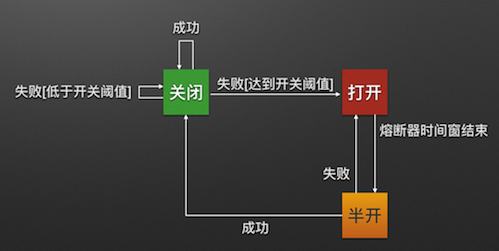
# 基本介绍

## 雪崩效应

在微服务架构中通常会有多个服务层调用，基础服务的故障可能会导致级联故障，进而造成整个系统不可用的情况，这种现象被称为服务雪崩效应。服务雪崩效应是一种因“服务提供者”的不可用导致“服务消费者”的不可用,并将不可用逐渐放大的过程。

## 熔断器（应用于服务调用端）



熔断器的原理很简单，如同电力过载保护器。**它可以实现快速失败，如果它在一段时间内侦测到许多类似的错误，会强迫其以后的多个调用快速失败，不再访问远程服务器，从而防止应用程序不断地尝试执行可能会失败的操作，**使得应用程序继续执行而不用等待修正错误，或者浪费CPU时间去等到长时间的超时产生。**熔断器也可以使应用程序能够诊断错误是否已经修正，如果已经修正，应用程序会再次尝试调用操作**。

## Hystrix特性

3.1 断路器机制

1. 断路器很好理解, 当Hystrix Command请求后端服务失败数量超过一定比例(默认50%), 断路器会切换到开路状态(Open)。这时所有请求会直接失败而不会发送到后端服务.
2. 断路器保持在开路状态一段时间后(默认5秒), 自动切换到半开路状态(HALF-OPEN). 这时会判断下一次请求的返回情况, 如果请求成功, 断路器切回闭路状态(CLOSED), 否则重新切换到开路状态(OPEN).
   1. Fallback

Fallback相当于是降级操作。对于查询操作, 我们可以实现一个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. fallback方法的返回值一般是设置的默认值或者来自缓存.**

* 1. 资源隔离

在Hystrix中, 主要通过线程池来实现资源隔离. 通常在使用的时候我们会根据调用的远程服务划分出多个线程池. 例如调用产品服务的Command放入A线程池, 调用账户服务的Command放入B线程池. 这样做的主要优点是运行环境被隔离开了. 这样就算调用服务的代码存在bug或者由于其他原因导致自己所在线程池被耗尽时, 不会对系统的其他服务造成影响。

# 二、熔断器的基本使用

## 1. 修改配置文件（服务调用方）

feign.hystrix.enabled=true

## 2. 创建回调类（当调用失败时）

//实现远程调用的方法

@Component

public class HelloRemoteHystrix implements HelloRemote{

@Override

public String hello(@RequestParam(value = "name") String name) {

return "hello" +name+", this messge send failed ";

}

}

## 3. 添加fallback属性

在HelloRemote类添加指定fallback类，在服务熔断的时候返回fallback类中的内容。

