流控降级最佳实践

> 作者：宿何（@sczyh30）
> 创作日期：2019-08-03
> 专栏地址：【稳定大于一切】

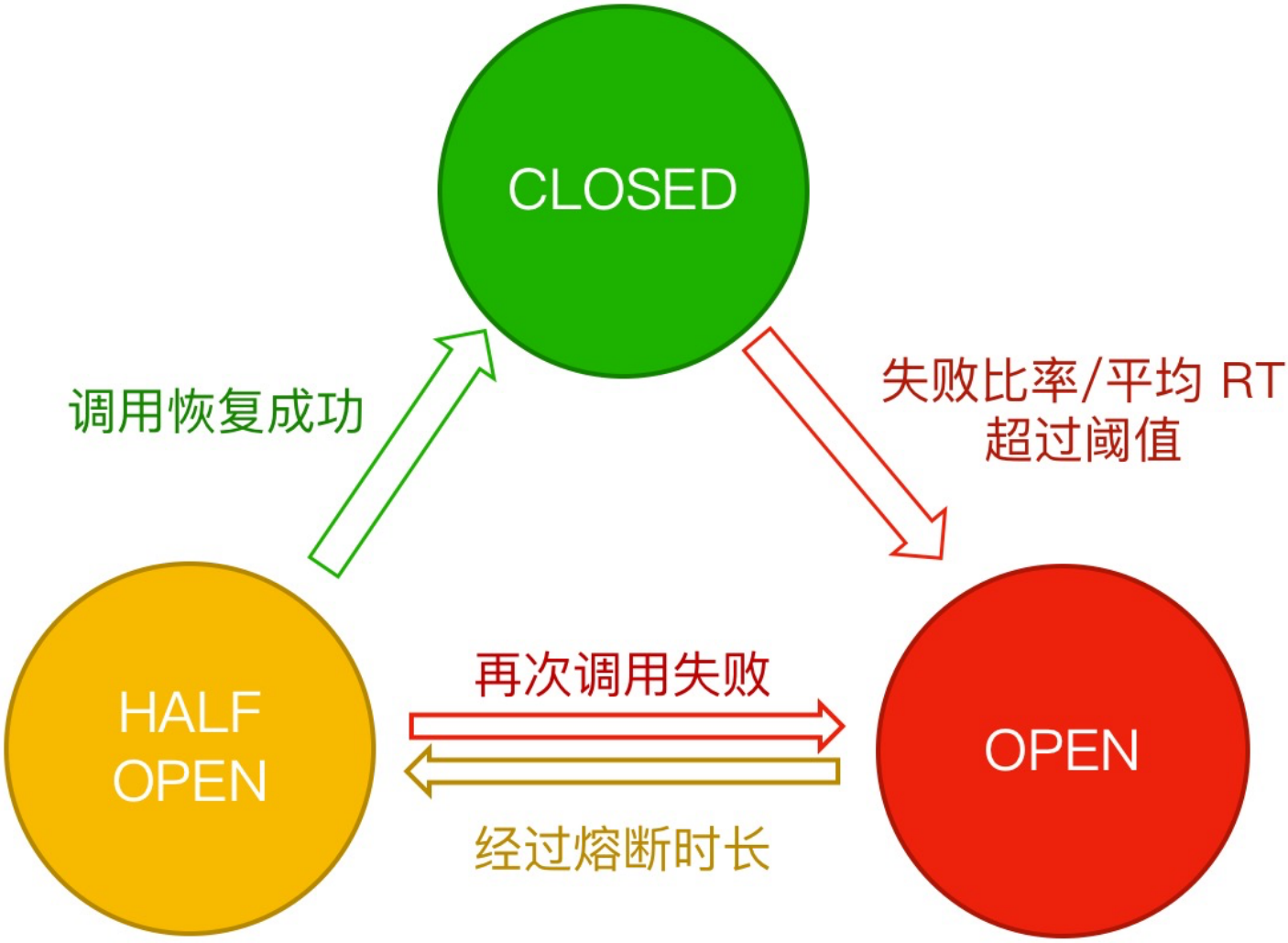
流量控制和熔断降级是保障微服务稳定性重要的一环。本文将会帮助读者理解流控降级的重要性，并介绍流控降级的各种最佳实践场景。如有遗漏或错误，欢迎补充指正。

目录

什么是流控降级

流控，即流量控制。流量控制在网络传输中是一个常用的概念，它用于调整网络包的发送数据。然而，从系统稳定性角度考虑，在处理请求的速度上，也有非常多的讲究。任意时间到来的请求往往是随机不可控的，而系统的处理能力是有限的。我们需要根据系统的处理能力对流量进行控制，在保证系统吞吐量比较高的同时又不会把系统打垮。

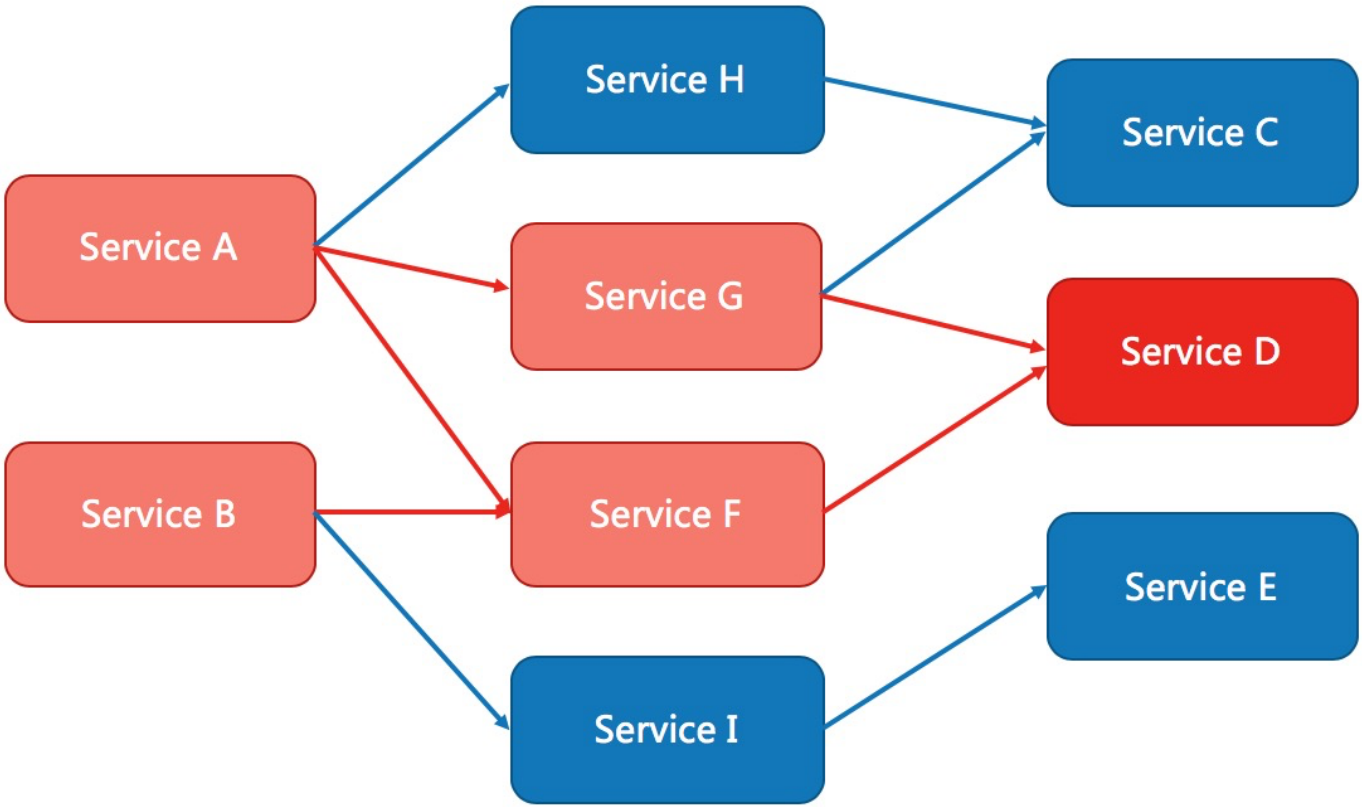
熔断降级则是在服务出现不稳定因素的时候暂时切断服务的调用，等待一段时间再进行尝试。一方面防止给不稳定服务“雪上加霜”，另一方面保护服务的调用方不被拖垮。这其实就是熔断器（Circuit Breaker）的思想：



为什么需要流控降级

流量是非常随机性的、不可预测的。前一秒可能还风平浪静，后一秒可能就出现流量洪峰了（例如双十一零点的场景）。然而我们系统的容量总是有限的，如果突然而来的流量超过了系统的承受能力，就可能会导致请求处理不过来，堆积的请求处理缓慢，CPU/Load 飙高，最后导致系统崩溃。因此，我们需要针对这种突发的流量来进行限制，在尽可能处理请求的同时来保障服务不被打垮。

一个服务常常会调用别的模块，可能是另外的一个远程服务、数据库，或者第三方 API 等。例如，支付的时候，可能需要远程调用银联提供的 API；查询某个商品的价格，可能需要进行数据库查询。然而，这个被依赖服务的稳定性是不能保证的。如果依赖的服务出现了不稳定的情况，请求的响应时间变长，那么调用服务的方法的响应时间也会变长，线程会产生堆积，最终可能耗尽业务自身的线程池，服务本身也变得不可用。



现代微服务架构都是分布式的，由非常多的服务组成。不同服务之间相互调用，组成复杂的调用链路。以上的问题在链路调用中会产生放大的效果。复杂链路上的某一环不稳定，就可能会层层级联，最终导致整个链路都不可用。因此我们需要对不稳定的服务进行熔断降级，暂时切断不稳定调用，避免局部不稳定因素导致整体的雪崩。

那么是不是服务的量级很小就不用进行限流防护了呢？是不是微服务的架构比较简单就不用引入熔断保护机制了呢？其实，这与请求的量级、架构的复杂程度无关。很多时候，可能正是一个非常边缘的服务出现故障而导致整体业务受影响，造成巨大损失。我们需要具有面向失败设计的意识，在平时就做好容量规划和强弱依赖的梳理，合理地配置流控降级规则，做好事前防护，而不是在线上出现问题以后再进行补救。

流控降级最佳实践

下面我们会介绍几个常见场景下的流控降级最佳实践，帮助大家合理地利用流控降级来保障服务的稳定性。

服务提供方流控

在服务提供方（Service Provider）的场景下，我们需要保护服务提供方自身不被流量洪峰打垮。我们通常根据服务提供方的服务能力进行流量控制，或针对特定的服务调用方进行限制。

为了保护服务提供方不被激增的流量拖垮影响稳定性，我们可以配置 QPS 模式的限流，当每秒的请求量超过设定的阈值时，会自动拒绝多余的请求。

根据调用方的需求来分配服务提供方的处理能力也是常见的限流方式。例如，有两个服务消费者 A 和 B 都向服务提供方发起调用请求，我们希望优先处理消费者 A 的请求，而只对来自服务 B 的请求进行限流。

服务调用方隔离/熔断

除了根据服务能力进行流量控制以外，还有一种常见的场景，即对依赖方进行隔离和降级。前面提到过，分布式系统中难免会有不稳定的服务节点（请求慢或请求处理失败）。如果调用端不引入自我保护的机制，那么有可能会被依赖的不稳定服务拖垮，导致自身也不可用。分布式系统中服务间的依赖关系是非常复杂的，一个服务不可用可能会导致上游数个服务不可用，最终会导致雪崩。

对这种情况，我们要在服务调用端（Service Consumer）对依赖的不稳定服务进行隔离和熔断。

服务隔离一般有两种方式：

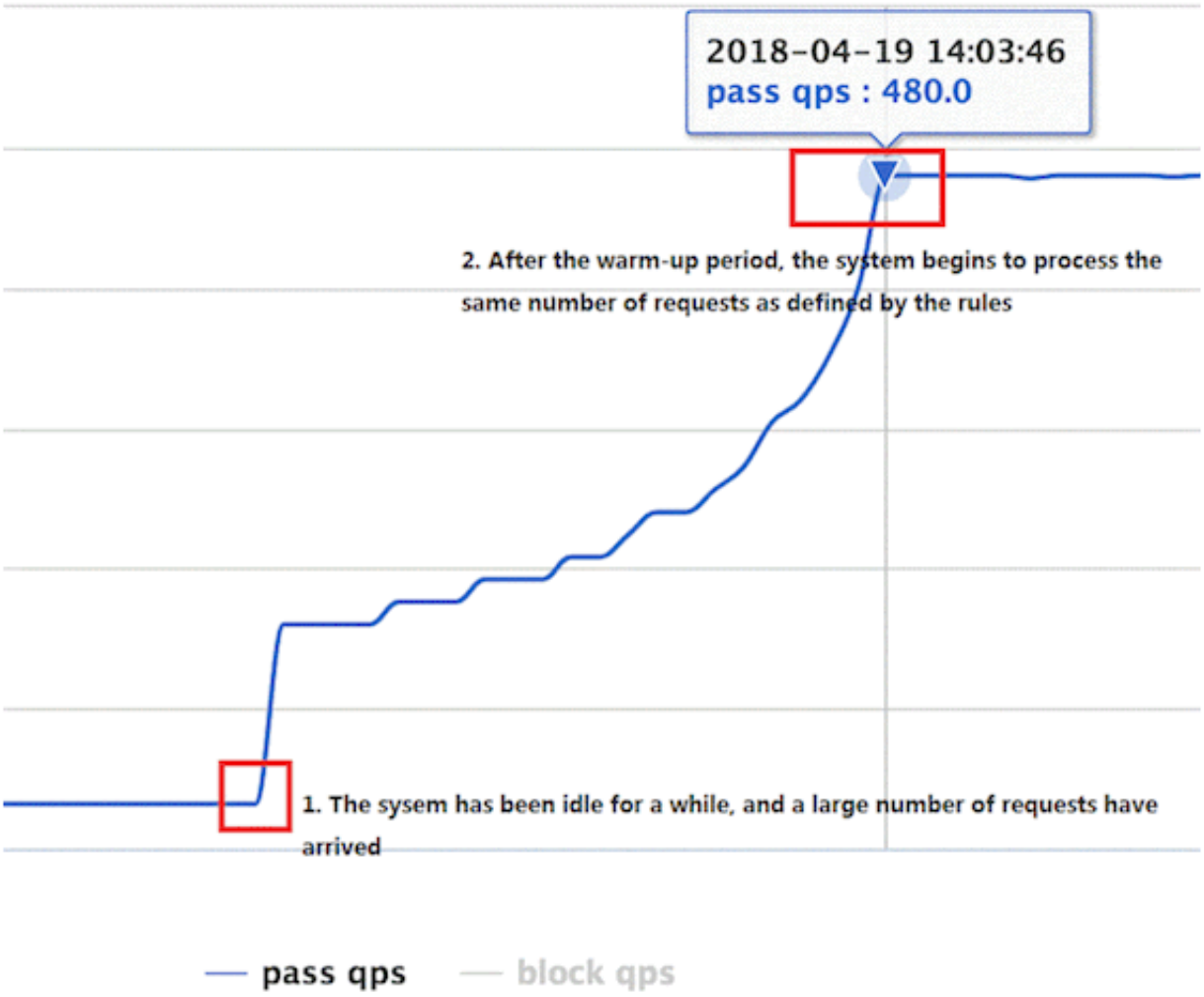
- **线程池隔离**：即针对不同的业务调用分别创建不同的线程池，不同服务调用都发生在不同的线程池中，在线程池排队、超时等阻塞情况时可以快速“掐断”。线程池隔离的好处是隔离度比较高，可以针对某个服务调用的线程池去进行处理而不影响其它资源，但是代价就是可能会创建比较多的线程池，线程上下文切换的 overhead 比较大，而且线程池执行会导致一些基于 ThreadLocal 的场景出现问题（如 Spring 事务）。Hystrix 主推线程池隔离模式。
- **信号量隔离**：即限制某个服务调用的并发量，而不是为不同服务显式创建线程池。这样的隔离比较轻量，overhead 比较小，但是效果不错，并且可以结合实时的响应时间对慢调用进行熔断，从而保障自身不被不稳定服务调用所拖垮。Sentinel 主推信号量隔离模式。

当服务依赖于多个下游服务，而某个下游服务调用非常慢或经常出错时，会严重影响当前服务的调用。我们可以借助熔断器的思想，当异常比率或业务调用的平均时长超过某个阈值后将调用进行熔断，直到一段时间过后再尝试恢复。熔断期间我们可以提供默认的处理逻辑（fallback），熔断期间的调用都会返回 fallback 的结果，而不会再去尝试本已非常不稳定的服务。

需要注意的是，即使服务调用方引入了熔断降级机制，我们还是需要在 HTTP 或 RPC 客户端配置请求超时时间，来做一个兜底的防护。

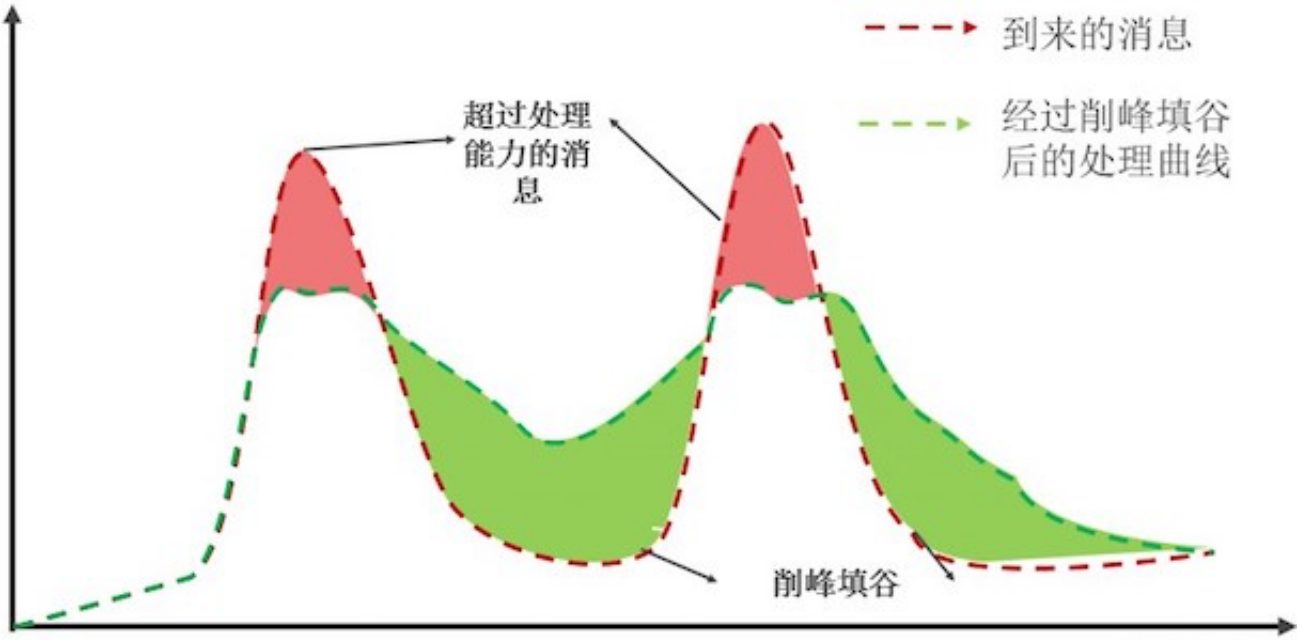
冷系统预热

当系统长期处于低水位的情况下，流量突然增加时，直接把系统拉升到高水位可能瞬间把系统压垮。我们可以利用 Guava 的 SmoothRateLimiter 或 Sentinel 的 Warm Up 流控模式，控制通过的流量缓慢增加，在一定时间内逐渐增加到阈值上限，而不是在一瞬间全部放行。这样可以给冷系统一个预热的时间，避免冷系统被压垮。



削峰填谷

请求的到来往往是没有规律的。很多时候流量可能都集中于某几秒的时间，而在接下来的一段时间内都没有很多请求。例如，某应用的处理能力是每秒 10 个请求。在某一秒，突然到来了 30 个请求，而接下来两秒，都没有请求到达。在这种情况下，如果直接拒绝 20 个请求，应用在接下来的两秒就会空闲。所以，需要把请求突刺均摊到一段时间内，让系统负载保持在请求处理水位之内，同时尽可能地处理更多请求，从而起到“削峰填谷”的效果。常见的场景为消息队列消费端处理消息的场景：



上图中，红色的部分代表超出消息处理能力的部分。观察得出，消息突刺往往都是瞬时的、不规律的，其后一段时间系统往往都会有空闲资源。把红色的那部分消息平摊到后面空闲时去处理，这样既可以保证系统负载处在一个稳定的水位，又可以尽可能地处理更多消息。我们可以利用 Leaky Bucket 算法 结合请求排队来让消息匀速消费，允许一部分消息排队等待处理，超出最大等待时长的消息直接拒绝处理。

Sentinel 提供了匀速排队模式，可以很好地适用这种场景。

网关流控

网关作为流量的入口，会接收到大量的用户请求。我们可以在网关层面，从流量入口处拦截激增的流量，防止下游服务被冲垮。网关流控常见的场景：

- 限制某个 API 的调用频率，比如对外提供的 OpenAPI 每小时总调用量不超过 30000 次。
- 针对每个请求的客户端 IP、Header 或者 URL 参数进行流控，比如限制每个 IP 的调用频次来进行防刷。
- 结合集群限流来限制某个下游服务的总调用量，这样多余的流量就直接在网关层被拒绝了，而不会再打到应用层。

对于使用了 Nginx 的场景，我们可以借助 Nginx 自带的 ngx*http*limit*req*module 模块来对经过 Nginx 的请求进行限制；对于常用的 Java API Gateway，如 Spring Cloud Gateway 和 Zuul，我们可以利用 Sentinel 的网关流量控制特性，配置网关流控规则来针对每个不同路由、不同的请求属性（如 HTTP Header、URL 参数等）进行流量控制。

流控降级规则的配置

需要注意的是，限流降级的配置是需要结合容量规划、依赖梳理来做的。我们可以借助 JMeter 或 阿里云 PTS 等压测工具对我们的服务进行全链路压测，了解每个服务的最大承受能力，来确定流控和熔断降级的阈值。同时，业务系统需要具备实时监控的能力，以便实时地根据流量情况做出相应的限流降级策略调整。

推荐工具&产品

- Sentinel - 阿里巴巴开源的，面向分布式服务架构的流控降级中间件
- AHAS —— 阿里云应用高可用服务，提供企业级的流量控制和监控能力
- resilience4j
- Hystrix（已停止维护）

相关文章&交流群

- Java 常用限流降级组件对比
- Nginx Rate Limiting
- Sentinel 开源交流群（钉钉群）：21977771

加入我们

【稳定大于一切】打造国内稳定性领域知识库，**让无法解决的问题少一点点，让世界的确定性多一点点。**

- GitHub 地址
- 钉钉群号：23179349