# Spring Cloud断路器Hystrix原理读书笔记

## 服务雪崩

假设一个场景，我们去苹果店买新出的iPhone，店员提供一站式服务：试用、支付、交货。但是由于大家都知道的原因，交货一直很慢很慢，客户、店员不停地去交货窗口提交请求，最终店里所有的店员和客户都堆积到交货窗口，而外面的客户不停地涌入店里，却没人服务，整个店都崩溃了——服务雪崩。

应对方法有几个：

1. 限流，比如排队，把队伍搞的曲里拐弯的，想进店要绕好远的路。也或者店里人数达到阈值就拉个警戒线不再放人进去。想想每个工作日早高峰。北京几个重要地铁站的外面，就是这个场景。
2. 缓存，店里多备货，不用去库房拿货，太耽误时间。
3. 扩容，增加店面积，店员。
4. 熔断，交货窗口直接关闭，告知店员和客户，您也别来了，来了也没用，后续怎么处理，你们商量着办就行了（fallBack）。
5. 服务降级，像试用这种相对不重要的环节，就直接PASS掉，节约时间，集中资源办好后面重要的环节。

## Hystrix

终于说到本文主角了。

为了保护系统免于服务雪崩，Hystrix的设计原则有：

隔离、熔断、命令模式。

### 隔离

这个很好理解，不再多解释。Hystrix的隔离机制有两种：

1. 线程池：给每个服务分配一个线程池，内含10（默认）个线程，当线程池满了，第11个请求过来的时候，开始排队，队伍上限为5（默认），第16个请求过来的时候，直接走fallback流程，不再对服务进行请求。

已经执行请求的线程在调用结束后反馈调用情况，交由Metrics保存，根据Metrics里的计数，计算服务的健康状态，从而决定是否熔断。

好处是，当并发高起来的时候，就算线程池满了，无法执行服务调用，也可以把请求放入队列等待，不至于立即fallback。

1. 信号量：有一个原子计数器，也即信号量，记录了现在运行线程的数量。当新请求进来的时候，先判断信号量是否达到阈值10（默认），如果小于，则+1，继续执行服务调用，执行完毕后，-1。如果已经达到阈值，则直接fallback。

相比线程池，好处是没有线程切换的开销。坏处就是一旦达到阈值，就直接fallback，不像线程池还有个队列可以缓冲一下

### 熔断

有三个状态：

1. 关，执行请求，收集执行情况。
2. 开，拒绝请求。根据执行失败与执行总数的比值，也即服务健康状况，与设定的阈值的比较，决定断路器是否打开。默认是错误超50%，且10秒内超过20个请求。
3. 半开半关，经过一段时间后，允许某请求通过并执行，如成功，则关闭断路器，反之继续打开断路器。

### 命令模式

基于命令模式，Hystrix将要执行请求的服务，封装到HystrixCommand或HystrixObservableCommand中，这里除了run()来执行具体的服务调用外，还包括线程池，断路器，getFallback()失败回调方法。

执行的流程：

1. 创建Command对象，执行调用。
2. 检查断路器是否开启。根据Metrics决定断路器的开关。

2.1、是，执行fallback。

2.2、否，进入3。

1. 检查线程池是否已满，并上报结果到Metrics。

3.1、是，执行fallback。

3.2、否，进入4。

1. 执行run()。
2. 检查执行是否失败，并上报结果到Metrics。

5.1、是，执行fallback。

5.2、否，进入6。

1. 检查执行是否超时。并上报结果到Metrics。

6.1、是，执行fallback。

**6.2、否，至此执行成功，返回结果。**

1. 检查fallback是否成功。

7.1、是，返回fallback结果。

7.2、否，抛异常。