**IDEA热部署**：<https://www.jianshu.com/p/32a91e375fcd>

# 一 微服务概述

微服务、微服务架构、SpringCloud

## 1 微服务与微服务架构

**微服务**(个体) 🡪 SpringBoot，强调的是服务的大小，关注的是某一个点，具体解决某一个问题或完成具体的任务功能。

**微服务架构**(整体) 🡪 SpringCloud，将传统的一站式应用(单一应用程序)拆分成一组小的服务(去耦合)，每个服务运行在其独立的进程中，可独立部署，能够自行单独启动或销毁，拥有自己独立的数据库，服务之间采用轻量级的通信机制互相沟通(通常是基于HTTP的RESTful API)。

**优点：**松耦合，可灵活部署，微服务只是业务逻辑的代码，不会和HTML,CSS 或其他界面组件混合，每个微服务都有自己的存储能力，可以有自己的数据库，也可以有统一数据库…

**缺点**：处理分布式系统的复杂性多服务运维难度，系统部署依赖、服务间通信成本、数据一致性、系统集成测试…

## 2 微服务技术栈

天上飞的理念 🡪 落地的实现

|  |  |
| --- | --- |
| **微服务条目** | **落地技术** |
| 服务开发 | **SpringBoot**、**Spring**、**SpringMVC** |
| 服务配置与管理 | Netflix公司的Archaius、阿里的Diamond等 |
| 服务注册与发现 | **Eureka**、Consul、Zookeeper等 |
| 服务调用 | Rest、RPC、gRPC |
| 服务熔断器 | **Hystrix**、Envoy等 |
| 负载均衡 | **Ribbon**、Nginx等 |
| 服务接口调用(客户端调用服务的简化工具) | **Feign**等 |
| 消息队列 | Kafka、RabbitMQ、ActiveMQ等 |
| 服务配置中心管理 | **SpringCloudConfig**、Chef等 |
| 服务路由（API网关） | **Zuul**等 |
| 服务监控 | Zabbix、Nagios、Metrics、Spectator等 |
| 全链路追踪 | Zipkin，Brave、Dapper等 |
| 服务部署 | Docker、OpenStack、Kubernetes等 |
| 数据流操作开发包 | SpringCloud Stream（封装与Redis,Rabbit、Kafka等发送接收消息） |
| 事件消息总线 | Spring Cloud Bus |
| …… | …… |

# 二 SpringCloud概述

**官网**

<https://spring.io/projects/spring-cloud>

https://springcloud.cc/spring-cloud-netflix.html

**开发API说明：**

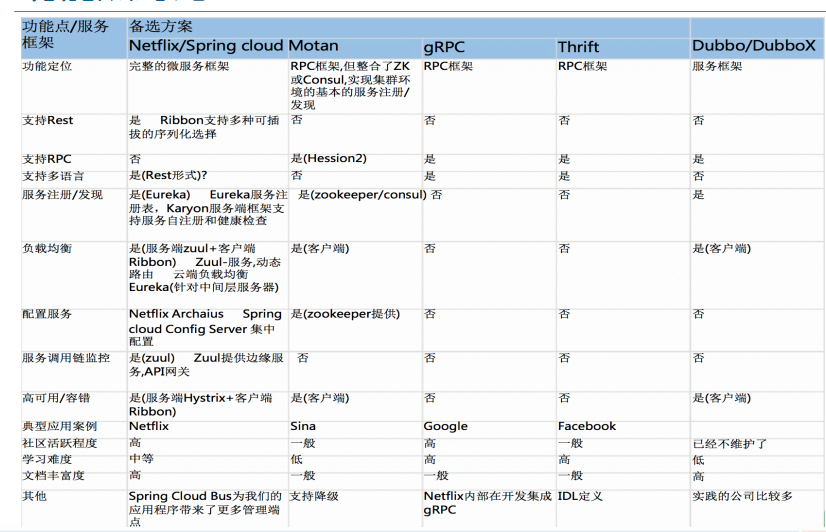
**https://springcloud.cc/spring-cloud-dalston.html**

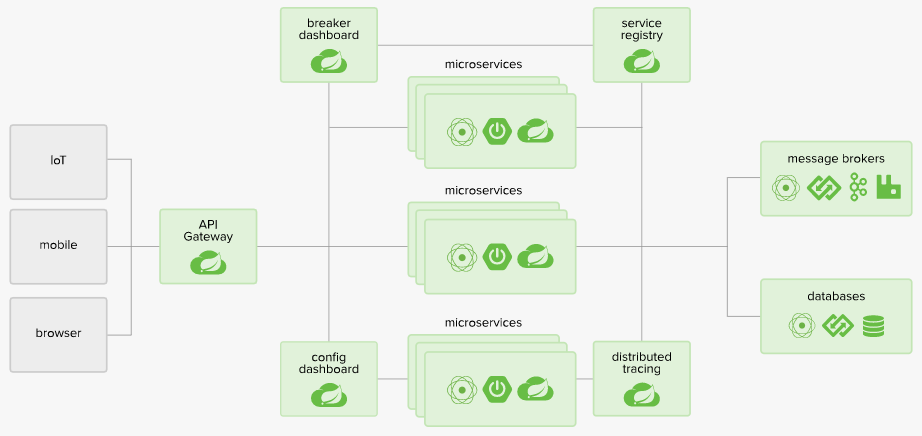
springcloud中国社区：http://springcloud.cn/

**springcloud中文网：https://springcloud.cc/**

**微服务的落地维度(多元技术的组合)，**

**SpringCloud = 分布式微服务架构下的一站式解决方案，是各个微服务架构落地技术的集合体，俗称微服务全家桶**





API Gateway -> zuul路由网关

breaker dashboard -> Hystrix断路器

service registry -> Eureka服务注册与发现

distributed tracing(分布式链路跟踪)

message brokers(消息队列)

database

# 三 Rest微服务构建案例工程模块

**父项目与子项目的依赖，**dependencies与 dependencyManagement的区别：

dependencies，即使在子项目中不写该依赖项，那么子项目仍然会从父项目中继承该依赖项。

dependencyManagement，只是声明依赖并不实现引入，因此子项目要声明需要用的依赖；只有在子项目中写了该依赖项，并且没有指定具体版本，才会从父项目中继承该项，并且version和scope都是读取父类的pom文件;如果子项目中指定了版本号，那么会使用子项目中指定的jar版本。

**约定 > 配置 > 编码**

**IDEA使用Lombok插件：**

1. IDEA中添加Lombok插件, File > Setting > Plugins > Lombok Plugin > install
2. File > Settings > Build > Compiler > Annotation Processors -> √ Enable annotation processing
3. 重启IDEA
4. 在Maven项目的pom.xml中添加配置

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<version>xxx</version>

</dependency>

**其他Model可通过该方式引用microservicecloud-api模块的相关资源(gav)**

<dependency>

<groupId>com.chxp</groupId>

<artifactId>microservicecloud-api</artifactId>

<version>${project.version}</version>

</dependency>

**RestTemplate** 是一种简单便捷的访问 restful 服务模板类，是Spring提供的用于访问Rest服务的客户端模板工具集，提供了多种便捷访问**远程Http服务**的方法。

(url,requestMap,ResponseBean.class) 🡪

(REST请求地址,请求参数,HTTP响应转换被转换成的对象类型)

# 四 Eureka服务注册与发现

(Zookeeper)

C**AP**：Consistency一致性、**Availability可用性**、**Partition tolerance分区容错性**

## 1 Eureka概念

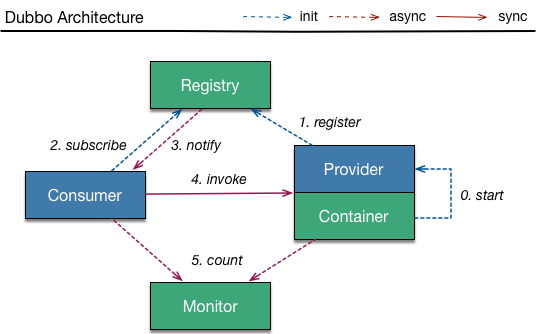
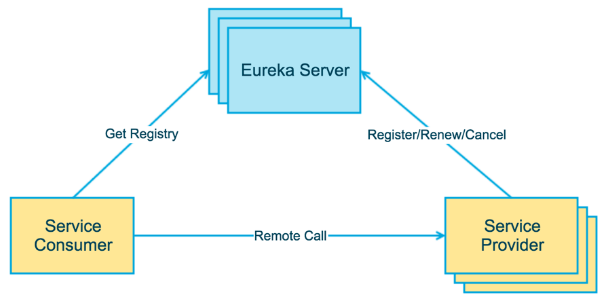
**Eureka**(Netflix公司开发的一个子模块，也是核心模块之一)基于REST的服务，用于定位服务以实现云端中间层服务发现和故障转移。**服务发现与注册**(只需使用服务的标识符就可以访问到服务，不需要修改服务调用的配置文件。功能类似于dubbo的注册中心(Zookeeper))

## 2 Eureka原理

**Eureka**采用C-S的设计架构，系统中的微服务使用 Eureka Client连接到 Eureka Server并维持心跳连接。系统的维护人员通过 Eureka Server 来监控系统中各个微服务是否正常运行。(SpringCloud 的一些模块(如Zuul)可以通过 Eureka Server 来发现系统中的其他微服务，并执行相关的逻辑)

**Eureka Server：**提供服务注册功能的服务器，是服务注册中心。各个节点启动后，会在Eureka Server中进行注册，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：**Java客户端，用于简化Eureka Server的交互，客户端同时也具备一个内置的、使用轮询(round-robin)负载算法的负载均衡器。在应用启动后，将会向Eureka Server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（默认90秒）。



**Eureka Server：**提供服务注册和发现。

**Service Provider：**服务提供方将自身服务注册到Eureka，使服务消费方能够找到。

**(Service Provider -> Eureka Client)**

**Service Consumer**：服务消费方从Eureka获取注册服务列表，从而消费服务。

## 3 构建步骤

**引入 SpringCloud 技术组件的基本步骤：**

1 新增相关的 Maven 坐标

<!-- eureka-server服务端 -->

<dependency>

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

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

</dependency>

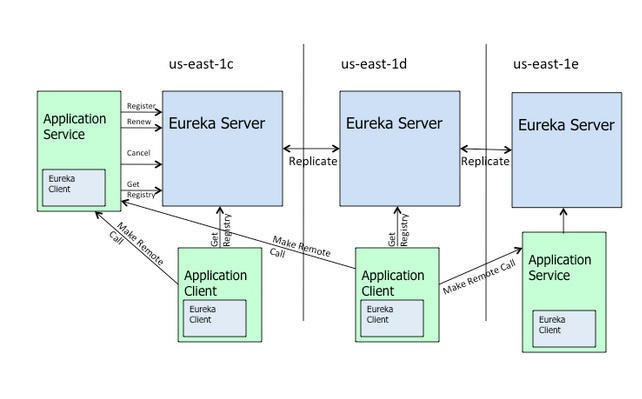
2 在主启动类上标注启动该技术组件的相关注解标签

@EnableEurekaServer

3 编写业务逻辑代码

**Eureka自我保护机制**：好死不如赖活着

## 4 集群配置



处于不同节点的eureka通过Replicate进行数据同步

Application Service为服务提供者

Application Client为服务消费者

Make Remote Call完成一次服务调用

服务启动后向Eureka注册，Eureka Server会将注册信息向其他Eureka Server进行同步，当服务消费者要调用服务提供者，则向服务注册中心获取服务提供者地址，然后会将服务提供者地址缓存在本地，下次再调用时，则直接从本地缓存中取，完成一次调用。

# 五 Ribbon负载均衡

(Nginx)

## 1 Ribbon概述

**LB**，即负载均衡(Load Balance)，在微服务或分布式集群中经常用的一种应用。 负载均衡简单的说就是将用户的请求平摊的分配到多个服务上从而达到系统的HA。

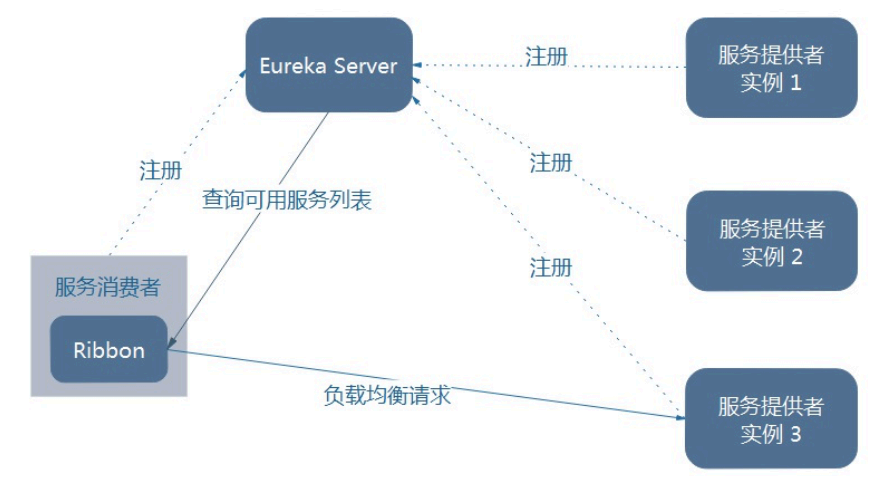
**集中式LB**，即在服务的消费方和提供方之间使用独立的LB设施(可以是硬件，如F5；可以是软件，如Nginx), 由该设施负责把访问请求通过某种策略转发至服务的提供方。

**进程内LB**，将LB逻辑集成到消费方，消费方从服务注册中心获知有哪些地址可用，然后自己再从这些地址中选择出一个合适的服务器。Ribbon属于进程内LB，它只是一个类库，集成于消费方进程，消费方通过它来获取到服务提供方的地址。

Spring Cloud **Ribbon**是基于Netflix Ribbon实现的一套 **客户端 负载均衡** 的工具。

简单的说，就是在配置文件中列出Load Balancer（简称LB）后面所有的机器，Ribbon会自动的基于某种规则（如简单轮询，随机连接等）去连接这些机器。也可以很容易使用Ribbon实现自定义的负载均衡算法。

## 2 Ribbon负载均衡



**第一步** 选择 Eureka Server，优先选择在同一个区域内负载较少的server。

**第二步** 根据用户指定的策略(比如轮询、随机和根据响应时间加权等)，在从server取到的服务注册列表中选择一个地址。

## 3 Ribbon核心组件IRule

## 4 Ribbon自定义

# 六 Feign负载均衡

(Nginx)

Feign：声明式的接口加注解Web服务客户端的调用。

Feign面向接口调用微服务

# 七 Hystrix断路器

假设微服务A调用微服务B和C，微服务B和C有调用其他的微服务，即所谓的“**扇出**”。如果扇出链路上某个微服务的调用响应时间过长或不可用，对微服务A的调用会占用越来越多的系统资源，进而引进系统的崩溃，即所谓的“**服务雪崩**”。

**服务依赖的保护措施：**

（1）熔断模式：如果某个目标服务调用慢或者有大量超时，熔断该服务的调用，对于后续调用请求不再继续调用，目标服务直接返回，快速释放资源，如果目标服务情况好转则恢复调用。

（2）隔离模式：可以对不同类型的请求使用线程池来隔离资源，每种类型的请求互不影响，如果一种类型的请求线程资源耗尽，则对后续的该类型请求直接返回，不再调用后续资源。。

（3）限流模式：上述的熔断模式和隔离模式都属于出错后的容错处理机制，限流模式则属于预防模式。限流模式主要是提前对各个类型的请求设置最高的QPS阈值，若高于设置的阈值则对该请求直接返回，不再调用后续资源。这种模式不能解决服务依赖的问题，只能解决系统整体资源分配问题，因为没有被限流的请求依然有可能造成雪崩效应。

## 1 Hystrix概述

**断路器**，本身是一种开关装置，当某个服务单元发生故障之后，通过断路器的故障监控（熔断保险丝），向调用方返回一个符合预期的可处理的备选响应（FallBack），而不是长时间的等待或者抛出调用方无法处理的异常，这样就保证了服务调用方的线程不会被长时间地不必要地占用，从而避免了故障在分布式系统中的蔓延，乃至雪崩。

**Hystrix**能够保证在一个依赖出问题的情况下，不会导致整体服务失败，避免级联故障，以提高分布式系统的弹性。

## 2 服务熔断

一般是某个服务故障或者异常引起的，当某个异常条件被触发，直接熔断整个服务，快速返回“错误”的响应信息，而不是一直等到此服务超时，对于后续调用请求，不在继续调用目标服务，直接返回，快速释放资源。如果目标服务情况好转则恢复调用。

熔断机制的注解 **@HystrixCommand**

## 3 服务降级

**在客户端实现完成的，与服务端没有关系**

所谓降级，一般是从整体负荷考虑，当某个服务熔断后，服务器将不会再被调用，此时客户端可以自己准备一个本地的fallback回调，返回一个缺省值，虽然服务水平下降，但至少可用，比直接挂掉强。

## 4 服务监控

**HystrixDashboard**，Hystrix会持续地记录所有通过Hystrix发起的请求的执行信息，并以统计报表和图形的形式展示给用户，包括每秒执行多少请求多少成功，多少失败等。

# 八 zuul路由网关

## 1 zuul概述

路由功能，负责将外部请求转发到具体的微服务实例上，是实现外部访问统一入口的基础。

过滤功能，负责对请求的处理过程进行干预，是实现请求校验、服务聚合等功能的基础。

Zuul注册为Eureka服务治理下的应用，从Eureka中获得其他微服务的相关信息，访问微服务都是通过Zuul跳转后获得。

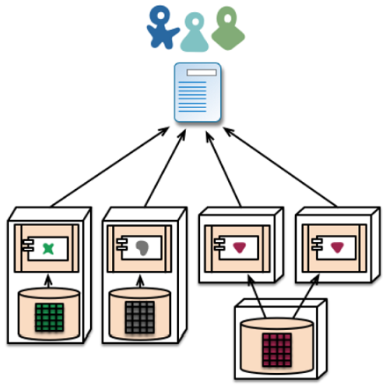
**代理 + 路由 + 过滤**

## 2 路由基本配置

## 3 路由访问映射规则

# 九 SpringCloud Config分布式配置中心

Git/GitHub



## 1 概述

SpringCloud Config为微服务架构中的微服务提供集中化的外部配置支持，配置服务器为各个不同微服务应用的所有环境提供了一个中心化的外部配置。

分为服务端和客户端：

**SpringCloud Config服务端，**也称为分布式配置中心，是一个独立的微服务应用，用来连接配置服务器并为客户端提供获取配置信息，加密/解密信息等访问接口。

**SpringCloud Config客户端，**是通过指定的配置中心来管理应用资源以及与业务相关的配置内容，并在启动的时候从配置中心获取和加载配置信息配置服务器，默认采用Git来存储配置信息。

集中管理配置文件，动态配置更新，配置信息以REST接口形式暴露。

## 2 服务端配置

## 3 客户端配置

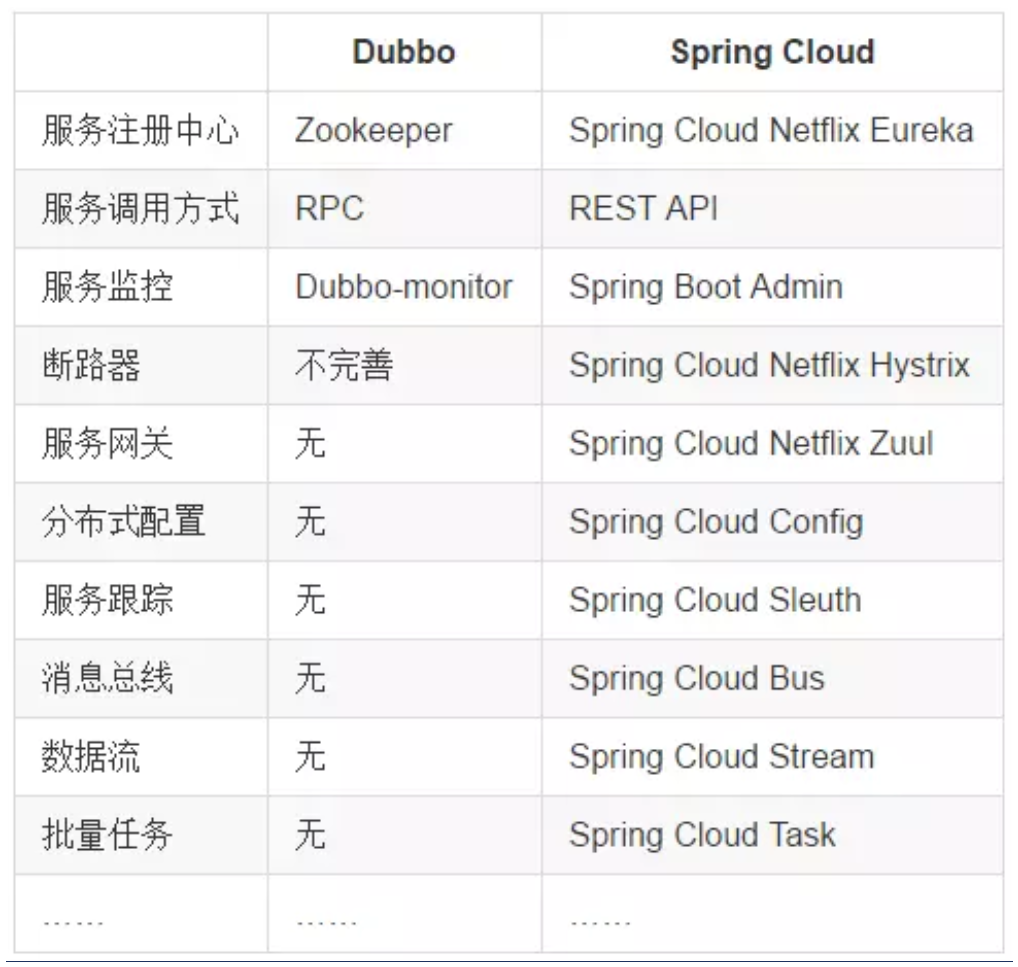
# 十 面试题

## 1 什么是微服务？

## 2 微服务之间是如何独立通讯的

## 3 SpringCloud和Dubbo有哪些区别？

品牌机与组装机的区别：



**通信机制：**Dubbo基于RPC远程过程调用，SpringCloud基于HTTP的RESTful API调用。

Dubbo的定位始终是一款RPC框架，而SpringCloud的目标是微服务架构下的一站式解决方案。

RPC(Romote Procedure Call)：远程过程调用，允许一台计算机程序远程调用另外一台计算机的子程序，不用关心底层网络通信。

## 4 SpringBoot和SpringCloud，请你谈谈对他们的理解

SpringBoot专注于快速、方便的开发单个微服务个体，SpringCloud是关注全局的微服务协调整理治理框架，它将SpringBoot开发的一个个单体微服务整合并管理起来，为各个微服务之间提供配置管理、服务发现、断路器、路由、微代理、事件总线、全局锁、决策竞选、分布式会话等集成服务。

SpringBoot可以离开SpringCloud独立使用开发项目，但SpringCloud离不开SpringBoot，属于依赖的关系。

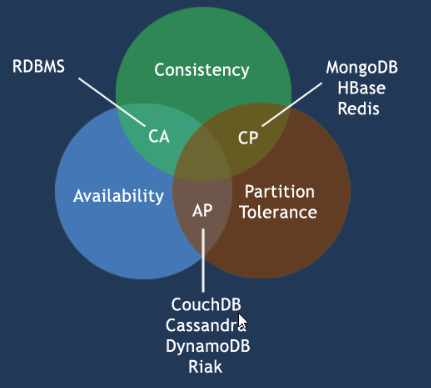
Keyword：Dubbo + Zookeeper -> SpringCloud(一整套技术的结合)、微服务

## 5 什么是服务熔断？什么是服务降级

## 6 微服务的优缺点分别是什么？说下你在项目开发中碰到的坑

## 7 你所知道的微服务技术栈有哪些？请列举一二

## 8 eureka和zookeeper都可以提供服务注册与发现的功能，请说说两个的区别？



ACID原则：A(Atomicity) 原子性、C(Consistency)一致性、I(Isolation)独立性、D(Durability)持久性

CAP原则：C(Consistency)强一致性、A(Availability)可用性、P(Partition tolerance)分区容错性，在一个分布式系统中，三者只可兼其二。由于P在是分布式系统中必须要保证的，因此只能在A和C之间进行权衡。

RDBMS(mysql/oracle/sqlServer) 🡪 ACID

NOSQL(redis/mongdb) 🡪 CAP

**eureka遵守AP，zookeeper遵守CP**

**Zookeeper**：当向注册中心查询服务列表时，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。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。在云部署的环境下，因网络问题使得zk集群失去master节点是较大概率会发生的事，虽然服务能够最终恢复，但是漫长的选举时间导致的注册长期不可用是不能容忍的。

**Eureka：**各个节点都是平等的，几个节点挂掉不会影响正常节点的工作，剩余的节点依然可以提供注册和查询服务。Eureka的客户端在向某个Eureka注册时发现连接失败，则会自动切换至其它节点，只要有一台Eureka还在，就能保证注册服务可用(保证可用性)，查到的信息可能不是最新的(不保证强一致性)。

Eureka有一种自我保护机制，如果在15分钟内超过85%的节点都没有正常的心跳，Eureka就认为客户端与注册中心出现了网络故障，此时会出现以下几种情况：

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它节点上(即保证当前节点依然可用)

3. 当网络稳定时，当前实例新的注册信息会被同步到其它节点中

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那样使整个注册服务瘫痪。