用

5.2.1 改造 eureka-client-a 的配置文件

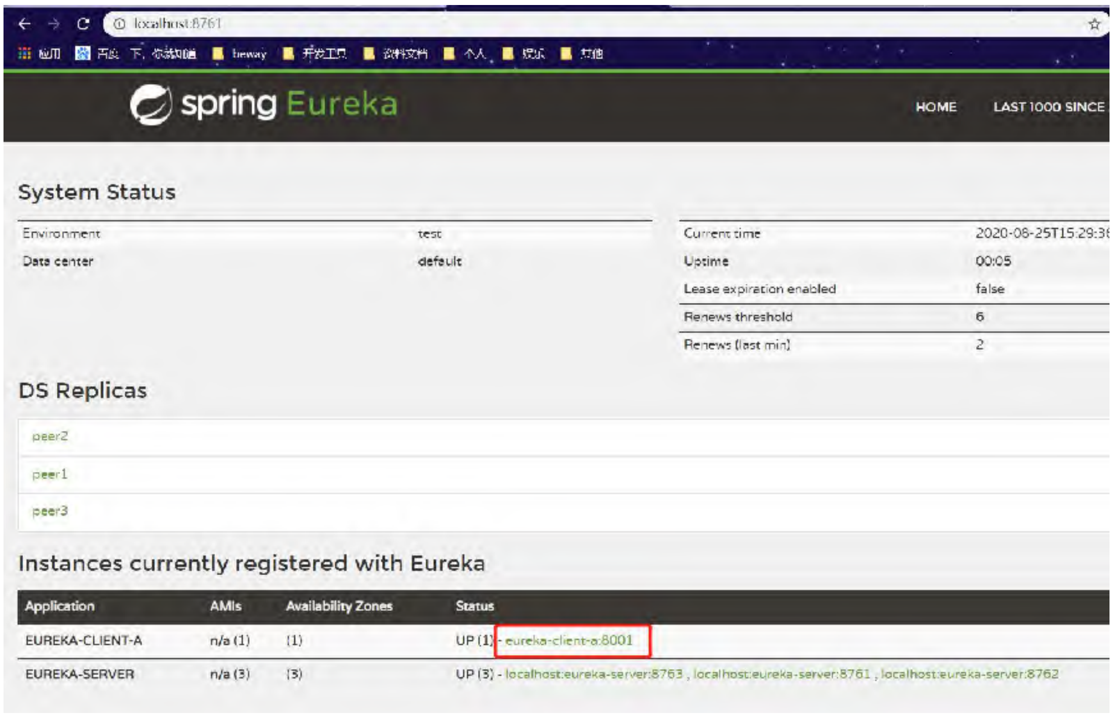


application.yml

```yaml
server:
    port: 8001
spring:
    application:
        name: eureka-client-a
eureka:
    client:
        service-url:    #eureka服务端和客户端的交互地址,集群使用,来分割
            defaultZone: http://peer1:8761/eureka/,http://peer2:8761/eureka/,http://peer3:8761/eureka/
        register-with-eureka: true  #注册自己到eureka-server上
        fetch-registry: true    #拉去服务列表
    instance:
        instance-id: ${spring.application.name}:${server.port}  #注册到eureka-server的实例名称,   默这样显示
        lease-renewal-interval-in-seconds: 30   #eureka客户端向服务端发送心跳时间，每隔30s发送一次
```

5.2.2 测试

不管哪一台 server 都注册成功了



System Status

Environment	test
Data center	default

Current time	2020-08-25T15:29:38
Uptime	00:05
Lease expiration enabled	false
Renews threshold	6
Renews (last min)	2

DS Replicas

peer2
peer1
peer3

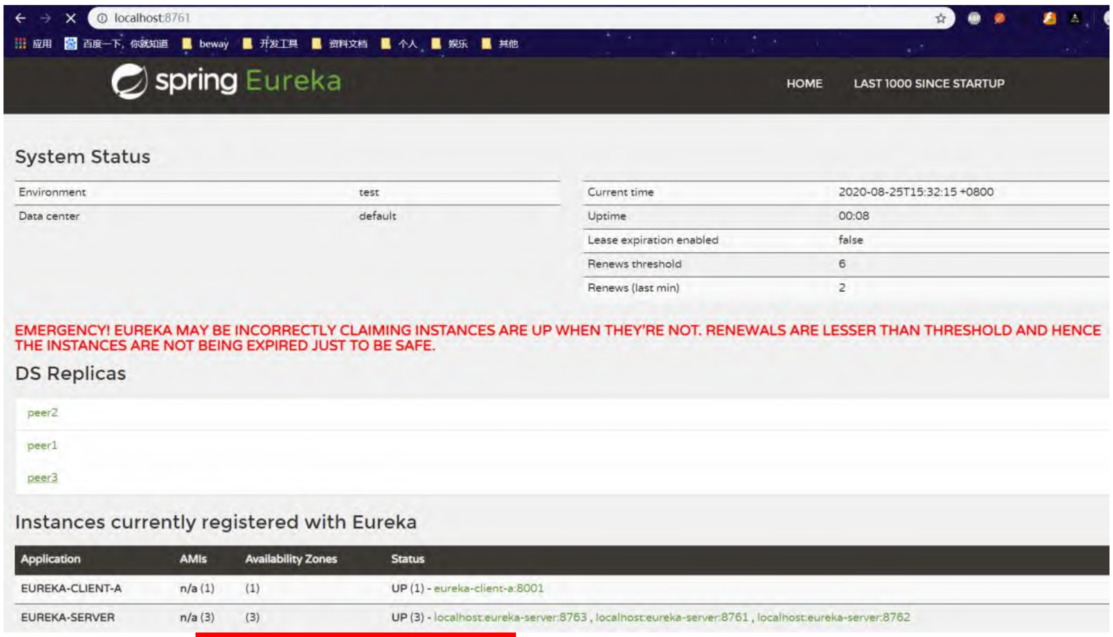
Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
EUREKA-CLIENT-A	n/a (1)	(1)	UP (1) - eureka-client-a:8001
EUREKA-SERVER	n/a (3)	(3)	UP (3) - localhost:eureka-server:8763 , localhost:eureka-server:8761 , localhost:eureka-server:8762

5.2.3 宕机一台 server

Eureka server 的集群里面，没有主机和从机的概念，节点都是对等的，只有集群里面有一个集群存活，就能保证服务的可用性。（主机 （写） 从 （读））

只要有一台存活，服务就能注册和调用



System Status

Environment	test
Data center	default

Current time	2020-08-25T15:32:15 +0800
Uptime	00:08
Lease expiration enabled	false
Renews threshold	6
Renews (last min)	2

EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.

DS Replicas

peer2
peer1
peer3

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
EUREKA-CLIENT-A	n/a (1)	(1)	UP (1) - eureka-client-a:8001
EUREKA-SERVER	n/a (3)	(3)	UP (3) - localhost:eureka-server:8763 , localhost:eureka-server:8761 , localhost:eureka-server:8762

了解一下一个**分布式数据一致性协议**　Paxos　raft

http://thesecretlivesofdata.com/raft/

zk 是 Paxos

eureka 没有分布式数据一致性的机制 节点都是相同的

nacos　raft

在有主从模式的集群中 一般都要遵循这样的协议 才可以稳定对外提供服务

Zookeeper　Paxos
Nacos　raft

6. Eureka 概念的理解

6.1 服务的注册

当项目启动时（eureka 的客户端），就会向 eureka-server 发送自己的**元数据（原始数据）**（运行的 ip，端口 port，健康的状态监控等，因为使用的是 http/ResuFul 请求风格），eureka-server 会在自己内部保留这些元数据(内存中)。（有一个服务列表）（restful 风格，以 http 动词的请求方式，完成对 url 资源的操作）

6.2 服务的续约

项目启动成功了，除了向 eureka-server 注册自己成功，还会**定时**的向 eureka-server 汇报自己，心跳，表示自己还活着。（修改一个时间）

6.3 服务的下线（主动下线）

当项目关闭时，会给 eureka-server 报告，说明自己要下机了。

6.4 服务的剔除（被动下线，主动剔除）

当项目超过了指定时间没有向 eureka-server 汇报自己，那么 eureka-server 就会认为此节点死掉了，会把它剔除掉，也不会放流量和请求到此节点了。

7.Eureka 源码分析

了解他的原理 出了问题排查 bug，优化你的代码

7.1 Eureka 运作原理的特点

Eureka-server 对外提供的是 restful 风格的服务

以 http 动词的形式对 url 资源进行操作 get post put delete

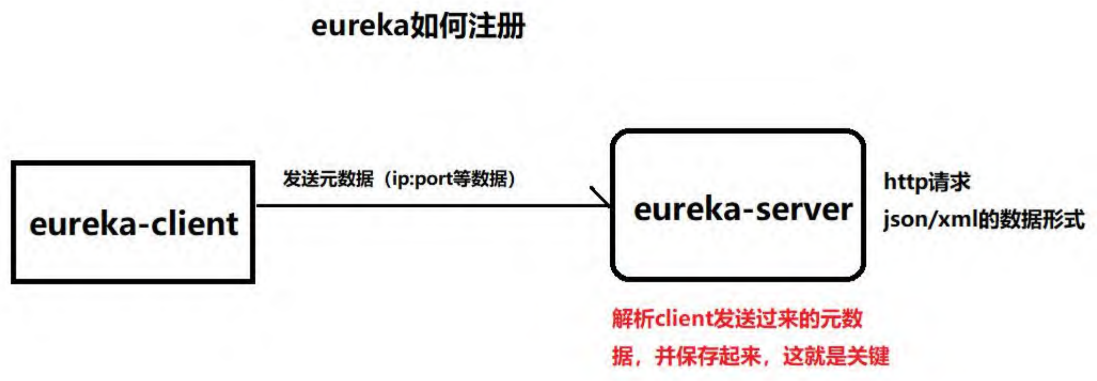
http 服务 + 特定的请求方式 + 特定的 url 地址

只要利用这些 restful 我们就能对项目实现注册和发现

只不过，eureka 已经帮我们使用 java 语言写了 client，让我们的项目只要依赖 client 就能实现注册和发现！

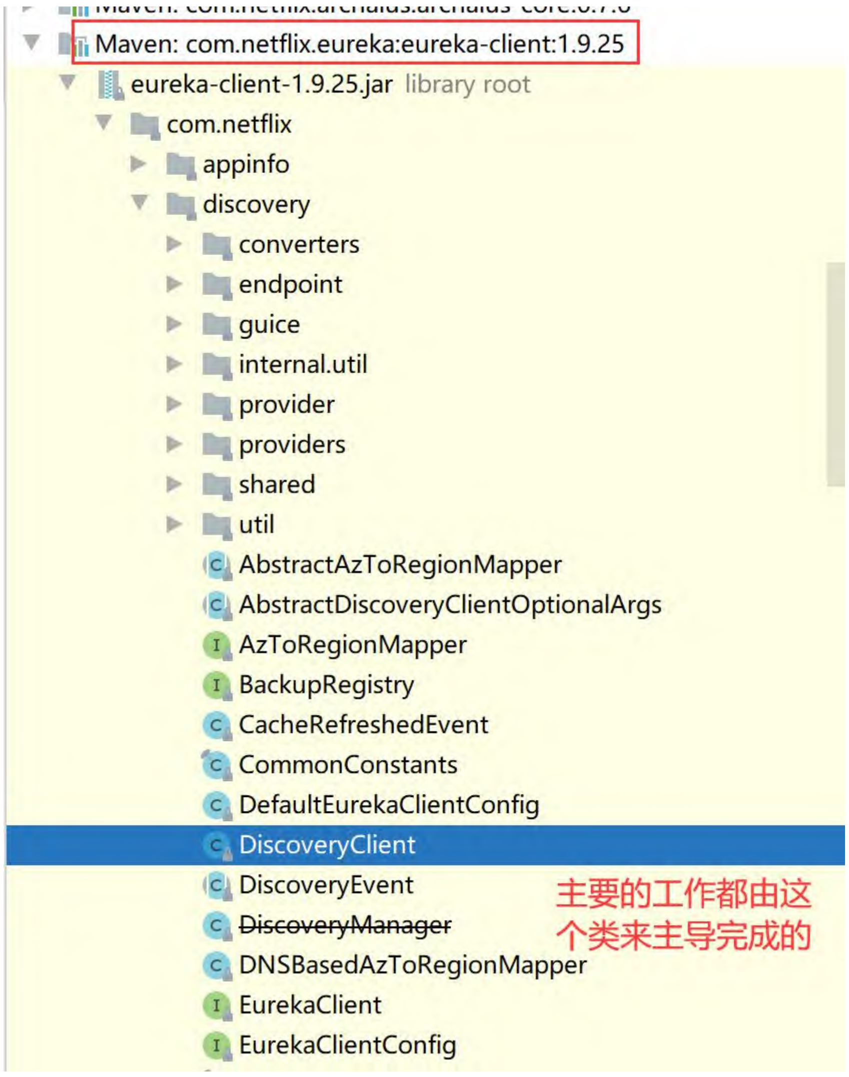
只要你会发起 Http 请求，那你就**有可能**自己实现服务的注册和发现。不管你是什么语言！

7.2 服务注册的源码分析【重点】


eureka如何注册
eureka-client
发送元数据（ip:port等数据）
eureka-server
http请求
json/xml的数据形式
解析client发送过来的元数据，并保存起来，这就是关键

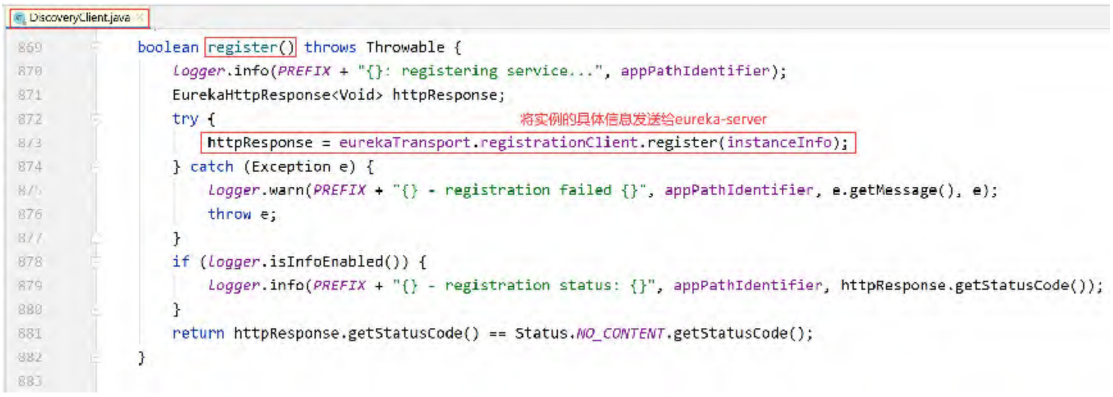
7.2.1 Eureka-client 发起注册请求

7.2.1.1 源码位置


Maven: com.netflix.eureka:eureka-client:1.9.25
eureka-client-1.9.25.jar library root
com.netflix
appinfo
discovery
converters
endpoint
guice
internal.util
provider
providers
shared
util
AbstractAzToRegionMapper
AbstractDiscoveryClientOptionalArgs
AzToRegionMapper
BackupRegistry
CacheRefreshedEvent
CommonConstants
DefaultEurekaClientConfig
DiscoveryClient
DiscoveryEvent
~~DiscoveryManager~~
DNSBasedAzToRegionMapper
EurekaClient
EurekaClientConfig

主要的工作都由这个类来主导完成的

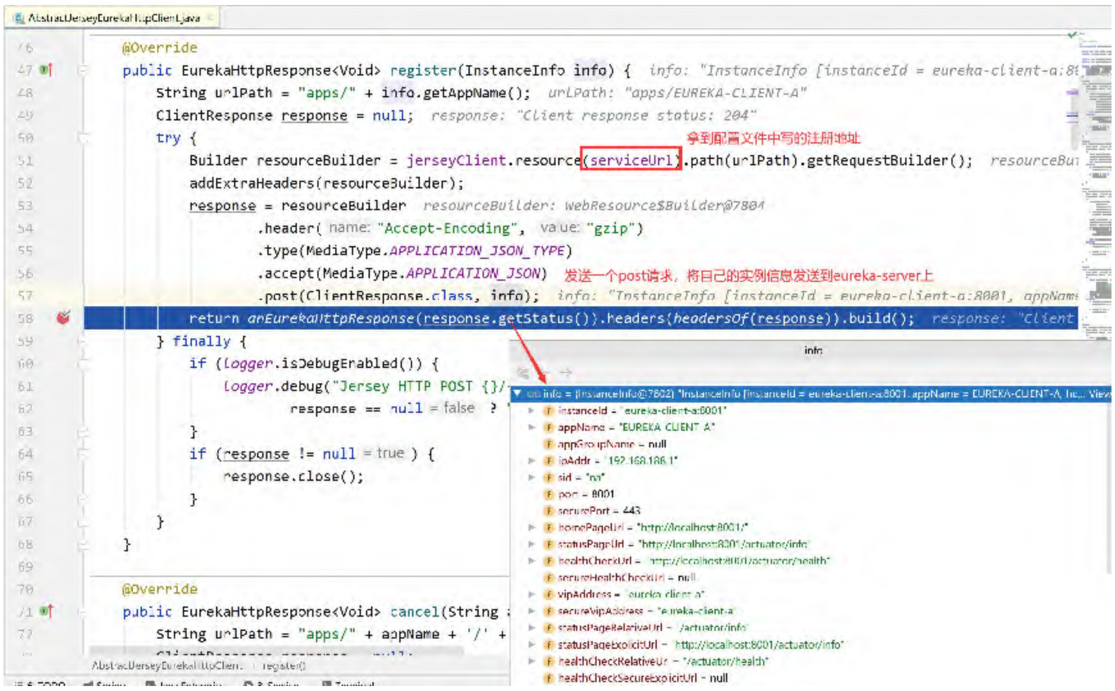
7.2.1.2 如何发送信息注册自己


DiscoveryClient.java

```java
    boolean register() throws Throwable {
        logger.info(PREFIX + "{}: registering service...", appPathIdentifier);
        EurekaHttpResponse<Void> httpResponse;
        try {
            httpResponse = eurekaTransport.registrationClient.register(instanceInfo);
        } catch (Exception e) {
            logger.warn(PREFIX + "{} - registration failed {}", appPathIdentifier, e.getMessage(), e);
            throw e;
        }
        if (logger.isInfoEnabled()) {
            logger.info(PREFIX + "{} - registration status: {}", appPathIdentifier, httpResponse.getStatusCode());
        }
        return httpResponse.getStatusCode() == Status.NO_CONTENT.getStatusCode();
    }
```

将实例的具体信息发送给eureka-server

7.2.1.3 真正的注册 AbstractJerseyEurekaHttpClient


AbstractJerseyEurekaHttpClient.java

```java
    @Override
    public EurekaHttpResponse<Void> register(InstanceInfo info) {   info: "InstanceInfo [instanceId = eureka-client-a:80
        String urlPath = "apps/" + info.getAppName();   urlPath: "apps/EUREKA-CLIENT-A"
        ClientResponse response = null;   response: "Client response status: 204"
        try {
            Builder resourceBuilder = jerseyClient.resource(serviceUrl).path(urlPath).getRequestBuilder();   resourceBu
            addExtraHeaders(resourceBuilder);
            response = resourceBuilder   resourceBuilder: WebResource$Builder@7804
                    .header( name: "Accept-Encoding",  value: "gzip")
                    .type(MediaType.APPLICATION_JSON_TYPE)
                    .accept(MediaType.APPLICATION_JSON)
                    .post(ClientResponse.class, info);   info: "InstanceInfo [instanceId = eureka-client-a:8001, appName
            return anEurekaHttpResponse(response.getStatus()).headers(headersOf(response)).build();   response: "Client
        } finally {
            if (logger.isDebugEnabled()) {
                logger.debug("Jersey HTTP POST {}/
                        response == null = false  ?
            }
            if (response != null = true ) {
                response.close();
            }
        }
    }

    @Override
    public EurekaHttpResponse<Void> cancel(String
        String urlPath = "apps/" + appName + '/' +
```

拿到配置文件中写的注册地址

发送一个post请求，将自己的实例信息发送到eureka-server上

```
info
▼ info = {InstanceInfo@7802} "InstanceInfo [instanceId = eureka-client-a:8001, appName = EUREKA-CLIENT-A, Ip... View
    instanceId = "eureka-client-a:8001"
    appName = "EUREKA CLIENT A"
    appGroupName = null
    ipAddr = "192.168.186.1"
    sid = "na"
    port = 8001
    securePort = 443
    homePageUrl = "http://localhost:8001/"
    statusPageUrl = "http://localhost:8001/actuator/info"
    healthCheckUrl = "http://localhost:8001/actuator/health"
    secureHealthCheckUrl = null
    vipAddress = "eureka-client-a"
    secureVipAddress = "eureka-client-a"
    statusPageRelativeUrl = "/actuator/info"
    statusPageExplicitUrl = "http://localhost:8001/actuator/info"
    healthCheckRelativeUrl = "/actuator/health"
    healthCheckSecureExplicitUrl = null
```

总结：

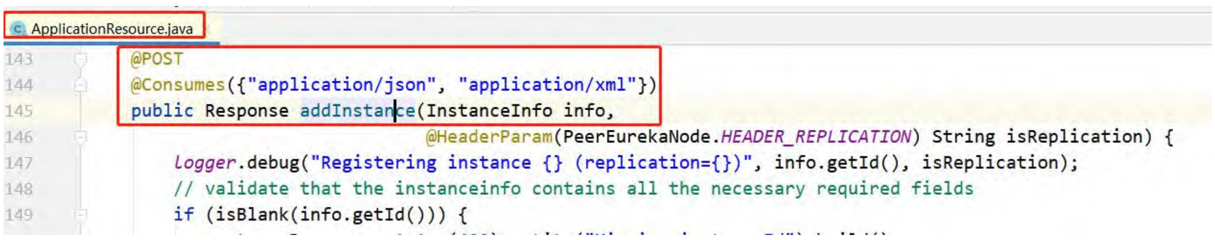
当 eureka 启动的时候，会向我们指定的 serviceUrl 发送请求，把自己节点的数据以 post 请求的方式，数据以 json 形式发送过去。

当返回的状态码为 204 的时候，表示注册成功。

7.2.2 Eureka-server 实现注册+保存

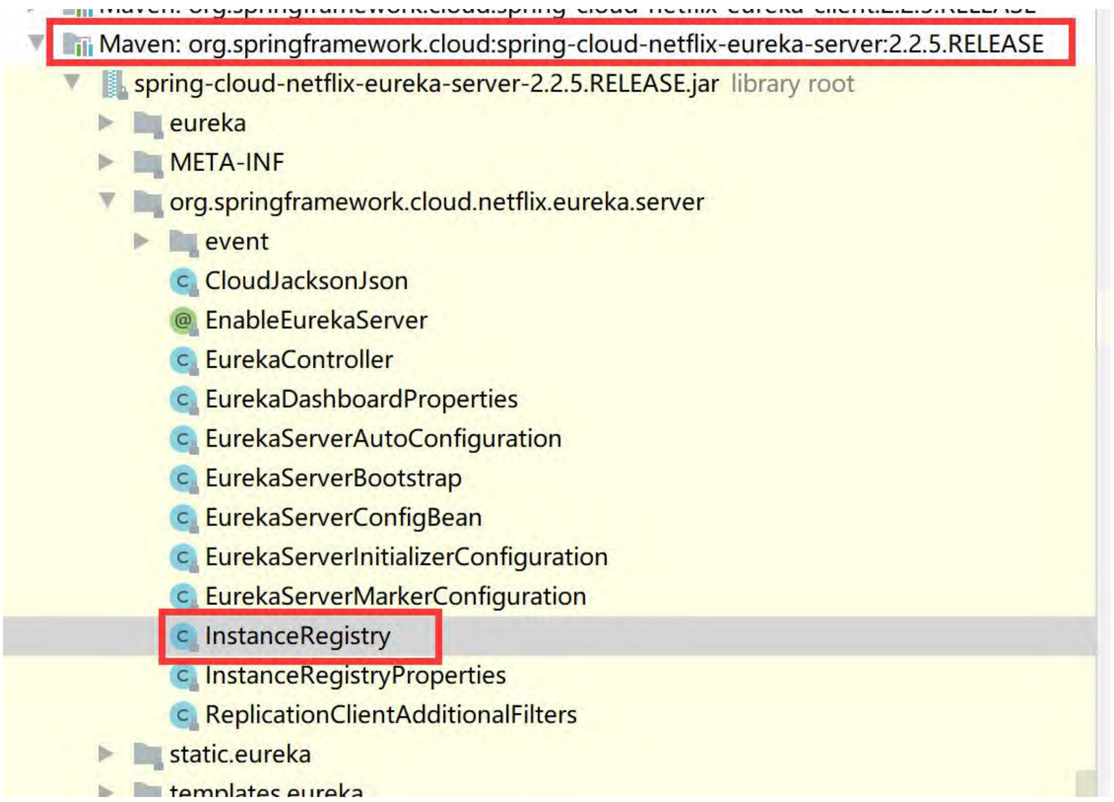
7.2.2.1 接受客户端的请求

com.netflix.eureka.resources.ApplicationResource

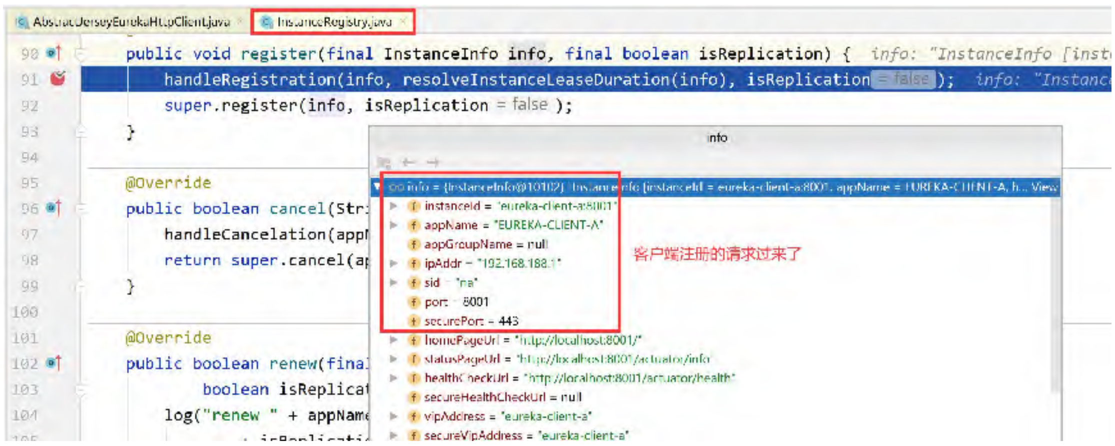

ApplicationResource.java

```
@POST
@Consumes({"application/json", "application/xml"})
public Response addInstance(InstanceInfo info,
                            @HeaderParam(PeerEurekaNode.HEADER_REPLICATION) String isReplication) {
    logger.debug("Registering instance {} (replication={})", info.getId(), isReplication);
    // validate that the instanceinfo contains all the necessary required fields
    if (isBlank(info.getId())) {
```

7.2.2.2 源码位置


```
▼ Maven: org.springframework.cloud:spring-cloud-netflix-eureka-server:2.2.5.RELEASE
  ▼ spring-cloud-netflix-eureka-server-2.2.5.RELEASE.jar library root
    ▶ eureka
    ▶ META-INF
    ▼ org.springframework.cloud.netflix.eureka.server
      ▶ event
        CloudJacksonJson
        EnableEurekaServer
        EurekaController
        EurekaDashboardProperties
        EurekaServerAutoConfiguration
        EurekaServerBootstrap
        EurekaServerConfigBean
        EurekaServerInitializerConfiguration
        EurekaServerMarkerConfiguration
        InstanceRegistry
        InstanceRegistryProperties
        ReplicationClientAdditionalFilters
    ▶ static.eureka
    ▶ templates.eureka
```

7.2.2.3 接受 client 的注册请求

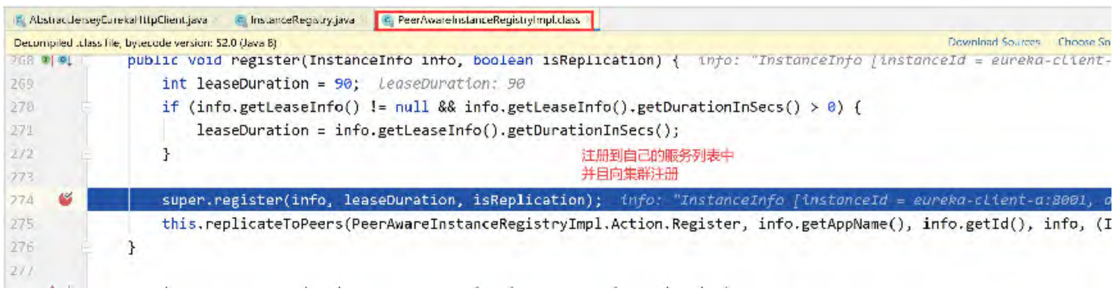

AbstractJerseyEurekaHttpClient.java　InstanceRegistry.java

```
public void register(final InstanceInfo info, final boolean isReplication) {   info: "InstanceInfo [inst
    handleRegistration(info, resolveInstanceLeaseDuration(info), isReplication = false);   info: "Instance
    super.register(info, isReplication = false );
}

@Override
public boolean cancel(Str
    handleCancelation(appN
    return super.cancel(ap
}

@Override
public boolean renew(final
        boolean isReplicat
    log("renew " + appName
```

```
info
oo info = {InstanceInfo@10102} "InstanceInfo [instanceId = eureka-client-a:8001, appName = EUREKA-CLIENT-A, h... View
  instanceId = "eureka-client-a:8001"
  appName = "EUREKA-CLIENT-A"
  appGroupName = null
  ipAddr = "192.168.188.1"
  sid = "na"
  port = 8001
  securePort = 443
  homePageUrl = "http://localhost:8001/"
  statusPageUrl = "http://localhost:8001/actuator/info"
  healthCheckUrl = "http://localhost:8001/actuator/health"
  secureHealthCheckUrl = null
  vipAddress = "eureka-client-a"
  secureVipAddress = "eureka-client-a"
```

客户端注册的请求过来了

7.2.2.4 处理请求（注册自己，向其他节点注册）

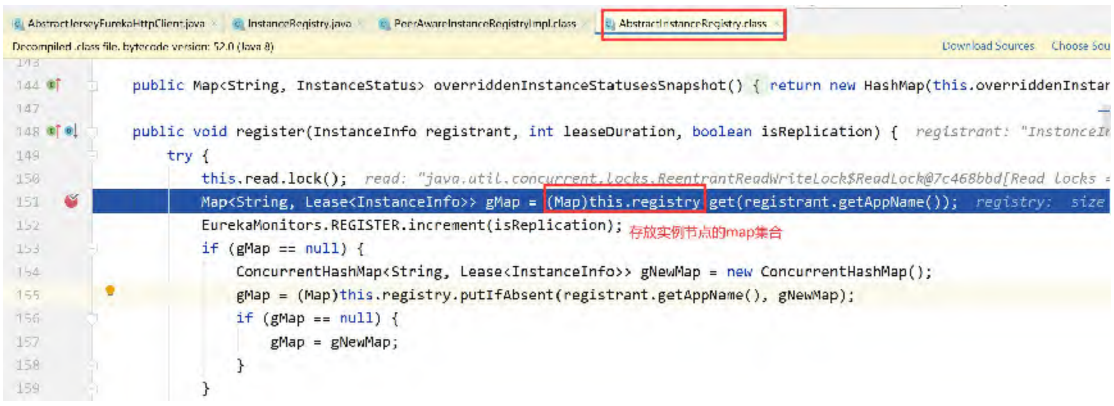


AbstractJerseyEurekaHttpClient.java　InstanceRegistry.java　PeerAwareInstanceRegistryImpl.class

Decompiled .class file, bytecode version: 52.0 (Java 8)　Download Sources　Choose So

```
public void register(InstanceInfo info, boolean isReplication) {  info: "InstanceInfo [instanceId = eureka-client-
    int leaseDuration = 90;  leaseDuration: 90
    if (info.getLeaseInfo() != null && info.getLeaseInfo().getDurationInSecs() > 0) {
        leaseDuration = info.getLeaseInfo().getDurationInSecs();
    }

    super.register(info, leaseDuration, isReplication);  info: "InstanceInfo [instanceId = eureka-client-a:8001, 
    this.replicateToPeers(PeerAwareInstanceRegistryImpl.Action.Register, info.getAppName(), info.getId(), info, (I
}
```

7.2.2.5 真正的注册自己



AbstractJerseyEurekaHttpClient.java　InstanceRegistry.java　PeerAwareInstanceRegistryImpl.class　AbstractInstanceRegistry.class

Decompiled .class file, bytecode version: 52.0 (Java 8)　Download Sources　Choose Sou

```
public Map<String, InstanceStatus> overriddenInstanceStatusesSnapshot() { return new HashMap(this.overriddenInstar

public void register(InstanceInfo registrant, int leaseDuration, boolean isReplication) {  registrant: "InstanceIr
    try {
        this.read.lock();  read: "java.util.concurrent.locks.ReentrantReadWriteLock$ReadLock@7c468bbd[Read locks =
        Map<String, Lease<InstanceInfo>> gMap = (Map)this.registry.get(registrant.getAppName());  registry:  size
        EurekaMonitors.REGISTER.increment(isReplication);
        if (gMap == null) {
            ConcurrentHashMap<String, Lease<InstanceInfo>> gNewMap = new ConcurrentHashMap();
            gMap = (Map)this.registry.putIfAbsent(registrant.getAppName(), gNewMap);
            if (gMap == null) {
                gMap = gNewMap;
            }
        }
```

7.2.2.6 具体源码分析

```
public void register(InstanceInfo registrant, int leaseDuration, boolean isReplication) {
        try {
            read.lock();
          //通过服务名称得到注册的实例
            Map<String, Lease<InstanceInfo>> gMap = registry.get(registrant.getAppName());
            REGISTER.increment(isReplication);
         //因为之前没有实例，肯定为 null
            if (gMap == null) {
          //新建一个集合来存放实例
                final    ConcurrentHashMap<String,    Lease<InstanceInfo>>    gNewMap    =    new
ConcurrentHashMap<String, Lease<InstanceInfo>>();
                gMap = registry.putIfAbsent(registrant.getAppName(), gNewMap);
                if (gMap == null) {
                    gMap = gNewMap;
                }
            }
          //gMap 就是该服务的实例
            Lease<InstanceInfo> existingLease = gMap.get(registrant.getId());
            // Retain the last dirty timestamp without overwriting it, if there is already a lease
            if (existingLease != null && (existingLease.getHolder() != null)) {
                Long                              existingLastDirtyTimestamp                              =
existingLease.getHolder().getLastDirtyTimestamp();
                Long registrationLastDirtyTimestamp = registrant.getLastDirtyTimestamp();
                logger.debug("Existing      lease      found      (existing={},      provided={}",
existingLastDirtyTimestamp, registrationLastDirtyTimestamp);
                // this is a > instead of a >= because if the timestamps are equal, we still take
the remote transmitted
                // InstanceInfo instead of the server local copy.
                if (existingLastDirtyTimestamp > registrationLastDirtyTimestamp) {
```

```
                logger.warn("There is an existing lease and the existing lease's dirty timestamp
{} is greater" +
                        " than the one that is being registered {}", existingLastDirtyTimestamp,
registrationLastDirtyTimestamp);
                logger.warn("Using the existing instanceInfo instead of the new instanceInfo as
the registrant");
                registrant = existingLease.getHolder();
            }
        } else {
            // The lease does not exist and hence it is a new registration
            synchronized (lock) {
                if (this.expectedNumberOfClientsSendingRenews > 0) {
                    // Since the client wants to register it, increase the number of clients
sending renews
                    this.expectedNumberOfClientsSendingRenews                                =
this.expectedNumberOfClientsSendingRenews + 1;
                    updateRenewsPerMinThreshold();
                }
            }
            logger.debug("No previous lease information found; it is new registration");
        }
      //新建一个服务的实例节点
        Lease<InstanceInfo> lease = new Lease<InstanceInfo>(registrant, leaseDuration);
        if (existingLease != null) {
            lease.setServiceUpTimestamp(existingLease.getServiceUpTimestamp());
        }
      //放到注册 map 的列表里
        gMap.put(registrant.getId(), lease);
        recentRegisteredQueue.add(new Pair<Long, String>(
                System.currentTimeMillis(),
                registrant.getAppName() + "(" + registrant.getId() + ")"));
        // This is where the initial state transfer of overridden status happens
        if (!InstanceStatus.UNKNOWN.equals(registrant.getOverriddenStatus())) {
            logger.debug("Found overridden status {} for instance {}. Checking to see if needs
to be add to the "
                        +          "overrides",          registrant.getOverriddenStatus(),
registrant.getId());
            if (!overriddenInstanceStatusMap.containsKey(registrant.getId())) {
                logger.info("Not   found   overridden   id   {}   and   hence   adding   it",
registrant.getId());
                overriddenInstanceStatusMap.put(registrant.getId(),
registrant.getOverriddenStatus());
            }
        }
        InstanceStatus                    overriddenStatusFromMap                            =
overriddenInstanceStatusMap.get(registrant.getId());
        if (overriddenStatusFromMap != null) {
            logger.info("Storing overridden status {} from map", overriddenStatusFromMap);
            registrant.setOverriddenStatus(overriddenStatusFromMap);
        }

        // Set the status based on the overridden status rules
        InstanceStatus  overriddenInstanceStatus  =  getOverriddenInstanceStatus(registrant,
existingLease, isReplication);
        registrant.setStatusWithoutDirty(overriddenInstanceStatus);

        // If the lease is registered with UP status, set lease service up timestamp
        if (InstanceStatus.UP.equals(registrant.getStatus())) {
            lease.serviceUp();
        }
        registrant.setActionType(ActionType.ADDED);
        recentlyChangedQueue.add(new RecentlyChangedItem(lease));
      //设置心跳时间等参数
        registrant.setLastUpdatedTimestamp();
        invalidateCache(registrant.getAppName(),              registrant.getVIPAddress(),
registrant.getSecureVipAddress());
            logger.info("Registered instance {}/{} with status {} (replication={})",
                    registrant.getAppName(),     registrant.getId(),     registrant.getStatus(),
isReplication);
        } finally {
            read.unlock();
        }
    }
```

7.2.3 服务注册总结

重要的类：

DiscoveryClient 里面的 register()方法完后注册的总体构造

AbstractJerseyEurekaHttpClient 里面的 register()方法具体发送注册请求（post)

InstanceRegistry 里面 register()方法接受客户端的注册请求

PeerAwareInstanceRegistryImpl 里面调用父类的 register()方法实现注册

AbstractInstanceRegistry 里面的 register()方法完成具体的注册保留数据到 map 集合

保存服务实例数据的集合：

第一个 key 是应用名称（全大写） spring.application.name

Value 中的 key 是应用的实例 id eureka.instance.instance-id

Value 中的 value 是 具体的服务节点信息

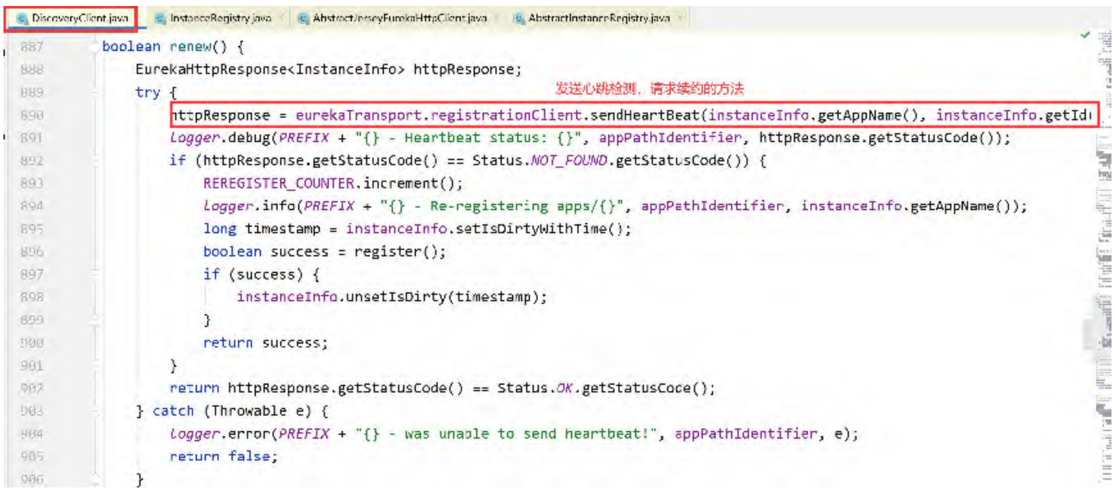
```
private final ConcurrentHashMap<String, Map<String,
Lease<InstanceInfo>>> registry
        = new ConcurrentHashMap<String, Map<String,
Lease<InstanceInfo>>>();
```

7.3 服务续约的源码分析

7.3.1 Eureka-client 发起续约请求

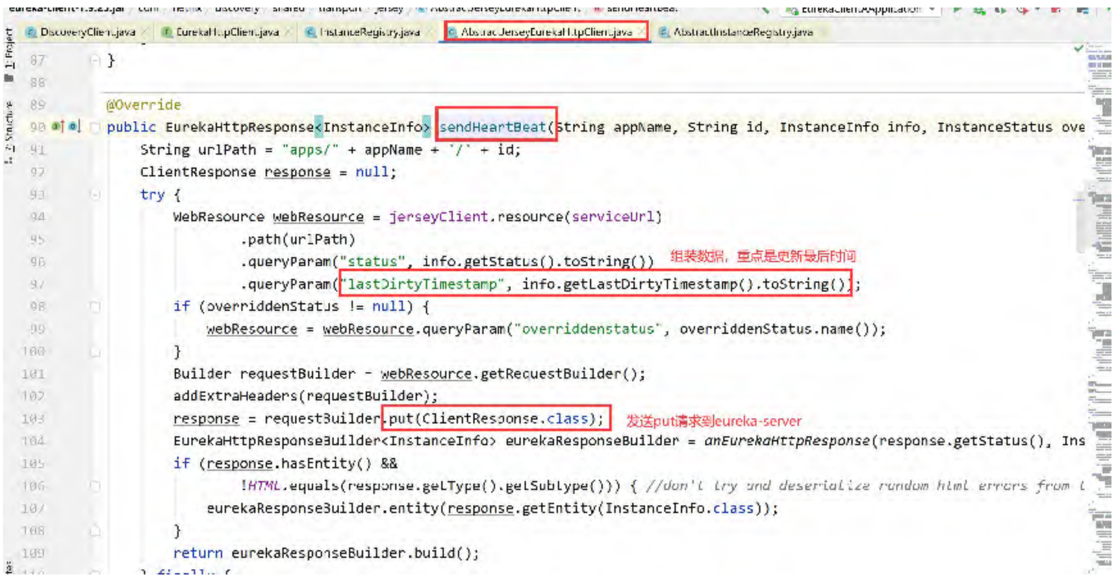
7.3.1.1 如何发请求续约自己

DiscoveryClient 的 renew()方法


DiscoveryClient.java
InstanceRegistry.java
AbstractJerseyEurekaHttpClient.java
AbstractInstanceRegistry.java

```
boolean renew() {
    EurekaHttpResponse<InstanceInfo> httpResponse;
    try {                                                        发送心跳检测，请求续约的方法
        httpResponse = eurekaTransport.registrationClient.sendHeartBeat(instanceInfo.getAppName(), instanceInfo.getId
        logger.debug(PREFIX + "{} - Heartbeat status: {}", appPathIdentifier, httpResponse.getStatusCode());
        if (httpResponse.getStatusCode() == Status.NOT_FOUND.getStatusCode()) {
            REREGISTER_COUNTER.increment();
            logger.info(PREFIX + "{} - Re-registering apps/{}", appPathIdentifier, instanceInfo.getAppName());
            long timestamp = instanceInfo.setIsDirtyWithTime();
            boolean success = register();
            if (success) {
                instanceInfo.unsetIsDirty(timestamp);
            }
            return success;
        }
        return httpResponse.getStatusCode() == Status.OK.getStatusCode();
    } catch (Throwable e) {
        logger.error(PREFIX + "{} - was unable to send heartbeat!", appPathIdentifier, e);
        return false;
    }
```

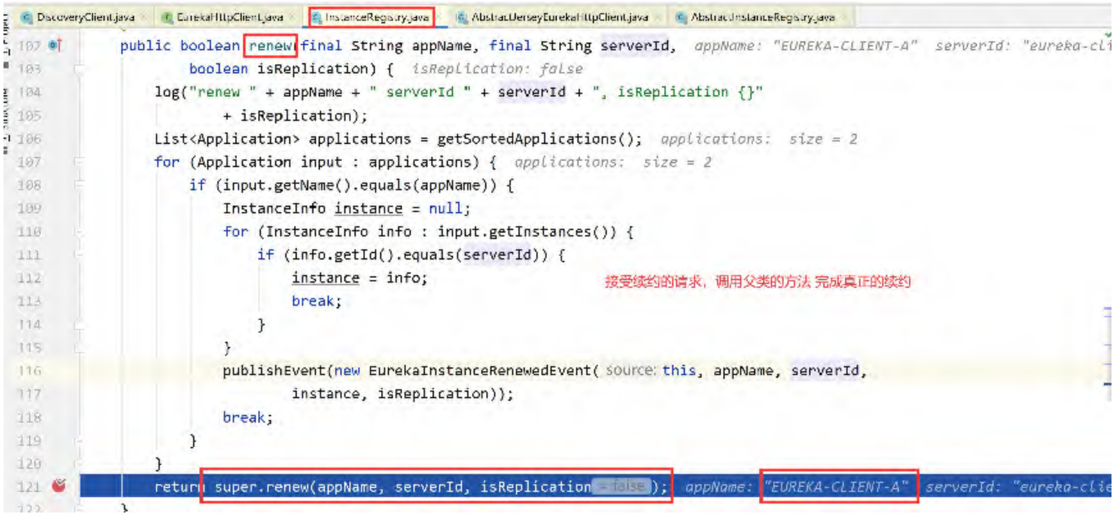
7.3.1.2 真正的请求续约自己（AbstractJerseyEurekaHttpClient）


DiscoveryClient.java
EurekaHttpClient.java
InstanceRegistry.java
AbstractJerseyEurekaHttpClient.java
AbstractInstanceRegistry.java

```
}

@Override
public EurekaHttpResponse<InstanceInfo> sendHeartBeat(String appName, String id, InstanceInfo info, InstanceStatus ove
    String urlPath = "apps/" + appName + '/' + id;
    ClientResponse response = null;
    try {
        WebResource webResource = jerseyClient.resource(serviceUrl)
                .path(urlPath)
                .queryParam("status", info.getStatus().toString())  组装数据，重点是更新最后时间
                .queryParam("lastDirtyTimestamp", info.getLastDirtyTimestamp().toString());
        if (overriddenStatus != null) {
            webResource = webResource.queryParam("overriddenstatus", overriddenStatus.name());
        }
        Builder requestBuilder = webResource.getRequestBuilder();
        addExtraHeaders(requestBuilder);
        response = requestBuilder.put(ClientResponse.class);  发送put请求到eureka-server
        EurekaHttpResponseBuilder<InstanceInfo> eurekaResponseBuilder = anEurekaHttpResponse(response.getStatus(), Ins
        if (response.hasEntity() &&
                !HTML.equals(response.getType().getSubtype())) { //don't try and deserialize random html errors from t
            eurekaResponseBuilder.entity(response.getEntity(InstanceInfo.class));
        }
        return eurekaResponseBuilder.build();
```

7.3.2 Eureka-server 实现续约操作

7.3.2.1 接受续约的请求

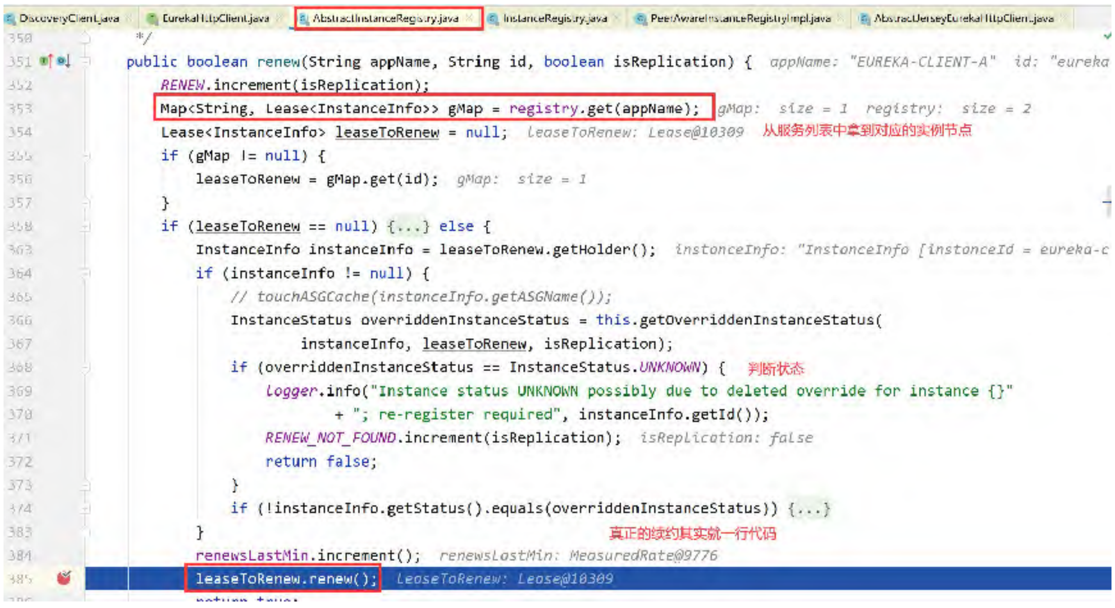


DiscoveryClient.java | EurekaHttpClient.java | InstanceRegistry.java | AbstractJerseyEurekaHttpClient.java | AbstractInstanceRegistry.java

```
public boolean renew(final String appName, final String serverId,   appName: "EUREKA-CLIENT-A"  serverId: "eureka-cli
        boolean isReplication) {  isReplication: false
    log("renew " + appName + " serverId " + serverId + ", isReplication {}"
            + isReplication);
    List<Application> applications = getSortedApplications();  applications:  size = 2
    for (Application input : applications) {  applications:  size = 2
        if (input.getName().equals(appName)) {
            InstanceInfo instance = null;
            for (InstanceInfo info : input.getInstances()) {
                if (info.getId().equals(serverId)) {
                    instance = info;
                    break;
                }
            }
            publishEvent(new EurekaInstanceRenewedEvent( source: this, appName, serverId,
                    instance, isReplication));
            break;
        }
    }
    return super.renew(appName, serverId, isReplication = false);  appName: "EUREKA-CLIENT-A"  serverId: "eureka-clie
}
```

接受续约的请求，调用父类的方法 完成真正的续约

7.3.2.2 真正的续约



DiscoveryClient.java | EurekaHttpClient.java | AbstractInstanceRegistry.java | InstanceRegistry.java | PeerAwareInstanceRegistryImpl.java | AbstractJerseyEurekaHttpClient.java

```
 */
public boolean renew(String appName, String id, boolean isReplication) {  appName: "EUREKA-CLIENT-A"  id: "eureka
    RENEW.increment(isReplication);
    Map<String, Lease<InstanceInfo>> gMap = registry.get(appName);  gMap:  size = 1  registry:  size = 2
    Lease<InstanceInfo> leaseToRenew = null;  leaseToRenew: Lease@10309
    if (gMap != null) {
        leaseToRenew = gMap.get(id);  gMap:  size = 1
    }
    if (leaseToRenew == null) {...} else {
        InstanceInfo instanceInfo = leaseToRenew.getHolder();  instanceInfo: "InstanceInfo [instanceId = eureka-c
        if (instanceInfo != null) {
            // touchASGCache(instanceInfo.getASGName());
            InstanceStatus overriddenInstanceStatus = this.getOverriddenInstanceStatus(
                    instanceInfo, leaseToRenew, isReplication);
            if (overriddenInstanceStatus == InstanceStatus.UNKNOWN) {
                logger.info("Instance status UNKNOWN possibly due to deleted override for instance {}"
                        + "; re-register required", instanceInfo.getId());
                RENEW_NOT_FOUND.increment(isReplication);  isReplication: false
                return false;
            }
            if (!instanceInfo.getStatus().equals(overriddenInstanceStatus)) {...}
        }
        renewsLastMin.increment();  renewsLastMin: MeasuredRate@9776
        leaseToRenew.renew();  leaseToRenew: Lease@10309
```

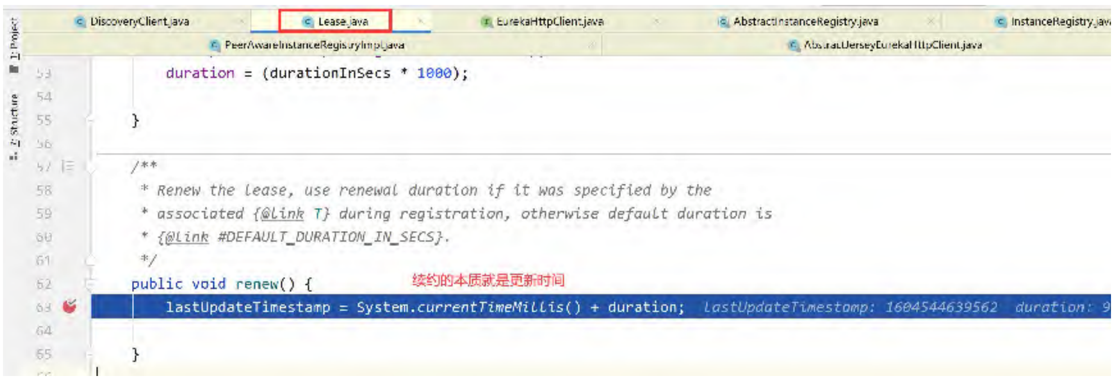
从服务列表中拿到对应的实例节点

判断状态

真正的续约其实就一行代码

7.3.2.3 续约的本质

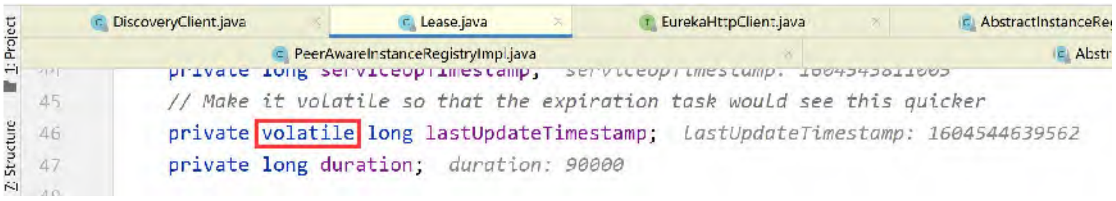
续约的本质就是修改了服务节点的最后更新时间


DiscoveryClient.java
Lease.java
EurekaHttpClient.java
AbstractInstanceRegistry.java
InstanceRegistry.java
PeerAwareInstanceRegistryImpl.java
AbstractJerseyEurekaHttpClient.java
1: Project
Z: Structure

```
        duration = (durationInSecs * 1000);

    }

    /**
     * Renew the lease, use renewal duration if it was specified by the
     * associated {@link T} during registration, otherwise default duration is
     * {@link #DEFAULT_DURATION_IN_SECS}.
     */
    public void renew() {                    续约的本质就是更新时间
        lastUpdateTimestamp = System.currentTimeMillis() + duration;   lastUpdateTimestamp: 1604544639562   duration: 9
    }
```

duration：代表注册中心最长的忍耐时间：

并不是 30s 没有续约就里面剔除，而是 30 +duration(默认是 90s) 期间内没有续约，才剔除服务



```
    private long serviceUpTimestamp;   serviceUpTimestamp: 1604543811005
    // Make it volatile so that the expiration task would see this quicker
    private volatile long lastUpdateTimestamp;   lastUpdateTimestamp: 1604544639562
    private long duration;   duration: 90000
```

Volatile 标识的变量是具有可见性的，当一条线程修改了我的剔除时间，其他线程就可以立马看到（应用场景：一写多读），后面在剔除里面有一个定时任务，去检查超时从而判断某一个服务是否应该被剔除

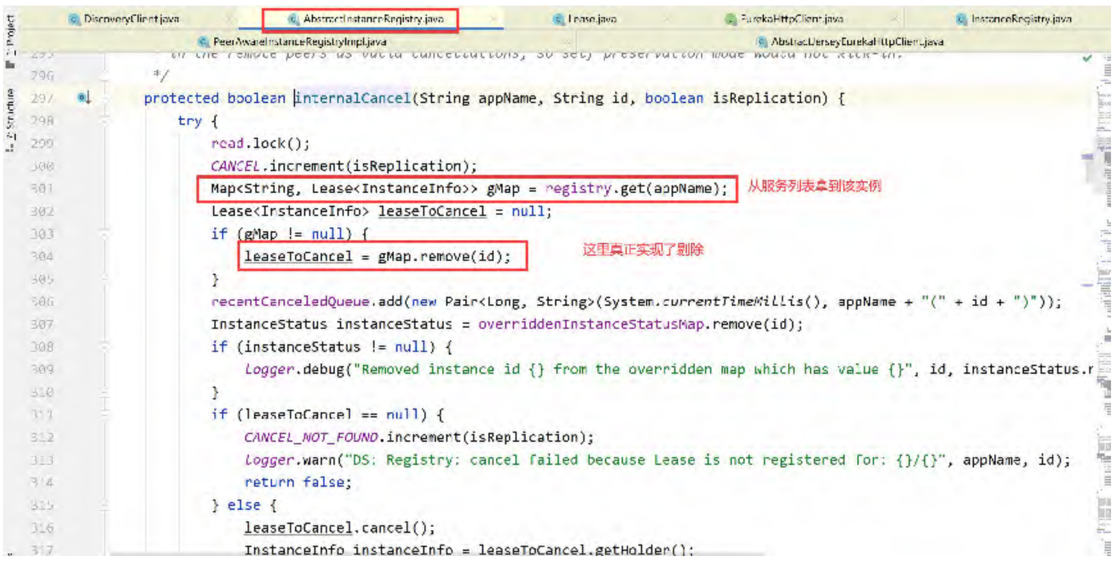
7.4 服务剔除的源码分析（被动下线）

7.4.1 Eureka-server 实现服务剔除

7.4.1.1 在 AbstractInstanceRegistry 的 evict()方法中筛选剔除的节点

```
public void evict(long additionalLeaseMs) {
        logger.debug("Running the evict task");

        if (!isLeaseExpirationEnabled()) {
            logger.debug("DS: lease expiration is currently disabled.");
            return;
        }

        // We collect first all expired items, to evict them in random order. For large
eviction sets,
        // if we do not that, we might wipe out whole apps before self preservation kicks
in. By randomizing it,
        // the impact should be evenly distributed across all applications.
        //创建一个新的集合来存放过期的服务实例
        List<Lease<InstanceInfo>> expiredLeases = new ArrayList<>();
        for    (Entry<String,    Map<String,    Lease<InstanceInfo>>>    groupEntry    :
registry.entrySet()) {
            Map<String, Lease<InstanceInfo>> leaseMap = groupEntry.getValue();
            if (leaseMap != null) {
              //循环
                for      (Entry<String,      Lease<InstanceInfo>>      leaseEntry      :
leaseMap.entrySet()) {
                    Lease<InstanceInfo> lease = leaseEntry.getValue();
                  //判断过期，加入集合中
                    if (lease.isExpired(additionalLeaseMs) && lease.getHolder() != null)
{
                        expiredLeases.add(lease);
                    }
                }
            }
        }

        // To compensate for GC pauses or drifting local time, we need to use current
registry size as a base for
        // triggering self-preservation. Without that we would wipe out full registry.
        int registrySize = (int) getLocalRegistrySize();
        int       registrySizeThreshold       =       (int)       (registrySize       *
serverConfig.getRenewalPercentThreshold());
        int evictionLimit = registrySize - registrySizeThreshold;

        int toEvict = Math.min(expiredLeases.size(), evictionLimit);
        if (toEvict > 0) {
            logger.info("Evicting {} items (expired={}, evictionLimit={})", toEvict,
expiredLeases.size(), evictionLimit);

            Random random = new Random(System.currentTimeMillis());
            for (int i = 0; i < toEvict; i++) {
                // Pick a random item (Knuth shuffle algorithm)
                int next = i + random.nextInt(expiredLeases.size() - i);
                Collections.swap(expiredLeases, i, next);
                Lease<InstanceInfo> lease = expiredLeases.get(i);

                String appName = lease.getHolder().getAppName();
                String id = lease.getHolder().getId();
                EXPIRED.increment();
                logger.warn("DS: Registry: expired lease for {}/{}", appName, id);
               //这整个方法并没有真的杀死过期的服务节点
                //下面这个方法才是真正干掉过期的服务
                internalCancel(appName, id, false);
            }
        }
    }
```

7.4.1.2 在 internalCancel 方法里面真正实现剔除



```java
protected boolean internalCancel(String appName, String id, boolean isReplication) {
    try {
        read.lock();
        CANCEL.increment(isReplication);
        Map<String, Lease<InstanceInfo>> gMap = registry.get(appName);
        Lease<InstanceInfo> leaseToCancel = null;
        if (gMap != null) {
            leaseToCancel = gMap.remove(id);
        }
        recentCanceledQueue.add(new Pair<Long, String>(System.currentTimeMillis(), appName + "(" + id + ")"));
        InstanceStatus instanceStatus = overriddenInstanceStatusMap.remove(id);
        if (instanceStatus != null) {
            logger.debug("Removed instance id {} from the overridden map which has value {}", id, instanceStatus.r
        }
        if (leaseToCancel == null) {
            CANCEL_NOT_FOUND.increment(isReplication);
            logger.warn("DS: Registry: cancel failed because Lease is not registered for: {}/{}", appName, id);
            return false;
        } else {
            leaseToCancel.cancel();
            InstanceInfo instanceInfo = leaseToCancel.getHolder();
```

7.4.1.3 在服务剔除中涉及到哪些重要的点

怎么删除一个集合里面过期的数据?

Redis 怎么清除过期的 key　LRU(热点 key)

1 定时 (k-thread)

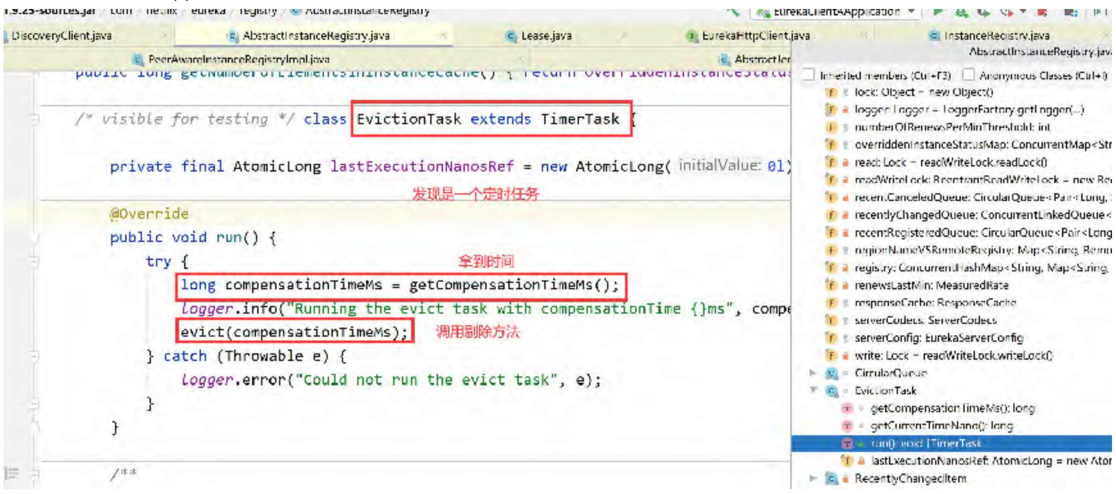
2 惰性　(在再次访问该 key 时有作用)

3 定期　(使用一个线程来完成清除任务)

定期 (实时性差) + 惰性

7.4.1.4 什么时候执行服务剔除操作呢?

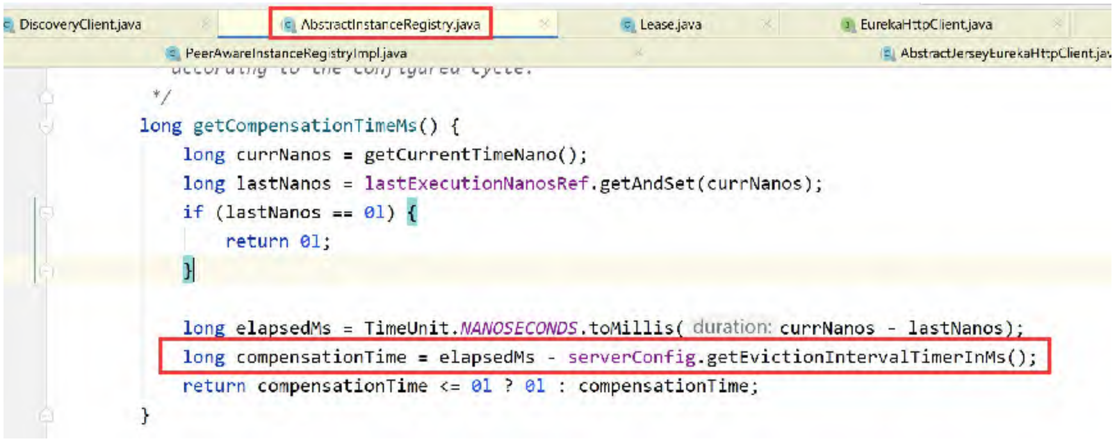
查看 evict()方法在哪里调用的



```java
/* visible for testing */ class EvictionTask extends TimerTask {

    private final AtomicLong lastExecutionNanosRef = new AtomicLong( initialValue: 0l);

    @Override
    public void run() {
        try {
            long compensationTimeMs = getCompensationTimeMs();
            logger.info("Running the evict task with compensationTime {}ms", compe
            evict(compensationTimeMs);
        } catch (Throwable e) {
            logger.error("Could not run the evict task", e);
        }
    }
```

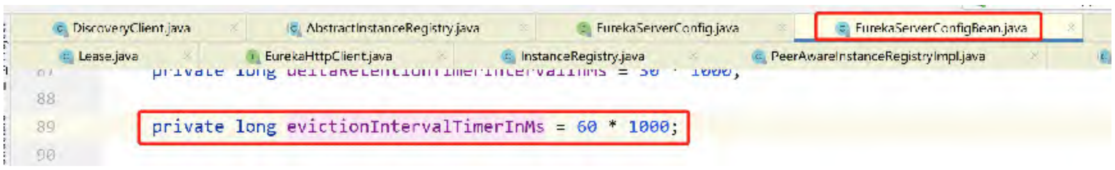
```
lock: Object = new Object()
logger: Logger = LoggerFactory.getLogger(...)
numberOfRenewsPerMinThreshold: int
overriddenInstanceStatusMap: ConcurrentMap<Str
read: Lock = readWriteLock.readLock()
readWriteLock: ReentrantReadWriteLock = new Re
recentCanceledQueue: CircularQueue<Pair<Long,
recentlyChangedQueue: ConcurrentLinkedQueue<
recentRegisteredQueue: CircularQueue<Pair<Long
regionNameVSRemoteRegistry: Map<String, Remo
registry: ConcurrentHashMap<String, Map<String,
renewsLastMin: MeasuredRate
responseCache: ResponseCache
serverCodecs: ServerCodecs
serverConfig: EurekaServerConfig
write: Lock = readWriteLock.writeLock()
CircularQueue
EvictionTask
    getCompensationTimeMs(): long
    getCurrentTimeNano(): long
    run(): void ↑TimerTask
    lastExecutionNanosRef: AtomicLong = new Ator
RecentlyChangedItem
```

具体查看多久执行一次呢?



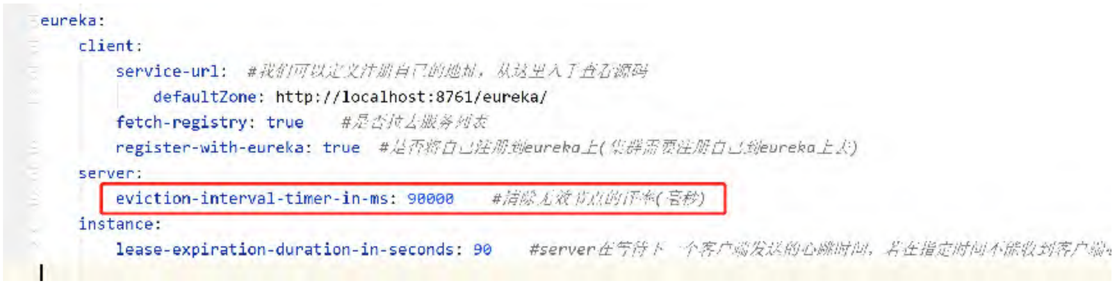
```
    according to the configured cycle.
    */
    long getCompensationTimeMs() {
        long currNanos = getCurrentTimeNano();
        long lastNanos = lastExecutionNanosRef.getAndSet(currNanos);
        if (lastNanos == 0l) {
            return 0l;
        }

        long elapsedMs = TimeUnit.NANOSECONDS.toMillis( duration: currNanos - lastNanos);
        long compensationTime = elapsedMs - serverConfig.getEvictionIntervalTimerInMs();
        return compensationTime <= 0l ? 0l : compensationTime;
    }
```

发现默认是 60s 执行一次



```
    private long deltaRetentionTimerIntervalInMs = 30 * 1000;

    private long evictionIntervalTimerInMs = 60 * 1000;
```

当然我们也可以自定义检测定时器的执行时间

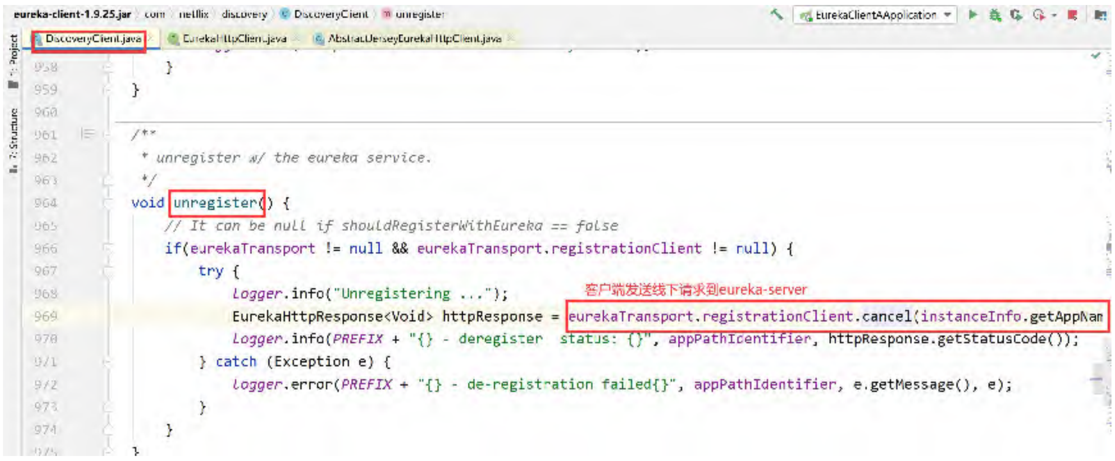


```
eureka:
  client:
    service-url:  #我们可以定义注册中心的地址，从这里入了查看源码
      defaultZone: http://localhost:8761/eureka/
    fetch-registry: true    #是否拉去服务列表
    register-with-eureka: true  #是否将自己注册到eureka上(集群需要注册自己到eureka上)
  server:
    eviction-interval-timer-in-ms: 90000    #清除无效节点的评率(毫秒)
  instance:
    lease-expiration-duration-in-seconds: 90    #server在等待下一个客户端发送的心跳时间，若在指定时间不能收到客户端
```

7.5 服务下线的源码分析

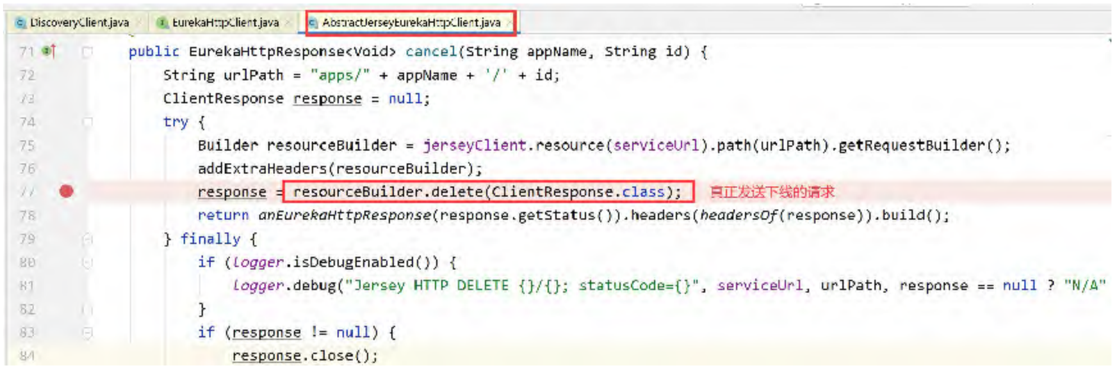
7.5.1 Eureka-client 发起下线请求

7.5.1.1 如何发起下线请求


eureka-client-1.9.25.jar › com › netflix › discovery › DiscoveryClient › unregister
EurekaClientAApplication
DiscoveryClient.java
EurekaHttpClient.java
AbstractJerseyEurekaHttpClient.java
客户端发送线下请求到eureka-server

```
        }
    }

    /**
     * unregister w/ the eureka service.
     */
    void unregister() {
        // It can be null if shouldRegisterWithEureka == false
        if(eurekaTransport != null && eurekaTransport.registrationClient != null) {
            try {
                logger.info("Unregistering ...");
                EurekaHttpResponse<Void> httpResponse = eurekaTransport.registrationClient.cancel(instanceInfo.getAppNam
                logger.info(PREFIX + "{} - deregister  status: {}", appPathIdentifier, httpResponse.getStatusCode());
            } catch (Exception e) {
                logger.error(PREFIX + "{} - de-registration failed{}", appPathIdentifier, e.getMessage(), e);
            }
        }
    }
```

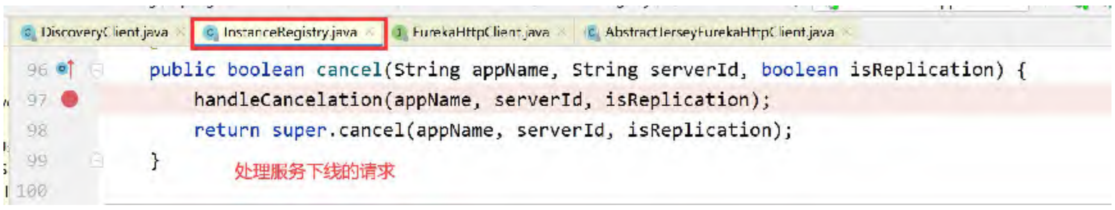
7.5.1.2 真正的发请求下线 AbstractJerseyEurekaHttpClient



```
    public EurekaHttpResponse<Void> cancel(String appName, String id) {
        String urlPath = "apps/" + appName + '/' + id;
        ClientResponse response = null;
        try {
            Builder resourceBuilder = jerseyClient.resource(serviceUrl).path(urlPath).getRequestBuilder();
            addExtraHeaders(resourceBuilder);
            response = resourceBuilder.delete(ClientResponse.class);
            return anEurekaHttpResponse(response.getStatus()).headers(headersOf(response)).build();
        } finally {
            if (logger.isDebugEnabled()) {
                logger.debug("Jersey HTTP DELETE {}/{}; statusCode={}", serviceUrl, urlPath, response == null ? "N/A"
            }
            if (response != null) {
                response.close();
```

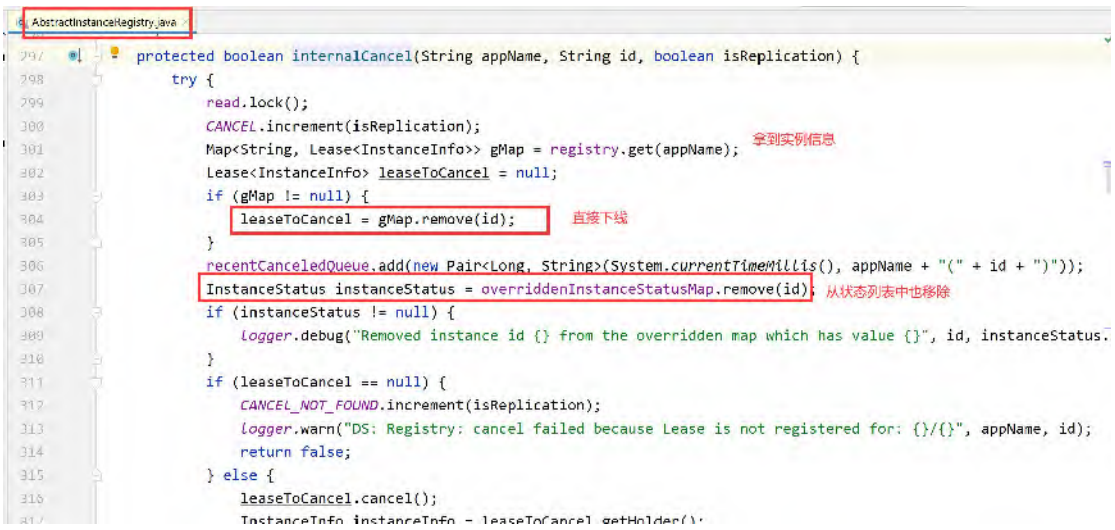
7.5.2 Eureka-server 处理下线请求

7.5.2.1 接受下线请求



```
    public boolean cancel(String appName, String serverId, boolean isReplication) {
        handleCancelation(appName, serverId, isReplication);
        return super.cancel(appName, serverId, isReplication);
    }
```

7.5.2.2 真正的下线服务



```java
protected boolean internalCancel(String appName, String id, boolean isReplication) {
    try {
        read.lock();
        CANCEL.increment(isReplication);
        Map<String, Lease<InstanceInfo>> gMap = registry.get(appName);
        Lease<InstanceInfo> leaseToCancel = null;
        if (gMap != null) {
            leaseToCancel = gMap.remove(id);
        }
        recentCanceledQueue.add(new Pair<Long, String>(System.currentTimeMillis(), appName + "(" + id + ")"));
        InstanceStatus instanceStatus = overriddenInstanceStatusMap.remove(id);
        if (instanceStatus != null) {
            logger.debug("Removed instance id {} from the overridden map which has value {}", id, instanceStatus.
        }
        if (leaseToCancel == null) {
            CANCEL_NOT_FOUND.increment(isReplication);
            logger.warn("DS: Registry: cancel failed because Lease is not registered for: {}/{}", appName, id);
            return false;
        } else {
            leaseToCancel.cancel();
            InstanceInfo instanceInfo = leaseToCancel.getHolder();
```

7.6 服务发现（源头）

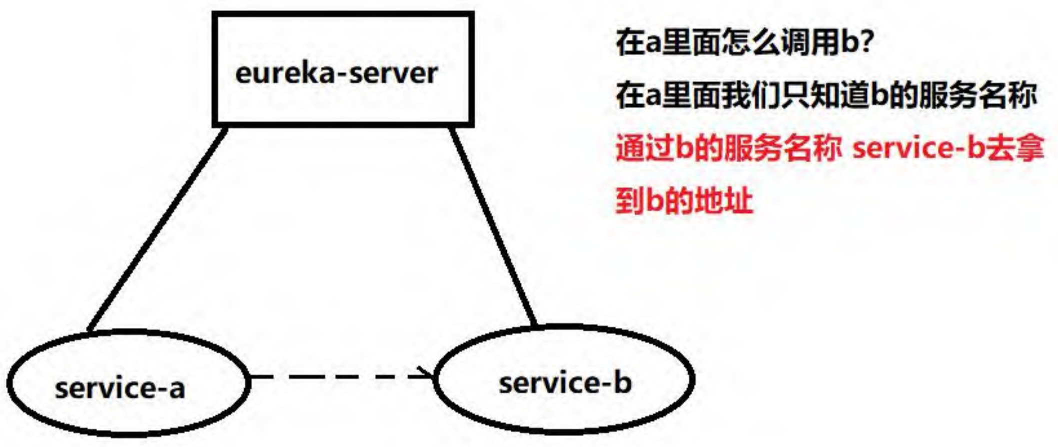
7.6.1 什么是服务发现

根据服务名称发现服务的实例过程

客户端会在本地缓存服务端的列表

拉取列表是有间隔周期的 （导致服务上线 客户端不能第一时间感知到 （可以容忍））

其实每次做服务发现 都是从本地的列表来进行的


eureka-server
service-a
service-b
在a里面怎么调用b?
在a里面我们只知道b的服务名称
通过b的服务名称 service-b去拿到b的地址

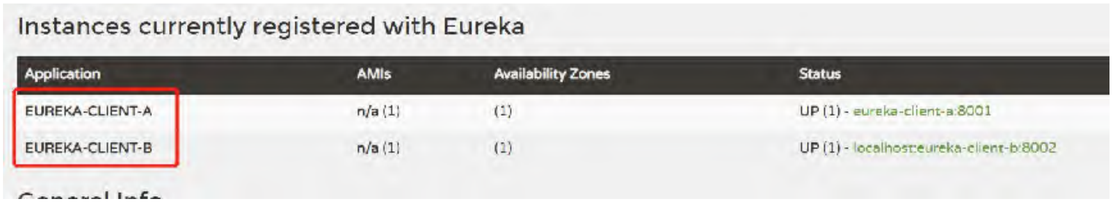
7.6.2 测试服务发现

启动 eureka-server 一台

启动服务 a

启动服务 b

确保服务都上线了



Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
EUREKA-CLIENT-A	n/a (1)	(1)	UP (1) - eureka-client-a:8001
EUREKA-CLIENT-B	n/a (1)	(1)	UP (1) - localhost:eureka-client-b:8002

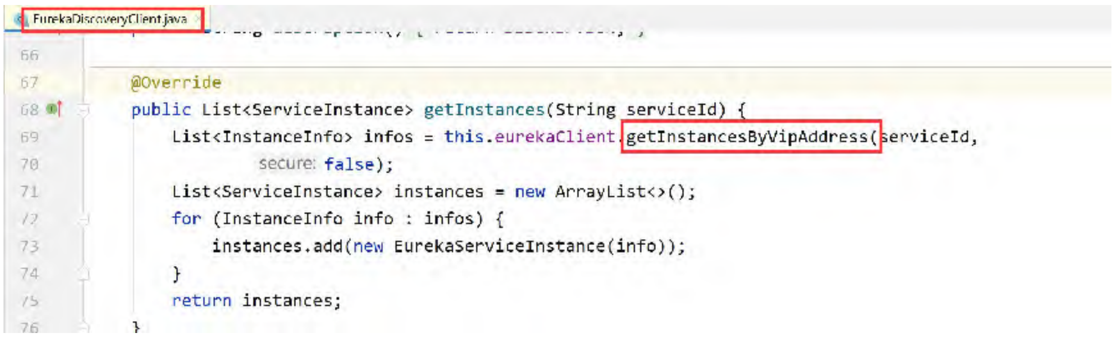
7.6.2.1 在 a 服务里面做服务发现

```java
package com.bjpowernode.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.cloud.client.ServiceInstance;
import org.springframework.cloud.client.discovery.DiscoveryClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.List;

/**
 * @Author: 北京动力节点
 */
@RestController
public class TestController {

    /**
     * 注入服务发现组件，我们的 eureka 已经实现了这个接口，所以 IOC 里面有这个对象
     */
    @Autowired
    private DiscoveryClient discoveryClient;

    /**
     * 服务发现
     *
     * @param serviceId
     * @return
     */
    @GetMapping("find")
    public String find(String serviceId) {
        //调用服务发现
        List<ServiceInstance> instances = discoveryClient.getInstances(serviceId);
        instances.forEach(System.out::print);
        return instances.toString();
    }
}
```

访问 http://localhost:8001/find?serviceId=eureka-client-b

7.6.3 服务发现的源码分析

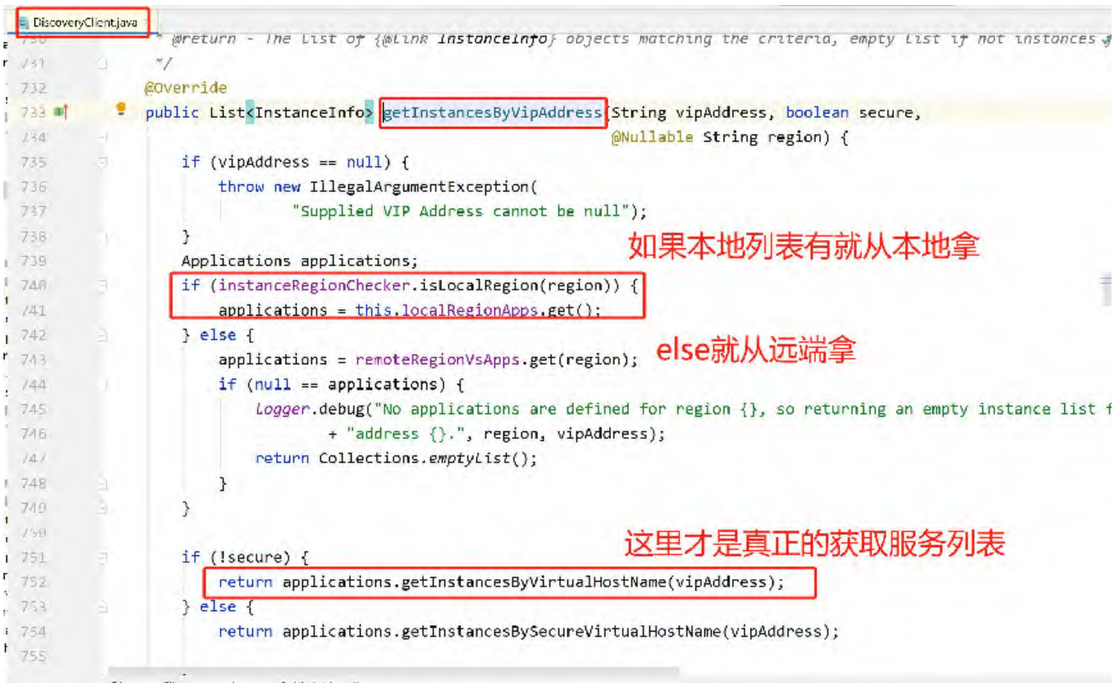
从 discoveryClient.getInstances(serviceId);方法进去，找到 eureka 的实现



EurekaDiscoveryClient.java

```java
@Override
public List<ServiceInstance> getInstances(String serviceId) {
    List<InstanceInfo> infos = this.eurekaClient.getInstancesByVipAddress(serviceId,
            secure: false);
    List<ServiceInstance> instances = new ArrayList<>();
    for (InstanceInfo info : infos) {
        instances.add(new EurekaServiceInstance(info));
    }
    return instances;
}
```

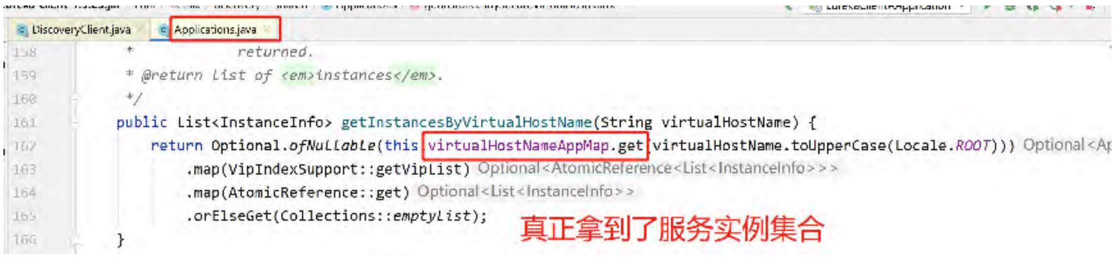
从 getInstancesByVipAddress 方法进去看到真正的服务发现



DiscoveryClient.java

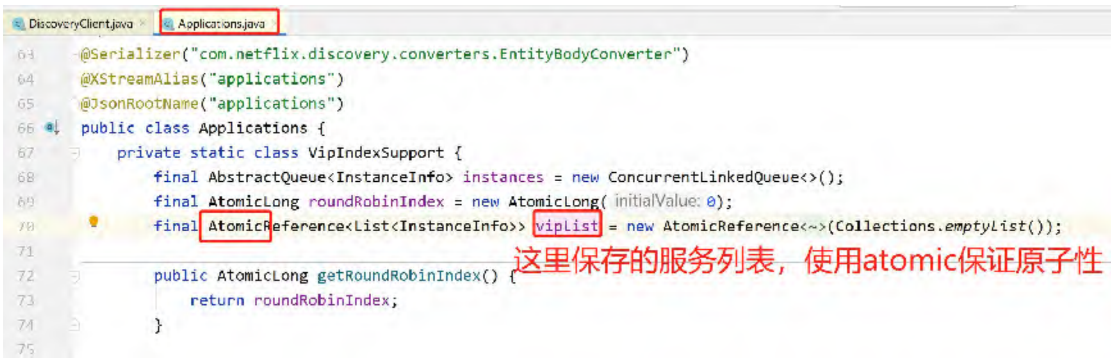
```java
 * @return - The list of {@link InstanceInfo} objects matching the criteria, empty list if not instances
 */
@Override
public List<InstanceInfo> getInstancesByVipAddress(String vipAddress, boolean secure,
                                                   @Nullable String region) {
    if (vipAddress == null) {
        throw new IllegalArgumentException(
                "Supplied VIP Address cannot be null");
    }
    Applications applications;
    if (instanceRegionChecker.isLocalRegion(region)) {
        applications = this.localRegionApps.get();
    } else {
        applications = remoteRegionVsApps.get(region);
        if (null == applications) {
            logger.debug("No applications are defined for region {}, so returning an empty instance list f
                    + "address {}.", region, vipAddress);
            return Collections.emptyList();
        }
    }

    if (!secure) {
        return applications.getInstancesByVirtualHostName(vipAddress);
    } else {
        return applications.getInstancesBySecureVirtualHostName(vipAddress);
```

在 getInstancesByVirtualHostName 方法里面做真正的服务发现



DiscoveryClient.java　Applications.java

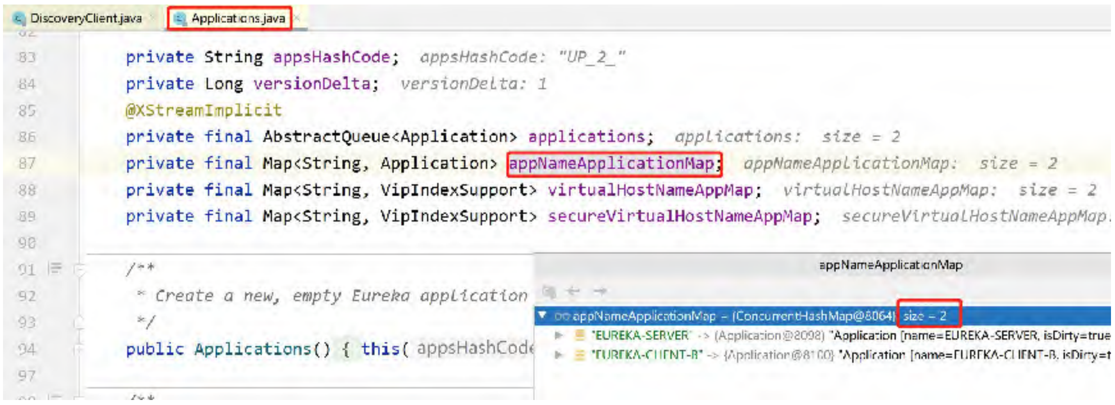
```java
 *          returned.
 * @return list of <em>instances</em>.
 */
public List<InstanceInfo> getInstancesByVirtualHostName(String virtualHostName) {
    return Optional.ofNullable(this.virtualHostNameAppMap.get(virtualHostName.toUpperCase(Locale.ROOT))) Optional<Ap
        .map(VipIndexSupport::getVipList) Optional<AtomicReference<List<InstanceInfo>>>
        .map(AtomicReference::get) Optional<List<InstanceInfo>>
        .orElseGet(Collections::emptyList);
}
```


DiscoveryClient.java | Applications.java

```
@Serializer("com.netflix.discovery.converters.EntityBodyConverter")
@XStreamAlias("applications")
@JsonRootName("applications")
public class Applications {
    private static class VipIndexSupport {
        final AbstractQueue<InstanceInfo> instances = new ConcurrentLinkedQueue<>();
        final AtomicLong roundRobinIndex = new AtomicLong( initialValue: 0);
        final AtomicReference<List<InstanceInfo>> vipList = new AtomicReference<~>(Collections.emptyList());

        public AtomicLong getRoundRobinIndex() {
            return roundRobinIndex;
        }
```

这里保存的服务列表，使用atomic保证原子性

7.6.3.1 在 eureka-client 客户端也有 map 集合存放服务列表?


DiscoveryClient.java | Applications.java

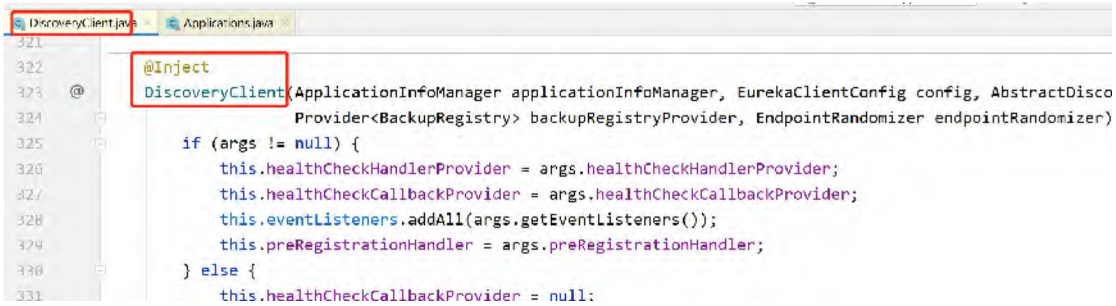
```
    private String appsHashCode;  appsHashCode: "UP_2_"
    private Long versionDelta;  versionDelta: 1
    @XStreamImplicit
    private final AbstractQueue<Application> applications;  applications:  size = 2
    private final Map<String, Application> appNameApplicationMap;  appNameApplicationMap:  size = 2
    private final Map<String, VipIndexSupport> virtualHostNameAppMap;  virtualHostNameAppMap:  size = 2
    private final Map<String, VipIndexSupport> secureVirtualHostNameAppMap;  secureVirtualHostNameAppMap

    /**
     * Create a new, empty Eureka application
     */
    public Applications() { this( appsHashCode
```

```
appNameApplicationMap
▼ appNameApplicationMap = {ConcurrentHashMap@8064} size = 2
  ▶ "EUREKA-SERVER" -> {Application@2098} "Application [name=EUREKA-SERVER, isDirty=true
  ▶ "EUREKA-CLIENT-B" -> {Application@8100} "Application [name=EUREKA-CLIENT-B, isDirty=t
```

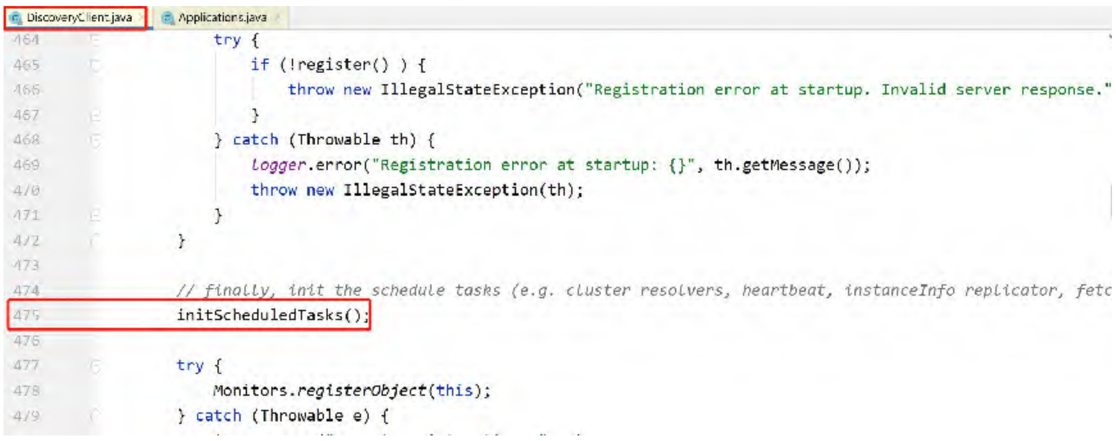
我们发现，当我们还没有做服务发现之前，集合里面已经有值了，说明项目启动的时候就去 server 端拉取服务列表并且缓存了起来

7.6.3.2 到底何时从 server 拉取服务放进去的呢?

在 eureka 的 DiscoverClient 类的一个构造方法里面，有一个任务调度线程池


DiscoveryClient.java | Applications.java

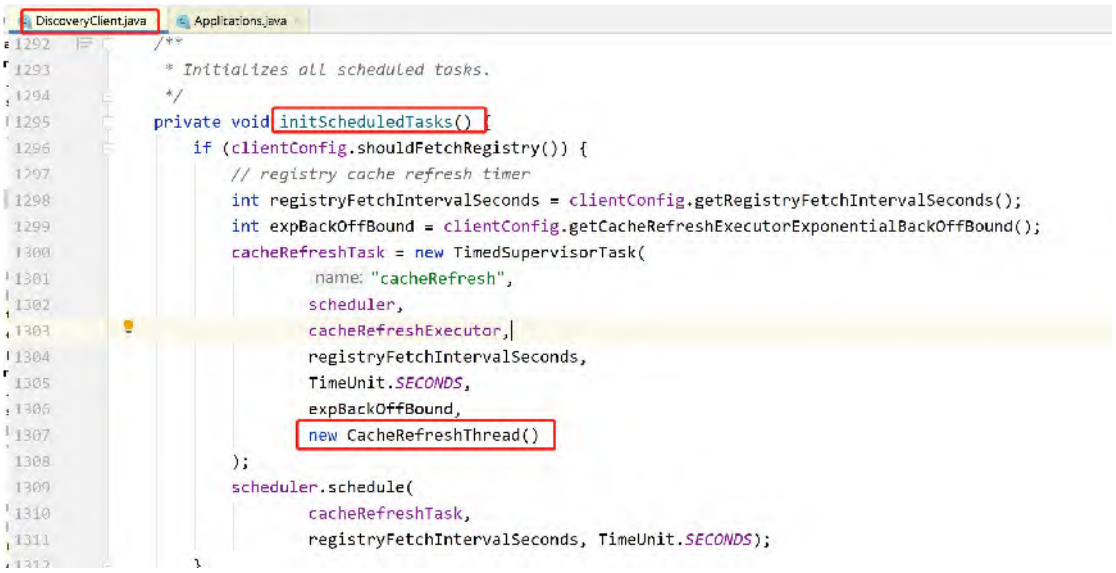
```
    @Inject
    DiscoveryClient(ApplicationInfoManager applicationInfoManager, EurekaClientConfig config, AbstractDisco
                    Provider<BackupRegistry> backupRegistryProvider, EndpointRandomizer endpointRandomizer)
        if (args != null) {
            this.healthCheckHandlerProvider = args.healthCheckHandlerProvider;
            this.healthCheckCallbackProvider = args.healthCheckCallbackProvider;
            this.eventListeners.addAll(args.getEventListeners());
            this.preRegistrationHandler = args.preRegistrationHandler;
        } else {
            this.healthCheckCallbackProvider = null;
```



DiscoveryClient.java Applications.java

```
            try {
                if (!register() ) {
                    throw new IllegalStateException("Registration error at startup. Invalid server response."
                }
            } catch (Throwable th) {
                logger.error("Registration error at startup: {}", th.getMessage());
                throw new IllegalStateException(th);
            }
        }

        // finally, init the schedule tasks (e.g. cluster resolvers, heartbeat, instanceInfo replicator, fetc
        initScheduledTasks();

        try {
            Monitors.registerObject(this);
        } catch (Throwable e) {
```

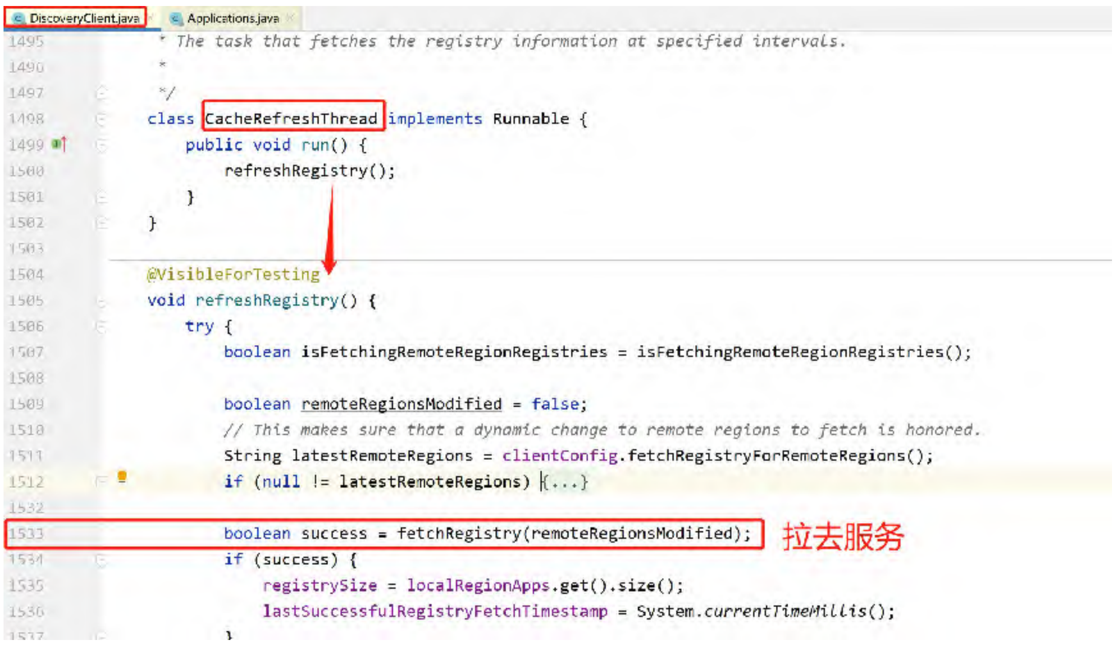
查看 initScheduledTasks()这个方法

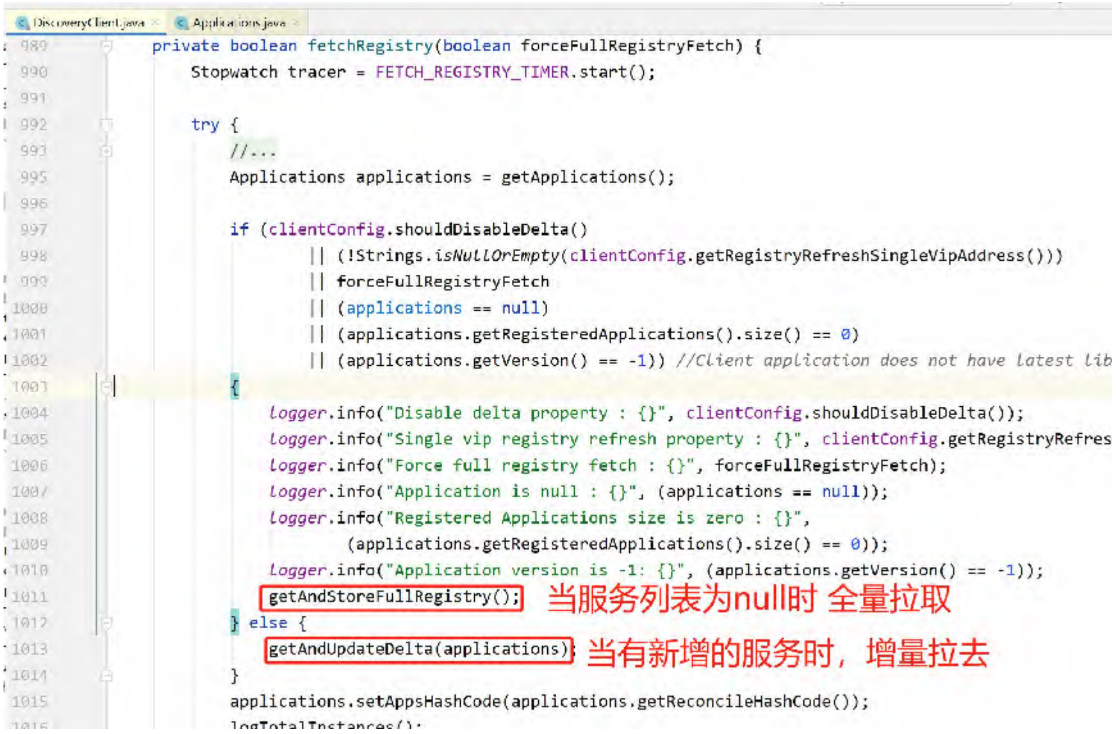


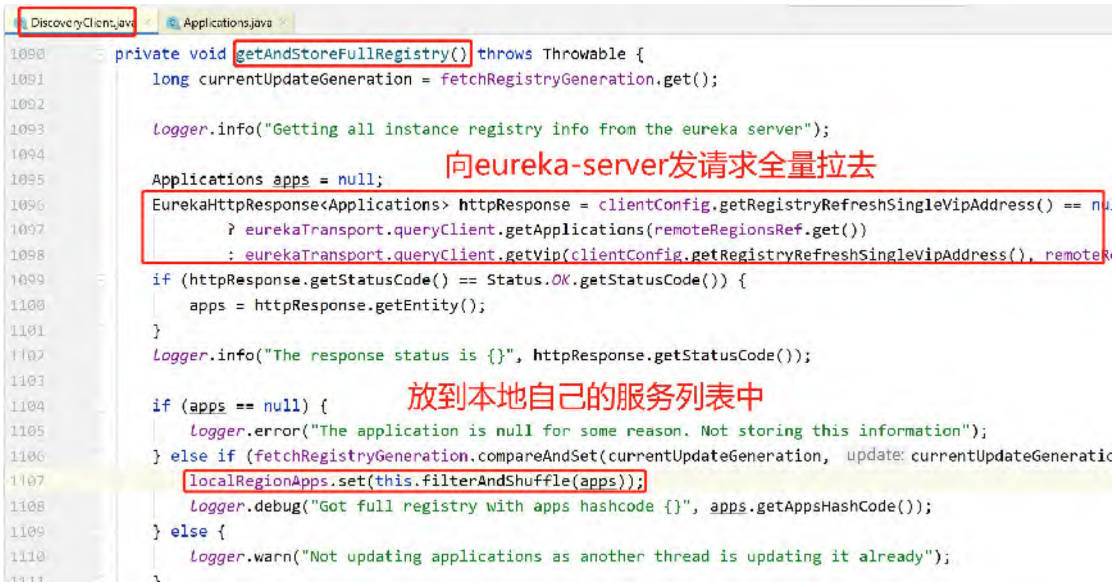
DiscoveryClient.java Applications.java

```
    /**
     * Initializes all scheduled tasks.
     */
    private void initScheduledTasks() {
        if (clientConfig.shouldFetchRegistry()) {
            // registry cache refresh timer
            int registryFetchIntervalSeconds = clientConfig.getRegistryFetchIntervalSeconds();
            int expBackOffBound = clientConfig.getCacheRefreshExecutorExponentialBackOffBound();
            cacheRefreshTask = new TimedSupervisorTask(
                    name: "cacheRefresh",
                    scheduler,
                    cacheRefreshExecutor,
                    registryFetchIntervalSeconds,
                    TimeUnit.SECONDS,
                    expBackOffBound,
                    new CacheRefreshThread()
            );
            scheduler.schedule(
                    cacheRefreshTask,
                    registryFetchIntervalSeconds, TimeUnit.SECONDS);
        }
```

在 CacheRefreshThread()中

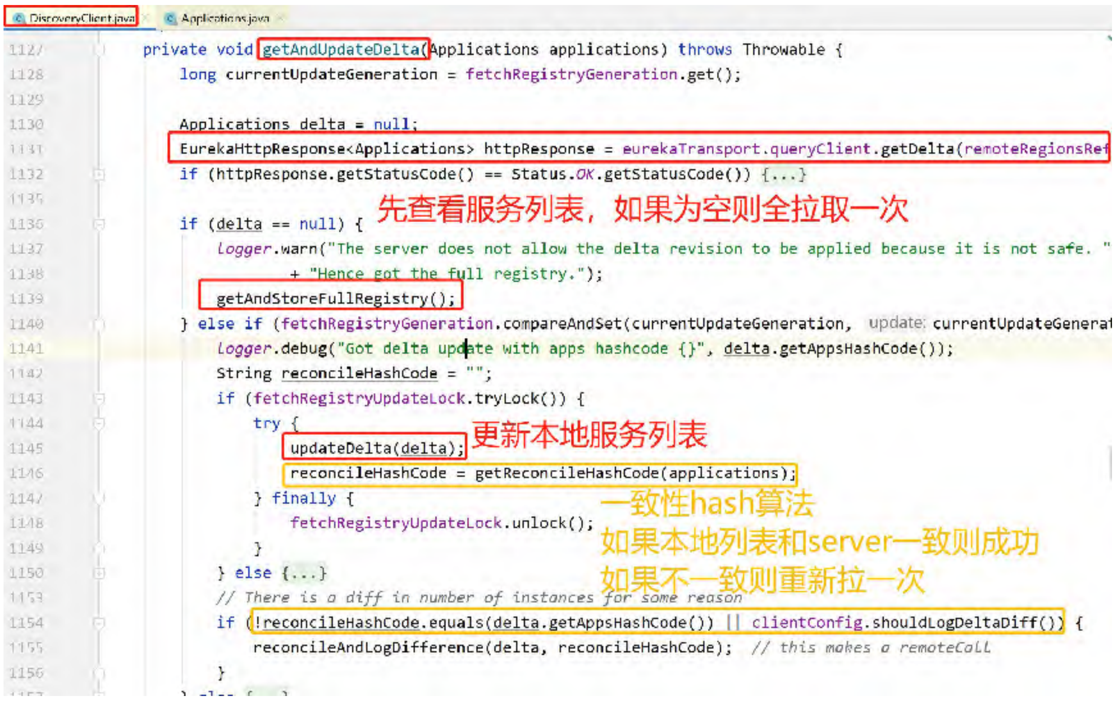

```java
DiscoveryClient.java    Applications.java
     * The task that fetches the registry information at specified intervals.
     *
     */
    class CacheRefreshThread implements Runnable {
        public void run() {
            refreshRegistry();
        }
    }

    @VisibleForTesting
    void refreshRegistry() {
        try {
            boolean isFetchingRemoteRegionRegistries = isFetchingRemoteRegionRegistries();

            boolean remoteRegionsModified = false;
            // This makes sure that a dynamic change to remote regions to fetch is honored.
            String latestRemoteRegions = clientConfig.fetchRegistryForRemoteRegions();
            if (null != latestRemoteRegions) {...}

            boolean success = fetchRegistry(remoteRegionsModified);  拉去服务
            if (success) {
                registrySize = localRegionApps.get().size();
                lastSuccessfulRegistryFetchTimestamp = System.currentTimeMillis();
            }
```
fetchRegistry()方法中判断决定是全量拉取还是增量拉取


```java
DiscoveryClient.java    Applications.java
    private boolean fetchRegistry(boolean forceFullRegistryFetch) {
        Stopwatch tracer = FETCH_REGISTRY_TIMER.start();

        try {
            //...
            Applications applications = getApplications();

            if (clientConfig.shouldDisableDelta()
                    || (!Strings.isNullOrEmpty(clientConfig.getRegistryRefreshSingleVipAddress()))
                    || forceFullRegistryFetch
                    || (applications == null)
                    || (applications.getRegisteredApplications().size() == 0)
                    || (applications.getVersion() == -1)) //Client application does not have latest lib
            {
                logger.info("Disable delta property : {}", clientConfig.shouldDisableDelta());
                logger.info("Single vip registry refresh property : {}", clientConfig.getRegistryRefres
                logger.info("Force full registry fetch : {}", forceFullRegistryFetch);
                logger.info("Application is null : {}", (applications == null));
                logger.info("Registered Applications size is zero : {}",
                        (applications.getRegisteredApplications().size() == 0));
                logger.info("Application version is -1: {}", (applications.getVersion() == -1));
                getAndStoreFullRegistry();  当服务列表为null时 全量拉取
            } else {
                getAndUpdateDelta(applications);  当有新增的服务时，增量拉去
            }
            applications.setAppsHashCode(applications.getReconcileHashCode());
            logTotalInstances();
```
getAndStoreFullRegistry()全量拉取



```java
private void getAndStoreFullRegistry() throws Throwable {
    long currentUpdateGeneration = fetchRegistryGeneration.get();

    logger.info("Getting all instance registry info from the eureka server");

    Applications apps = null;
    EurekaHttpResponse<Applications> httpResponse = clientConfig.getRegistryRefreshSingleVipAddress() == nu
            ? eurekaTransport.queryClient.getApplications(remoteRegionsRef.get())
            : eurekaTransport.queryClient.getVip(clientConfig.getRegistryRefreshSingleVipAddress(), remoteR
    if (httpResponse.getStatusCode() == Status.OK.getStatusCode()) {
        apps = httpResponse.getEntity();
    }
    logger.info("The response status is {}", httpResponse.getStatusCode());

    if (apps == null) {
        logger.error("The application is null for some reason. Not storing this information");
    } else if (fetchRegistryGeneration.compareAndSet(currentUpdateGeneration,  update: currentUpdateGeneratic
        localRegionApps.set(this.filterAndShuffle(apps));
        logger.debug("Got full registry with apps hashcode {}", apps.getAppsHashCode());
    } else {
        logger.warn("Not updating applications as another thread is updating it already");
```

getAndUpdateDelta()增量拉取



```java
private void getAndUpdateDelta(Applications applications) throws Throwable {
    long currentUpdateGeneration = fetchRegistryGeneration.get();

    Applications delta = null;
    EurekaHttpResponse<Applications> httpResponse = eurekaTransport.queryClient.getDelta(remoteRegionsRef
    if (httpResponse.getStatusCode() == Status.OK.getStatusCode()) {...}

    if (delta == null) {
        logger.warn("The server does not allow the delta revision to be applied because it is not safe. "
                + "Hence got the full registry.");
        getAndStoreFullRegistry();
    } else if (fetchRegistryGeneration.compareAndSet(currentUpdateGeneration,  update: currentUpdateGenerat
        logger.debug("Got delta update with apps hashcode {}", delta.getAppsHashCode());
        String reconcileHashCode = "";
        if (fetchRegistryUpdateLock.tryLock()) {
            try {
                updateDelta(delta);
                reconcileHashCode = getReconcileHashCode(applications);
            } finally {
                fetchRegistryUpdateLock.unlock();
            }
        } else {...}
        // There is a diff in number of instances for some reason
        if (!reconcileHashCode.equals(delta.getAppsHashCode()) || clientConfig.shouldLogDeltaDiff()) {
            reconcileAndLogDifference(delta, reconcileHashCode);  // this makes a remoteCall
        }
```

7.6.3.3 服务发现总结

重要的类：

DiscoveryClient 类里面的构造方法执行线程初始化调用

CacheRefreshThread 类里面的 run 方法执行服务列表的拉取（方便后期做服务发现）

fetchRegistry()方法去判断全量拉取还是增量拉取

全量拉取发生在：当服务列表为 null 的情况 当项目刚启动就全量拉取
增量拉取发生：当列表不为 null ，只拉取 eureka-server 的修改的数据(注册新的服务，上线服务)

eureka 客户端会把服务列表缓存到本地 为了提高性能

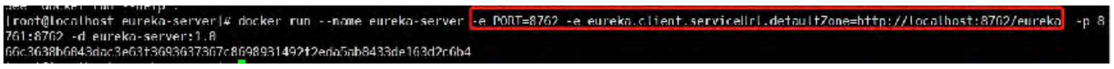
但是有脏读问题，当你启动一个新的应用的时候 不会被老的应用快速发现

8.Eureka-docker 部署

8.1 打包 eureka-server 前修改配置文件，可自定义

```yaml
server:
    port: ${PORT:8761} #为什么是8761，其他端口就会报错
spring:
    application:
        name: eureka-server #应用名称
eureka:
    client:
        service-url: #我们可以定义注册自己的地址，从这里入手查看源码
            defaultZone: ${EUREKA_SERVER:http://localhost:8761/eureka}
        fetch-registry: true    #是否拉去服务列表
        register-with-eureka: true  #是否将自己注册到eureka 上(集群需要注册自己到eureka 上去)
    server:
        eviction-interval-timer-in-ms: 90000    #清除无效节点的评率(毫秒)
    instance:
        lease-expiration-duration-in-seconds: 90    #server 在等待下一个客户端发送的心跳时间，若
在指定时间不能收到客户端心跳，则剔除此实例并且禁止流量
        instance-id: ${eureka.instance.hostname}:${spring.application.name}:${server.port}
        hostname: ${APP_HOST:locahost} #主机地址
        prefer-ip-address: ${IP_ADDRESS:true} #显示名称
```

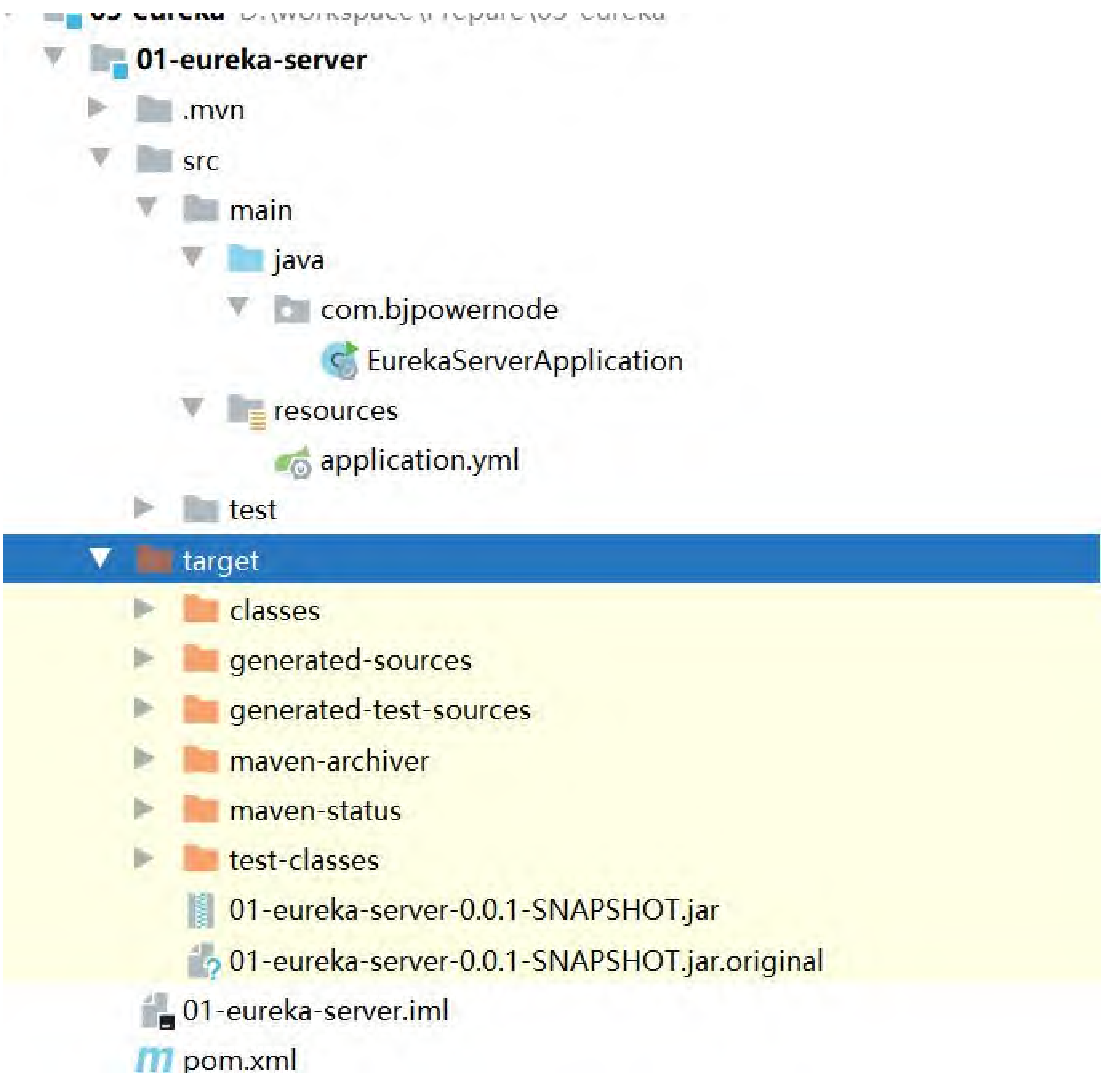
可以用这种方式把变量写活，不要写死，docker 在运行的时候是从环境变量里面去取值的，很多项目在部署的时候都需要稍微修改的。如下



```
[root@localhost eureka-server]# docker run --name eureka-server -e PORT=8762 -e eureka.client.serviceUrl.defaultZone=http://localhost:8762/eureka -p 8761:8762 -d eureka-server:1.0
66c3638b6843dac3e63f3693637307c8698931492f2eda5ab8433de163d2c6b4
```

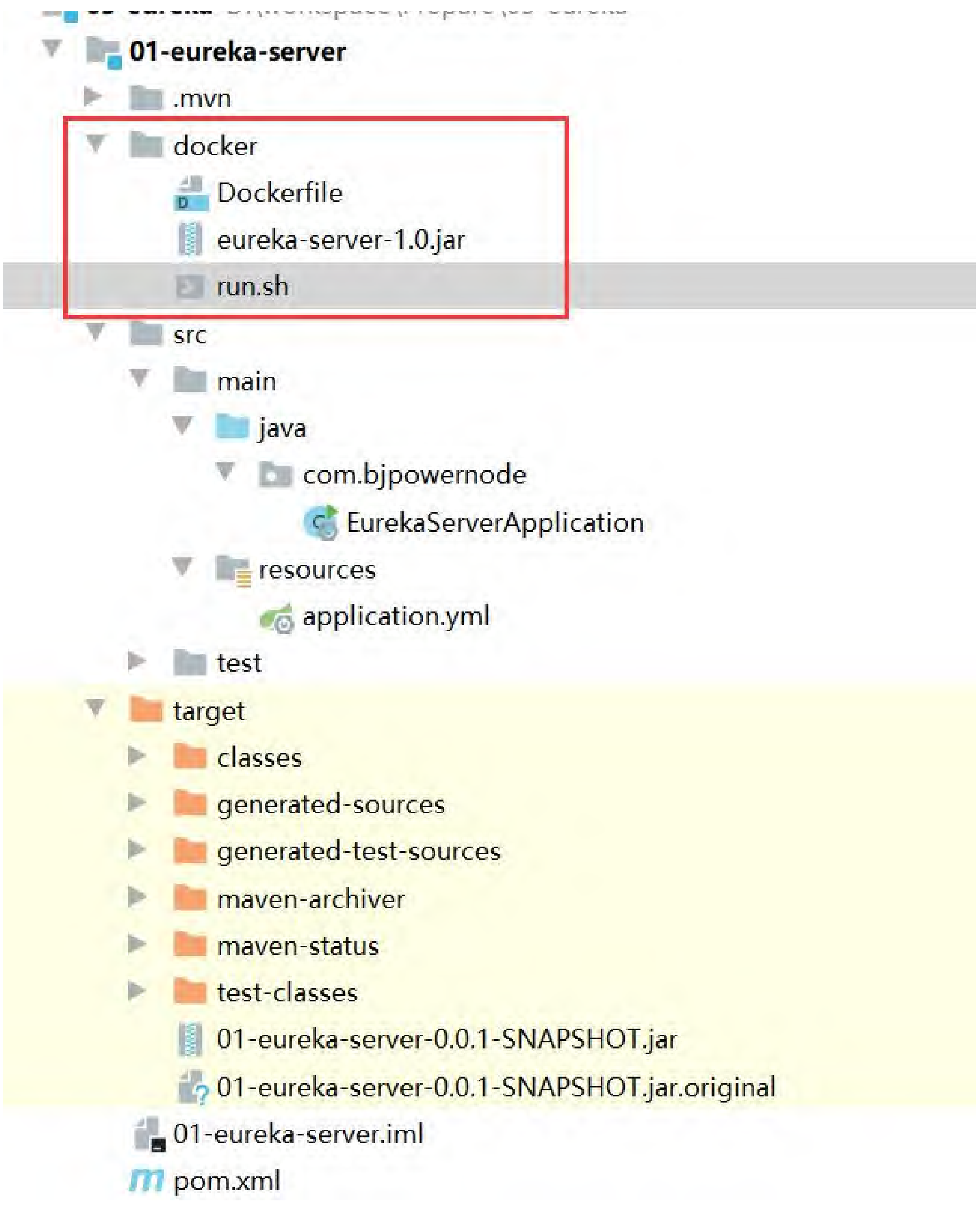
8.2 打包 eureka-server

Eureka-server 本质就是一个 springboot 项目，我们用自带的 maven 打包插件打成 jar 包



```
▼ 01-eureka-server
    ▶ .mvn
    ▼ src
        ▼ main
            ▼ java
                ▼ com.bjpowernode
                      EurekaServerApplication
            ▼ resources
                  application.yml
        ▶ test
    ▼ target
        ▶ classes
        ▶ generated-sources
        ▶ generated-test-sources
        ▶ maven-archiver
        ▶ maven-status
        ▶ test-classes
          01-eureka-server-0.0.1-SNAPSHOT.jar
          01-eureka-server-0.0.1-SNAPSHOT.jar.original
      01-eureka-server.iml
      pom.xml
```

8.3 创建文件夹，编写 Dockerfile 和 run.sh 脚本



Dockerfile

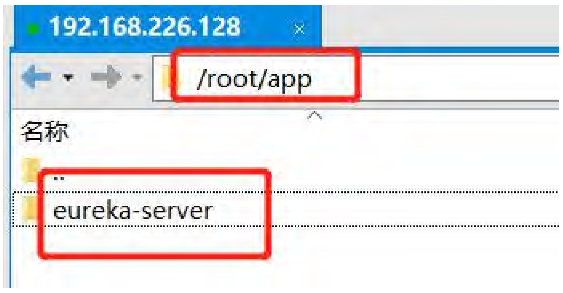
```
FROM openjdk:8
ENV workdir=/root/app/eureka-server
COPY . ${workdir}
WORKDIR ${workdir}
EXPOSE 8761
```

```
CMD ["java","-jar","eureka-server-1.0.jar"]
```

run.sh

```
cd .. && docker build ./eureka-server -t eureka-server:1.0
```

8.4 在服务器创建文件夹，注意路径和名称



这里的路径和 Dockerfile 里面的 env 变量一致

文件夹名称和 run.sh 脚本里的一致

8.5 执行构建和运行

修改 shell 脚本权限

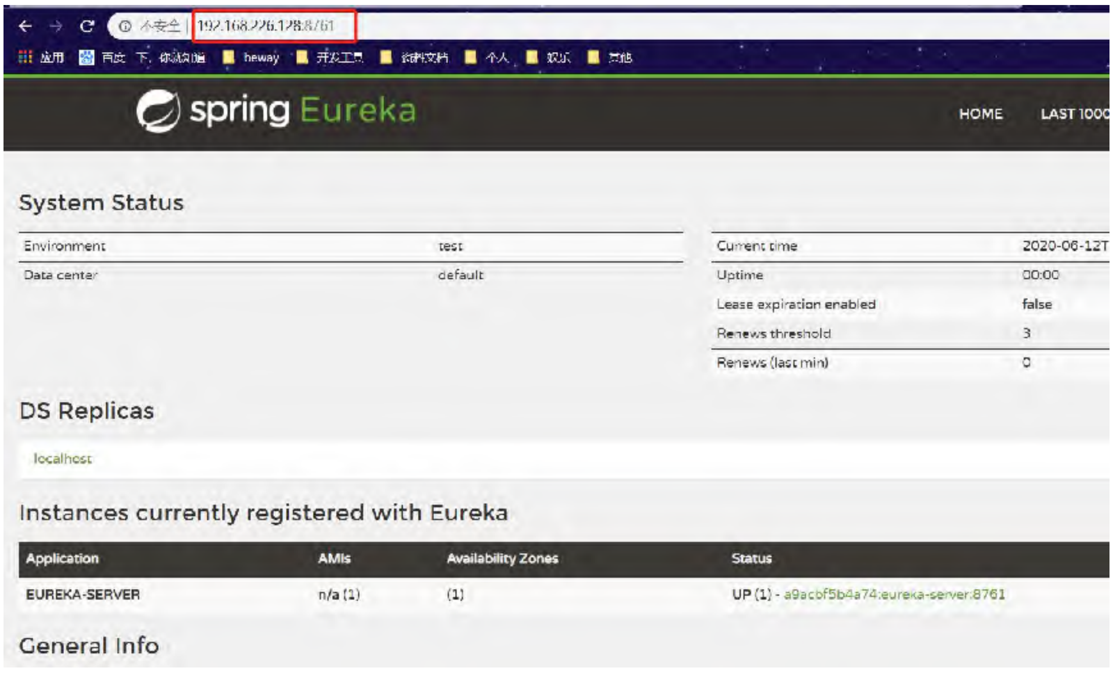
1.chmod 777 run.sh

2.执行./run.sh 或者在/root/app/ 路径下执行

```
docker build ./eureka-server -t eureka-server:1.0
```

3.执行 docker run --name eureka-server -p 8761:8761 -d eureka-server:1.0

8.6 测试访问



不安全 | 192.168.226.128:8761
应用 百度一下，你就知道 heway 开发工具 资料文档 个人 娱乐 其他
spring Eureka
HOME
LAST 1000

System Status

Environment	test
Data center	default

Current time	2020-06-12T
Uptime	00:00
Lease expiration enabled	false
Renews threshold	3
Renews (last min)	0

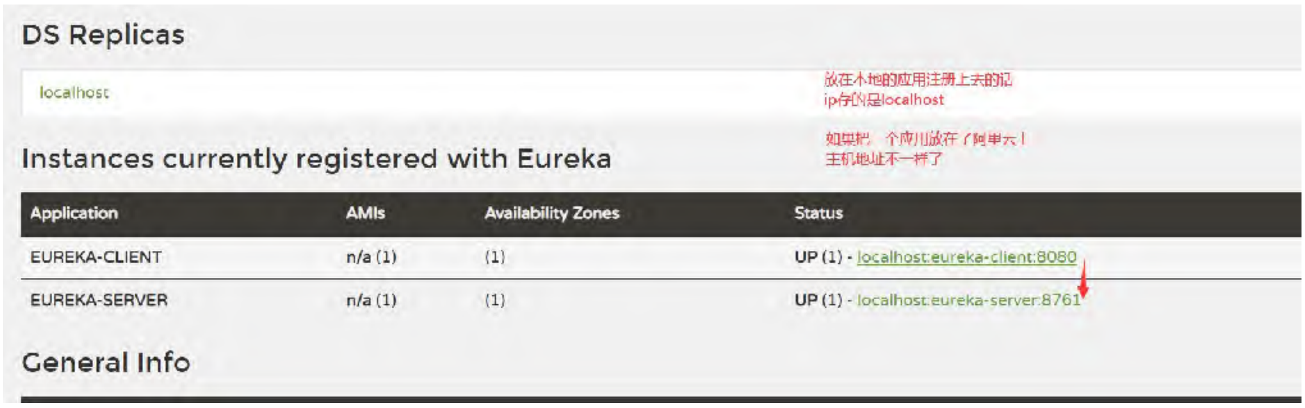
DS Replicas

localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
EUREKA-SERVER	n/a (1)	(1)	UP (1) - a9acbf5b4a74:eureka-server:8761

General Info



DS Replicas

localhost

放在本地的应用注册上去的话
ip存的是localhost

如果把一个应用放在了阿里云上
主机地址不一样了

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
EUREKA-CLIENT	n/a (1)	(1)	UP (1) - localhost:eureka-client:8080
EUREKA-SERVER	n/a (1)	(1)	UP (1) - localhost:eureka-server:8761

General Info

在开发阶段 最好都在一个局域网内网开发

部署阶段 都是公网地址