链滴

Spring Cloud（二）Eureka 服务注册与发现

作者：wlgzs-sjl

原文链接：https://ld246.com/article/1607220286690

来源网站：链滴

什么是Eureka?

Netflix 在设计Eureka 时，遵循的就是AP原则。

CAP原则又称CAP定理，指的是在一个分布式系统中

一致性（Consistency）

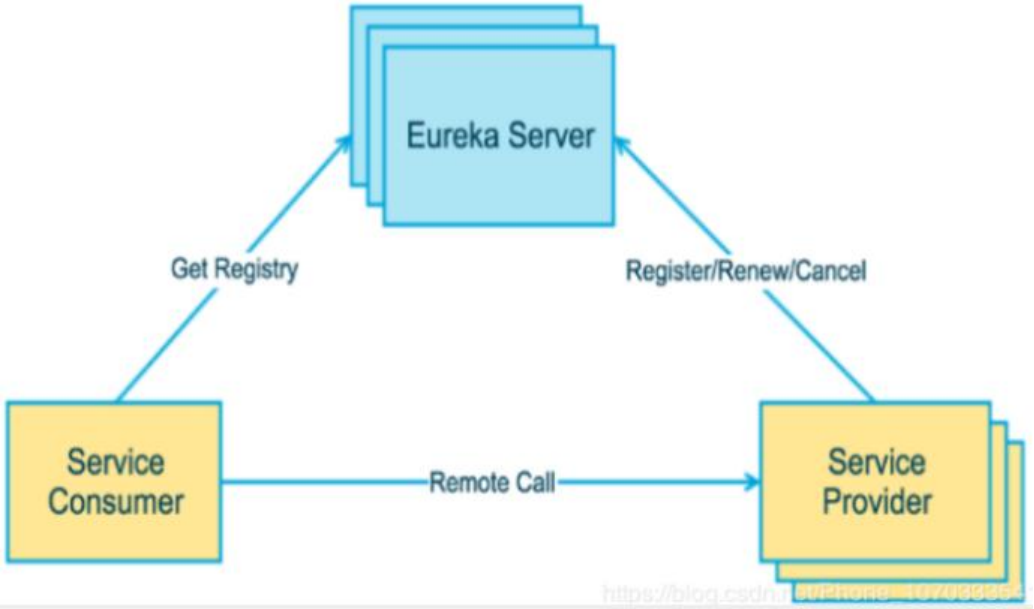
可用性（Availability）

分区容错性（Partition tolerance）

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。

  Eureka是Netflflix的一个子模块，也是核心模块之一。Eureka是一个基于RE
T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
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
调用的配置文件了，功能类似于Dubbo的注册中心，比如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
。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，SpringClou
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
。



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中行注册，这样EurekaServer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（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，EurekaSer er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：提供服务的注册于发现。
- 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。

项目构建

建立springcloud-eureka-7003模块

编辑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 http://maven.apache.org/xsd
maven-4.0.0.xsd">
    <parent>
        <artifactId>springcloud</artifactId>
        <groupId>com.wlgzs</groupId>
        <version>1.0-SNAPSHOT</version>
```

```
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>springcloud-eureka-7003</artifactId>

    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-eureka-server</artifactId>
            <version>1.4.6.RELEASE</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
        </dependency>
    </dependencies>

</project>
```

application.yml

```
server:
  port: 7003

eureka:
  instance:
    hostname: localhost
  client:
    register-with-eureka: false #表示是否注册自己
    fetch-registry: false #false表示自己是注册中心
    service-url:
    defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/
```

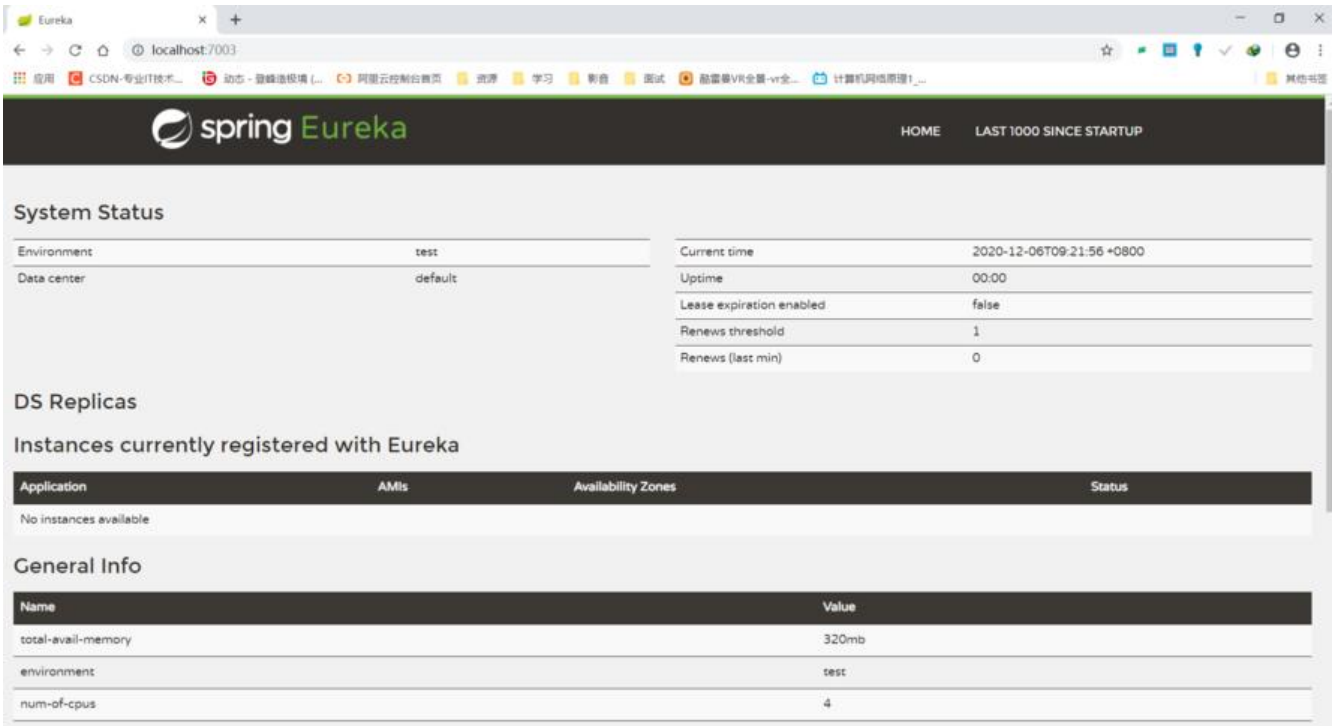
编写主启动类

```
package com.wlgzs.springcloud;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

/**
 * @author wlgzs-sjl
 * @date 2020/11/22 17:07
 */
@SpringBootApplication
@EnableEurekaServer
public class EurekaServer_7003 {
    public static void main(String[] args) {
        SpringApplication.run(EurekaServer_7003.class,args);
    }
}
```

启动，访问测试：



System Status：系统信息

DS Replicas：服务器副本

Instances currently registered with Eureka：已注册的微服务列表

General Info：一般信息

Instance Info：实例信息

建立Service Provider：将 8001 的服务入驻到 7003 的eureka中

修改8001服务的pom文件，增加eureka的支持

```
<!-- Eureka -->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-eureka</artifactId>
            <version>1.4.6.RELEASE</version>
        </dependency>
```

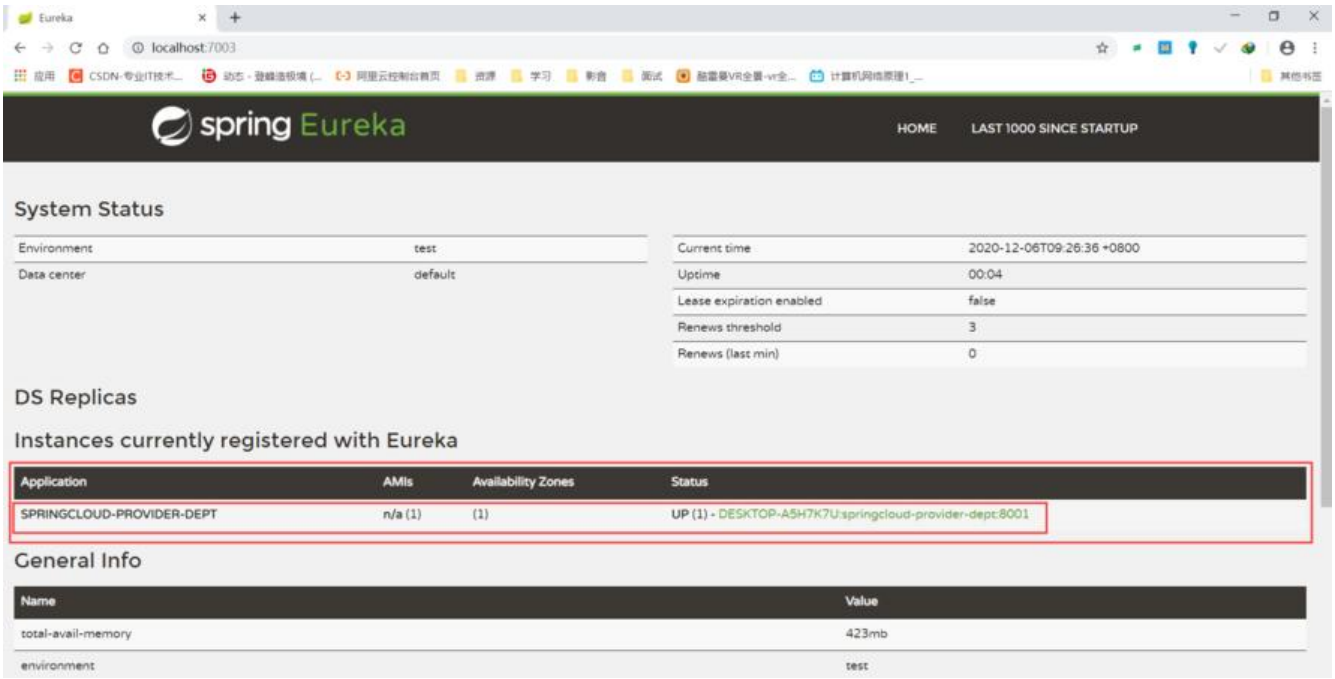
yaml 中配置 eureka 的支持

```
eureka:
  client:
    service-url:
      defaultZone: http://localhost:7003/eureka
```

8001 的主启动类注解支持

```
@SpringBootApplication
@EnableEurekaClient  // 本服务启动之后会自动注册进Eureka中
public class DeptProvider_8001 {
    public static void main(String[] args) {
        SpringApplication.run(DeptProvider_8001.class,args);
    }
}
```

启动7001，再启动8001，测试访问



Eureka的自我保护机制

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
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
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
服务。Eureka通过 **自我保护机制** 来解决这个问题--**Eureka Server 在运行期间会去统计心跳失败
例在 15 分钟之内是否超过 85%，如果超过 85%，那么这个节点就会进入自我保护模式。**一旦进
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
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
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
。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。

服务发现 Discovery

对于注册进eureka里面的微服务，可以通过服务发现来获得该服务的信息。【对外暴露服务】

修改springcloud-provider-dept-8001工程中的DeptController

```
@Autowired
    private DiscoveryClient client;

    @GetMapping("/discovery")
    public Object discovery() {
        //获得微服务列表清单
        List<String> list = client.getServices();
        System.out.println("client.getServices()==>" + list);
        // 得到一个具体的微服务!
        List<ServiceInstance> serviceInstanceList = client.getInstances("springcloud-provider-de
t");
        for (ServiceInstance serviceInstance : serviceInstanceList) {
            System.out.println(
                    serviceInstance.getServiceId() + "\t" +
                            serviceInstance.getHost() + "\t" +
                            serviceInstance.getPort() + "\t" +
                            serviceInstance.getUri());
        }
        return this.client;
    }
```

主启动类增加一个注解

```
@SpringBootApplication
@EnableEurekaClient
@EnableDiscoveryClient
public class DeptProvider_8001 {
    public static void main(String[] args) {
        SpringApplication.run(DeptProvider_8001.class,args);
    }
}
```

启动Eureka服务，启动8001提供者，后台输出：

consumer访问服务

在springcloud-consumer-dept-80中修改DeptConsumerController增加个方法

```
@GetMapping("/discovery")
```

```
public Object discovery() {
    return restTemplate.getForObject(REST_URL_PREFIX + "/discovery", Object.class);
}
```

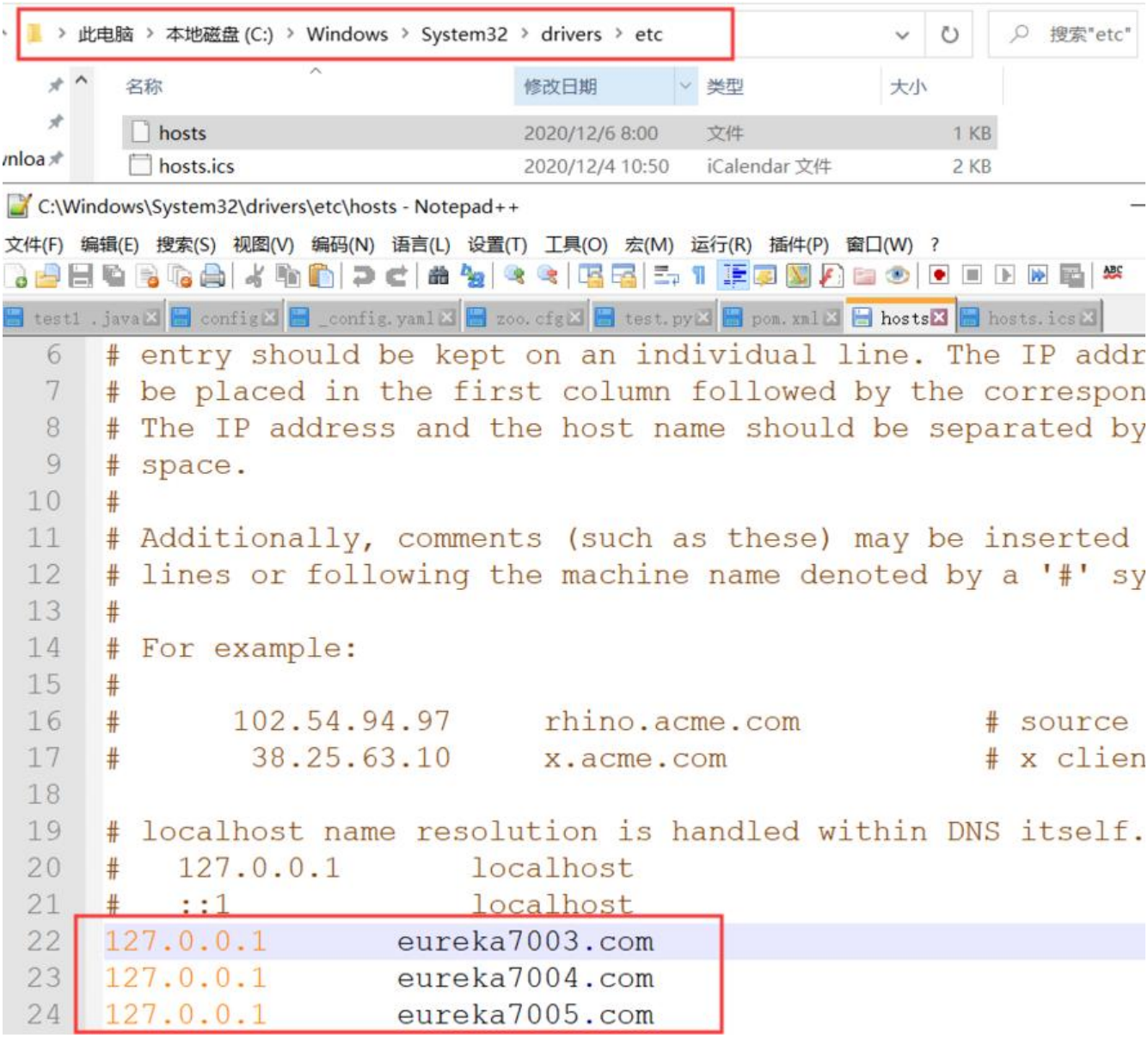
集群配置

新建工程springcloud-eureka-7004、springcloud-eureka-7005；

按照7003为模板粘贴POM；

修改7004和7005的主启动类；

修改映射配置，windows域名映射。



原文链接：Spring Cloud（二）Eureka 服务注册与发现

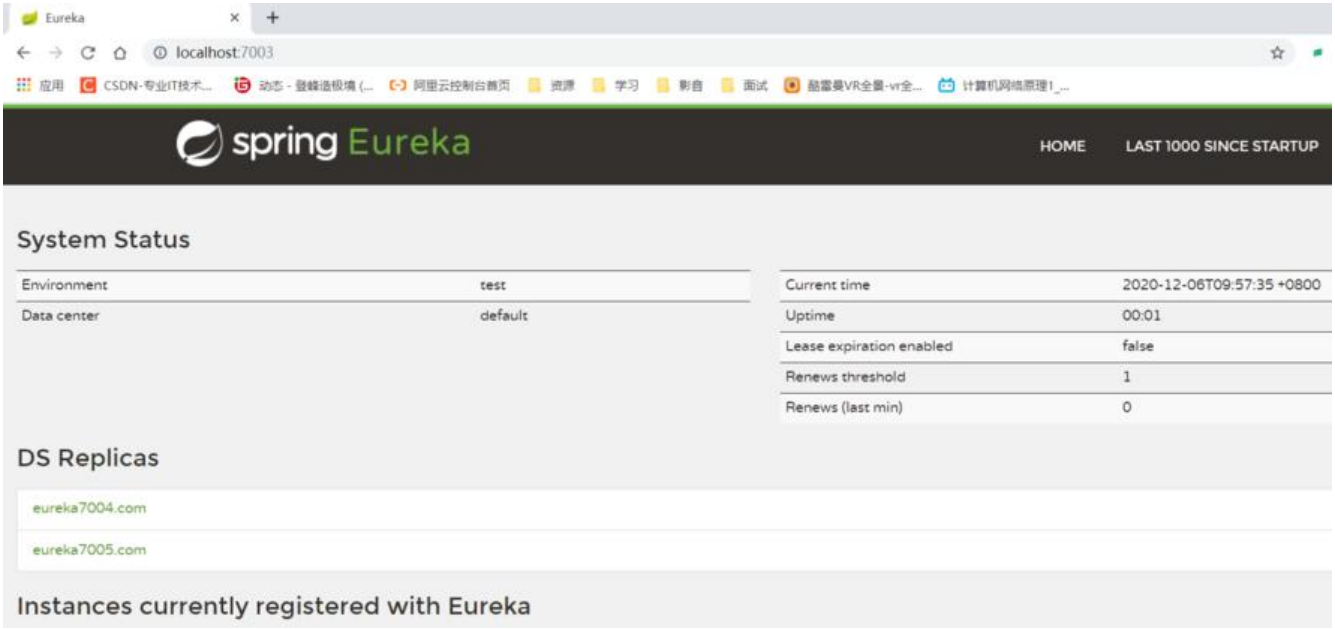
修改3个EurekaServer的yml文件

```yaml
server:
  port: 7003

eureka:
  instance:
    hostname: eureka7003.com
  client:
    register-with-eureka: false #表示是否注册自己
    fetch-registry: false #false表示自己是注册中心
    service-url:
      # defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/
      defaultZone: http://eureka7004.com:7004/eureka/,http://eureka7005.com:7005/eureka/
```

7004，7005同上（只需要修改defaultZone的地址和端口）

启动集群测试



对比Zookeeper

回顾CAP原则

RDBMS （Mysql、Oracle、sqlServer）===>ACID

NoSQL（redis、mongdb）===> CAP

ACID是什么?

- A（Atomicity）原子性
- C（Consistency） 一致性
- I （Isolation）隔离性
- D（Durability）持久性

CAP是什么?

- C（Consistency）强一致性
- A（Availability）可用性
- P（Partition tolerance）分区容错性

CAP的三进二：CA、AP、CP

CAP理论的核心

- 一个分布式系统不可能同时很好的满足一致性，可用性和分区容错性这三个需求。
- 根据CAP原理，将NoSQL数据库分成了满足CA原则，满足CP原则和满足AP原则三大类：
  - CA：单点集群，满足一致性，可用性的系统，通常可扩展性较差
  - CP：满足一致性，分区容错性的系统，通常性能不是特别高
  - AP：满足可用性，分区容错性的系统，通常可能对一致性要求低一些

作为服务注册中心，Eureka比Zookeeper好在哪里?

Zookeeper保证的是CP

  当向注册中心查询服务列表时，我们可以容忍注册中心返回的是几分钟以前注册信息，但不能接受服务直接down掉不可用。也就是说，服务注册功能**对一致性的要求高于可用性**但是zk会出现这样一种情况，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，就导致在选举期间注册服务瘫痪。在云部署的环境下，因为网络问题使得zk集群失去master节点是较概率会发生的事件，虽然服务最终能够恢复，但是漫长的选举时间导致的注册长期不可用是不能容忍。

Eureka保证的是AP

  Eureka优先保证可用性。Eureka各个节点都是平等的，几个节点挂掉不会响正常节点的工作，剩余的节点依然可以提供注册和查询服务。而Eureka的客户端在向某个Eureka册时，如果发现连接失败，则会自动切换至其他节点，只要有一台Eureka还在，就能保住注册服务的用性，只不过查到的信息可能不是最新的，除此之外，Eureka还有一种自我保护机制，如果在15分内超过85%的节点都没有正常的心跳，那么Eureka就认为客户端与注册中心出现了网络故障，此时会现以下几种情况：

1. Eureka不再从注册列表中移除因为长时间没收到心跳而应该过期的服务
2. Eureka仍然能够接受新服务的注册和查询请求，但是不会被同步到其他节点上（即保证当前节点依可用）
3. 当网络稳定时，当前实例新的注册信息会被同步到其他节点中

因此，Eureka可以很好的应对因网络故障导致部分节点失去联系的情况，而不会像zookeeper那样整个注册服务瘫痪