38 微服务常见面试题分析

更新时间：2019-08-02 16:24:34

> 为中华之崛起而读书。
>
> ——周恩来

随着越来越多的公司使用 Spring Cloud 技术，一些公司招聘岗位条件都会有一条 Spring Cloud 实战项目经验优先 等描述。我在拉勾网上使用 springcloud 关键字进行检索，查询到了 30 多页的招聘需求，全部都是和 Java 工程师招聘相关。

可以预见的是，Spring Cloud 慢慢会成为大多数公司面试考察的知识点之一，本节课为大家梳理 Spring Cloud 技术栈中常见的考察点。回顾这些高频的面试题，其实也是对 Spring Cloud 知识体系做一次整体的梳理。

微服务相关

谈谈你对微服务的理解

微服务架构是一种架构模式或者说是一种架构风格，它提倡将单一应用程序划分为一组小的服务，每个服务运行在其独立的自己的进程中，服务之间相互协调、互相配合，为用户提供最终服务。

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）,每个服务都围绕着具体的业务进行构建，并且能够被独立的构建在生产环境、类生产环境等。

另外，应避免统一的、集中式的服务管理机制，对具体的一个服务而言，应根据业务上下文，选择合适的语言、工具对其进行构建，可以有一个非常轻量级的集中式管理来协调这些服务，可以使用不同的语言来编写服务，也可以使用不同的数据存储。

微服务的优缺点？

微服务的优点

- 复杂度可控：在将应用分解的同时，规避了原本复杂度无止境的积累。每一个微服务专注于单一功能，并通过定义良好的接口清晰表述服务边界。由于体积小、复杂度低，每个微服务可由一个小规模开发团队完全掌控，易于

保持高可维护性和开发效率。

- 独立部署：由于微服务具备独立的运行进程，所以每个微服务也可以独立部署。当某个微服务发生变更时无需编译、部署整个应用。由微服务组成的应用相当于具备一系列可并行的发布流程，使得发布更加高效，同时降低对生产环境所造成的风险，最终缩短应用交付周期。
- 技术选型灵活：微服务架构下，技术选型是去中心化的。每个团队可以根据自身服务的需求和行业发展的现状，自由选择最适合的技术栈。由于每个微服务相对简单，故需要对技术栈进行升级时所面临的风险就较低，甚至完全重构一个微服务也是可行的。
- 容错：当某一组件发生故障时，在单一进程的传统架构下，故障很有可能在进程内扩散，形成应用全局性的不可用。在微服务架构下，故障会被隔离在单个服务中。若设计良好，其他服务可通过重试、平稳退化等机制实现应用层面的容错。
- 扩展：单块架构应用也可以实现横向扩展，就是将整个应用完整的复制到不同的节点。当应用的不同组件在扩展需求上存在差异时，微服务架构便体现出其灵活性，因为每个服务可以根据实际需求独立进行扩展。

微服务的缺点

- 技术难度提高，微服务架构下会比传统架构多出几倍的应用数，他们相互之间的调用复杂度成指数级别增长，传统的技术方案很难满足微服务架构模式，因此需要使用更多的技术来保障整个架构的稳定性。
- 运维复杂，微服务架构下对运维有了新的调整，一次性部署几百台服务成为了常见的工作，因此自动化部署、监控、运维手段都需要提升。
- 性能影响，微服务之间大多是跨进程访问，并且因为服务切割的粒度变小，调用的链条越长，所以在性能上有所损耗，需要权衡。

微服务架构下如何解决数据问题

每个微服务都有自己独立的数据库，导致后台关联查询业务实现起来比较复杂，是大家普遍都会遇到的一个问题。我们也算是踩过一段时间的坑。主要有三个解决方案。

严格按照微服务的划分来做，微服务相互独立，各微服务数据库也独立，后台需要展示数据时，调用各微服务的接口来获取对应的数据，再进行数据处理后展示出来，这是标准的用法，也是最麻烦的用法。

将业务高度相关的表放到一个库中，将业务关系不是很紧密的表严格按照微服务模式来拆分，这样既可以使用微服务，也避免了数据库分散导致后台系统统计功能难以实现，是一个折中的方案。

数据库严格按照微服务的要求来切分，以满足业务高并发，实时或者准实时将各微服务数据库数据同步到NoSQL数据库中，在同步的过程中进行数据清洗，用来满足后台业务系统的使用，推荐使用MongoDB、HBase等。

三种方案在不同的公司我都使用过，第一种方案适合业务较为简单的小公司；第二种方案，适合在原有系统之上，慢慢演化为微服务架构的公司；第三种适合大型高并发的互联网公司。

一般一个微服务享用这个数据库的管理权，其他服务如果需要写入或者读取此服务内的数据，都应该通过此服务的接口来调用。

微服务架构下如何解决事务问题

分布式事务就是指事务的参与者、支持事务的服务器、资源服务器以及事务管理器分别位于不同的分布式系统的不同节点之上。 简单的说，就是一次大的操作由不同的小操作组成，这些小的操作分布在不同的服务器上，且属于不同的应用，分布式事务需要保证这些小操作要么全部成功，要么全部失败。本质上来说，分布式事务就是为了保证不同数据库的数据一致性。

分布式事务的目的是保障分库数据一致性，而跨库事务会遇到各种不可控制的问题，如个别节点永久性宕机，像单机事务一样的 ACID 是无法奢望的。另外，业界著名的 CAP 理论也告诉我们，对分布式系统，需要将数据一致性和系统可用性、分区容忍性放在天平上一起考虑。

目前主流的方案有以下三种：

二阶段提交 2PC，强一致性

二阶段提交的算法思路可以概括为：参与者将操作成败通知协调者，再由协调者根据所有参与者的反馈情况，决定各参与者是否要提交操作还是中止操作。

2PC 的核心原理是通过提交分阶段和记日志的方式，记录下事务提交所处的阶段状态，在组件宕机重启后，可通过日志恢复事务提交的阶段状态，并在这个状态节点重试。

消息机制，最终一致性

核心思想是将需要分布式处理的任务通过消息或者日志的方式来异步执行，消息或日志可以存到本地文件、数据库或消息队列，再通过业务规则进行失败重试，它要求各服务的接口是幂等的。

借助消息队列，在处理业务逻辑的地方，发送消息，业务逻辑处理成功后，提交消息，确保消息是发送成功的，之后消息队列投递来进行处理，如果成功，则结束，如果没有成功，则重试，直到成功，不过仅仅适用业务逻辑中，第一阶段成功，第二阶段必须成功的场景。

TCC 补偿模式，最终一致性

TCC 属于补偿型柔性事务，本质也是一个两阶段型事务，这与 JTA 是极为相似的，但是与 JTA 的不同点是，JTA 属于资源层事务，而 TCC 是服务层事务。

在一个长事务（ long-running ）中 ，一个由两台服务器一起参与的事务，服务器 A 发起事务，服务器 B 参与事务，B 的事务需要人工参与，所以处理时间可能很长。如果按照 ACID 的原则，要保持事务的隔离性、一致性，服务器 A 中发起的事务中使用到的事务资源将会被锁定，不允许其他应用访问到事务过程中的中间结果，直到整个事务被提交或者回滚。这就造成事务A中的资源被长时间锁定，系统的可用性将不可接受。

WS-BusinessActivity 提供了一种基于补偿的 long-running 的事务处理模型。还是上面的例子，服务器 A 的事务如果执行顺利，那么事务 A 就先行提交，如果事务 B 也执行顺利，则事务 B 也提交，整个事务就算完成。但是如果事务 B 执行失败，事务 B 本身回滚，这时事务 A 已经被提交，所以需要执行一个补偿操作，将已经提交的事务 A 执行的操作反操作，恢复到未执行前事务 A 的状态。这样的 SAGA 事务模型，是牺牲了一定的隔离性和一致性的，但是提高了 long-running 事务的可用性。

特别建议：在微服务架构下，能不使用分布式事务尽量不要使用，如果非得使用的话，结合自己的业务分析，看看自己的业务比较适合哪一种，选择强一致，还是最终一致即可。

Spring Cloud 相关

Spring Boot 和 Spring Cloud 有什么关系？

Spring Boot 是一套快速开发工具集合，重构了整个开发流程，简化了人们使用 Spring 的难度，让普通开发人员可以基于 Spring Boot 快速构建应用开发。Spring Boot 专注于应用个体。

Spring Cloud 是一系列框架的有序集合。它利用 Spring Boot 的开发便利性巧妙地简化了分布式系统基础设施的开发，如服务发现注册、配置中心、消息总线、负载均衡、断路器、数据监控等，Spring Cloud 专注于全局的服务治理。

总结一下，Spring Boot 是快速开发工具，专业于应用个体；Spring Cloud 基于 Spring Boot 开发，因此继承了 Spring Boot 的特性，另外 Spring Cloud 更专注于服务治理。

使用 Spring Cloud 有什么优势？

- Spring Cloud 来源于 Spring，质量、稳定性、持续性都可以得到保证
- Spring Cloud天然支持 Spring Boot，更加便于业务落地。
- Spring Cloud 发展非常的快，可以看到业界越来越多的公司选择了 Spring Cloud
- Spring Cloud 是 Java 领域最适合做微服务的框架。
- 相比于其它框架,Spring Cloud 对微服务周边环境的支持力度最大。
- 对于中小企业来讲，使用门槛较低。

Spring Cloud 和 Dubbo 有什么区别

- Dubbo，是阿里巴巴服务化治理的核心框架，并被广泛应用于中国各互联网公司；Spring Cloud 来源于 Spring 家族产品，Spring 是一家历史悠久的开源公司。
- Dubbo 是一个非常优秀的服务治理框架，并且在服务治理、灰度发布、流量分发比 Spring Cloud 更全面。
- Spring Cloud 几乎考虑了服务治理的方方面面，拥有 Spring Boot 的快速开发的特性，易用使用和部署。
- Dubbo 关注的领域是 Spring Cloud 的一个子集，Dubbo 专注服务治理，Spring Cloud 覆盖整个微服务架构领域。
- Dubbo 使用 RPC 调用效率很高一些，Spring Cloud 使用 Http 调用效率低，使用更简单。
- Dubbo 曾经使用范围更广泛，现在 Spring Cloud 使用更普遍。

Netflix 和 Spring Cloud 有什么关系

Netflix 是一家成功实践微服务架构的互联网公司，几年前，Netflix 就把它的几乎整个微服务框架栈开源贡献给了社区。Spring 背后的 Pivotal 公司在 2015 年推出的 Spring Cloud 开源产品，主要对 Netflix 开源组件的进一步封装，方便 Spring 开发人员构建微服务基础框架。

Netflix 在 Spring Cloud 体系内贡献的产品主要有：Eureka、Hystrix、Zuul、ribbon、turbine、archaius 等，最新版本中这些产品都处于维护状态。

Rest 和 Rpc 对比

- Rest 风格的系统交互更方便，Rpc 调用服务提供方和调用方式之间依赖太强
- Rest 调用系统性能较低，Rpc 调用效率比 Rest 高。
- Rest 的灵活性可以跨系统跨语言调用，Rpc 只能在同语言内调用
- Rest 可以和 swagger 等工具整合，自动输出接口 API 文档

Spring Cloud 组件

Eeruka 续约与剔除

服务实例启动后，会周期性地向 Eureka Server 发送心跳以续约自己的信息，避免自己的注册信息被剔除。续约的方式与服务注册基本一致：首先更新自身状态，再同步到其它 Peer。

如果 Eureka Server 在一段时间内没有接收到某个微服务节点的心跳，Eureka Serve r将会注销该微服务节点（自我保护模式除外）。

Eeruka 可能会出现服务注册延迟的情况，为什么

服务端的更改可能需要2分钟才能传播到所有客户端，这是因为 Eureka 有三处缓存和一处延迟造成的。

- Eureka Server 对注册列表进行缓存，默认时间为 30s。
- Eureka Client 对获取到的注册信息进行缓存，默认时间为 30s。
- Ribbon 会从上面提到的 Eureka Client 获取服务列表，将负载均衡后的结果缓存30s。

如果不是在 Spring Cloud 环境下使用这些组件(Eureka, Ribbon)，服务启动后并不会马上向 Eureka 注册，而是需要等到第一次发送心跳请求时才会注册。心跳请求的发送间隔默认是30s。Spring Cloud 对此做了修改，服务启动后会马上注册。

Eureka 自我保护机制是什么

默认情况下，如果 Eureka Server 在一定 90s 内没有接收到某个微服务实例的心跳，会注销该实例。但是在微服务架构下服务之间通常都是跨进程调用，网络通信往往会面临着各种问题，比如微服务状态正常，网络分区故障，导致此实例被注销。当 Eureka Server 检测到大规模服务异常时，会自动进入自我保护机制。

Eureka Server 进入自我保护机制，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

Eureka 和 Consul 有什么区别

功能对比见下表：

Feature	Euerka	Consul
服务健康检查	可配支持	服务状态，内存，硬盘等
多数据中心	—	支持
kv 存储服务	—	支持
一致性	—	raft
cap	ap	cp
使用接口(多语言能力)	http（sidecar）	支持 http 和 dns
watch 支持	支持 long polling/大部分增量	全量/支持long polling
自身监控	metrics	metrics
安全	—	acl /https
编程语言	Java	go
Spring Cloud 集成	已支持	已支持

Consul 服务注册相比 Eureka 会稍慢一些。因为 Consul 的 raft 协议要求必须过半数的节点都写入成功才认为注册成功, Leader 挂掉时，重新选举期间整个 Consul 不可用。Eureka 服务注册快，但可能会出现数据不一致的情况。

Ribbon 和 Feign 有什么区别

Ribbon
是一个基于 HTTP 和 TCP 客户端的负载均衡器
它可以在客户端配置 ribbonServerList（服务端列表），然后轮询请求以实现均衡负载。Feign 是在 Ribbon 的基础上进行了一次改进，是一个使用起来更加方便的 HTTP 客户端，让调用者更像是调用一个本地方法一样调用远程服务。

总结为三点：

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

相对繁琐。Feign 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。

什么是服务熔断？什么是服务降级？

熔断机制是应对雪崩效应的一种微服务链路保护机制。当某个微服务不可用或者响应时间太长时，会进行服务降级，进而熔断该节点微服务的调用，快速返回"错误"的响应信息。当检测到该节点微服务调用响应正常后恢复调用链路。在 Spring Cloud 框架里熔断机制通过 Resilience4j 实现，Resilience4j 会监控微服务间调用的状况，当失败的调用到一定阈值如果失败，就会启动熔断机制。

服务降级，当某个服务熔断之后，客户端将不再调用此请求，此时客户端返回一个本地的 fallback 回调，返回一个缺省值。这样做，虽然会出现局部的错误，但可以避免因为一个服务挂机，而影响到整个架构的稳定性。

服务网关的作用

有以下作用：

- 简化客户端调用复杂度，统一处理外部请求。
- 数据裁剪以及聚合，根据不同的接口需求，对数据加工后对外。
- 多渠道支持，针对不同的客户端提供不同的网关支持。
- 遗留系统的微服务化改造，可以作为新老系统的中转组件。
- 统一处理调用过程中的安全、权限问题

Spring Cloud 体系内的网关有 Zuul、Spring Cloud Gateway，最新版本推荐使用后者。

Spring Cloud bus 的作用

Spring cloud bus 通过轻量消息代理连接各个分布的节点。这会用在广播状态的变化（例如配置变化）或者其他的消息指令。Spring bus 的一个核心思想是通过分布式的启动器对 Spring Boot 应用进行扩展，也可以用来建立一个多个应用之间的通信频道。

目前唯一实现的方式是用 AMQP 消息代理作为通道，同样特性的设置（有些取决于通道的设置）在更多通道的文档中。

分布式追踪 Sleuth 工作原理

当我们项目中引入 `spring-cloud-starter-sleuth` 包后，每次链路请求都会添加一串追踪信息，格式是 `[server-name, main-traceId,sub-spanId,boolean]` 。

- server-name：服务结点名称；
- main-traceId：一条链路唯一的 ID，为 TraceID
- sub-spanId：链路中每一环的 ID，为 SpanID
- boolean：是否将信息输出到 Zipkin 等服务收集和展示。

这种机制是如何实现的呢？我们知道 Spring Cloud 微服务是通过 Http 协议通信的，所以 Sleuth 的实现也是基于 Http 的，为了在数据的收集过程中不能影响到正常业务，Sleuth 会在每个请求的 Header 上添加跟踪需求的重要信息，例如：

X-A1-TraceId：对应 TraceID；
X-A1-SpanId：对应 SpanID；
X-A1-ParentSpanId：前面一环的 SpanID；
X-A1-Sampled：是否被选中抽样输出；
X-Span-Name：工作单元名称。

这样在数据收集时，只需要将 Header 上的相关信息发送给对应的图像工具即可，图像工具根据上传的数据，按照 Span 对应的逻辑进行分析、展示。

总结

本节为大家介绍了常见的 Spring Cloud 面试题，跟着面试题回顾了整个 Spring Cloud 生态技术，研究不同时期 Spring Cloud 技术栈的发展，也可以同步了解行业的技术动态。在日常工作中，遇到问题时多做思考，在面试时其实很容易覆盖掉 Spring Cloud 大多数面试问题。也希望大家在以后的面试中，遇到 Spring Cloud 相关问题，如数家珍。

}

37 Sentinel 基本用法
39 专栏总结