

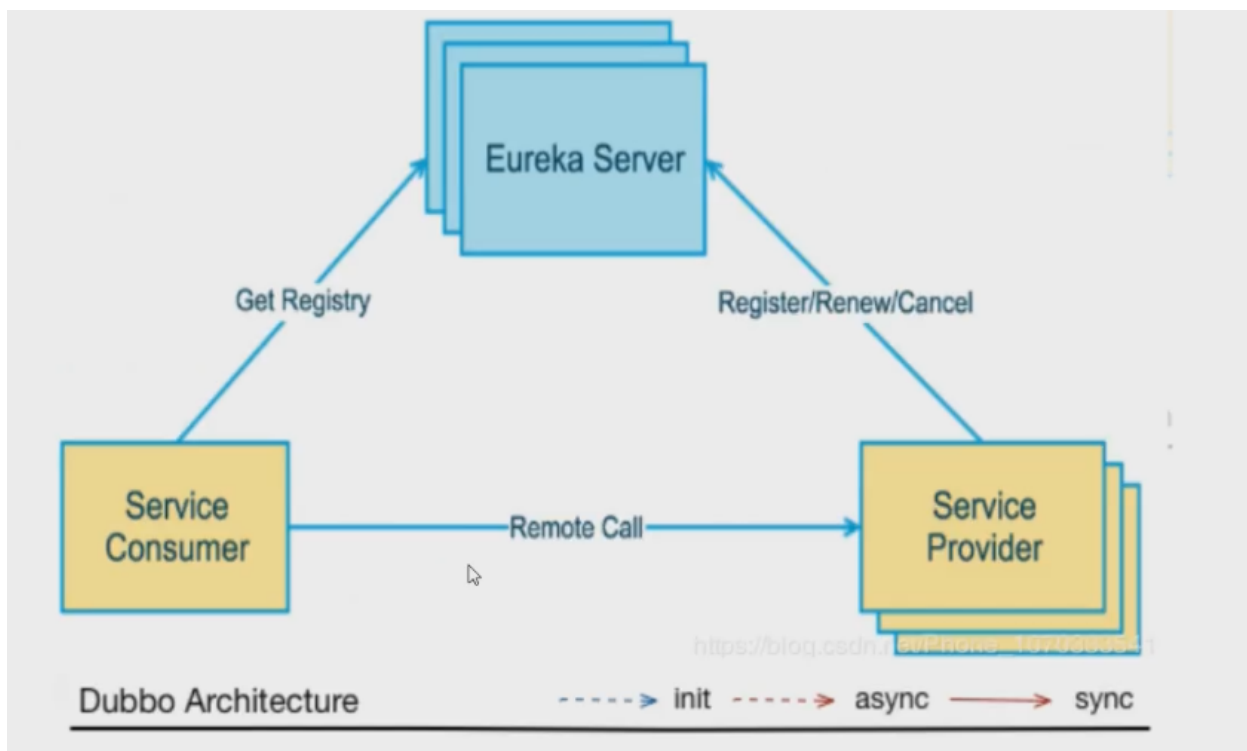
Eureka服务注册与发现

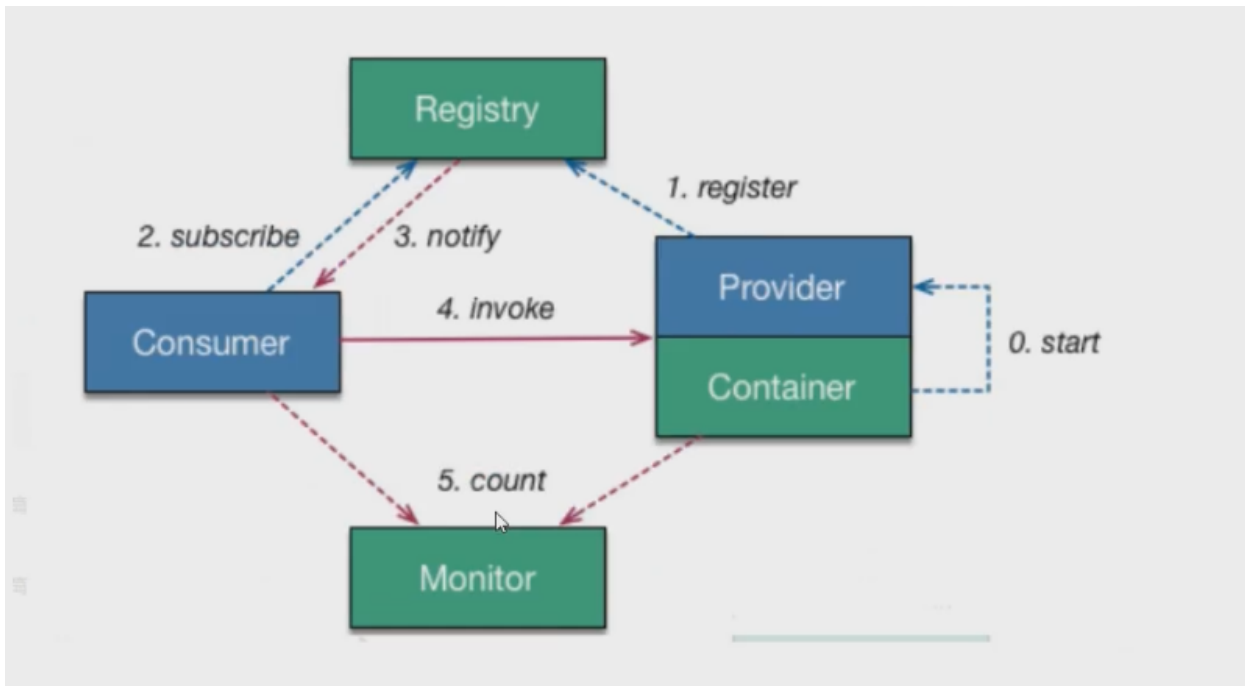
5.1、什么是Eureka

- Eureka：怎么读？
- Netflix 在设计Eureka时，遵循的就是AP原则
- Eureka是Netflix的一个子模块，也是核心模块之一。Eureka是一个基于REST的服务，用于定位服务，以实现云端中间层服务发现和故障转移，服务注册与发现对于微服务来说是非常重要的，有了服务发现与注册，只需要使用服务的标识符，就可以访问到服务，而不需要修改服务调用的配置文件了，功能类似于Dubbo的注册中心，比如Zookeeper；

5.2、原理讲解

- Eureka的基本架构
 - SpringCloud 封装了NetFlix公司开发的Eureka模块来实现服务注册和发现（对比Zookeeper）
 - Eureka采用了C-S的架构设计，EurekaServer 作为服务注册功能的服务器，他是服务注册中心
 - 而系统中的其他微服务。使用Eureka的客户端连接到EurekaServer并维持心跳连接。这样系统的维护人员就可以通过EurekaServer来监控系统中各个微服务是否正常运行，SpringCloud的一些其他模块（比如Zuul）就可以通过EurekaServer来发现系统中的其他微服务，并执行相关的逻辑；
 - 和Dubbo架构对比



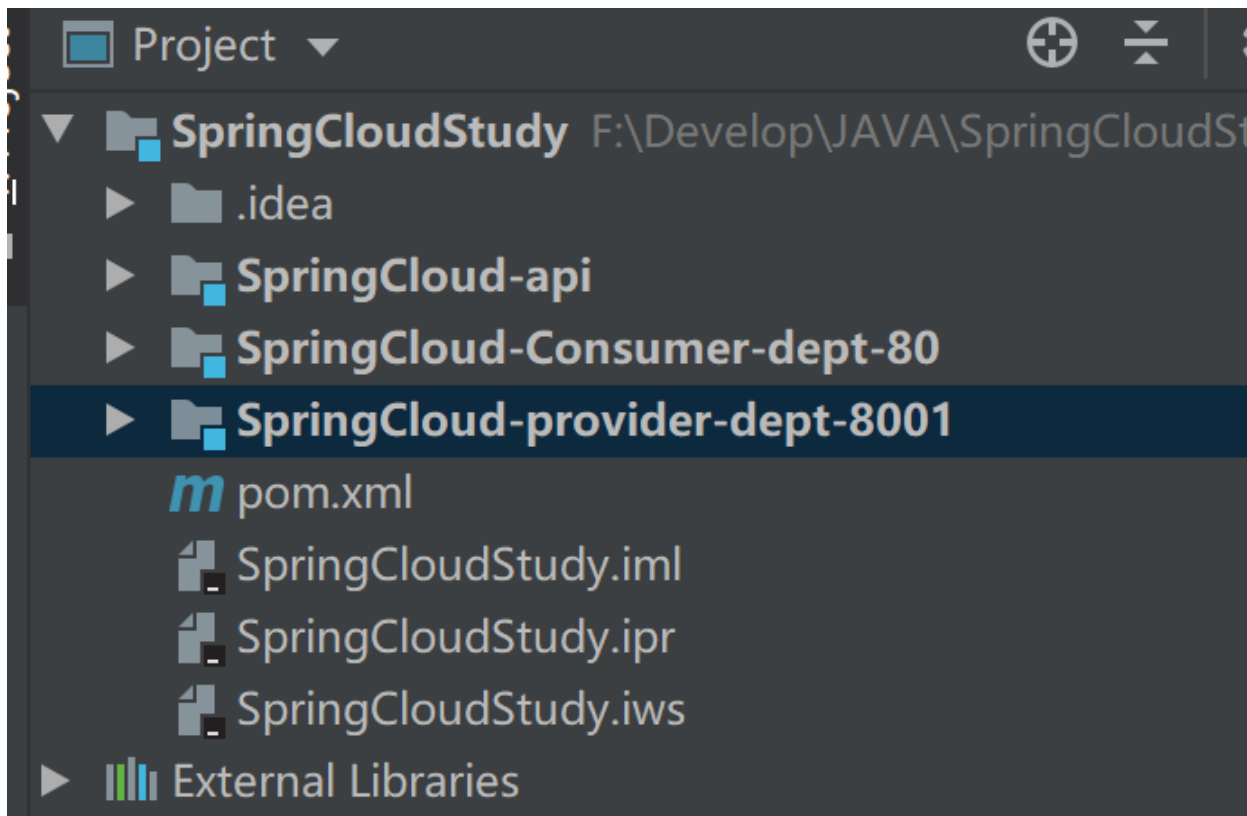


- Eureka 包含两个组件：**Eureka Server** 和 **Eureka Client**。
- Eureka Server 提供服务注册服务，各个节点启动后，会在 EurekaServer 中进行注册，这样 Eureka Server 中的服务注册表中将会存储所有可用服务节点的信息，服务节点的信息可以在界面中直观的看到。
- Eureka Client 是一个 Java 客户端，用于简化 EurekaServer 的交互，客户端同时也具备一个内置的，使用轮询负载算法的负载均衡器。在应用启动后，将会向 EurekaServer 发送心跳（默认周期为 30 秒）。如果 Eureka Server 在多个心跳周期内没有接收到某个节点的心跳，EurekaServer 将会从服务注册表中把这个服务节点移除掉（默认周期为 90 秒）

- 三大角色

- Eureka Server：提供服务的注册与发现。
- Service Provider：将自身服务注册到 Eureka 中，从而使消费方能够找到。
- Service Consumer：服务消费方从 Eureka 中获取注册服务列表，从而找到消费服务。

目前架构状态。



SpringCloud-Consumer-dept-80: 服务消费者

SpringCloud-provider-dept-8001: 服务提供者

因此: 我们需要创建一个提供注册与发现的服务

演示详解:

一、创建提供注册与发现的服务模块



SpringCloud-Consumer-dept-80: 服务消费者

SpringCloud-provider-dept-8001: 服务提供者

SpringCloud-EurEka-Server-8002: 提供注册与发现的服务

1. 导入依赖

```

1 <dependencies>
2 <!-- https://mvnrepository.com/artifact/org.springframework.cloud/spring
   -cloud-starter-eureka-server -->
3 <dependency>
4 <groupId>org.springframework.cloud</groupId>
5 <artifactId>spring-cloud-starter-eureka-server</artifactId>
6 <version>1.4.7.RELEASE</version>
7 </dependency>
8 <!--热部署-->
9 <dependency>
10 <groupId>org.springframework.boot</groupId>
11 <artifactId>spring-boot-devtools</artifactId>
12 </dependency>
13 </dependencies>
14 ##注意到点：版本对应问题，Springboot最好使用2.2.x的

```

2. 编写配置文件

application.yml

```

1 #配置端口
2 server:
3   port: 8002
4
5 #Eureka配置
6 eureka:
7   instance:
8     hostname: localhost #Eureka服务器实例名称
9   client:
10     register-with-eureka: false #表示是否向Eureka注册自己，既然是server提供注册与发现 所以自己不注册的
11     fetch-registry: false #fetch-registry如果为false，则表示自己为注册中心
12     service-url: #监控页面 我们访问的 提供监控的页面
13     defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/

```

3. 开启功能

```

1 @SpringBootApplication
2 @EnableEurekaServer //EnableEurekaServer服务端的启动类，可以接受别人注册进来
3 public class SpringCloudEurekaServer_8002 {
4   public static void main(String[] args) {
5     SpringApplication.run(SpringCloudEurekaServer_8002.class,args);
6   }
7 }

```

访问监控页面测试，是否开启注册与发现成功！

System Status			
Environment	test	Current time	2020-12-21T15:57:58 +0800
Data center	default	Uptime	00:00
		Lease expiration enabled	false
		Renews threshold	1
		Renews (last min)	0
DS Replicas			
Instances currently registered with Eureka			
Application	AMIs	Availability Zones	Status
No instances available			
General Info			
Name	Value		
total-avail-memory	456mb		
environment	test		
num-of-cpus	16		
current-memory-usage	89mb (19%)		

4. 配置类的实现。（需要的情况）

二、提供服务到注册中心，供消费者查询

SpringCloud-provider-dept-8001：提供服务到注册中心，供消费者查询

1. 导入依赖

```
1  ##在原先的基础上加上eureka客户端和 eureka -->完善监控信息（看个人需求，一般团队开发需要用到监控信息）
2  <!--eureka客户端-->
3  <!-- https://mvnrepository.com/artifact/org.springframework.cloud/spring-cloud-starter-netflix-eureka-client -->
4  <dependency>
5    <groupId>org.springframework.cloud</groupId>
6    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
7    <version>2.2.5.RELEASE</version>
8  </dependency>
9  <!--eureka -->完善监控信息-->
10 <dependency>
11   <groupId>org.springframework.boot</groupId>
12   <artifactId>spring-boot-starter-actuator</artifactId>
13 </dependency>
```

2. 编写配置文件

```
1  #Eureka配置 我们应该将服务提供到哪里去（注册中心）
2  eureka:
```

```

3  client:
4  service-url:
5  defaultZone: http://localhost:8002/eureka/
6  instance: #（这里是下面图示）
7  instance-id: SpringCloud-provider-dept-8001 #可以通过这个任意修改Eureka注册实例显示的status信息
8
9  #Info配置 也就是监控信息 （这里是下面图示）
10 info:
11  app.name: lidadaibiao
12  app.email: 945827167@qq.com

```

3. 开启功能

```

1
2 @SpringBootApplication
3 @EnableEurekaClient //在服务启动后自动注册到Eureka中 ---》springcloud-eureka-8002
4 @EnableDiscoveryClient //服务发现，针对团队开发 下面会记载到。
5 public class DeptProvider_8001 {
6     public static void main(String[] args) {
7         SpringApplication.run(DeptProvider_8001.class,args);
8     }
9 }

```

先启动SpringCloud-Eureka-Server-8002：注册与发现 再启动 SpringCloud-provider-dept-8001：提供服务到注册中心，供消费者查询

```

2020-12-21 16:22:54.652 INFO 15276 --- [ restartedMain] com.netflix.discovery.DiscoveryClient : Discovery Client initialized at timestamp 1608538974650 with initial
2020-12-21 16:22:54.654 INFO 15276 --- [ restartedMain] o.s.c.n.e.s.EurekaServiceRegistry : Registering application SPRINGCLOUD-PROVIDER-DEPT with eureka with
2020-12-21 16:22:54.655 INFO 15276 --- [ restartedMain] com.netflix.discovery.DiscoveryClient : Saw local status change event StatusChangeEvent [timestamp=1608538
2020-12-21 16:22:54.657 INFO 15276 --- [nfoReplicator-0] com.netflix.discovery.DiscoveryClient : DiscoveryClient_SPRINGCLOUD-PROVIDER-DEPT/SpringCloud-provider-dep
2020-12-21 16:22:54.696 INFO 15276 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8001 (http) with context path ''
2020-12-21 16:22:54.697 INFO 15276 --- [ restartedMain] .s.c.n.e.s.EurekaAutoServiceRegistration : Updating port to 8001
2020-12-21 16:22:54.720 INFO 15276 --- [nfoReplicator-0] com.netflix.discovery.DiscoveryClient : DiscoveryClient_SPRINGCLOUD-PROVIDER-DEPT/SpringCloud-provider-dep
2020-12-21 16:22:54.959 INFO 15276 --- [ restartedMain] c.l.springcloud.DeptProvider_8001 : Started DeptProvider_8001 in 5.298 seconds (JVM running for 6.099)
2020-12-21 16:22:55.390 INFO 15276 --- [192.168.101.103] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2020-12-21 16:22:55.390 INFO 15276 --- [192.168.101.103] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2020-12-21 16:22:55.397 INFO 15276 --- [192.168.101.103] o.s.web.servlet.DispatcherServlet : Completed initialization in 7 ms
2020-12-21 16:22:55.400 ERROR 15276 --- [192.168.101.103] com.alibaba.druid.pool.DruidDataSource : testWhileIdle is true, validationQuery not set
2020-12-21 16:22:55.407 INFO 15276 --- [192.168.101.103] com.alibaba.druid.pool.DruidDataSource : {dataSource-1} inited
Mon Dec 21 16:22:55 CST 2020 WARN: Establishing SSL connection without server's identity verification is not recommended. According to MySQL 5.5.45+, 5.6.26+ and 5.7.6

```

访问测试：

我们可以在监控页面进行服务提供实例查看

← → ↻ localhost:8002 ☆ 更新

System Status

Environment	test	Current time	2020-12-21T16:23:44 +0800
Data center	default	Uptime	00:01
		Lease expiration enabled	false
		Renews threshold	3
		Renews (last min)	1

DS Replicas

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
SPRINGCLOUD-PROVIDER-DEPT	n/a (1)	(1)	UP (1) - SpringCloud-provider-dept-8001

General info

服务提供实例

- 1 instance:
- 2 instance-id: SpringCloud-provider-dept-8001 #可以通过这个任意修改Eureka注册实例显示的status信息

Application	AMIs	Availability Zones	Status
SPRINGCLOUD-PROVIDER-DEPT	n/a (1)	(1)	UP (1) - SpringCloud-provider-dept-8001

当我们点击进去查看info的时候，会显示我们所配置的监控信息

- 1 #Info配置 也就是监控信息 （这里是下面图示）
- 2 info:
- 3 app.name: lidadaibiao
- 4 app.email: 945827167@qq.com
- 5 <!--eureka -->完善监控信息--> 看网址 actuator/info
- 6 <dependency>
- 7 <groupId>org.springframework.boot</groupId>
- 8 <artifactId>spring-boot-starter-actuator</artifactId>
- 9 </dependency>

← → ↻ ⚠ 不安全 | laptop-sfulgcaf:8001/actuator/info

```
{"app":{"name":"lidadaibiao","email":"945827167@qq.com"}}
```

4. 配置类的实现。（需要的情况下）

三、Eureka的自我保护机制

场景：当我们突然中断提供服务者，发现注册与发现的监控页面会出现以下情况。

System Status

Environment	test	Current time	2020-12-21T16:36:29 +0800
Data center	default	Uptime	00:13
		Lease expiration enabled	false
		Renews threshold	3
		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

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
SPRINGCLOUD-PROVIDER-DEPT	n/a (1)	(1)	UP (1) - SpringCloud-provider-dept-8001

还在

General Info

这是Eureka的自我保护机制：

自我保护机制：好死不如赖活着

一句话总结：某时刻某一个微服务不可以用了，eureka不会立刻清理，依旧会对该微服务的信息进行保存！

- 默认情况下，如果EurekaServer在一定时间内没有接收到某个微服务实例的心跳，EurekaServer将会注销该实例（默认90秒）。但是当网络分区故障发生时，微服务与Eureka之间无法正常通行，以上行为可能变得非常危险了--因为微服务本身其实是健康的，此时本不应该注销这个服务。Eureka通过 **自我保护机制** 来解决问题--当EurekaServer节点在短时间内丢失过多客户端时（可能发生了网络分区故障），那么这个节点就会进入自我保护模式。一旦进入该模式，EurekaServer就会保护服务注册表中的信息，不再删除服务注册表中的数据（也就是不会注销任何微服务）。当网络故障恢复后，该EurekaServer节点会自动退出自我保护模式。
- 在自我保护模式中，EurekaServer会保护服务注册表中的信息，不再注销任何服务实例。当它收到的心跳数重新恢复到阈值以上时，该EurekaServer节点就会自动退出自我保护模式。它的设计哲学就是宁可保留错误的服务注册信息，也不盲目注销任何可能健康的服务实例。一句话：好死不如赖活着
- 综上，自我保护模式是一种应对网络异常的安全保护措施。它的架构哲学是宁可同时保留所有微服务（健康的微服务和不健康的微服务都会保留），也不盲目注销任何健康的微服务。使用自我保护模式，可以让Eureka集群更加的健壮和稳定
- 在SpringCloud中，可以使用 `eureka.server.enable-self-preservation = false` 禁用自我保护模式【不推荐关闭自我保护机制】

自我保护机制：好死不如赖活着

一句话总结：某时刻某一个微服务不可以用了，eureka不会立刻清理，依旧会对该微服务的信息进行保存！

- 默认情况下，如果EurekaServer在一定时间内没有接收到某个微服务实例的心跳，EurekaServer将会注销该实例（默认90秒）。但是当网络分区故障发生时，微服务与Eureka之间无法正常通行，以上行为可能变得非常危险了--因为微服务本身其实是健康的，**此时本不应该注销这个服务**。Eureka通过 **自我保护机制** 来解决问题--当EurekaServer节点在短时间丢失过多客户端时（可能发生了网络分区故障），那么这个节点就会进入自我保护模式。一旦进入该模式，EurekaServer就会保护服务注册表中的信息，不再删除服务注册表中的数据（也就是不会注销任何微服务）。当网络故障恢复后，该EurekaServer节点会自动退出自我保护模式。
- 在自我保护模式中，EurekaServer会保护服务注册表中的信息，不再注销任何服务实例。当它收到的心跳数重新恢复到阈值以上时，该EurekaServer节点就会自动退出自我保护模式。它的设计哲学就是宁可保留错误的服务注册信息，也不盲目注销任何可能健康的服务实例。一句话：好死不如赖活着
- 综上，自我保护模式是一种应对网络异常的安全保护措施。它的架构哲学是宁可同时保留所有微服务（健康的微服务和不健康的微服务都会保留），也不盲目注销任何健康的微服务。使用自我保护模式，可以让Eureka集群更加的健壮和稳定
- 在SpringCloud中，可以使用 `eureka.server.enable-self-preservation = false` 禁用自我保护模式【不推荐关闭自我保护机制】

四、Eureka的服务发现 (Discovery)

- 对于注册到Eureka里面的微服务，我们可以通过服务发现来获得该服务信息（向外暴露服务信息）
- 编写我们提供服务到注册中心的，提供服务者Controller层进行服务发现

```
1 @Autowired
2 private DiscoveryClient discoveryClient; #注入服务发现客户端 import org.springframework.cloud.client.discovery.DiscoveryClient;
3 @GetMapping("/eureka/discovery/")
4 public Object discovery(){
5     //获得微服务列表的清单
6     List<String> services = discoveryClient.getServices();
7     System.out.println("微服务清单: "+services);
8     //得到一个具体的，微服务实例信息 getInstances(String serviceId); serviceId 服务ID
9     List<ServiceInstance> serviceInstanceList =
10     discoveryClient.getInstances("SPRINGCLOUD-PROVIDER-DEPT");
11     for (ServiceInstance serviceInstance : serviceInstanceList) {
12         System.out.println("URI:"+serviceInstance.getUri());
13         System.out.println("port:"+serviceInstance.getPort());
14         System.out.println("host:"+serviceInstance.getHost());
15         System.out.println("ServiceId:"+serviceInstance.getServiceId());
16     }
17     return this.discoveryClient;
18 }
```

- 需要在启动类中开启服务发现

```
1 @SpringBootApplication
2 @EnableEurekaClient //在服务启动后自动注册到Eureka中 ---》springcloud-eureka
-7001
3 @EnableDiscoveryClient //开启服务发现，针对团队开发
4 public class DeptProvider_8001 {
5     public static void main(String[] args) {
6         SpringApplication.run(DeptProvider_8001.class,args);
7     }
8 }
```

测试:

← → ↻ localhost:8001/eureka/discovery/

{"discoveryClients":[{"services":["springcloud-provider-dept"],"order":0}, {"services":["springcloud-provider-dept"],"order":0}]}

