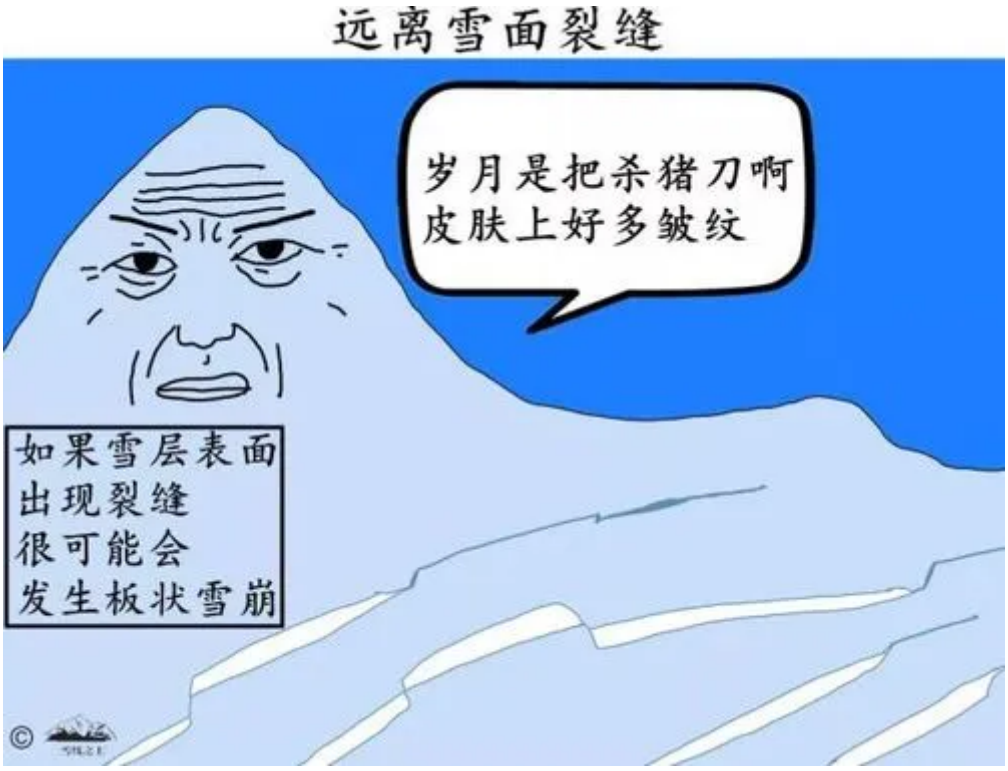
【第55话：微服务中灾难性雪崩效应是什么？如何防止】

Hello 小伙伴们，这解决给大家讲解下什么是微服务中的雪崩效应，以及如何防止。

首先我们来讲一下什么是灾难性雪崩效应？

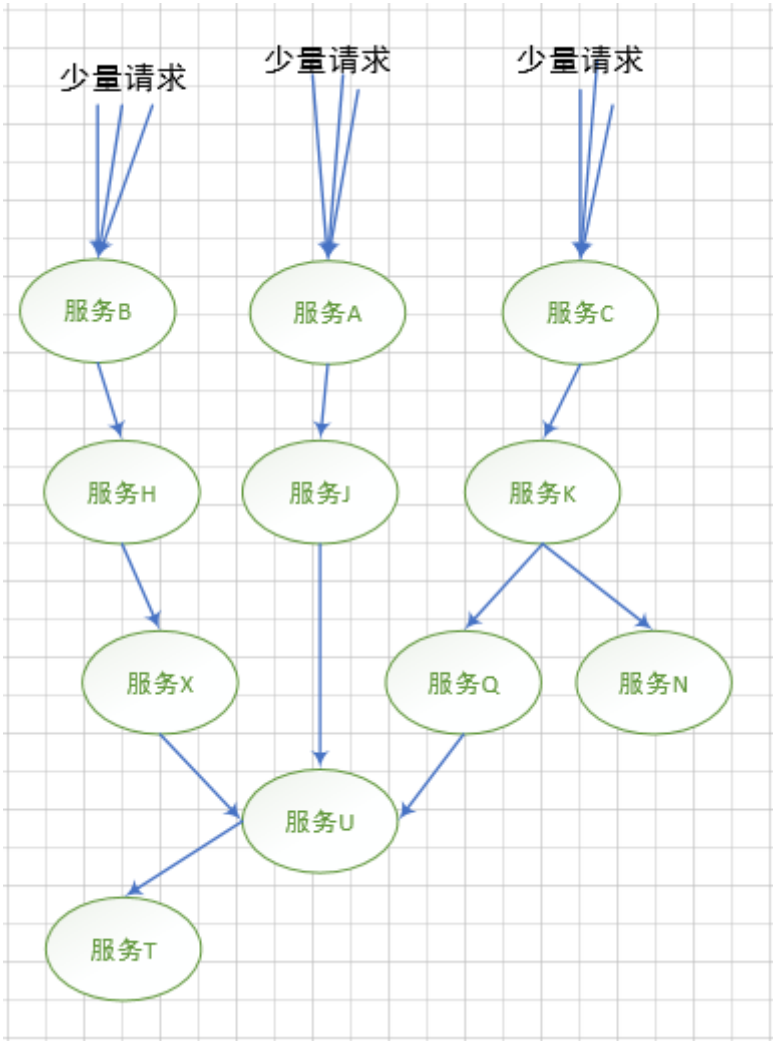
先来看看自然界的雪崩。俗话说：当雪崩发生时，没有一片雪花是无辜的！！！



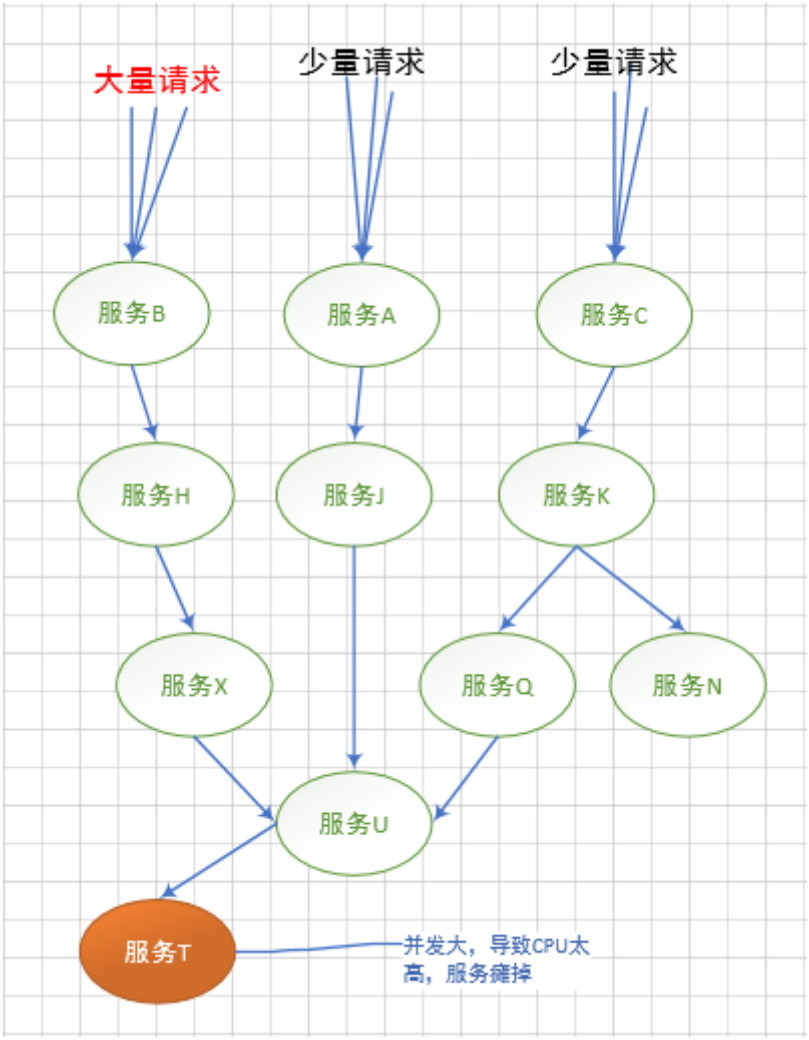
那么微服务架构种的灾难性雪崩效应是如何发生的？

在微服务架构的项目中，尤其是中大型项目，肯定会出现一个服务调用其他的服务，其他服务又调用别的服务，服务和服务之间形成了一种链式的调用关系。

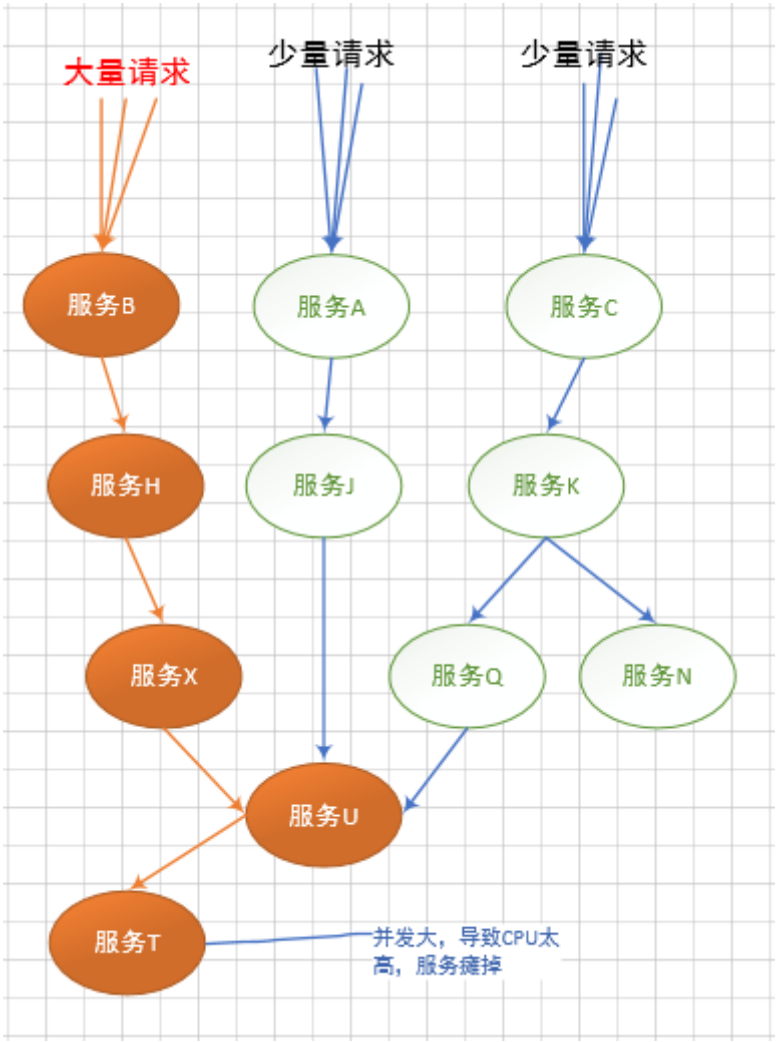
当少量请求时，对于整个服务链条是没有过多的影响的。



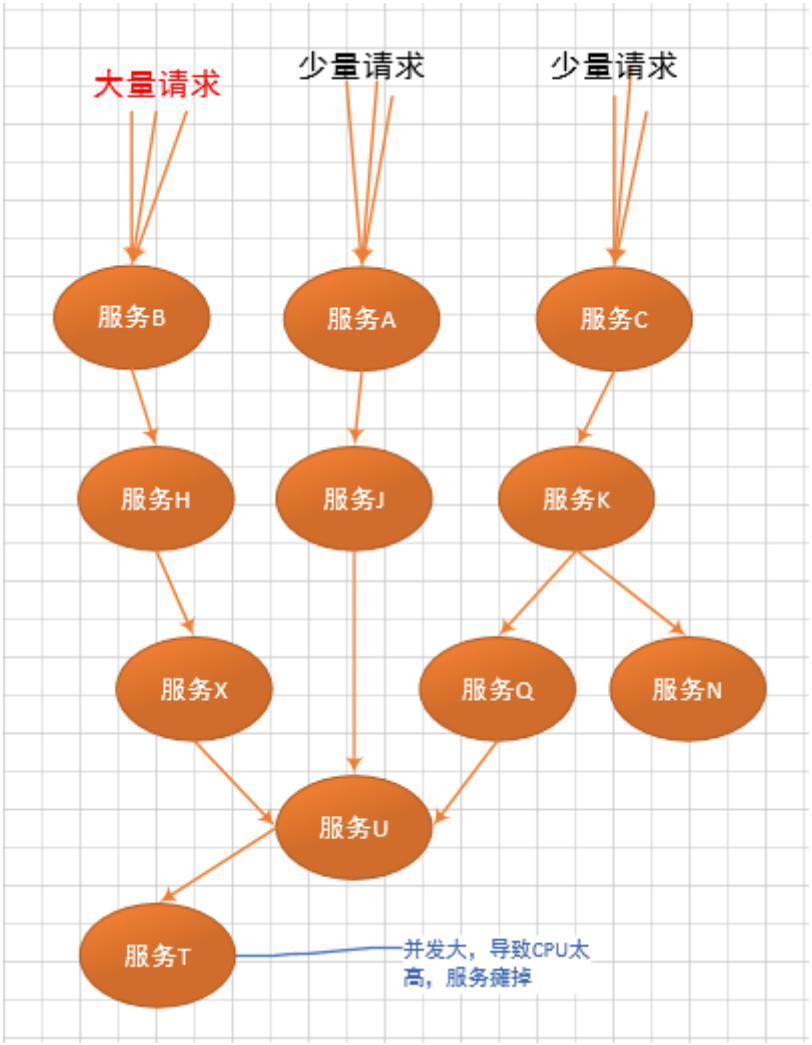
虽然每个服务的请求都是少量的，但是最终都访问服务T。所以对于服务T来说请求量就是比较大的。所在的服务器CPU压力比较高。



当其中某一个服务突然遇到大量请求时。整个链条上所有服务负载骤增。



导致服务U和服务T的负载过高。运行性能下降。会导致其他调用服务U和服务T的链条出现问题。从而所有的项目可能都出现的问题。这种情况就称为灾难性的雪崩效应。



总结起来灾难性雪崩效应发生的原因可以简单归结为下述三种：

1. 服务提供者（Application Service）不可用。如：硬件故障、程序BUG、缓存击穿、并发请求量过大等。
2. 重试加大流量。如：用户重试、代码重试逻辑等。
3. 服务调用者（Application Client）不可用。如：同步请求阻塞造成的资源耗尽等。

雪崩效应最终的结果就是：服务链条中的某一个服务不可用，导致一系列的服务不可用，最终造成服务逻辑崩溃。这种问题造成的后果，往往是无法预料的。

那么，如何防止灾难性雪崩效应的发生呢，总结起来有下面几点：

1. 降级

超时降级、资源不足时(线程或信号量)降级，降级后可以配合降级接口返回托底数据。实现一个fallback方法, 当请求后端服务出现异常的时候, 可以使用fallback方法返回的值.

此方案保证的是：即使某服务出现了问题，整个项目还可以继续运行。

拿王者荣耀举例，服务发生异常：

你看到的（内部结果，受不了了啊！！）：

别人看到的（降级结果，人机比真人厉害，求求你别上线！！）：

2. 熔断

当失败率(如因网络故障/超时造成的失败率高)达到阀值自动触发降级，熔断器触发的快速失败会进行快速恢复。

通俗理解：熔断就是具有特定条件的降级，当出现熔断时在设定的时间内容就不在请求Application Service了。所以在代码上熔断和降级都是一个注解

此方案保证的是：即使某服务出现了问题，整个项目还可以继续运行，且一段时间内，出现问题的服务不会被访问。

服务发生异常(严重问题)：

你看到的(心里不爽)（内部结果）：

别人看到的(大快人心)（熔断后的结果）：

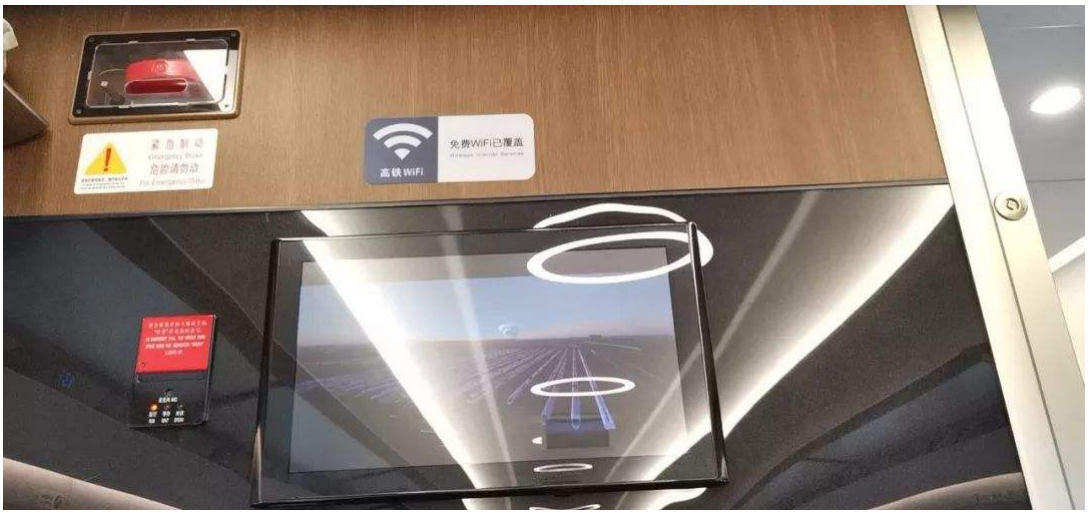
3. 请求缓存

提供了请求缓存。服务A调用服务B，如果在A中添加请求缓存，第一次请求后走缓存了，就不在访问服务B了，即使出现大量请求时，也不会对B产生高负载。

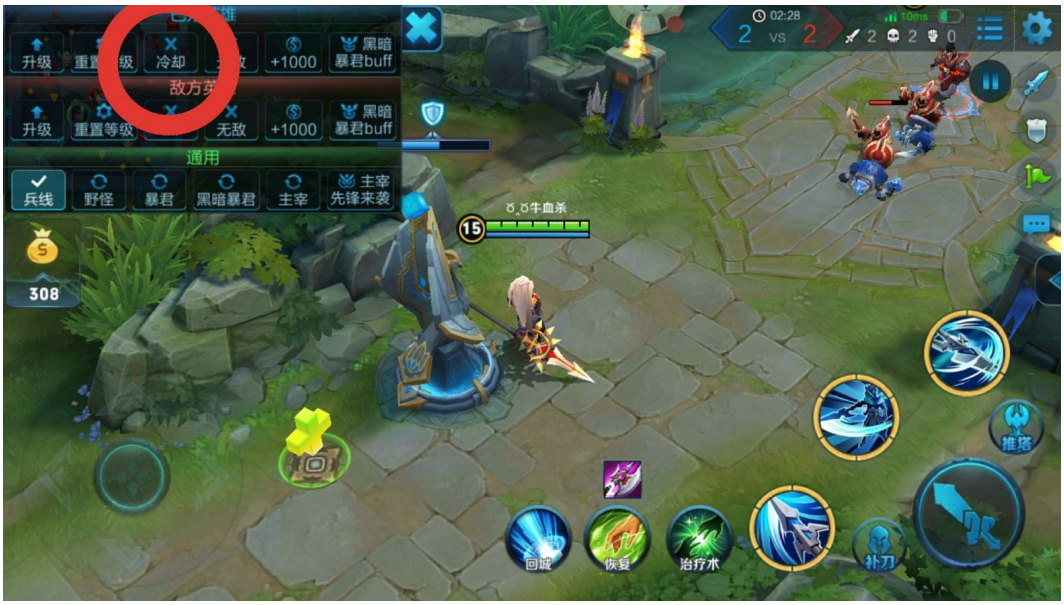
请求缓存可以使用Spring Cache实现。

此方案保证的是：减少对Application Service的调用。

问题发生（信号不好，减少与远程服务器的连接）：



你看到的（自以为的快乐，单机也很开心）：



别人看到的（自欺欺人的乐趣，别人以为你在排位）：

4. 请求合并

提供请求合并。当服务A调用服务B时，设定在5毫秒内所有请求合并到一起，对于服务B的负载就会大大减少，解决了对于服务B负载激增的问题。

此方案保证的是：减少对Application Service的调用。

我以为的（平稳流量）：

问题来了（实际流量）：

解决办法（还是大壶过瘾）：

5. 隔离

隔离分为线程池隔离和信号量隔离。通过判断线程池或信号量是否已满，超出容量的请求直接降级，从而达到限流的作用。

未隔离时，资源公用，所以混乱：

隔离后，资源独享，秩序井然：