# ****1Spring Cloud的服务注册和发现如何实现？****

Eureka server提供服务注册和发现

Eureka Server提供服务注册服务。各个服务提供者节点启动后，会在Eureka Server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

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

Service Consumer服务消费方从Eureka获取注册服务列表，从而能够消费服务,

# 谈谈你对微服务架构的理解

传统项目架构使用单体架构，就是一个项目集成了所有的功能模块。例如：一个商城的后台管理系统，可能包含用户管理、商品管理、订单管理等模块。发布项目的时候，需要先把项目整体进行打包，然后再把打包后的文件发布到服务器上面。这样的缺点非常明显。

1，项目的复杂度高  
一个大型项目里面可能包含许多功能模块，而且模块之间肯定会存在一定的依赖关系，项目的复杂度非常高。如果其中某个模块出现问题，就有可能导致整个项目运行失败。

2，部署时间长，影响范围大  
由于大型项目的复杂度高，项目的构件和部署的时间也会随之增加。而且，如果某一个功能模块出现问题，那么解决问题后需要重新把整个项目重新发布。在发布过程中项目的所有功能都无法使用。

3，功能扩展能力有限  
单体架构的项目无法结合业务模块特点进行功能扩展，只能作为一个整体进行扩展，程序的伸缩性较差。

微服务架构

与传统架构方式不同，微服务架构把一个大型项目按照不同的功能拆分为不同的子模块。发布项目的时候，可以对各个子模块进行单独发布。每一个子模块都是运行在自己的进程中。模块之间通过一些通信机制进行数据通信。例如上面的商城后台系统，如果按照微服务架构来设计，每一个业务功能模块都可以作为一个子项目单独发布

1. 便于模块化开发  
   在一个大型项目里面，不同人员可能负责不同的模块。

（2）部署应用的效率更高，模块之间的影响较少不同模块可以单独部署。由于单个模块的代码量较少，部署的时候服务启动速度会更快。而且即使某个模块发生改变，只需要重新发布该模块即可，不会影响其他模块的功能使用。

（3）不受所使用技术的限制  
开发人员可以根据项目业务需要和团队成员能力的特点，对不同模块选择不同的技术来实现。

缺陷

（1）使用成本高  
微服务架构底层是基于分布式技术来实现，需要有更多服务器的硬件支持，从而导致硬件成本增加。

（2）对项目设计和开发人员的技术要求更高  
对于一个分布式系统来说，系统容器、网络延迟、高并发、事务控制都对软件设计和开发人员带来新的挑战。

（3）不同子模块中可能会存在相同功能  
有些功能由于没有达到分解为一个微服务的程度（比如一个日期的工具类），这时候需要在各个子模块中都需要定义相同功能，导致有些功能重复。

# 3.微服务实现框架有哪些

Java语言相关微服务框架

Dubbo：国内最早开源的 RPC 框架，由阿里巴巴公司开发并于 2011 年末对外开源，仅支持 Java 语言。

Motan：微博内部使用的 RPC 框架，于 2016 年对外开源，仅支持 Java 语言。

Tars：腾讯内部使用的 RPC 框架，于 2017 年对外开源，仅支持 C++ 语言。

Spring Cloud：国外 Pivotal 公司 2014 年对外开源的 RPC 框架，仅支持 Java 语言

而跨语言平台的开源 RPC 框架主要有以下几种。

gRPC：Google 于 2015 年对外开源的跨语言 RPC 框架，支持多种语言。

Thrift：最初是由 Facebook 开发的内部系统跨语言的 RPC 框架，2007 年贡献给了 Apache 基金，成为 Apache 开源项目之一，支持多种语言。

如果你的业务场景仅仅局限于一种语言的话，可以选择跟语言绑定的 RPC 框架中的一种；

如果涉及多个语言平台之间的相互调用，就应该选择跨语言平台的 RPC 框架。

# 4.springcloud五大组件

springcloud是springboot的超集，是微服务的全家桶。通过springcloud可以实现中小企业的快速部署。

　　springcloud项目是多个独立的子项目集合而成，所以它的版本管理比较特殊，版本名使用字母排序。

如果使用springboot 2.0以上版本，我们应该使用springcloud F版本。

接着介绍下它的常用五个组件：

（1）服务发现-- Netflix Eureka

　　　 由两个组件组成：Eureka服务端和Eureka客户端

Eureka服务端用作服务注册中心，支持集群部署

Eureka客户端是一个java客户端，用来处理服务注册与发现。

工作原理： 在应用启动时，Eureka客户端向服务端注册自己的服务信息，同时将服务端的服务信息缓存到本地。客户端会和服务端进行周期性的进行心跳交互进行更新服务信息。

(2) 客户端负载均衡 -- Netflix Ribbon

　　　　提供客户端的软件负载均衡算法。它基于Http和Tcp的客户端负载均衡，使得面向REST请求时变换为客户端的负载服务调用。

(3) 熔断器 -- Netflix Hystrix

作用：断路器，保护系统，控制故障范围。为了保证高可用，单个服务通常会集群部署。当网络或者其他原因导致单个服务出现问题，调用这个服务就会出现线程阻塞，此时如果大量请求进入，Servlet容器的线程资源就会消耗完毕，最终导致服务瘫痪。服务与服务之间的依赖，故障会传播，对整个微服务框架造成灾难性的严重后果，这个就是服务故障的“雪崩”效应。

（4）服务网关--Netflix Zuul

　　　　作用：api网关，路由，负载均衡等作用

类型Nginx 和F5，可以实现反向代理。

在微服务架构中，后端服务往往不直接暴露给前端，而是通过一个Api网关根据请求的Url，路由到响应的服务（MVC机制）。当添加Api网关后，在第三方调用端和服务提供方之间就创建了一面墙，这面墙直接与调用方进行通信进行权限控制。后将请求均衡分发给后台服务端。

(5) 分布式配置 -- Spring Cloud Config

　　　　作用：配置管理

提供服务端和客户端，服务器存储后端的默认实现使用git，因此它轻松支持标签版本的配置环境。Config是静态配置的。

如果需要动态配置，可以使用spring cloud bus进行动态配置更新。

# 5.单片，SOA和微服务架构有什么区别?

SOA 微服务

遵循“ 尽可能多的共享 ”架构方法 遵循“ 尽可能少分享 ”的架构方法

重要性在于 业务功能 重用 重要性在于“ 有界背景 ” 的概念

他们有 共同的 治理 和标准 他们专注于 人们的 合作 和其他选择的自由

使用 企业服务总线（ESB） 进行通信 简单的消息系统

它们支持 多种消息协议 他们使用 轻量级协议 ，如 HTTP / REST 等。

多线程， 有更多的开销来处理I / O. 单线程 通常使用Event Loop功能进行非锁定I / O处理

最大化应用程序服务可重用性 专注于 解耦

传统的关系数据库 更常用 现代 关系数据库 更常用

系统的变化需要修改整体 系统的变化是创造一种新的服务

DevOps / Continuous Delivery正在变得流行，但还不是主流 专注于DevOps /持续交付

单片架构类似于大容器，其中应用程序的所有软件组件组装在一起并紧密封装。

一个面向服务的架构是一种相互通信服务的集合。通信可以涉及简单的数据传递，也可以涉及两个或多个协调某些活动的服务。

微服务架构是一种架构风格，它将应用程序构建为以业务域为模型的小型自治服务集合。

# 6**除了eureka和zookeeper注册中心还知道其他服务注册中心么？服务注册是如何实现的？**

1.ZooKeeper保证的是CP,Eureka保证的是AP  
ZooKeeper在选举期间注册服务瘫痪,虽然服务最终会恢复,但是选举期间不可用的  
Eureka各个节点是平等关系,只要有一台Eureka就可以保证服务可用,而查询到的数据并不是最新的

自我保护机制会导致  
Eureka不再从注册列表移除因长时间没收到心跳而应该过期的服务  
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(高可用)  
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(最终一致性)  
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一样使得整个注册系统瘫痪

2.ZooKeeper有Leader和Follower角色,Eureka各个节点平等  
3.ZooKeeper采用过半数存活原则,Eureka采用自我保护机制解决分区问题  
4.Eureka本质上是一个工程,而ZooKeeper只是一个进程

# 7,作为服务注册中心，Eureka比Zookeeper好在哪里?

著名的CAP理论指出，一个分布式系统不可能同时满足C(一致性)、A(可用性)和P(分区容错性)。由于分区容错性P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。

因此，Zookeeper 保证的是CP, Eureka 则是AP。

Zookeeper保证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保证APEureka看明白了这一点，因此在设计时就优先保证可用性。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。而Eureka的客户端在向某个Eureka注册或时如果发现连接失败，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)，只不过查到的信息可能不是最新的(不保证强一致性)。

除此之外，Eureka还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，那么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：

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)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。

# 8,springboot自动配置的原理

在spring程序main方法中 注解@SpringBootApplication或者@EnableAutoConfiguration

在启动时扫描项目所依赖的每个starter中寻找包含spring.factories文件的JAR

根据spring.factories配置加载AutoConfigure类

根据 @Conditional注解的条件，进行自动配置并将Bean注入Spring Context

# 9,如何重新加载Spring Boot上的更改，而无需重新启动服务器？

**DevTools模块完全满足开发人员的需求。该模块将在生产环境中被禁用。它还提供H2数据库控制台以更好地测试应用程序。**

**添加【修改代码】自动重启功能**

**添加开发者工具集=====spring-boot-devtools**

# 10如何覆盖Spring Boot项目的默认属性？

**这可以通过在application.properties文件中指定属性来完成。**

**例如，在Spring MVC应用程序中，您必须指定后缀和前缀。这可以通过在application.properties文件中输入下面提到的属性来完成。**

**对于后缀 - spring.mvc.view.suffix: .jsp**

**对于前缀 - spring.mvc.view.prefix: /WEB-INF/**

# **11,**如何覆盖Spring Boot项目的默认属性？

这可以通过在application.properties文件中指定属性来完成。

例如，在Spring MVC应用程序中，您必须指定后缀和前缀。这可以通过在application.properties文件中输入下面提到的属性来完成。

* 对于后缀 - spring.mvc.view.suffix: .jsp
* 对于前缀 - spring.mvc.view.prefix: /WEB-INF/

**<dependencies>**

**<dependency>**

**<groupId>org.springframework.boot</groupId>**

**<artifactId>spring-boot-devtools</artifactId>**

**<optional>true</optional>**

**</dependency>**

**</dependencies>**

# 12.springcloud如何实现服务的注册和发现

**服务在发布时 指定对应的服务名（服务名包括了IP地址和端口） 将服务注册到注册中心（eureka或者zookeeper）**

**这一过程是springcloud自动实现 只需要在main方法添加@EnableDisscoveryClient 同一个服务修改端口就可以启动多个实例**

**调用方法：传递服务名称通过注册中心获取所有的可用实例 通过负载均衡策略调用（ribbon和feign）对应的服务**

# **13.ribbon和feign区别**

Ribbon添加maven依赖 spring-starter-ribbon 使用@RibbonClient(value=“服务名称”) 使用RestTemplate调用远程服务对应的方法  
feign添加maven依赖 spring-starter-feign 服务提供方提供对外接口 调用方使用 在接口上使用@FeignClient(“指定服务名”)

Ribbon和Feign的区别：

Ribbon和Feign都是用于调用其他服务的，不过方式不同。

1.启动类使用的注解不同，Ribbon用的是@RibbonClient，Feign用的是@EnableFeignClients。

2.服务的指定位置不同，Ribbon是在@RibbonClient注解上声明，Feign则是在定义抽象方法的接口中使用@FeignClient声明。

3.调用方式不同，

Ribbon需要自己构建http请求，模拟http请求然后使用RestTemplate发送给其他服务，步骤相当繁琐。

Feign则是在Ribbon的基础上进行了一次改进，采用接口的方式，将需要调用的其他服务的方法定义成抽象方法即可，

不需要自己构建http请求。不过要注意的是抽象方法的注解、方法签名要和提供服务的方法完全一致。

# **14,什么是服务熔断?什么是服务降级**

当下游服务因访问压力过大而响应变慢或失败，上游服务为了保护系统整体的可用性，可以暂时切断对下游服务的调用。这种牺牲局部，保全整体的措施就叫做熔断

服务降级则相对于整体应用完全可用状态而言的，因为部分不可用从保全大部分服务可用的这种服务状态成为服务降级

# **15.Hystrix 实现原理**

在执行的时候 Hystrix 会解析 Command 的隔离规则来创建 RxJava Scheduler 并在其上调度执行，若是线程池模式则 Scheduler 底层的线程池为配置的线程池，若是信号量模式则简单包装成当前线程执行的 Scheduler  
Hystrix 的 Command 强依赖于隔离规则配置的原因是隔离规则会直接影响 Command 的执行。

# 16什么是耦合和凝聚力？

组件之间依赖关系强度的度量被认为是耦合。一个好的设计总是被认为具有高内聚力和低耦合性。

面试官经常会问起凝聚力。它也是另一个测量单位。更像是一个模块内部的元素保持结合的程度。

必须记住，设计微服务的一个重要关键是低耦合和高内聚的组合。当低耦合时，服务对其他服务的依赖很少。这样可以保持服务的完整性。在高内聚性中，将所有相关逻辑保存在服务中成为可能。否则，服务将尝试彼此通信，从而影响整体性能。

# 17,您将如何在微服务上执行安全测试？

您需要独立测试各个部分。有三种常见的程序：

代码扫描 - 确保任何代码行都没有错误并且可以复制。

灵活性 - 安全解决方案应该是灵活的，以便可以根据系统的要求进行调整。

适应性 - 安全协议应该灵活和更新，以应对黑客或安全漏洞的新威胁。

# 16,什么是幂等性?它是如何使用的？

幂等性指的是这样一种场景:您重复执行一项任务，但最终结果保持不变或类似。

幂等性主要用作数据源或远程服务，当它接收一组以上指令时，它只处理一组指令。

# 18微服务之间是如何通讯的？

第一种：远程过程调用（Remote Procedure Invocation）

直接通过远程过程调用来访问别的service。

示例：REST、gRPC、Apache、Thrift

优点：

简单，常见。因为没有中间件代理，系统更简单

缺点：

只支持请求/响应的模式，不支持别的，比如通知、请求/异步响应、发布/订阅、发布/异步响应降低了可用性，因为客户端和服务端在请求过程中必须都是可用的

第二种：消息

使用异步消息来做服务间通信。服务间通过消息管道来交换消息，从而通信。

示例：Apache Kafka、RabbitMQ

优点:

把客户端和服务端解耦，更松耦合 提高可用性，因为消息中间件缓存了消息，直到消费者可以消费

支持很多通信机制比如通知、请求/异步响应、发布/订阅、发布/异步响应

缺点:

消息中间件有额外的复杂性

# 19你所知道的微服务技术栈有哪些？请列举一二

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

# 20 Spring Cloud Alibaba

同 Spring Cloud 一样，Spring Cloud Alibaba 也是一套微服务解决方案，包含开发分布式应用微服务的必需组件，方便开发者通过 Spring Cloud 编程模型轻松使用这些组件来开发分布式应用服务。

依托 Spring Cloud Alibaba，您只需要添加一些注解和少量配置，就可以将 Spring Cloud 应用接入阿里微服务解决方案，通过阿里中间件来迅速搭建分布式应用系统。

作为 Spring Cloud 体系下的新实现，Spring Cloud Alibaba 跟官方的组件或其它的第三方实现如 Netflix, Consul，Zookeeper 等对比，具备了更多的功能:

# 21、Spring Cloud Alibaba 包含组件

这幅图是 Spring Cloud Alibaba 系列组件，其中包含了阿里开源组件，阿里云商业化组件，以及集成Spring Cloud 组件。

阿里开源组件

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。

Sentinel：把流量作为切入点，从流量控制、熔断降级、系统负载保护等多个维度保护服务的稳定性。

RocketMQ：开源的分布式消息系统，基于高可用分布式集群技术，提供低延时的、高可靠的消息发布与订阅服务。

Dubbo：这个就不用多说了，在国内应用非常广泛的一款高性能 Java RPC 框架。

Seata：阿里巴巴开源产品，一个易于使用的高性能微服务分布式事务解决方案。

Arthas：开源的Java动态追踪工具，基于字节码增强技术，功能非常强大。