**《****Spring Cloud Netflix》**

**第02节**

**服务注册和服务发现-Eureka的使用**

1. **Spring Cloud Netflix**

该项目是Spring Cloud的核心子项目，是对Netflix公司一系列开源产品的封装。它为Spring Boot应用提供了自配置的整合，只需要通过一些简单的注解，就可以快速地在Spring Cloud的应用中使用起来。

它主要提供的模块包括：

服务发现注册（Eureka）

客户端负载均衡（Ribbon）

断路器（Hystrix）

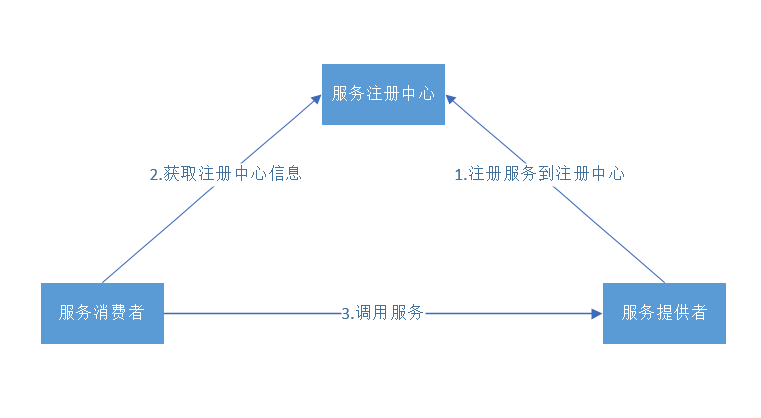
智能路由（Zuul）

开源地址：

<http://netflix.github.io/>

<https://github.com/Netflix>

1. **服务注册和服务发现**



调用关系说明：

1.服务提供者在启动时，向注册中心注册自己提供的服务。

2.服务消费者在启动时，向注册中心订阅自己所需的服务。

3.注册中心返回服务提供者地址给消费者。

4.服务消费者从提供者地址中调用消费者。

**注意！ 下面的服务端指：注册中心，客户端指：提供者和消费者**

1. **如何使用Eureka进行服务注册和发现**

**1、****服务端添加依赖**

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-eureka-server</artifactId>

</dependency>

**2、服务端添加配置**

# server (eureka 默认端口为：8761)

server.port=8761

# spring

spring.application.name=spring-cloud-server

# eureka

# 是否注册到eureka

eureka.client.register-with-eureka=false

# 是否从eureka获取注册信息

eureka.client.fetch-registry=false

# eureka服务器的地址（注意：地址最后面的 /eureka/ 这个是固定值）

eureka.client.serviceUrl.defaultZone=http://localhost:${server.port}/eureka/

**3、服务端添加注解**

@EnableEurekaServer

**4、客户端添加依赖**

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-eureka</artifactId>

</dependency>

**5、客户端添加配置**

**提供者**

# server

server.port=7777

# spring

spring.application.name=spring-cloud-provider

# eureka

eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka/

**消费者**

# server

server.port=8888

# spring

spring.application.name=spring-cloud-consumer

# eureka

eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka/

**6、客户端添加注解**

@EnableEurekaClient

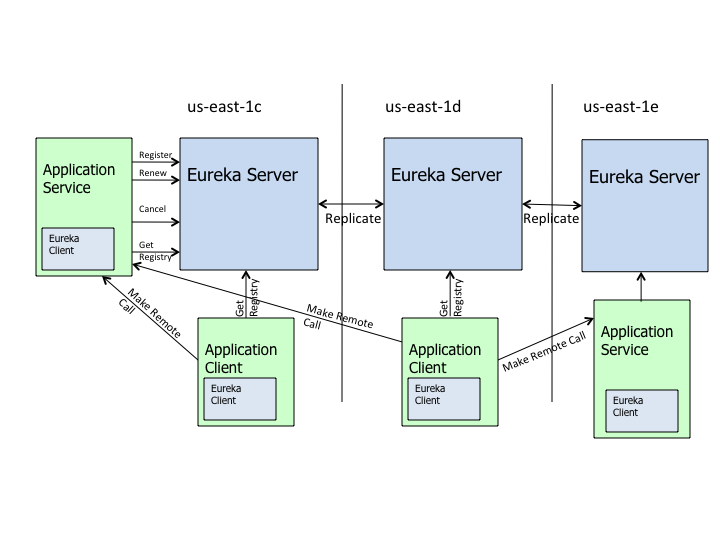
注意：

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.

分析：是由于Eureka进入了保护模式。

在保护模式下，Eureka Server将会尝试保护其服务注册表中的信息，暂时不会注销服务注册表中的服务。

1. **基本流程**：



1. 最左边的client（即服务提供者）发起us-east-1c注册请求；
2. Eureka Server集群中的其他两个node（us-east-1d和us-east-1e进行Replicate复制）；
3. 图下放的两个client（即服务消费者）分别向三个server获取注册信息及Get Registry。
4. **和Zookeeper的对比**

**1、分布式系统的CAP理论：**

一致性（C）：所有的节点上的数据时刻保持同步。

可用性（A）：每个请求都能接受到一个响应，无论响应成功或失败。

分区容错性（P）：系统应该能持续提供服务，即使系统内部有消息丢失（分区）。

由于分区容错性在是分布式系统中必须要保证的，因此我们只能在A和C之间进行权衡。

在此Zookeeper保证的是CP, 而Eureka则是AP。

**2、Zookeeper保证CP**

ZooKeeper是个 CP的，即任何时刻对ZooKeeper的访问请求能得到一致的数据结果，同时系统对网络分割具备容错性、但是它不能保证每次服务请求的可用性(注：也就是在极端环境下，ZooKeeper可能会丢弃一些请求，消费者程序需要重新请求才能获得结果)。

例如：当master节点因为网络故障与其他节点失去联系时，剩余节点会重新进行leader选举。问题在于，选举leader的时间太长，30 ~ 120s, 且选举期间整个zk集群都是不可用的，这就导致在选举期间注册服务瘫痪。

**3、Eureka保证AP**

Eureka看明白了这一点，因此在设计时就优先保证可用性。我们可以容忍注册中心返回的是几分钟以前的注册信息，但不能接受服务直接down掉不可用。也就是说，服务注册功能对可用性的要求要高于一致性。

如果Eureka服务节点在短时间里丢失了大量的心跳连接(注：可能发生了网络故障)，那么这个 Eureka节点会进入“自我保护模式”，同时保留那些“心跳死亡”的服务注册信息不过期。此时，这个Eureka节点对于新的服务还能提供注册服务，对于“死亡”的仍然保留，以防还有客户端向其发起请求。当网络故障恢复后，这个Eureka节点会退出“自我保护模式”。Eureka的哲学是，同时保留“好数据”与“坏数据”总比丢掉任何数据要更好。

**4、总结**

Eureka作为单纯的服务注册中心来说要比zookeeper更加“专业”，因为注册服务更重要的是可用性，我们可以接受短期内达不到一致性的状况。

当然，这也要看具体的使用场景。