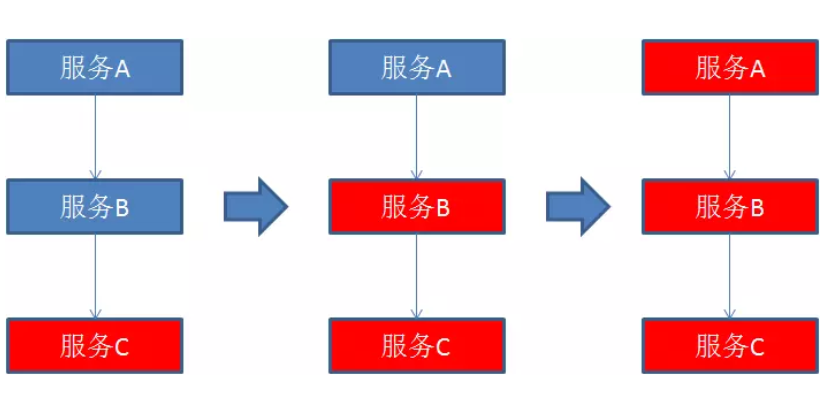
# 前言

在微服务架构中，根据业务来拆分成一个个的服务，服务与服务之间可以相互调用（RPC），在Spring Cloud**可以用Feign+Ribbon来进行服务之间的调用**。为了保证其高可用，单个服务通常会集群部署。由于网络原因或者自身的原因，服务并不能保证100%可用。如客户端访问A服务，而A服务需要调用B服务，B服务需要调用C服务，由于网络原因或者自身的原因，如果B服务或者C服务不能及时响应，A服务将处于阻塞状态，直到B服务C服务响应。此时若有大量的请求涌入，容器的线程资源会被消耗完毕，导致服务瘫痪。**服务与服务之间的依赖性，故障会传播，造成连锁反应，会对整个微服务系统造成灾难性的严重后果，这就是服务故障的“雪崩”效应。**

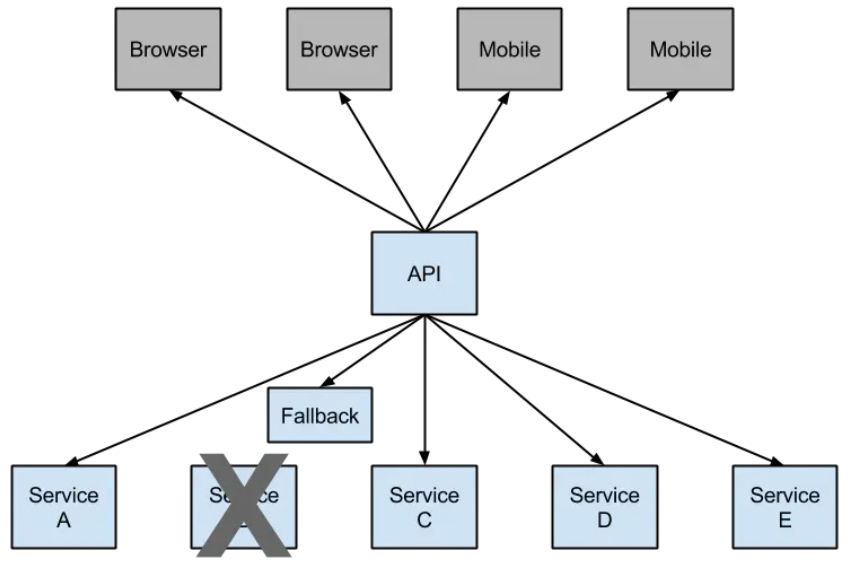


# 服务雪崩解决方案

* **熔断模式**：这种模式主要是参考电路熔断，如果一条线路电压过高，保险丝会熔断，防止火灾。放到我们的系统中，如果某个目标服务调用慢或者有大量超时，此时，熔断该服务的调用，对于后续调用请求，不在继续调用目标服务，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。
* **隔离模式**：这种模式就像对系统请求按类型划分成一个个小岛的一样，当某个小岛被火少光了，不会影响到其他的小岛。例如可以对不同类型的请求使用线程池来资源隔离，每种类型的请求互不影响，如果一种类型的请求线程资源耗尽，则对后续的该类型请求直接返回，不再调用后续资源。这种模式使用场景非常多，例如将一个服务拆开，对于重要的服务使用单独服务器来部署，再或者公司最近推广的多中心。
* **限流模式**：上述的熔断模式和隔离模式都属于出错后的容错处理机制，而限流模式则可以称为预防模式。限流模式主要是提前对各个类型的请求设置最高的QPS阈值，若高于设置的阈值则对该请求直接返回，不再调用后续资源。这种模式不能解决服务依赖的问题，只能解决系统整体资源分配问题，因为没有被限流的请求依然有可能造成雪崩效应。漏斗算法，令牌桶算法。

# 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。



# 服务降级

* 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！(**弃车保帅，壮士断腕，刮骨疗伤**)
* 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！

# 服务熔断和服务降级的区别和联系

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**！**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！

# Hystrix(黑丝锤科丝)是什么

Hystrix中文是豪猪的意思，豪猪浑身长满了长刺，可以保护自己。**Hystrix是一个用于处理分布式系统的延迟和容错的开源库，在Spring Cloud中防雪崩的利器就是Spring Cloud Hystrix**，Spring Cloud Hystrix是基于Netflix Hystrix实现的。在分布式系统里，许多依赖不可避免的会调用失败，比如超时、异常等，Hystrix能够保证在一个依赖出问题的情况下，不会导致整体服务失败，避免级联故障，以提高分布式系统的弹性。

# Hystrix能干啥

* **服务熔断**

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. 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).

* **服务降级**

Fallback相当于是降级操作, 我们可以实现一个fallback方法, 当请求后端服务出现异常或者服务熔断后的时候, 直接调用本地fallback方法，告知后面的请求服务不可用了，不要再来了**。**

* **服务容错**
* **线程和信号隔离**

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.比如说，一个服务调用两外两个服务，你如果调用两个服务都用一个线程池，那么如果一个服务卡在哪里，资源没被释放。后面的请求又来了，导致后面的请求都卡在哪里等待，导致你依赖的A服务把你卡在哪里，耗尽了资源，也导致了你另外一个B服务也不可用了。这时如果依赖隔离，某一个服务调用A B两个服务，如果这时我有100个线程可用，我给A服务分配50个，给B服务分配50个，这样就算A服务挂了，我的B服务依然可以用。

* **请求缓存**

比如一个请求过来请求我userId=1的数据，你后面的请求也过来请求同样的数据，这时我不会继续走原来的那条请求链路了，而是把第一次请求缓存过了，把第一次的请求结果返回给后面的请求。

* **请求合并**

我依赖于某一个服务，我要调用N次，我发了N条请求发然后拿到一堆结果，这时候我们可以把多个请求合并成一个请求，这样我们只需调用一次远程服务，提升了效率。

* **近乎实时的服务监控**

**。。。**

# Hystrix服务熔断降级初步配置

1. **在服务消费者中引入Hystrix的依赖**

|  |
| --- |
| <!-- https://mvnrepository.com/artifact/org.springframework.cloud/spring-cloud-starter-hystrix --> <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-hystrix</artifactId>  <version>1.4.7.RELEASE</version> </dependency> |

1. **在服务消费者启动类上加上如下注解**

@EnableCircuitBreaker @EnableHystrix

1. **在服务消费者需要被保护的API接口方法上加@HystrixCommand注解**
2. **测试**

# Hystrix默认超时时间

由于客户端请求服务端方法时，服务端方法响应超过1秒将会触发降级，所以我们可以配置Hystrix默认的超时配置。

|  |
| --- |
| hystrix:  command:  default:  execution:  isolation:  thread:  timeoutInMilliseconds: 3000 |

# Feign对于Hystrix的支持

1. **在服务消费者中引入Hystrix的依赖**
2. **在服务消费者配置文件中开启Feign对Hystrix的支持**

|  |
| --- |
| feign:  hystrix:  enabled: true |

1. **新建一个Feign Client接口的实现类，实现类编写接口中方法熔断触发后的降级逻辑**
2. **修改Feign Client接口的FeignClient注解添加对降级类的支持**
3. **测试**

# Hystrix监控平台

除了隔离依赖服务的调用以外，Hystrix还提供了准实时的调用监控（Hystrix Dashboard），Hystrix会持续地记录所有通过Hystrix发起的请求的执行信息，比如每秒的请求数和成功数等，这些监控数据对于分析系统请求的调用情况很有用。

前提是你的项目中已经包含了spring-boot-starter-actuator和spring-cloud-starter-hystrix，之后可以直接通过http://localhost:7908/actuator/hystrix.stream来访问，但是页面展示的并不直观。

**暴露所有的端点**

|  |
| --- |
| **management**:  **endpoints**:  **web**:  **exposure**:  **include**: "\*" |

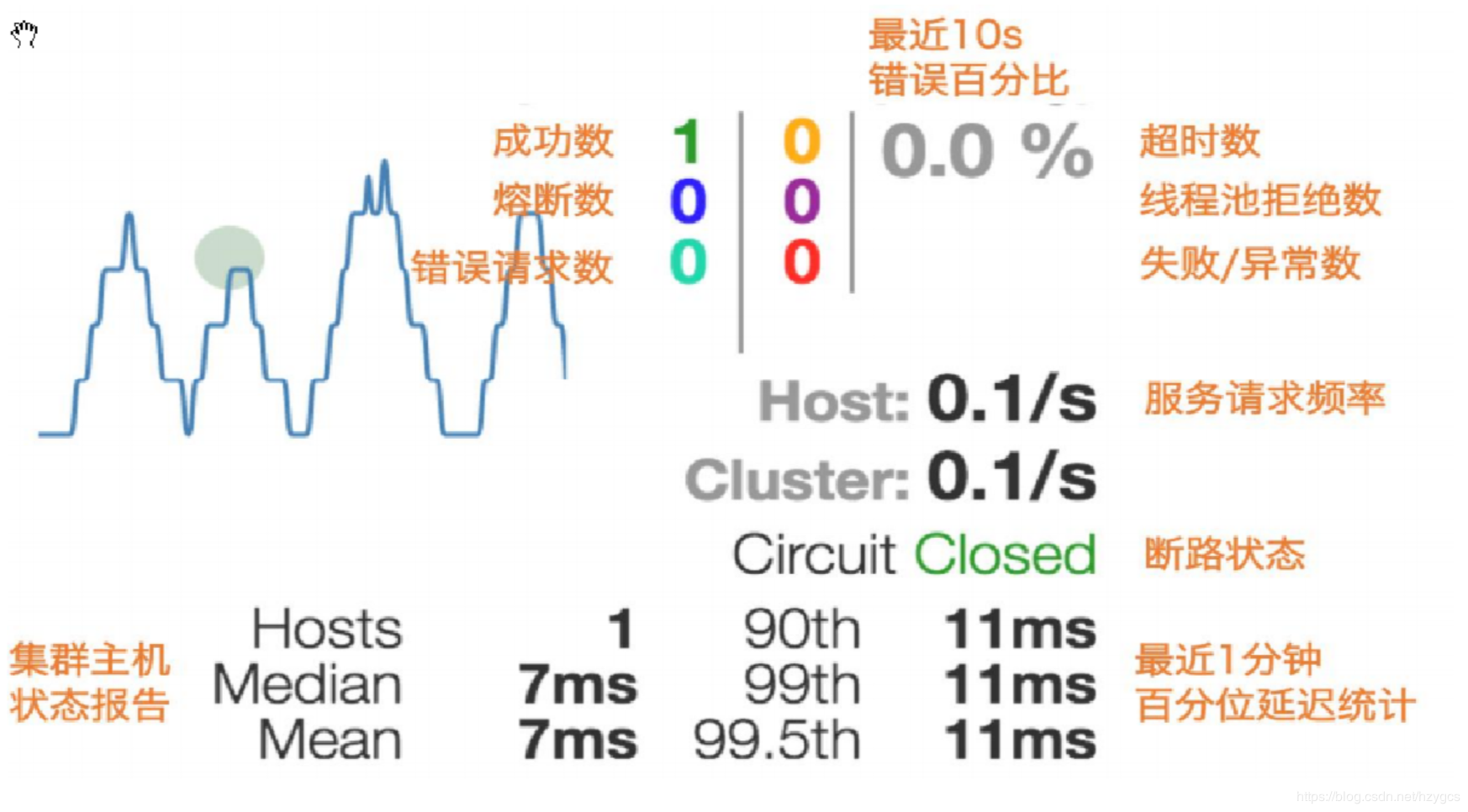
# Hystrix Dashboard 服务监控

刚刚讨论了Hystrix的监控，但访问/hystrix.stream接口获取的都是已文字形式展示的信息。很难通过文字直观的展示系统的运行状态，所以Spring Cloud也提供了Hystrix Dashboard的整合，对监控内容转化成可视化界面。还提供了基于图形化的DashBoard（仪表板）监控平台。Hystrix仪表板可以显示每个断路器（被@HystrixCommand注解的方法）的状态。

|  |
| --- |
| 1. **新建一个用于服务监控的项目** 2. **导入依赖** 3. **在启动类上添加@EnableHyStrixDashboard注解** 4. **测试** |

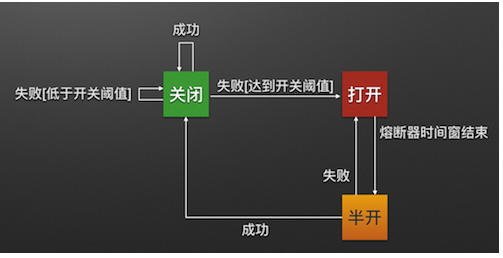
**下面来介绍下图形中各元素的具体含义：**

* 实心圆:共有两种含义。通过颜色的变化代表了实例的健康程度，它的健康度从绿色、黄色、橙色、红色递减。通过圆的大小来代表请求流量的大小，流量越大该实心圆就越大。所以通过该实心圆的展示，就可以在大量的实例中快速的发现故障实例和高压力实例。
* 曲线:记录2分钟内流量的相对变化，可以通过它来观察到流量的上升和下降趋势。
* 其他数量指标：



# Hystrix断路器的状态

熔断器有三个状态 CLOSED 、 OPEN 、 HALF\_OPEN 熔断器默认关闭状态，当触发熔断后状态变更为OPEN ,在等待到指定的时间，Hystrix会放请求检测服务是否开启，这期间熔断器会变为 HALF\_OPEN 半开启状态，熔断探测服务可用则继续变更为 CLOSED 关闭熔断器。



**Closed** ：关闭状态（断路器关闭），所有请求都正常访问。代理类维护了最近调用失败的次数，如果某次调用失败，则使失败次数加1。如果最近失败次数超过了在给定时间内允许失败的阈值，则代理类切换到断开(Open)状态。此时代理开启了一个超时时钟，当该时钟超过了该时间，则切换到半断开（Half-Open）状态。该超时时间的设定是给了系统一次机会来修正导致调用失败的错误。

**Open** ：打开状态（断路器打开），所有请求都会被降级。Hystix会对请求情况计数，当一定时间内失败请求百分比达到阈值，则触发熔断，断路器会完全关闭。默认失败比例的阈值是50%，请求次数最少不低于20次。

**Half Open** ：半开状态，open状态不是永久的，打开后会进入休眠时间（默认是5S）。随后断路器会自动进入半开状态。此时会释放1次请求通过，若这个请求是健康的，则会关闭断路器，否则继续保持打开，再次进行5秒休眠计时。

# 测试Hystrix断路器的状态

熔断器的默认触发阈值是20次请求，不好触发。休眠时间时5秒，时间太短，不易观察，为了测试方便，我们可以通过配置修改熔断策略：

|  |
| --- |
| hystrix:  command:  default:  execution:  isolation:  thread:  timeoutInMilliseconds: 6000 #默认的连接超时时间1秒,若1秒没有返回数据,自动的触发降级逻辑  circuitBreaker:  enabled: true  requestVolumeThreshold: 5  errorThresholdPercentage: 10  sleepWindowInMilliseconds: 10000 |

requestVolumeThreshold：触发熔断的最小请求次数，默认20

errorThresholdPercentage：触发熔断的失败请求最小占比，默认50%

sleepWindowInMilliseconds：熔断多少秒后去尝试请求

# Hystrix Turbine