微服务面试题

面试官：Spring Cloud 5大组件有哪些？

候选人：

早期我们一般认为的Spring Cloud五大组件是

- Eureka ：注册中心
- Ribbon ：负载均衡
- Feign ：远程调用
- Hystrix： 服务熔断
- Zuul/Gateway ：网关

随着SpringCloudAlibba在国内兴起，我们项目中使用了一些阿里巴巴的组件

- 注册中心/配置中心 Nacos
- 负载均衡 Ribbon
- 服务调用 Feign
- 服务保护 sentinel
- 服务网关 Gateway

面试官：服务注册和发现是什么意思？Spring Cloud 如何实现服务注册发现？

候选人：

我理解的是主要三块大功能，分别是服务注册 、服务发现、服务状态监控

我们当时项目采用的eureka作为注册中心，这个也是spring cloud体系中的一个核心组件

服务注册：服务提供者需要把自己的信息注册到eureka，由eureka来保存这些信息，比如服务名称、ip、端口等等

服务发现：消费者向eureka拉取服务列表信息，如果服务提供者有集群，则消费者会利用负载均衡算法，选择一个发起调用

服务监控：服务提供者会每隔30秒向eureka发送心跳，报告健康状态，如果eureka服务90秒没接收到心跳，从eureka中剔除

面试官：我看你之前也用过nacos、你能说下nacos与eureka的区别？

候选人：

我们当时xx项目就是采用的nacos作为注册中心，选择nacos还要一个重要原因就是它支持配置中心，不过nacos作为注册中心，也比eureka要方便好用一些，主要相同不同点在于几点：

- 共同点

Nacos与eureka都支持服务注册和服务拉取，都支持服务提供者心跳方式做健康检测

- Nacos与Eureka的区别

①Nacos支持服务端主动检测提供者状态：临时实例采用心跳模式，非临时实例采用主动检测模式

②临时实例心跳不正常会被剔除，非临时实例则不会被剔除

③Nacos支持服务列表变更的消息推送模式，服务列表更新更及时

④Nacos集群默认采用AP方式，当集群中存在非临时实例时，采用CP模式；Eureka采用AP方式

面试官：你们项目负载均衡如何实现的？

候选人：

是这样~~

在服务调用过程中的负载均衡一般使用SpringCloud的Ribbon 组件实现，Feign的底层已经自动集成了Ribbon，使用起来非常简单

当发起远程调用时，ribbon先从注册中心拉取服务地址列表，然后按照一定的路由策略选择一个发起远程调用，一般的调用策略是轮询

面试官：Ribbon负载均衡策略有哪些？

候选人：

我想想啊，有很多种，我记得几个：

- RoundRobinRule：简单轮询服务列表来选择服务器

- WeightedResponseTimeRule：按照权重来选择服务器，响应时间越长，权重越小
- RandomRule：随机选择一个可用的服务器
- ZoneAvoidanceRule：区域敏感策略，以区域可用的服务器为基础进行服务器的选择。使用Zone对服务器进行分类，这个Zone可以理解为一个机房、一个机架等。而后再对Zone内的多个服务做轮询(默认)

面试官：如果想自定义负载均衡策略如何实现？

候选人：

提供了两种方式：

1，创建类实现IRule接口，可以指定负载均衡策略，这个是全局的，对所有的远程调用都起作用

2，在客户端的配置文件中，可以配置某一个服务调用的负载均衡策略，只是对配置的这个服务生效远程调用

面试官：什么是服务雪崩，怎么解决这个问题？

候选人：

服务雪崩是指一个服务失败，导致整条链路的服务都失败的情形，一般我们在项目解决的话就是两种方案，第一个是服务降级，第二个是服务熔断，如果流量太大的话，可以考虑限流

服务降级：服务自我保护的一种方式，或者保护下游服务的一种方式，用于确保服务不会受请求突增影响变得不可用，确保服务不会崩溃，一般在实际开发中与feign接口整合，编写降级逻辑

服务熔断：默认关闭，需要手动打开，如果检测到 10 秒内请求的失败率超过 50%，就触发熔断机制。之后每隔 5 秒重新尝试请求微服务，如果微服务不能响应，继续走熔断机制。如果微服务可达，则关闭熔断机制，恢复正常请求

面试官：你们的微服务是怎么监控的？

候选人：

我们项目中采用的skywalking进行监控的

1，skywalking主要可以监控接口、服务、物理实例的一些状态。特别是在压测的时候可以看到众多服务中哪些服务和接口比较慢，我们可以针对性的分析和优化。

2，我们还在skywalking设置了告警规则，特别是在项目上线以后，如果报错，我们分别设置了可以给相关负责人发短信和发邮件，第一时间知道项目的bug情况，第一时间修复

面试官：你们项目中有没有做过限流？怎么做的？

候选人：

我当时做的xx项目，采用就是微服务的架构，因为xx因为，应该会有突发流量，最大QPS可以达到2000，但是服务支撑不住，我们项目都通过压测最多可以支撑1200QPS。因为我们平时的QPS也就不到100，为了解决这些突发流量，所以采用了限流。

【版本1】

我们当时采用的nginx限流操作，nginx使用的漏桶算法来实现过滤，让请求以固定的速率处理请求，可以应对突发流量，我们控制的速率是按照ip进行限流，限制的流量是每秒20

【版本2】

我们当时采用的是spring cloud gateway中支持局部过滤器RequestRateLimiter来做限流，使用的是令牌桶算法，可以根据ip或路径进行限流，可以设置每秒填充平均速率，和令牌桶总容量

面试官：限流常见的算法有哪些呢？

候选人：

比较常见的限流算法有漏桶算法和令牌桶算法

漏桶算法是把请求存入到桶中，以固定速率从桶中流出，可以让我们的服务做到绝对的平均，起到很好的限流效果

令牌桶算法在桶中存储的是令牌，按照一定的速率生成令牌，每个请求都要先申请令牌，申请到令牌以后才能正常请求，也可以起到很好的限流作用

它们的区别是，漏桶和令牌桶都可以处理突发流量，其中漏桶可以做到绝对的平滑，令牌桶有可能会产生突发大量请求的情况，一般nginx限流采用的漏桶，spring cloud gateway中可以支持令牌桶算法

面试官：什么是CAP理论？

候选人：

CAP主要是在分布式项目下的一个理论。包含了三项，一致性、可用性、分区容错性

- 一致性(Consistency)是指更新操作成功并返回客户端完成后，所有节点在同一时间的数据完全一致(强一致性)，不能存在中间状态。
- 可用性(Availability) 是指系统提供的服务必须一直处于可用的状态，对于用户的每一个操作请求总是能够在有限的时间内返回结果。
- 分区容错性(Partition tolerance) 是指分布式系统在遇到任何网络分区故障时，仍然需要能够保证对外提供满足一致性和可用性的服务，除非是整个网络环境都发生了故障。

面试官：为什么分布式系统中无法同时保证一致性和可用性？

候选人：

嗯，是这样的~~

首先一个前提，对于分布式系统而言，分区容错性是一个最基本的要求，因此基本上我们在设计分布式系统的时候只能从一致性（C）和可用性（A）之间进行取舍。

如果保证了一致性（C）：对于节点N1和N2，当往N1里写数据时，N2上的操作必须被暂停，只有当N1同步数据到N2时才能对N2进行读写请求，在N2被暂停操作期间客户端提交的请求会收到失败或超时。显然，这与可用性是相悖的。

如果保证了可用性（A）：那就不能暂停N2的读写操作，但同时N1在写数据的话，这就违背了一致性的要求。

面试官：什么是BASE理论？

候选人：

嗯，这个也是CAP分布式系统设计理论

BASE是CAP理论中AP方案的延伸，核心思想是即使无法做到强一致性（StrongConsistency，CAP的一致性就是强一致性），但应用可以采用适合的方式达到最终一致性（Eventual Consitency）。它的思想包含三方面：

1、Basically Available（基本可用）：基本可用是指分布式系统在出现不可预知的故障的时候，允许损失部分可用性，但不等于系统不可用。

2、Soft state（软状态）：即是指允许系统中的数据存在中间状态，并认为该中间状态的存在不会影响系统的整体可用性，即允许系统在不同节点的数据副本之间进行数据同步的过程存在延时。

3、Eventually consistent（最终一致性）：强调系统中所有的数据副本，在经过一段时间的同步后，最终能够达到一个一致的状态。其本质是需要系统保证最终数据能够达到一致，而不需要实时保证系统数据的强一致性。

面试官：你们采用哪种分布式事务解决方案？

候选人：

我们当时是xx项目，主要使用到的seata的at模式解决的分布式事务

seata的AT模型分为两个阶段：

1、阶段一RM的工作：① 注册分支事务 ② 记录undo-log（数据快照）③ 执行业务sql并提交 ④报告事务状态

2、阶段二提交时RM的工作：删除undo-log即可

3、阶段二回滚时RM的工作：根据undo-log恢复数据到更新前

at模式牺牲了一致性，保证了可用性，不过，它保证的是最终一致性

面试官：分布式服务的接口幂等性如何设计？

候选人：

嗯，我们当时有一个xx项目的下单操作，采用的token+redis实现的，流程是这样的

第一次请求，也就是用户打开了商品详情页面，我们会发起一个请求，在后台生成一个唯一token存入redis，key就是用户的id，value就是这个token，同时把这个token返回前端

第二次请求，当用户点击了下单操作会后，会携带之前的token，后台先到redis进行验证，如果存在token，可以执行业务，同时删除token；如果不存在，则直接返回，不处理业务，就保证了同一个token只处理一次业务，就保证了幂等性

面试官：xxl-job路由策略有哪些？

候选人：

xxl-job提供了很多的路由策略，我们平时用的较多就是：轮询、故障转移、分片广播...

面试官：xxl-job任务执行失败怎么解决？

候选人：

有这么几个操作

第一：路由策略选择故障转移，优先使用健康的实例来执行任务

第二，如果还有失败的，我们在创建任务时，可以设置重试次数

第三，如果还有失败的，就可以查看日志或者配置邮件告警来通知相关负责人解决

面试官：如果有大数据量的任务同时都需要执行，怎么解决？

候选人：

我们会让部署多个实例，共同去执行这些批量的任务，其中任务的路由策略是分片广播

在任务执行的代码中可以获取分片总数和当前分片，按照取模的方式分摊到各个实例执行就可以了