# 03 | 熔断：熔断-恢复-熔断-恢复，抖来抖去怎么办？

2023-06-21 邓明 来自北京



《后端工程师的高阶面经》

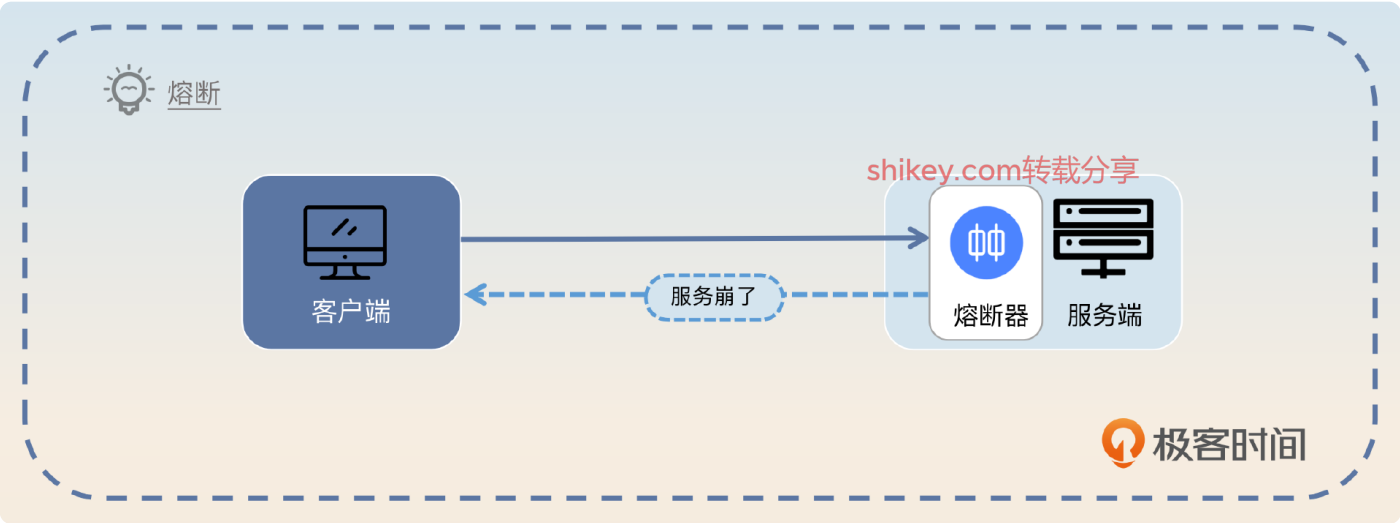
你好，我是大明。今天我们继续学习微服务架构，这节课我们讨论一个新的主题：熔断。

在微服务架构里面，熔断 - 限流 - 降级一般是连在一起讨论的，熔断作为微服务架构可用性保障的重要手段之一，是我们必须要掌握的，而且要能够说清楚自己在实践中是怎么利用熔断来提高系统的可用性的。

所以今天我就来给你梳理一下关于熔断有哪些知识是我们必须要掌握的，另外我还会在这个基础上给你提供一个全方位的熔断面试方案，帮助你在面试中脱颖而出。

## 前置知识

熔断在微服务架构里面是指当微服务本身出现问题的时候，它会拒绝新的请求，直到微服务恢复。




看图，你会不会觉得有点困惑？服务端明明返回了错误的响应，怎么还说熔断提高了系统的可用性呢？

答案就是**熔断可以给服务端恢复的机会**。试想这么一个场景，CPU 使用率已经 100% 了，服务端因此触发了熔断。那么拒绝了新来的请求之后，服务端的 CPU 使用率就会在一段时间内降到 100% 以内。

回到熔断的基本定义上来，我们可以提炼出两个点进一步讨论。



- 怎么判断微服务出现了问题？
- 怎么知道微服务恢复了？

接下来我们要讨论的亮点也是围绕这两个方面来进行的。

## 判定服务的健康状态

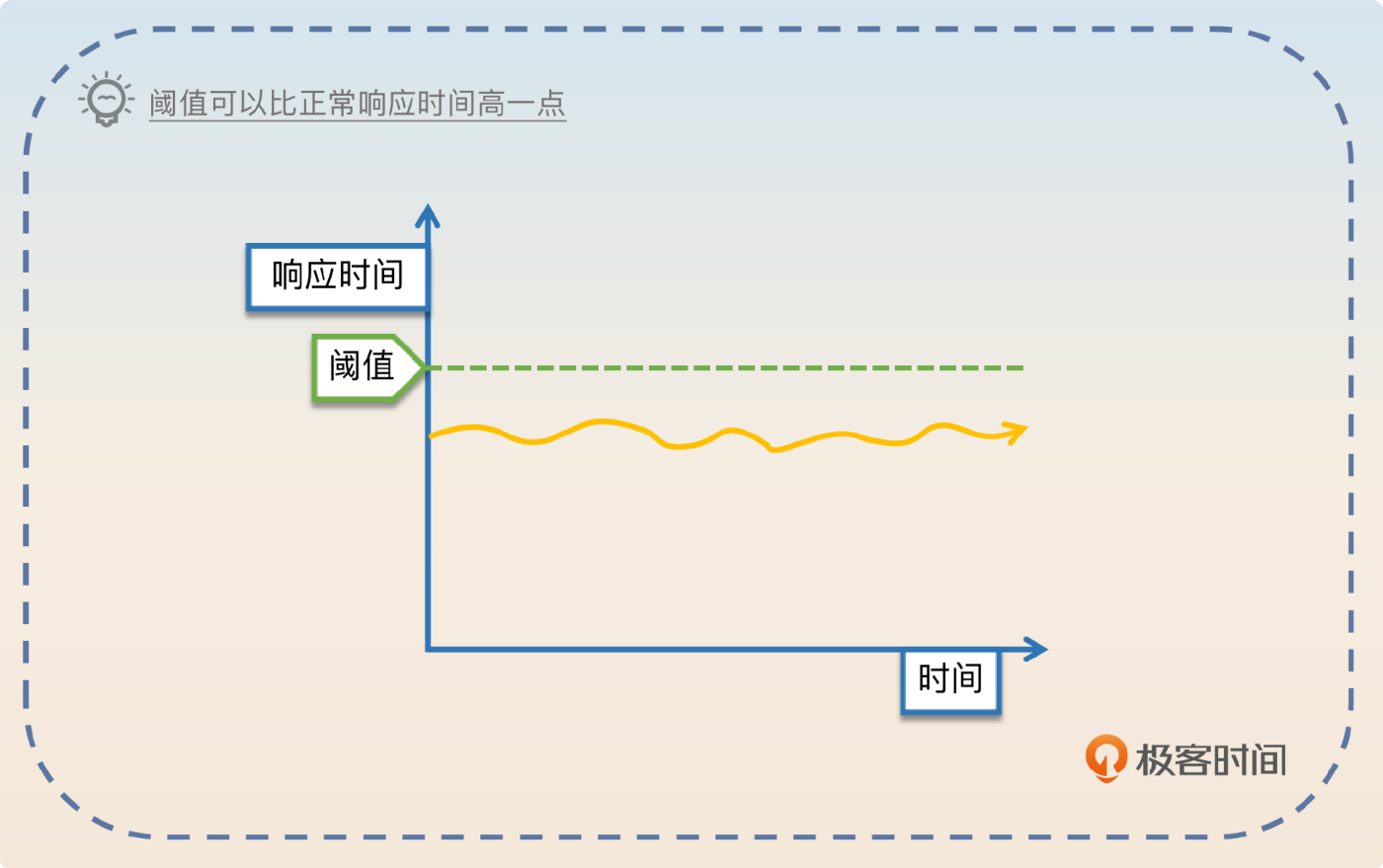
第一个问题，判断微服务是否出现了问题，它有点儿像我们在负载均衡里面讨论的动态算法。本质上也是要求你根据自己的业务来选择一些指标，代表这个服务器的健康程度。比如说一般可以考虑使用响应时间、错误率。

不管选择什么指标，都要考虑两个因素：**一是阈值如何选择；二是超过阈值之后，要不要持续一段时间才触发熔断。**

比如我们把响应时间作为指标，那么响应时间超过多少应该触发熔断呢？这是根据业务来决定的。比如说如果业务对响应时间的要求是在 1s 以内，那么你的阈值就可以设定在 1s，或者稍高一点，留点容错的余地也可以。



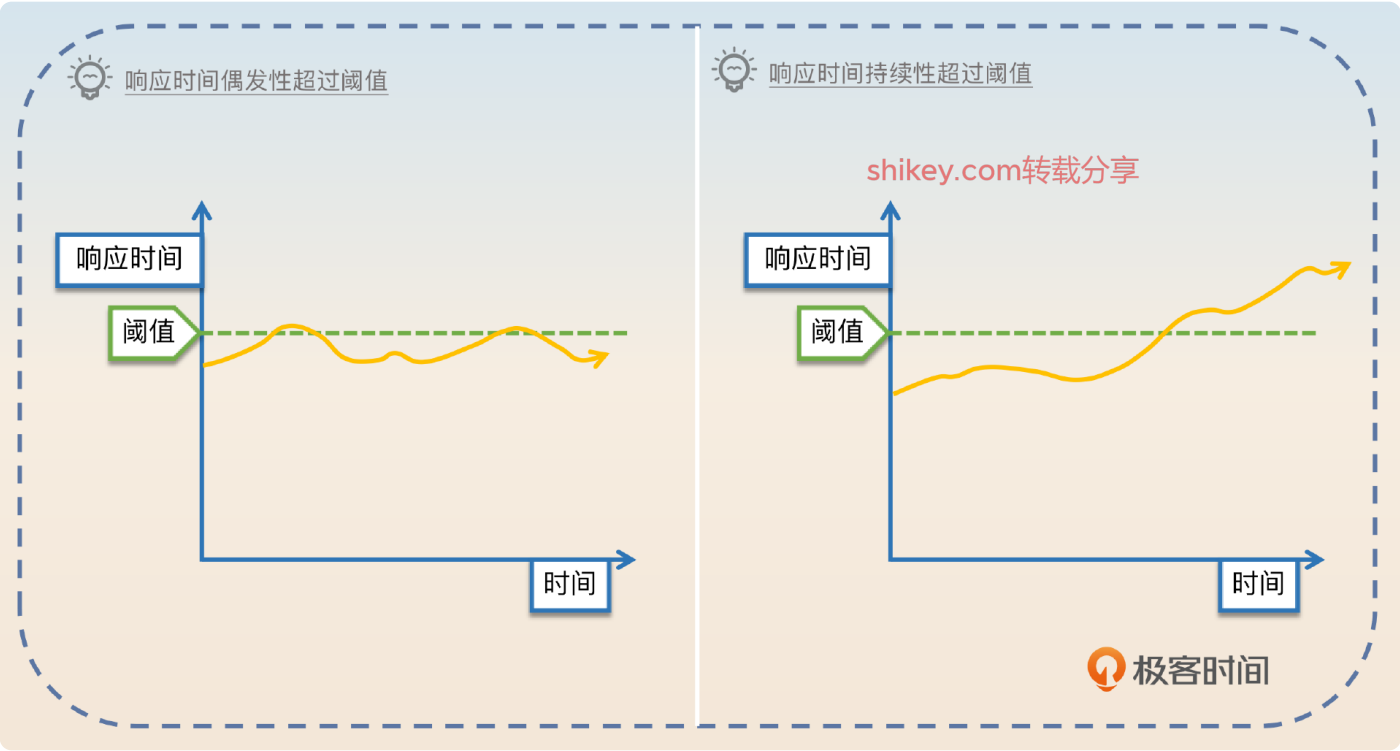
那么如果你的产品经理没跟你说这个业务对响应时间的要求，你就可以根据它的整体响应时间设定一个阈值，原则上阈值应该明显超过正常响应时间。比如你经过一段时间的观测之后，发现这个服务的 99 线是 1s，那么你可以考虑将熔断阈值设定为 1.2s。




那么是不是响应时间一旦超过了阈值就立刻熔断呢？一般也不是，而是**要求响应时间超过一段时间之后才触发熔断**。这主要是出于两个考虑，一个是响应时间可能是偶发性地突然增长；另外一个则是防止抖动。防止抖动这个问题后面我会和你进一步讨论。

那么这个“一段时间”究竟有多长，很大程度上就依赖个人经验了。如果时间过短，可能会频繁触发熔断，然后又恢复，再熔断，再恢复……反过来，如果时间过长，那就可能会导致该触发熔断的时候迟迟没有触发。

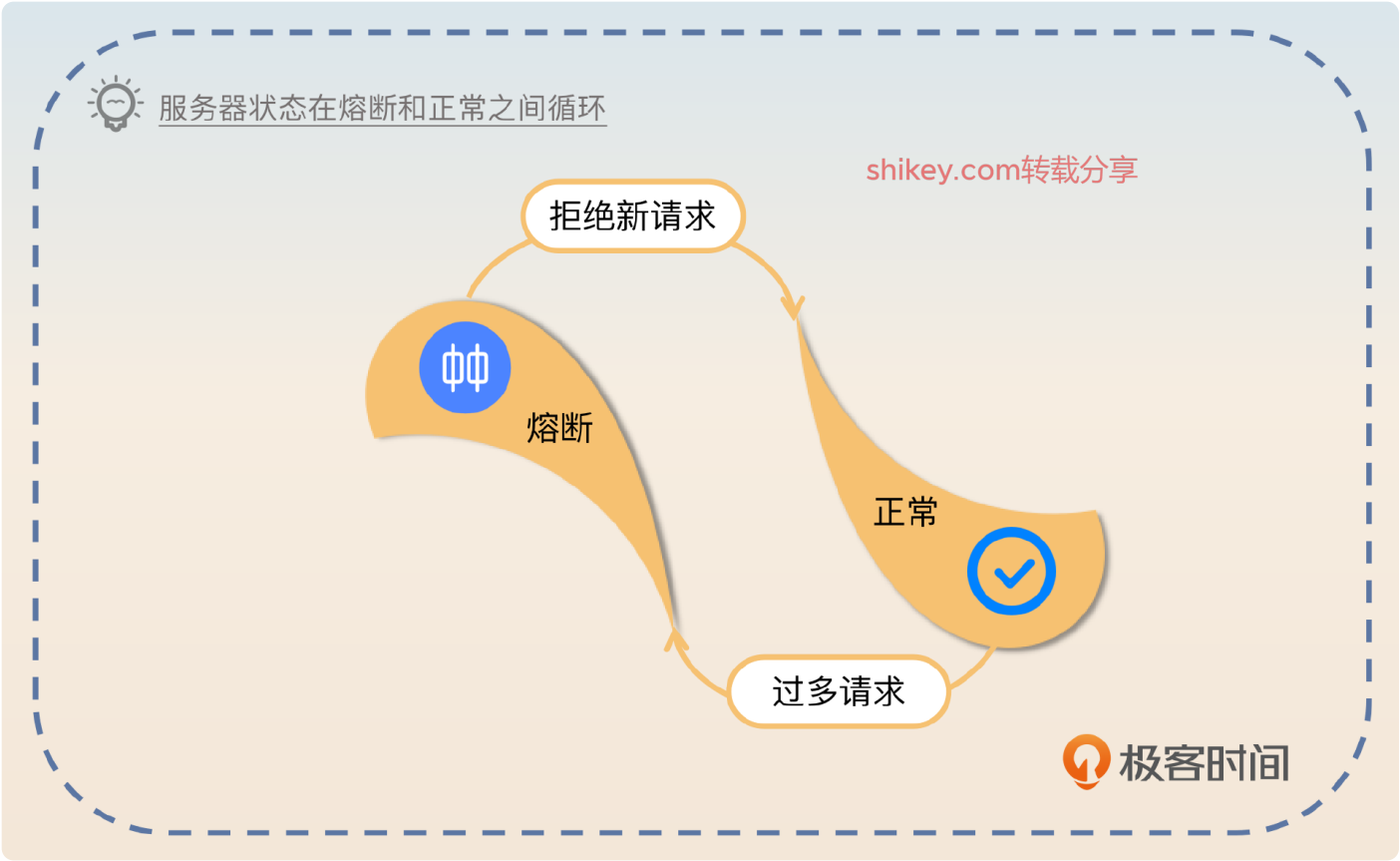
你可以根据经验来设定一个值，比如说三十秒或者一分钟。




当然最简单的做法就是超过阈值就直接触发熔断，但是采取这种策略就要更加小心抖动问题。

## 服务恢复正常

第二个问题，一个服务熔断之后要考虑恢复。比如说如果我们判断一个服务响应时间过长，进入了熔断状态。那么十分钟过后，已接收的请求已经被处理完了，即服务恢复正常了，那么它就要退出熔断状态，继续接收新请求。



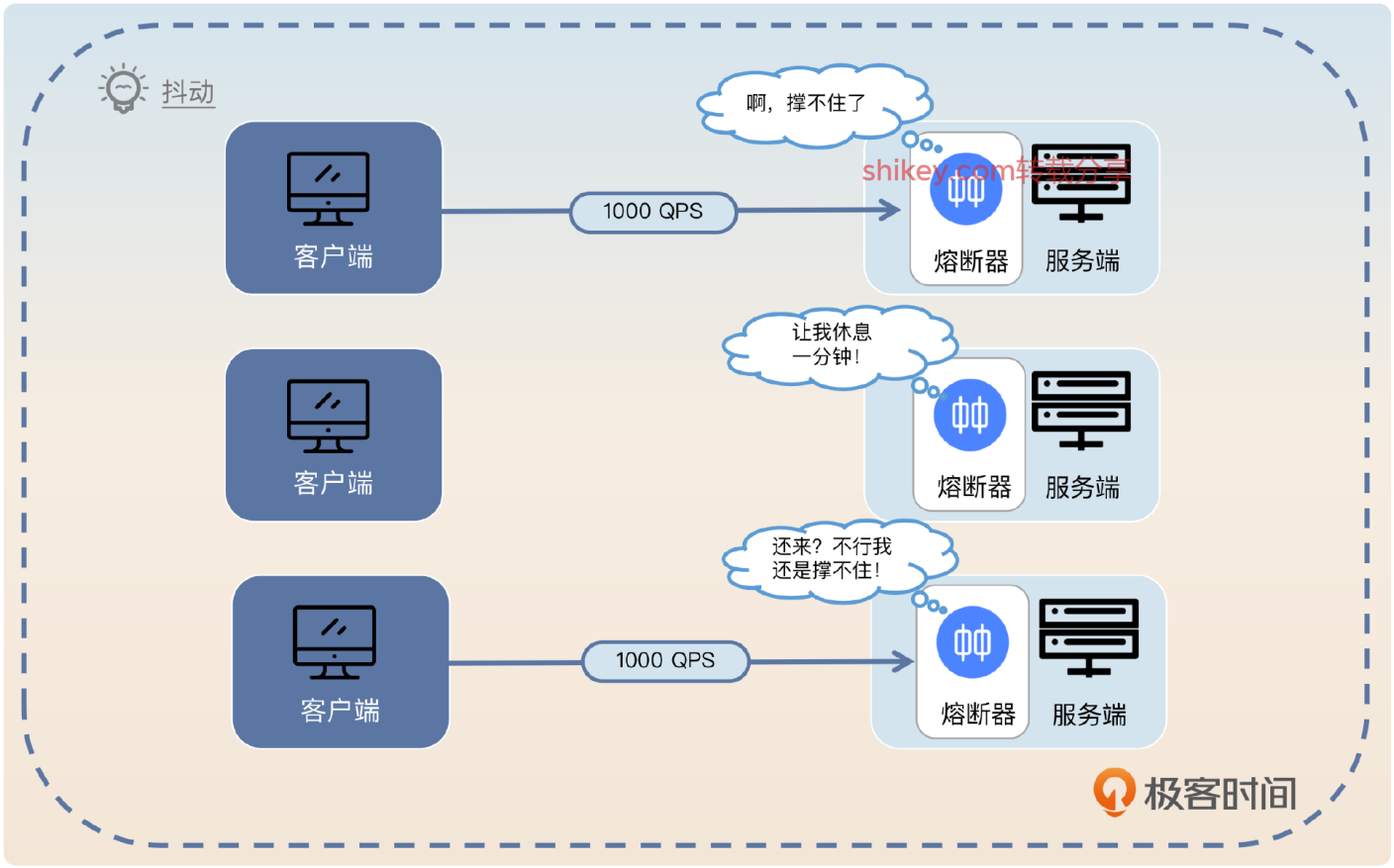

因此在触发熔断之后，就要考虑检测服务是否已经恢复正常。

很可惜，这方面微服务框架都做得比较差。大多数情况下就是触发熔断之后保持一段时间，比如说一分钟，一分钟之后就认为服务已经恢复正常，继续处理新请求。

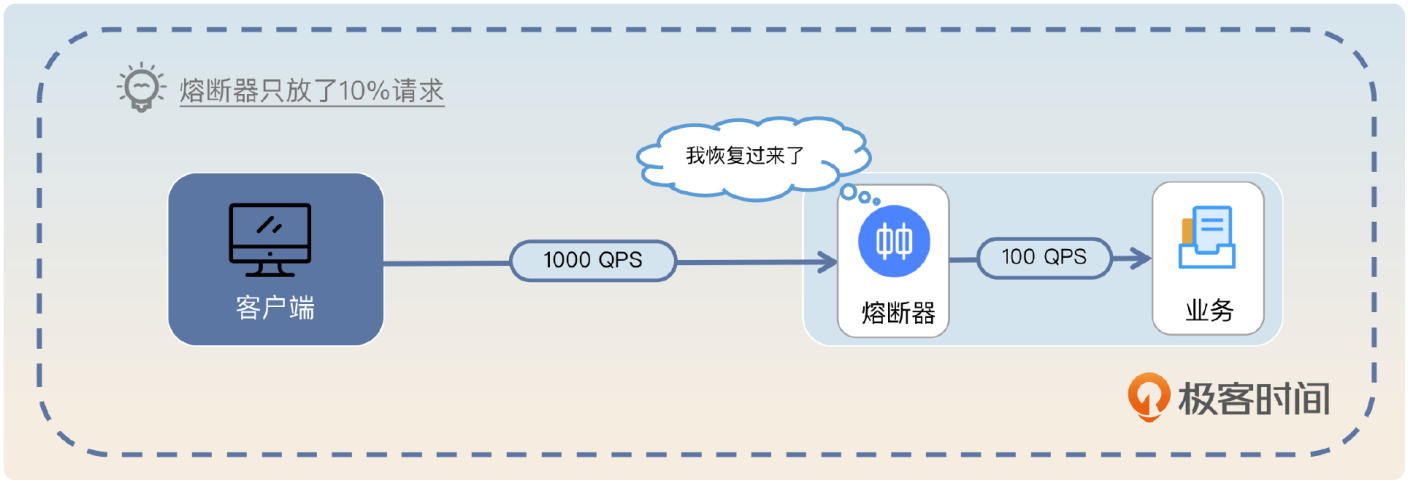
不过这里就涉及到我前面多次提到的抖动问题了。所谓**抖动就是服务频繁地在正常 - 熔断两个状态之间切换**。

引起抖动的原因是多样的，比如说前面提到的一旦超过阈值就进入熔断状态，或者我们这里说的恢复策略不当也会引起抖动。再比如刚刚我们提到的“一分钟后就认为服务已经恢复正常，继续处理新请求”就容易引发抖动问题。

你试想一下，如果本身熔断是高并发引起的。那么在一分钟后，并发依旧很高，这时候你一旦直接恢复正常，然后高并发的流量打过来，服务是不是又会触发熔断？




而要解决这个抖动问题，就需要在恢复之后控制住流量。比如说按照 10%、20%、30%……逐步递增，而不是立刻恢复 100% 的流量。




显然你能够看出来这种做法还是不够好。因为在这种逐步放开流量的措施下，依旧有请求因为熔断不会被处理。那么一个自然的想法就是，**能不能让客户端来控制这个流量？**简单来说就是服务端触发熔断之后，客户端就直接不再请求这个节点了，而是换一个节点。等到恢复了之后，客户端再逐步对这个节点放开流量。

当然可以，这也是我给出的亮点方案。

## 面试准备



这些就是关于熔断你要了解的基础知识，不过如果你想要彻底掌握，还需要把这些知识点和实际工作联系在一起。所以我建议你在面试之前，要弄清楚你所在的公司有没有用熔断来治理微服务。如果有，那么你需要进一步弄清楚下面这些情况。

1. 你们公司是怎么判断微服务出现故障的？比如说错误率、响应时间等等。
2. 你们公司是怎么判断微服务已经从故障中恢复过来的？
3. 在判断微服务已经恢复过来之后，有没有采取什么措施来防止抖动的问题？

关于熔断最佳的面试策略是把它作为你构建一个高可用微服务架构的一环。例如你在介绍某一个微服务项目的时候可以这样说。

> 这是一个高可用的微服务系统，为了保证它的可用性，我采取了限流、降级、熔断等措施。

此外，如果面试官问到服务治理以及提高系统可用性的方法之类的问题，你也可以用熔断来回答。又或者面试官问到了限流或者降级，那么你就可以尝试把话题引到熔断上面。此外，如果面试官问到某个服务崩溃了怎么办？这个问题相当于是在问怎么提高可用性防止服务崩溃，以及万一服务真崩溃了你也要有措施防止拖累别的服务，那么熔断就是一个可用的手段。

如果你现在有时间，在学完这节课的内容之后，就可以尝试在公司内部落地一下熔断，并且可以试试我给你的亮点方案，来加深印象以及对细节的把控。

## 基本思路

当面试官问“你有没有用过熔断”或者“怎么保障微服务可用性”的时候，你就可以介绍你使用的熔断。但是要根据我在前置知识里面的提示，你在面试的时候要说清楚什么时候判定服务需要触发熔断，为什么选用这个指标。

假如说你准备用**响应时间**来作为指标，那么你可以这么回答，关键词是**持续超过阈值**。

> 为了保障微服务的可用性，我在我的核心服务里面接入了熔断。针对不同的服务，我设计了不同的微服务熔断策略。



> 比如说最简单的熔断策略就是根据响应时间来进行。当响应时间超过阈值一段时间之后就会触发熔断。我一般会根据业务情况来选择这个阈值，例如，如果产品经理要求响应时间是 1s，那么我会把阈值设定在 1.2s。如果响应时间超过 1.2s，并且持续三十秒，就会触发熔断。在触发熔断的情况下，新请求会被拒绝，而已有的请求还是会被继续处理，直到服务恢复正常。

这里面试官就可能有很多种问法，但是我在前置知识里面都讨论到了。虽然他的问题可能千奇百怪，不过万变不离这几问。

1. 这阈值还可以怎么确定？那么你就回答还可以根据观测到的响应时间数据来确定。
2. 这个持续三十秒是如何计算出来的？这个问题其实可以坦白回答是基于个人经验，然后你解释一下过长或者过短的弊端就可以了。
3. 为什么多了 0.2s？那么你可以解释是留了余地，防止偶发性的响应时间变长的情况。
4. 怎么判断服务已经恢复正常了？那么你可以回答等待一段固定的时间，然后尝试逐渐放开流量。



如果你在实践中根据自己的业务特征选用了一些比较罕见的指标，或者你设计的触发熔断的条件比较有特色，那么也可以用自己的实际方案。

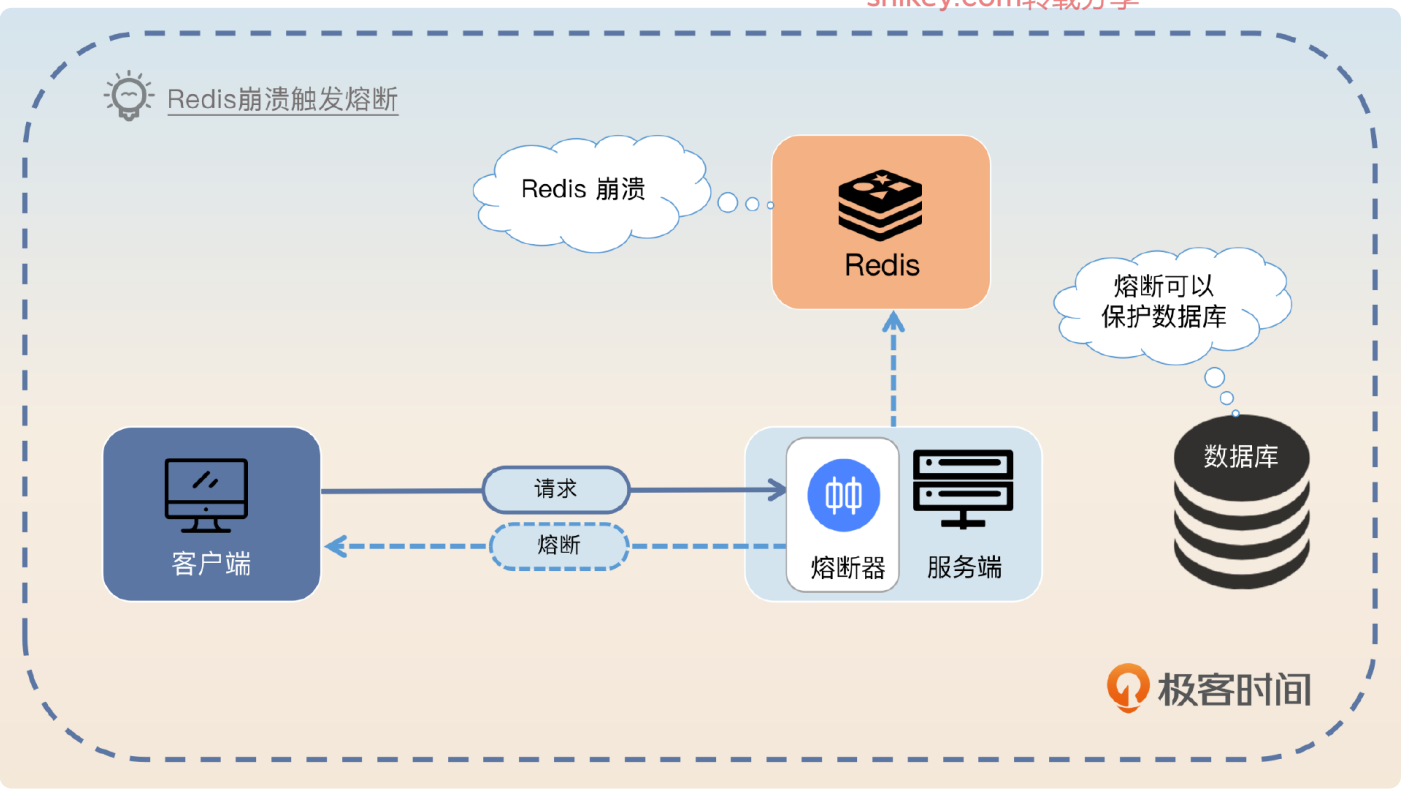
这里我给你另外一个微创新的方案，关键词是**缓存崩溃**。

> 我还设计过一个很有趣的熔断方案。我的一个接口并发很高，对缓存的依赖度非常严重。所以我的熔断策略是要是缓存不可用，比如说 Redis 崩溃了，那么我就会触发熔断。这里如果我不熔断的话，请求会因为 Redis 崩溃而全部落到 MySQL 上，基本上会压垮 MySQL。

> 在触发熔断之后，我会额外开启一个线程（如果是 Go 就换成 Goroutine）持续不断地 ping Redis。如果 Redis 恢复了，那么我就会退出熔断状态，新来的请求就不会被拒绝了。

这里我用 Redis 来作为例子，你可以将 Redis 替换为 MemCache 之类的，甚至你还可以将缓存替换成你业务上任何一个关键的第三方依赖。






这个方案里面我还留了一些可以引导的点。

缓存问题：在这里我提到了 Redis 失效，这种情况类似于缓存雪崩，那么你很自然地就可以把话题引导到如何处理缓存击穿、穿透、雪崩这些经典问题上。

高可用 MySQL：我在这里使用的是熔断来保护 MySQL，类似地，你也可以考虑用限流来保护 MySQL。

最后我提到了退出熔断状态，如果面试官了解抖动问题，那么他就肯定会追问“你是一次性放开全部流量吗？”，那么你就可以阐述抖动的问题，然后总结一下。

> 我这种逐步放开流量的方案其实还是有缺陷的，还有一些更加高级的做法，但是需要负载均衡来配合。

这个总结就是你留下的鱼饵，为了引出下面我给你展示的亮点方案。

## 亮点方案

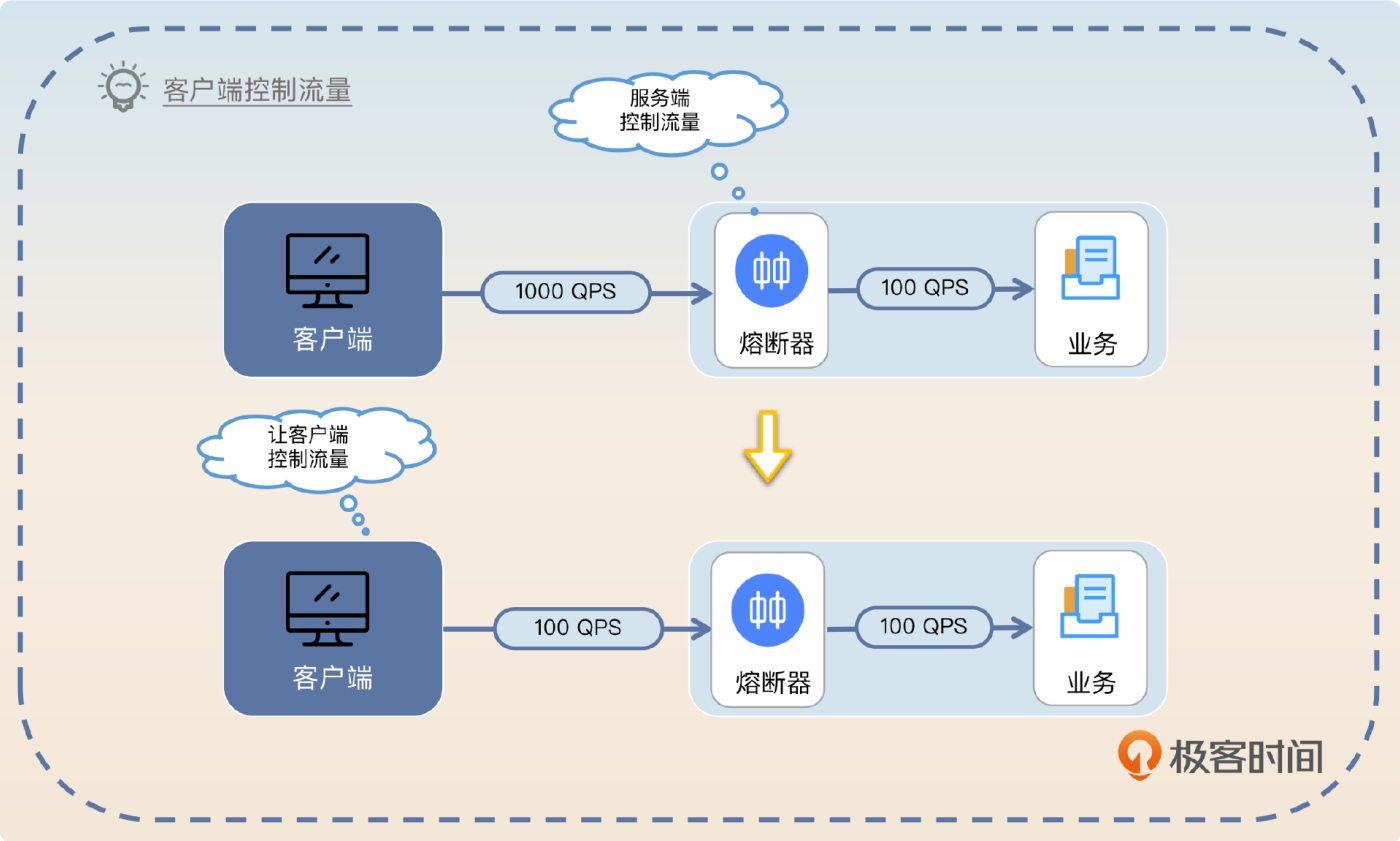
前面的基本思路如果你能答好，差不多也能通过跟熔断有关的面试了，而且有不小的概率能够给面试官留下你技术很不错的印象。但是你还可以进一步展示你在服务治理和服务可用性保证上的独到见解。这就需要用到下面我要给你讲的综合了负载均衡算法和熔断措施的方案了。



这个方案很简单，在落地的时候也不是很难。

我在讲抖动与恢复的时候提到，恢复的时候可以逐步放开流量。那么你是否注意到，这个放开流量是在服务端处理的，也就是说服务端还是收到了 100% 的流量，只不过只有部分流量会被放过去并且被正常处理。

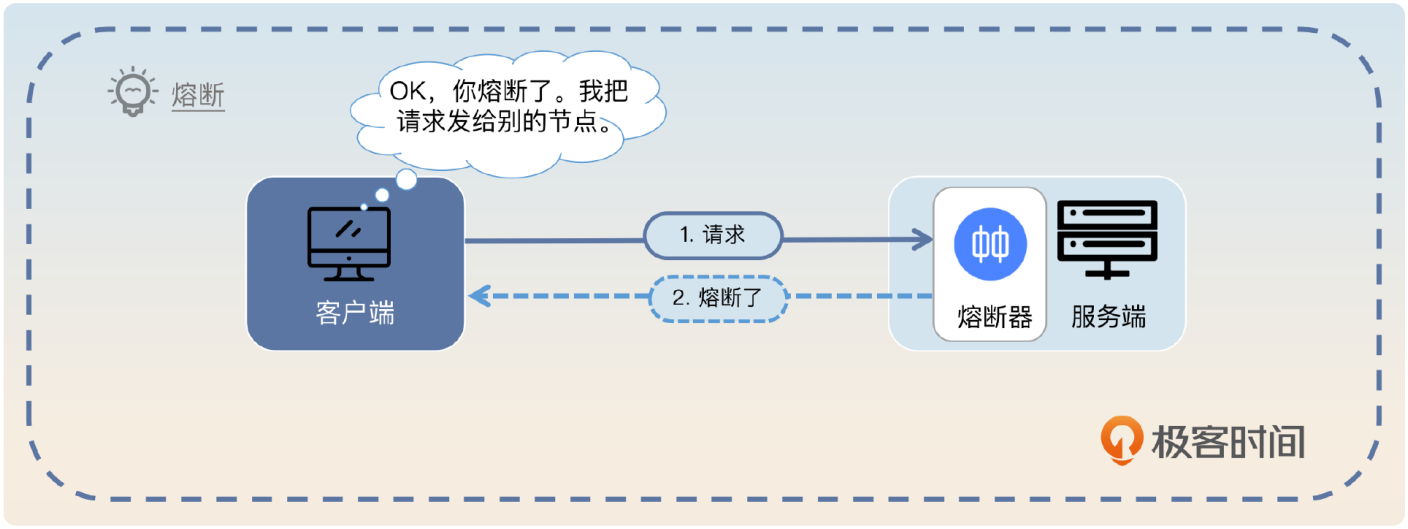
那么一个自然的想法就是**为什么不直接让客户端来控制这个流量呢?**

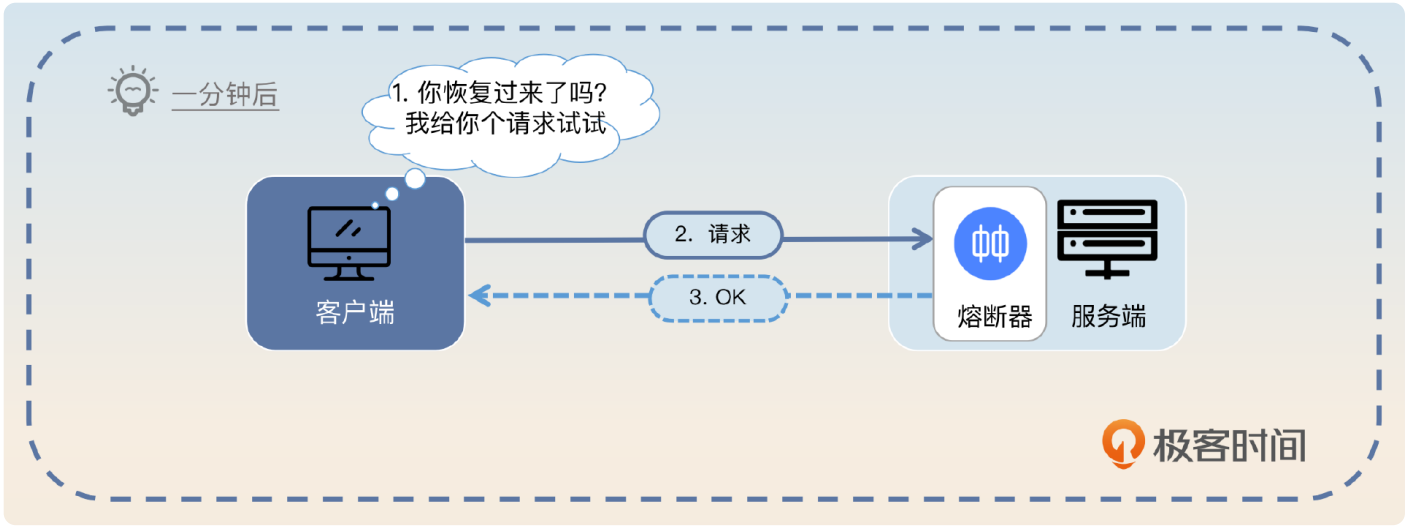


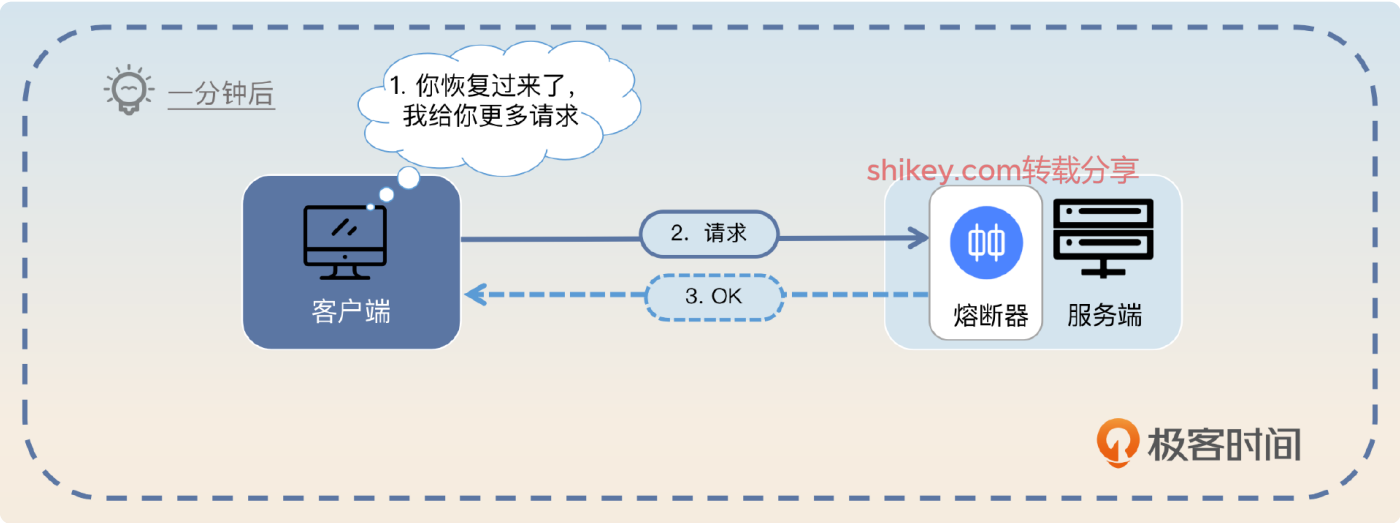

进一步结合我在负载均衡里面谈到的根据调用结果来调整负载均衡策略的讨论，是不是可以让客户端也采用这种负载均衡策略？答案是可以的。

整体流程：

1. 服务端在触发熔断的时候，会返回一个代表熔断的错误。
2. 客户端在收到这个错误之后，就会把这个服务端节点暂时挪出可用节点列表。后续所有的新请求都不会再打到这个触发了熔断的服务端节点上了。 
3. 客户端在等待一段时间后，逐步放开流量。
4. 如果服务端正常处理了新来的请求，那么客户端就加大流量。
5. 如果服务端再次返回了熔断响应，那么客户端就会再一次将这个节点挪出可用列表。
6. 如此循环，直到服务端完全恢复正常，客户端也正常发送请求到该服务端节点。






那么这里你就可以这样回答，关键词是**负载均衡**。

> 整体思路是利用负载均衡来控制流量。如果一个服务端节点触发了熔断，那么客户端在做负载均衡的时候就可以将这个节点挪出可用列表，后续请求会发给别的节点。在经过一段时间之后，客户端可以尝试发请求给该节点。如果该节点正确处理了，那客户端就可以加大流量。否则客户端就要再一次等待一段时间。

到这里你还可以自己杠自己一下，就是**万一所有可用节点都触发熔断了，应该怎么办？**你就可以这样来说。

> 这个方案是需要兜底的，比如说如果因为某些原因数据库出问题，导致某个服务所有的节点都触发了熔断，那么客户端就完全没有可用节点了。不过这个问题本身熔断解决不了，负载均衡也解决不了，只能通过监控告警之后人手工介入处理了。

## 面试思路总结

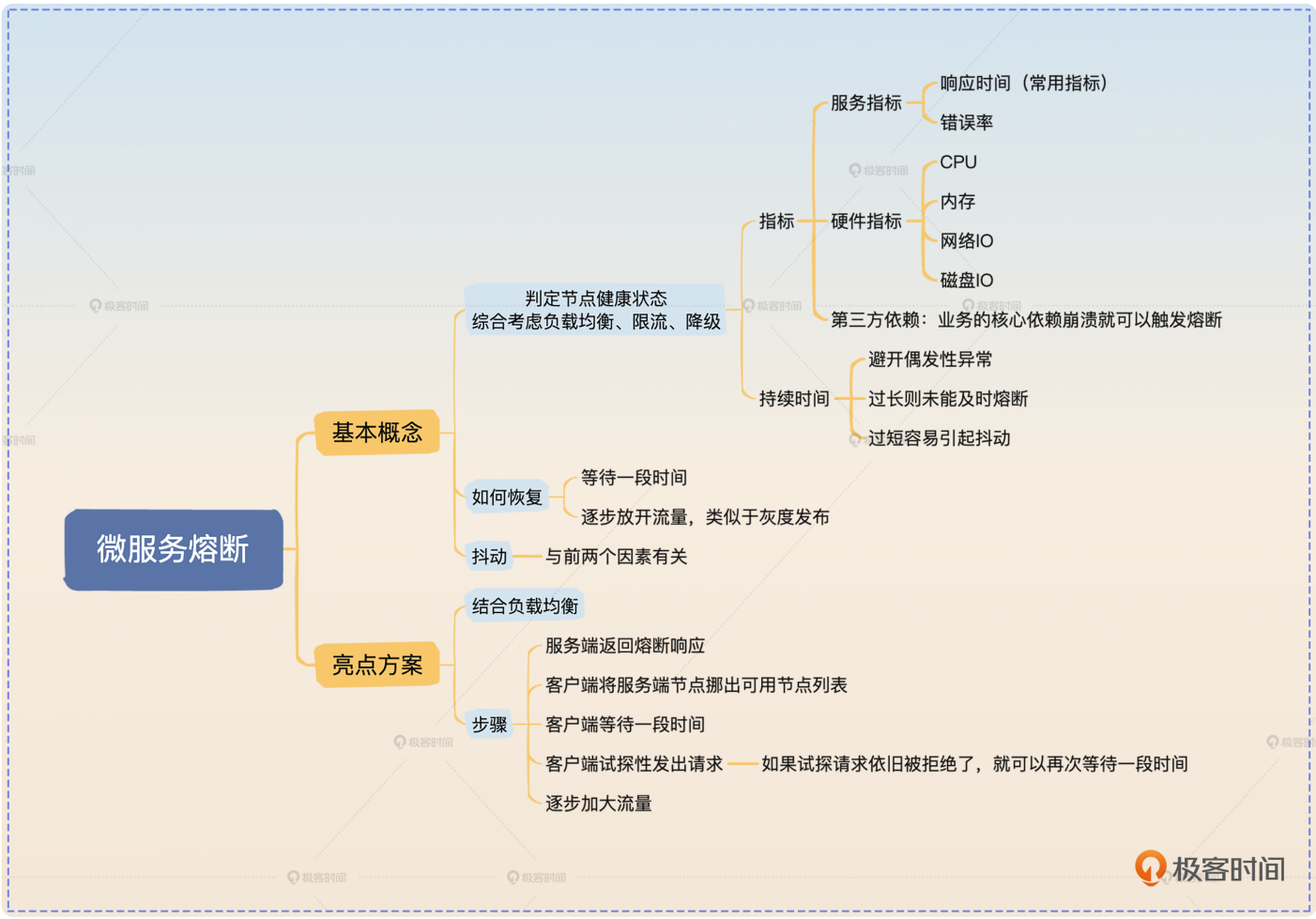
这节课我们主要解决的是熔断问题。我们讨论了熔断的基本概念，怎么判定服务是否熔断，以及熔断后如何恢复的问题。其中的难点是抖动的问题，为了防止抖动，我们需要合理判定节点的健康状况，在恢复期间尽可能等待一段时间，然后逐步放开流量。

最后我给出了一个**综合运用负载均衡和熔断的方案，重点在于客户端控制流量，并根据服务端节点的状况来操作可用节点列表**。你在学习的时候注意把亮点方案和前面学习的负载均衡内容

结合在一起，同时我也非常建议你在实际工作中尝试应用一下熔断，让它来保护你的系统，提高系统可用性。



同样地，我也整理这节课的思维导图，你可以参考。




除了熔断相关知识，这节课我还希望你学会综合运用各种技术手段来设计精巧的方案。在这里我给出了一个负载均衡 + 熔断的方案，后续你也可以看到更多这个技巧的应用。

最后我再强调一下，熔断面试的最好方案是把它作为你**构建高可用微服务**的一环，也就是说，你可以认为前面讨论的负载均衡，还有接下来要讨论的限流、降级、隔离等措施，都是你整个高可用方案的一环。

## 思考题

我在负载均衡和熔断两节课里面都提到一个关键点，即如何判定一个服务是否处于健康状态，这方面你有什么自己的思考？欢迎你把你的答案分享在评论区，也欢迎你把这节课的内容分享给需要的朋友，我们下节课再见！



## 精选留言 (14)

**小手冰凉*^O^***

2023-06-21 来自北京

老师好，业界好的熔断方案都是怎么做的呢？

> 作者回复: 现在的主流是根据错误率来判断要不要熔断，这个错误可以是业务错误，也可以是系统错误（比如中间件本身的错误）。当错误率比较高的时候就会触发熔断。
> 我之前就用过这种策略，比较好实现。

共 2 条评论 >　　3

**penbox**

2023-07-02 来自四川

在负载均衡和熔断两节课都提到了一个关键点，如何判断一个服务是否处于健康状态，这方面你有什么自己的思考？
学习了目前这两节课，我感觉服务端节点目前是什么状态，都是靠猜测或试探来得到的。
在负载均衡算法里面，靠连接数或响应时间来判断服务器权重，这两个指标都是客户端对服务端状态的猜测，并不是服务端的真实情况。动态调整权重里面的成加败减，要反复试探多次，才能把权重调整为正确的服务端状态。
在熔断这里，熔断之后到底恢复没有，不管是服务端和客户端都不知道，只能放点流量过去试一试。
目前好像还缺少手段来判断服务端当前的真实情况。有没有可能类似注册中心一样，在微服务框架中引入一个"健康中心"。每个服务端根据若干个指标，对自己打一个分，定时上报给健康中心。不管是客户端还是服务端自己，发现不对劲的时候，去健康中心里找到服务端对应的分数，就能直接判断出这个服务端的真实情况。

> 作者回复: 赞！基本上就是靠猜和试探，试探也是固定套路，你已经总结得很对了。
> 你考虑得非常深远了。
> 这种健康中心的想法应该说之前也有人探讨过，后来就是发现监控做得好，不太需要这么一个健康

中心。比如说我们的监控都会采集各个机器的 CPU，内存之类的东西。
所以现在其实是缺了一个能够综合运用各种监控指标打分的东西，但是一直以来也没有谁研究出来了通用的打分机制——适合各种业务的打分机制。
早期我还想过，能不能借助 AI 来学习人判断节点是否健康的思路，然后让 AI 来监控、打分、执行容错。还可以进一步整合日志分析、流量预判等，做得非常高级。
当然，这都是吹牛逼，我稍稍尝试过就放弃了。



2

**xyu**
2023-06-29 来自浙江

为什么说客户端利用负载均衡来控制流量相比服务端逐步放开流量是亮点方案?
我说一下我的理解，不知道对不对：
从整个系统来看，使用客户端控制流量的方案使得整个系统的可用性更好；客户端通过调用返回情况来调整权重，交互的过程实际上也是一个考察对方恢复情况的过程，然后根据你的恢复情况来加大流量，从而避免服务器又再次陷入熔断；如果使用服务端逐步放开流量的方案，机械地按比例的放开很可能会让服务器再次陷入熔断。

作者回复: 对的！按比例是偷懒的做法，你讲的这是基于反馈的流量恢复，是更加高级的方案。

共 2 条评论 >

2

**3.0的A7**
2023-06-21 来自北京

这里的客户端是指的是网关还是真实发起请求的客户端呢?
我理解是网关，比如nginx之类
如果是真实发起请求的客户端，比如移动app、web页面,那这个的开发成本是不是有点大呢?

共 2 条评论 >

2

**Bin**
2023-06-28 来自广东

What：什么是「熔断」
在微服务架构里面是指当微服务本身出现问题的时候，它会拒绝新的请求，直到微服务恢复。

Why：微服务架构为什么要使用「熔断」
可以给服务端恢复的机会
个人理解：
「熔断」机制就是在系统高负载/ 服务崩溃的时候，拒绝新的请求。相当于提供给服务/系统

一段"喘息"时间，"喘息"时间的意义在于：
1. 服务端不再接受新请求，服务的负载可以降低下来，防止服务雪崩
2. 服务端崩溃的情况下，可以重启服恢复务，可以理解为是一种故障转移（failover）手段
3. 上游服务收到「熔断」信息返回，可以进行兜底方案处理：例如业务补偿、请求转发、同步转异步等



When：什么时候进行和结束「熔断」
微服务本身「出现问题」的时候
「熔断」的开始和结束时机判断就在于以下问题的判断：
1. 判断微服务出现问题
2. 判断微服务恢复服务

「熔断」开启：判断出现问题
判断健康状态
1. 选择业务「指标」
每个微服务实际上都是附带着业务属性（微服务的划分是根据业务的职责和边界来划分）
所以可以根据业务来选择「指标」来代表微服务的健康状态。
常见的指标可以是：「响应时间的99线」，「错误率」等
2. 设置合适的「阈值」
这也是根据业务来决定的。
假设我们选择「响应时间」作为指标。
- 根据业务要求的「指标」阈值设置
- 观察业务整体的「指标」来设置

3. 设置持续多长时间才触发「熔断」
需要持续一段时间之后才触发「熔断」，因为：
- 防止「指标」 偶发性的突然增长
- 防止「服务抖动」

「持续时间」的设置，依赖个人经验

「熔断」结束：判断恢复服务
现状：大多数是触发「熔断」后保持「一段时间」，就认为服务已经恢复正常，继续处理新请求
存在的问题：「抖动」
解决方案：
1.「服务端控制流量」：恢复服务之后，比如说10%，20%逐步递增，不是100%恢复流量
这种做法不够好。依旧有请求因为熔断不会被处理。
2.「客户端控制流量」：服务端熔断后，客户端不再请求这个节点，而是换一个节点。恢复之后，客户端逐步放开流量

亮点方案：客户端结合「负载均衡」控制流量，解决「抖动」问题



Question

面试中，经常在你讲完一些场景解决方案之后，面试官就会问你有自己实现吗？或者有没有落地在自己的项目中？如果回答有，面试官就会问遇到过什么问题没有？你回答没有，感觉自己给面试官的感觉又只是停留在理论上，没有实践经验。想问老师，对于类似这些问题，应该怎么去破局呢？

作者回复: 我愿称你为课代表，总结太强了！

Question：其实我在面试的也会有这种困境，毕竟有些业务我确实没有解决过，只是听说过。

这里要分情况讨论：
1. 面试官问的场景是你确实要解决的，不过你用了 A 方案，但是你面试的时候还回答了 B，C 方案。这种，我一般就是先讲自己的 A 方案，讲完之后看情况，要是面试官觉得 A 不符合口味，就接着讲自己了解的 B 方案和 C 方案。正常来说，在做方案选型的时候，我多半比较过 B 和 C，但是最终选了 A。我只需要讲完 B C 之后讲一下自己当初的决策理由就可以。因为本身我落地的是 A，B 和 C 被我放弃了，问题也不太大。
2. 如果是这个场景跟我业务没有关系，比如说我的项目经历是支付方向，问我的场景是订单方向，我也就只能把公司的解决方案拿出来说说。问到一些答不出来的细节很正常，毕竟我项目经历又不是这个地方的。不过一般要避免面试官不按套路出牌，也就是最好别让自己陷入这种境地。

你在面试前，可以提前准备一下各种方案，包括如果真落地可能遇到什么问题，又或者你去搜搜业界落地的技术博客，也能回答上来一些细节性的问题。

2

**TimJuly**
2023-06-28 来自北京

引申一个问题，发现故障时（可能是自身引起，也可能是下游引起），怎么判断是单机故障还是集群故障

作者回复: 严格来说，如果你的异常机制（Go 中的错误机制）设计得好，你是知道的。比如说，你们有严格的错误码机制，每一个业务都有自己的错误码前缀，那么你根据错误码就知道究竟是谁出错了，也就是你能区别是自身、还是下游。
而如果没有这种机制，你基本上判断不出来是自身，还是下游引起的。这时候对你来说已经不重要

了，你直接将自身熔断了就可以。
而单机故障还是集群故障，你站在调用者角度永远都是认为下游是单机故障。比如说 A 调用 B，如果 A 调不通，它只会认为是它调用的 B 的那个节点崩了，而不会认为 B 整个集群都崩了。



集群故障是需要第三方来监控的。比如说你有 prometheus 监控，你发现所有的节点都各种超时，或者异常很多，这样你才能判定是整个集群都出问题了。

如果你利用的是负载均衡 fail-over 机制，或者你的客户端会给服务端发送心跳，那么服务端所有节点都调不通，也可以认为是集群崩溃了。

1

**Geek_948740**
2023-06-27 来自浙江

同问 这篇文章里的客户端是指什么 app和电脑的话做这些策略成本很高吧

作者回复: 这里的客户端基本上都是指服务调用中的客户端。比如说 A 调用 B 的接口，那么 A 就是客户端。

整个专栏，基本上都是这种语义，如果是APP 之类的，我一般都会直接说 APP，或者用户。

1

**码小呆**
2023-06-24 来自广东

我们是夜莺,上面写一个监控脚本,定时去请求对应服务的接口,一旦有服务请求不通过,就在群里实现告警,不过,如果公司的项目都上了prometheus ,一旦流量请求出现问题,就直接在企微群发送告警,这样子应该也是可以的

作者回复: 赞！确实可以的。不过面试的时候你还可以讲讲你们公司有没有什么自动处理告警的机制。因为有些故障是可以通过程序来恢复的。
不过你们的真的是服务一不通就告警吗？还是说会有考虑一分钟多少次调不通才告警？

共 2 条评论>

1

**hurrier**
2023-06-21 来自广东

大明老师 如果请求下流qps很低 比如几分钟才请求一次 按错误率、响应时间熔断感觉都不准确，有更好的方案吗

作者回复: 第一种思路是按错误类型。这个可以是跟业务相关的，也可以是跟业务没关的。比如说当你发现返回了某一个特定的错误（或者异常），就认为系统本身出了故障，那么就可以触发熔断。第二种思路是检测业务关键路径上的依赖。比如说你严重依赖于某一个中间件，例如 Redis，那么当你发现 Redis 连不上，比如说心跳已经 ping 不通了，那么就可以触发熔断（后面学了降级，你就应该考虑应该尽可能先降级）。

QPS 很低的话，你加一个限流会更好，防止突发流量。



共 2 条评论> 1

**peter**
2023-06-21 来自北京

请教老师两个问题：
Q1：服务恢复框架做得不好，是因为这个问题本身难以解决吗？
文中"很可惜，这方面微服务框架都做得比较差。大多数情况下就是触发熔断之后保持一段时间，比如说一分钟"。 为什么框架做得差？是投入不够？还是说此问题本身难以解决？或者说难以解决是微服务架构本身固有的问题？
Q2：熔断以后的恢复，业界一般是怎么做的？人工干预吗？

作者回复: 1. 首先应该是投入不够。这个东西本身是比较容易解决的，但是大部分微服务框架都没怎么做。微服务框架研发者都倾向于提供一些简单的、通用的策略，所以类似这种确实不怎么做。
2. 不是人工干预，最常见的做法就是熔断一个固定的时间，然后直接恢复。做得精致一点就是熔断之后，过一段时间，要试探性的放一些流量。如果这些流量被正常处理了，那么加大流量。

总的来说，如果你能为 gRPC 提供根据专栏内容涉及的一些扩展实现，放过去你的 Github 之类的，在面试的时候是可以介绍一下的，这样能够进一步证明你的代码能力。

1

**小晨**
2023-07-04 来自江苏

是否存在这种场景，通过客户端控制流量，熔断某个服务，将原本应该给这个服务的流量给其他节点，导致其他节点因为流量增加也导致也触发熔断，即原本可能只要熔断一个节点，但因为这个节点熔断了，流量分给了其他节点，其他节点的压力增加，进而触发熔断，造成"雪崩"?

作者回复: 会！你已经领悟到了面试另外一个可以提及的亮点。正常来说，你后端的节点处理能力是有冗余的，那么一两个节点熔断，这些冗余的处理能力刚好够用。



但是如果你要是大部分节点都已经快要撑不住流量了，那么你这时候一两台机器熔断，就会导致其它节点也撑不住。

所以这种流量的场景，用限流会更好。


**拾掇拾掇**

2023-07-04 来自浙江

我好奇这个试探性放点流量是怎么放？count计数下几个然后就不放，等待放进去流量的响应情况吗

作者回复: 差不多。不嫌麻烦的话，是可以做得很仔细的，最开始就像你说的，固定放几个请求试探。如果成功就步入下一个状态，比如说 0-100 随机一个数，小于 10 就放过去做试探，也就是 10% 的比例。后面如果收到了正常响应，就加大流量，不然就缩小流量或者退回上一个状态。

说实在的，我也没搞过这么复杂的实现。我就是搞过两步走的，第一阶段放几个请求，后面就是按照比例放。


**MClink**

2023-06-25 来自广东

如何判定一个服务是否处于健康状态。以GO语言实现的微服务举例。随便讲几个指标
1、服务依赖的数据库健康情况
1.1 数据库的CPU是否处于非高负载情况
1.2 数据库响应时长是否处于正常响应水平
1.3 数据库连接是否可以正常使用，无断连情况，sql 或命令语句是否可以正常执行

2. 服务所处环境，pod （或机器）的健康情况
2.1 CPU是否正常，使用率没有太高
2.2 磁盘IO是否正常
2.3 整体负载是否正常
2.4 内存使用率是否正常

3. 服务本身是否健康
3.1 gorutine 的数量是否正常
3.2 服务的内存占用是否正常
3.3 服务网络情况是否正常
3.3 服务是否存在Panic

4.服务有没有出现大量报错日志（框架的或者业务上的）
5.服务的接口APM P99 指标是否正常，响应时长没有突然暴涨

作者回复: 赞！所以当你掌握了这些之后，你就可以随便设计熔断、限流和降级的算法了。可以尝试自己搞一些有新意的出来，然后在面试中用用。

共 3 条评论>



。
2023-06-21 来自北京

写的很好，可以多更点吗😀

作者回复: 这个得我们小编安排，哈哈哈，我尽量写快点