谈谈服务雪崩、降级与熔断

孤独烟 Java后端技术 2月14日

作者：孤独烟

微信公众号：孤独烟（ID:zrj_guduyan）

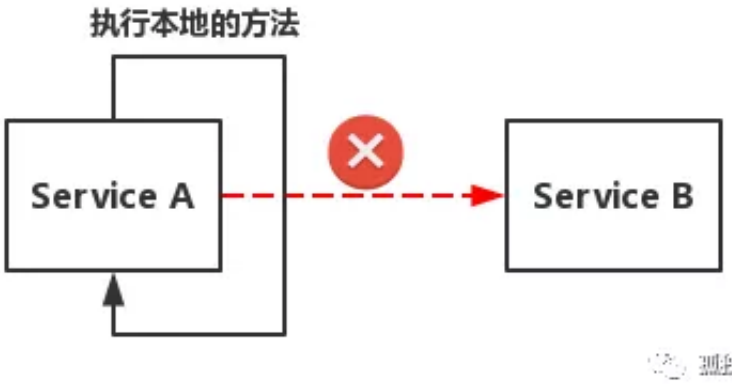
引言

首先，之所以谈这个话题呢，是发现现在很多人对微服务的设计缺乏认识，所以写一篇扫盲文。当然，考虑到目前大多微服务的文章都是口水文，烟哥争取将实现方式讲透，点清楚，让大家有所收获！

OK，我要先说明一下，我有很长一段时间将**服务降级**和**服务熔断**混在一起，认为是一回事！

为什么我会有这样的误解呢？

针对下面的情形，如图所示



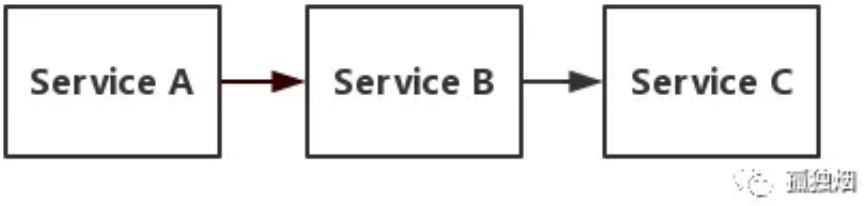
当 Service A 调用 Service B ，失败多次达到一定阀值， Service A 不会再去调 Service B ，而会去执行本地的降级方法！
对于这么一套机制:在Spring cloud中结合Hystrix,将其称为熔断降级!

所以我当时就以为是一回事了，毕竟熔断和降级是一起发生的，而且这二者的概念太相近了！后面接触了多了，发现自己理解的还是太狭隘了，因此本文中带着点我自己的见解，大家如果有不同意见，请轻喷！毕竟还有很多人认为两者是一致的！

正文

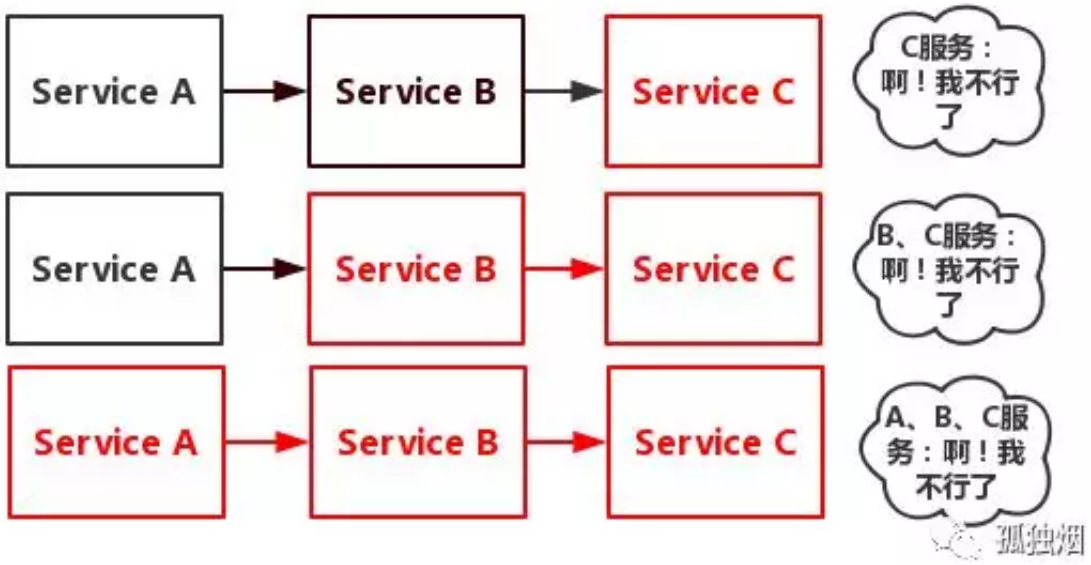
服务雪崩

OK，我们从服务雪崩开始讲起！假设存在如下调用链



而此时， Service A 的流量波动很大，流量经常会突然性增加！那么在这种情况下，就算 Service A 能扛得住请求， Service B 和 Service C 未必能扛得住这突发的请求。
此时，如果 Service C 因为抗不住请求，变得不可用。那么 Service B 的请求也会

阻塞，慢慢耗尽 Service B 的线程资源， Service B 就会变得不可用。紧接着， Service A 也会不可用，这一过程如下图所示



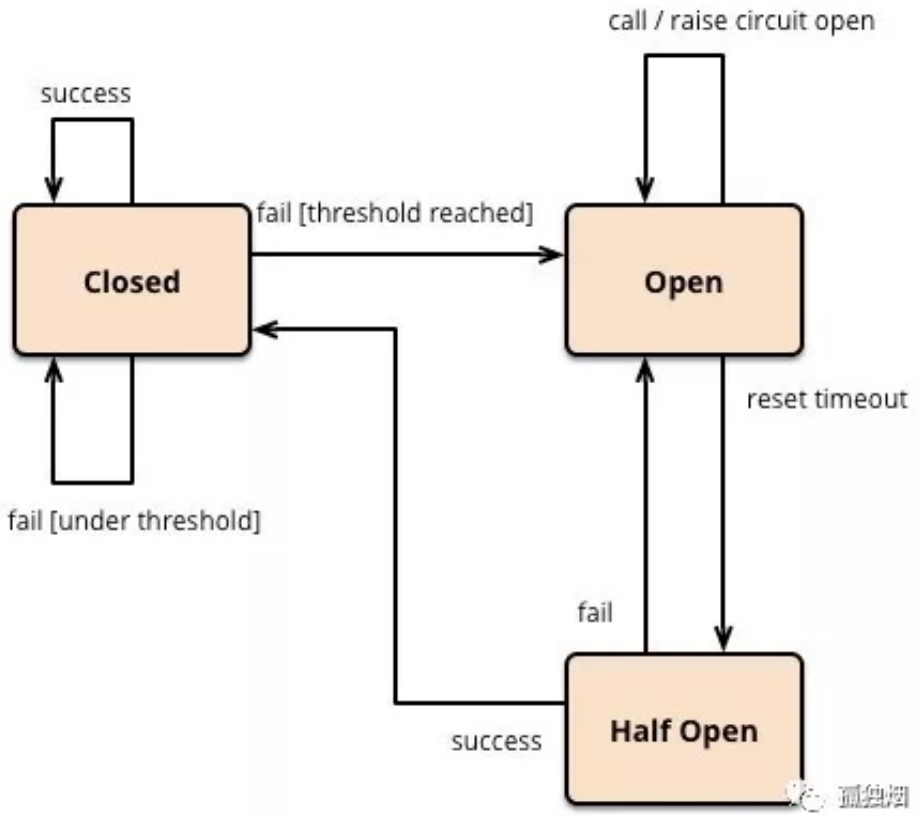
如上图所示，一个服务失败，导致整条链路的服务都失败的情形，我们称之为服务雪崩。

ps: 谁发明的这个词，真是面试装13必备！

那么，**服务熔断**和**服务降级**就可以视为解决服务雪崩的手段之一。

服务熔断

那么，什么是服务熔断呢？
服务熔断：当下游的服务因为某种原因突然变得不可用或响应过慢，上游服务为了保证自己整体服务的可用性，不再继续调用目标服务，直接返回，快速释放资源。如果目标服务情况好转则恢复调用。
需要说明的是熔断其实是一个框架级的处理，那么这套熔断机制的设计，基本上业内用的是 断路器模式 ，如 Martin Fowler 提供的状态转换图如下所示



- 最开始处于 closed 状态，一旦检测到错误到达一定阈值，便转为 open 状态；
- 这时候会有个 reset timeout ，到了这个时间了，会转移到 half open 状态；
- 尝试放行一部分请求到后端，一旦检测成功便回归到 closed 状态，即恢复服务；

业内目前流行的熔断器很多，例如阿里出的Sentinel,以及最多人使用的Hystrix
在Hystrix中，对应配置如下

```
//滑动窗口的大小，默认为20
circuitBreaker.requestVolumeThreshold
//过多长时间，熔断器再次检测是否开启，默认为5000，即5s钟
circuitBreaker.sleepWindowInMilliseconds
//错误率，默认50%
circuitBreaker.errorThresholdPercentage
```

每当20个请求中，有50%失败时，熔断器就会打开，此时再调用此服务，将会直接返回失败，不再调远程服务。直到5s钟之后，重新检测该触发条件，判断是否把熔断器关闭，或者继续打开。

这些属于框架层级的实现，我们只要实现对应接口就好！

服务降级

那么，什么是服务降级呢？
这里有两种场景:

- 当下游的服务因为某种原因**响应过慢**，下游服务主动停掉一些不太重要的业务，释放出服务器资源，增加响应速度！
- 当下游的服务因为某种原因**不可用**，上游主动调用本地的一些降级逻辑，避免卡顿，迅速返回给用户！

其实乍看之下，很多人还是不懂熔断和降级的区别!

其实应该要这么理解:

- 服务降级有很多种降级方式！如开关降级、限流降级、熔断降级!
- 服务熔断属于降级方式的一种！

可能有的人不服，觉得熔断是熔断、降级是降级，分明是两回事啊！其实不然，因为从实现上来说，熔断和降级必定是一起出现。因为当发生**下游服务不可用**的情况，这个时候为了对最终用户负责，就需要**进入上游的降级逻辑**了。因此，将熔断降级视为降级方式的一种，也是可以说的通的！

我撇开框架，以最简单的代码来说明！上游代码如下

```
try{
    //调用下游的helloWorld服务
    xxRpc.helloWorld();
}catch(Exception e){
    //因为熔断，所以调不通
    doSomething();
}
```

注意看，下游的helloWorld服务因为熔断而调不通。此时上游服务就会进入catch里头的代码块，那么catch里头执行的逻辑，你就可以理解为降级逻辑!
什么，你跟我说你不捕捉异常，直接丢页面？
OK，那我甘拜下风，当我理解错误!

服务降级大多是属于一种业务级别的处理。当然，我这里要讲的是另一种降级方式，也就是开关降级!这也是我们生产上常用的另一种降级方式！

做法很简单，做个开关，然后将开关放配置中心！在配置中心更改开关，决定哪些服务进行降级。至于配置变动后，应用怎么监控到配置发生了变动，这就不是本文该讨论的范围。
那么，在应用程序中部下开关的这个过程，业内也有一个名词，称为**埋点**！

那接下来最关键的一个问题，哪些业务需要埋点？
一般有以下方法
(1)简化执行流程
自己梳理出核心业务流程和非核心业务流程。然后在非核心业务流程上加上开关，一旦发现系统扛不住，关掉开关，结束这些次要流程。

(2)关闭次要功能
一个微服务下肯定有很多功能，那自己区分出主要功能和次要功能。然后次要功能加上开关，需要降级的时候，把次要功能关了吧！

(3)降低一致性
假设，你在业务上发现执行流程没法简化了，愁啊！也没啥次要功能可以关了，桑心啊！那只能降低一致性了，即将核心业务流程的同步改异步，将强一致性改最终一致性！

可是这些都是手动降级，有办法自动降级么？
这里我摸着良心说，我们在生产上没弄自动降级！因为一般需要降级的场景，都是可以预见的，例如某某活动。假设，平时真的有突发事件，流量异常，也有监控系统发邮件通知，提醒我们去降级！
当然，这并不代表自动降级不能做，因此以下内容可以认为我在胡说八道，因为我在生产上没实践过，只是头脑大概想了下，如果让我来做自动降级我会怎么实现：

- (1)自己设一个阈值，例如几秒内失败多少次，就启动降级
- (2)自己做接口监控(有兴趣的可以了解一下Rxjava)，达到阈值就走推送逻辑。怎么推呢？比如你配置是放在Git上，就用Jgit去改配置中心的配置。如果配置放数据库，就用Jdbc去改。
- (3)改完配置中心的配置后，应用就可以自动检测到配置的变化，进行降级！(这句不了解的，了解一下配置中心的热刷新功能)