# 关键词

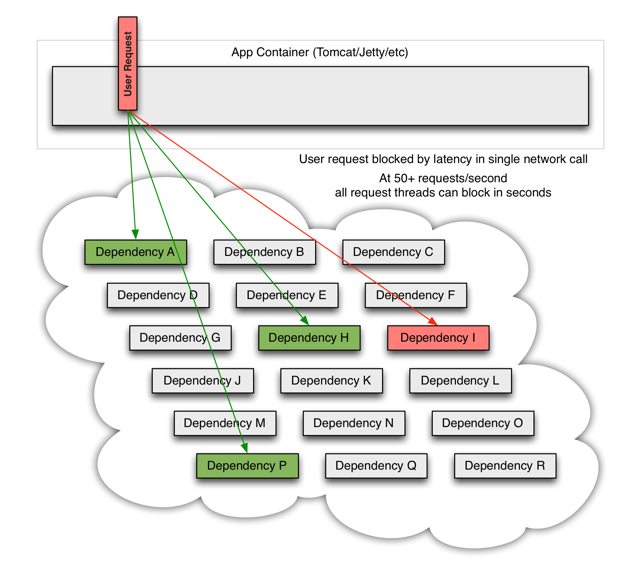
熔断 限流

性能杀手 响应时间

资源 隔离 线程资源

# 雪崩效应

服务雪崩效应是一种因“服务提供者的不可用”（原因）导致“服务调用者不可用”（结果），并将不可用逐渐放大的现象。



# 雪崩效应形成原因

我把服务雪崩的参与者简化为 **服务提供者** 和 **服务调用者**, 并将服务雪崩产生的过程分为以下三个阶段来分析形成的原因:

1. 服务提供者不可用
2. 重试加大流量
3. 服务调用者不可用

**服务不可用** :

* 硬件故障
* 程序Bug
* 缓存击穿
* 用户大量请求

**重试加大流量** :

* 用户重试
* 代码逻辑重试

**服务调用者不可用**:

* 同步等待造成的资源耗尽

# 雪崩的应对策略

针对造成服务雪崩的不同原因, 可以使用不同的应对策略:

1. 流量控制
2. 改进缓存模式
3. 服务自动扩容
4. 服务调用者降级服务

**流量控制** :

* 网关限流

Nginx + Lua的网关进行流量控制, OpenResty

**改进缓存模式** :

* 缓存预加载
* 同步改为异步刷新

**服务自动扩容** :

* AWS的auto scaling

**服务调用者降级服务**:

* 资源隔离
* 对依赖服务进行分类
* 不可用服务的调用快速失败

资源隔离

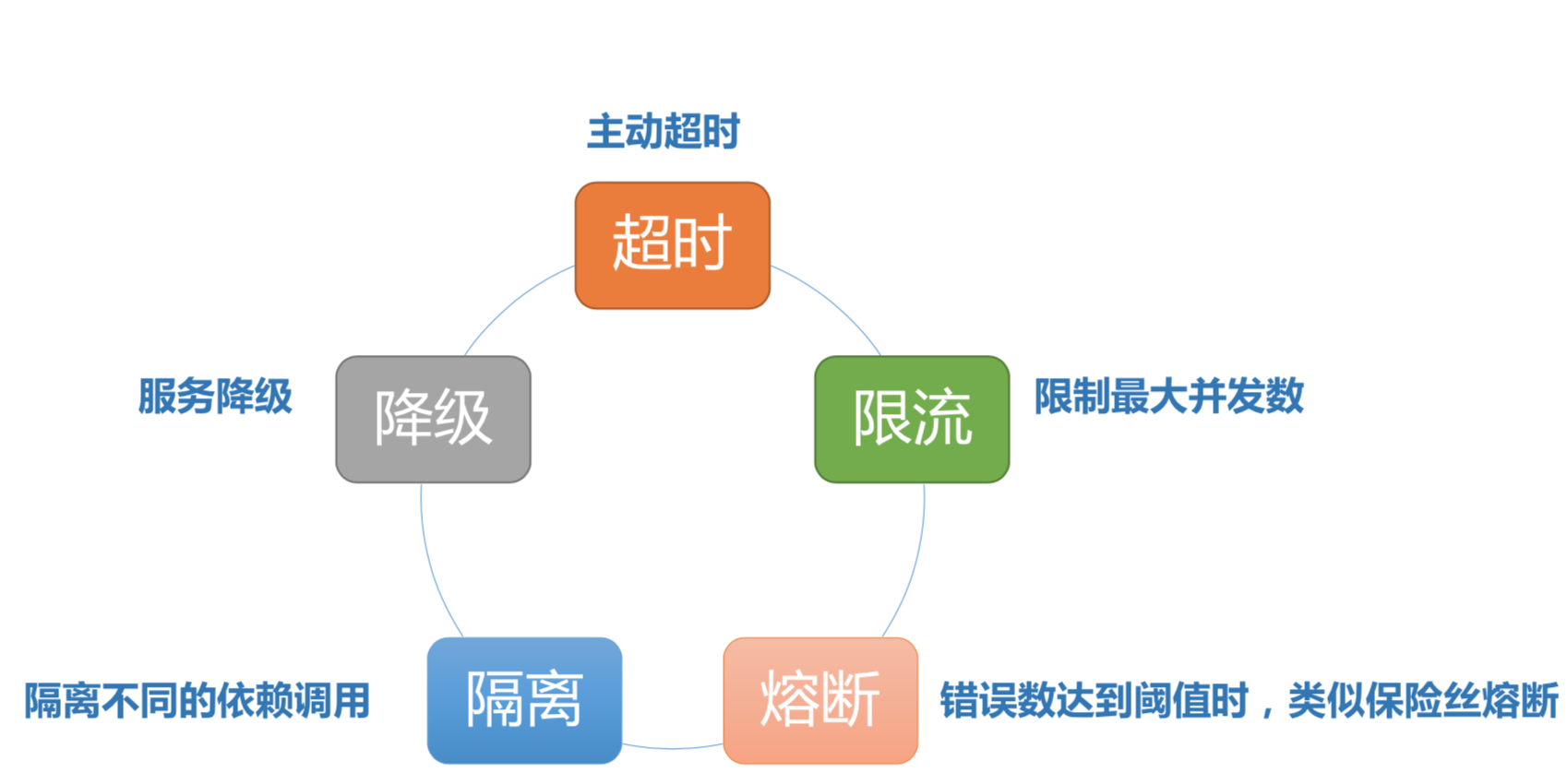
对调用服务的线程池进行隔离

依赖服务分为: 强依赖和弱依赖. 强依赖服务不可用会导致当前业务中止,而弱依赖服务的不可用不会导致当前业务的中止.

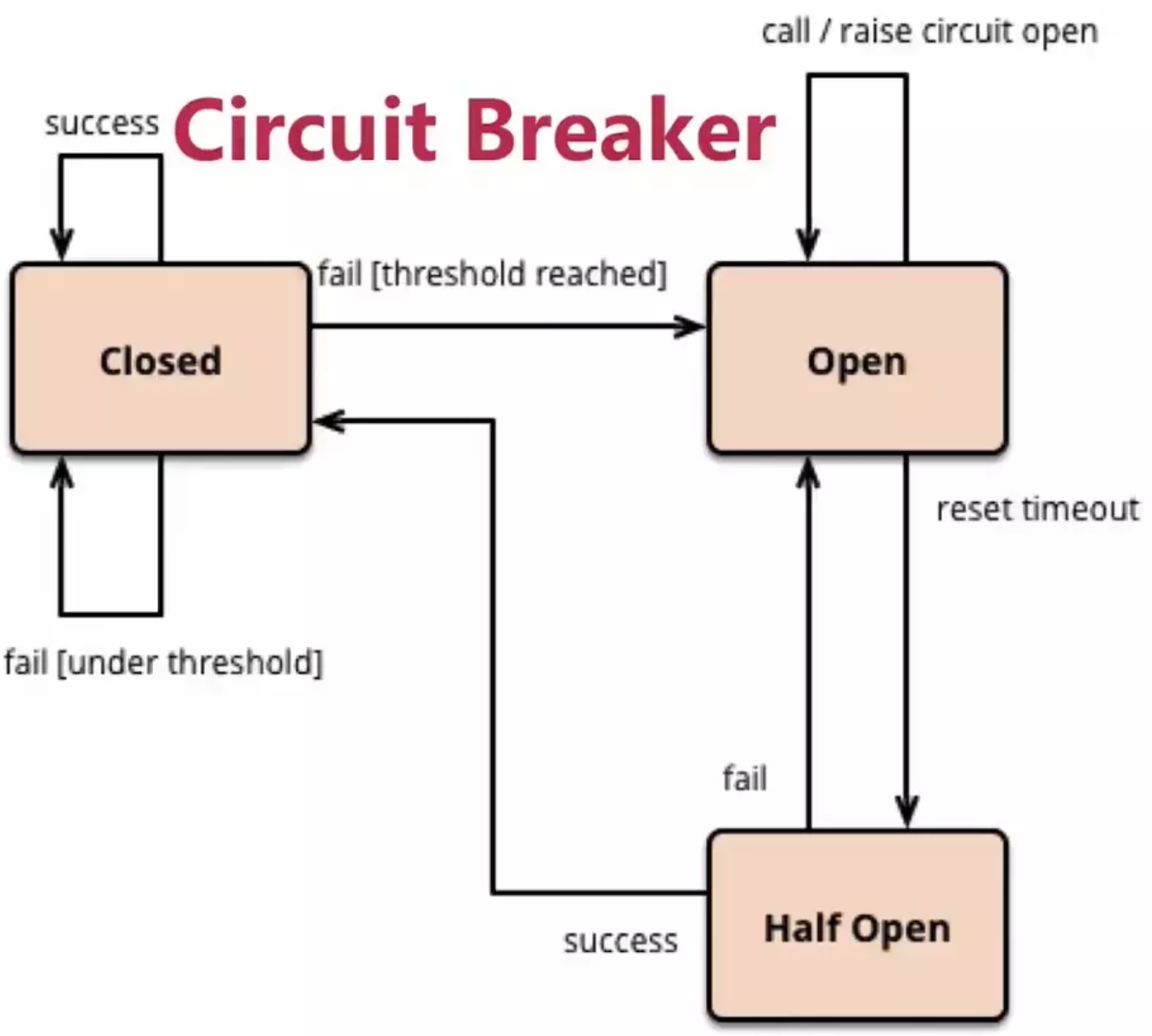
不可用服务的调用快速失败

**超时机制**, **熔断器** 和熔断后的 **降级方法**

## 基本容错模式



# 断路器模式



这里面有三种状态closed、open、half open。

closed就是关闭状态，即服务请求是成功的，当请求失败到异常次数后，就会触发熔断，进入open模式。

open就是打开状态，此时会对服务直接返回错误，快速失败。但是涉及到一个时钟选项，默认的时钟到了这个时间后，就会进入half open半熔断状态。

half open半熔断状态：如果调用都成功或者成功率达到一定比例，就会认为服务恢复了，进入closed模式。否则就认为服务未成功，又回到open状态。

@HystrixProperty(name ="circuitBreaker.enabled", value ="true"), //设置打开熔断

@HystrixProperty(name ="circuitBreaker.requestVolumeThreshold", value ="10"),    //请求数达到后才计算错误率

@HystrixProperty(name ="circuitBreaker.errorThresholdPercentage", value ="40"),    //成功率超过这个数字就代表服务恢复了

@HystrixProperty(name ="circuitBreaker.sleepWindowInMilliseconds", value ="10000"), //熔断时间，即设置一个时间窗口。当失败次数达到熔断是，就会进入这个时间窗口，这时候默认返回服务降级的处理逻辑，过了这个窗口时间，服务恢复了就会采用原来的处理逻辑，如果服务未恢复就进入新的时间窗口。

# 舱壁隔离模式



# Hystrix工作目标

* 防止任何单个依赖项用尽所有容器（例如Tomcat）用户线程。

Preventing any single dependency from using up all container (such as Tomcat) user threads.

* 脱落负载并快速失败而不是排队。

Shedding load and failing fast instead of queueing.

* 在可行的情况下提供回退以保护用户免于失败。

Providing fallbacks wherever feasible to protect users from failure.

* 使用隔离技术（例如隔板，泳道和断路器模式）来限制任何一个依赖项的影响。

Using isolation techniques (such as bulkhead, swimlane, and circuit breaker patterns) to limit the impact of any one dependency.

* 通过近实时指标，监控和警报优化发现时间

Optimizing for time-to-discovery through near real-time metrics, monitoring, and alerting

* 通过Hystrix的大多数方面的配置更改的低延迟传播和对动态属性更改的支持来优化恢复时间，这允许您使用低延迟反馈循环进行实时操作修改。

Optimizing for time-to-recovery by means of low latency propagation of configuration changes and support for dynamic property changes in most aspects of Hystrix, which allows you to make real-time operational modifications with low latency feedback loops.

* 防止整个依赖关系客户端执行中的故障，而不仅仅是网络流量。

Protecting against failures in the entire dependency client execution, not just in the network traffic.

# Hystrix如何实现其目标

* 将对外部系统（或“依赖项”）的所有调用包含在通常在单独线程中执行的对象HystrixCommand或HystrixObservableCommand对象中。

Wrapping all calls to external systems (or “dependencies”) in a HystrixCommand or HystrixObservableCommand object which typically executes within a separate thread

* 为每个依赖项维护一个小的线程池（或信号量）; 如果它变满，将立即拒绝发往该依赖项的请求而不是排队。

Maintaining a small thread-pool (or semaphore) for each dependency; if it becomes full, requests destined for that dependency will be immediately rejected instead of queued up.

* 统计调用成功，失败（客户端引发的异常），超时和线程拒绝。

Measuring successes, failures (exceptions thrown by client), timeouts, and thread rejections.

* 如果服务的错误百分比超过阈值，则手动或自动地使断路器跳闸以停止对特定服务的所有请求一段时间。

Tripping a circuit-breaker to stop all requests to a particular service for a period of time, either manually or automatically if the error percentage for the service passes a threshold.

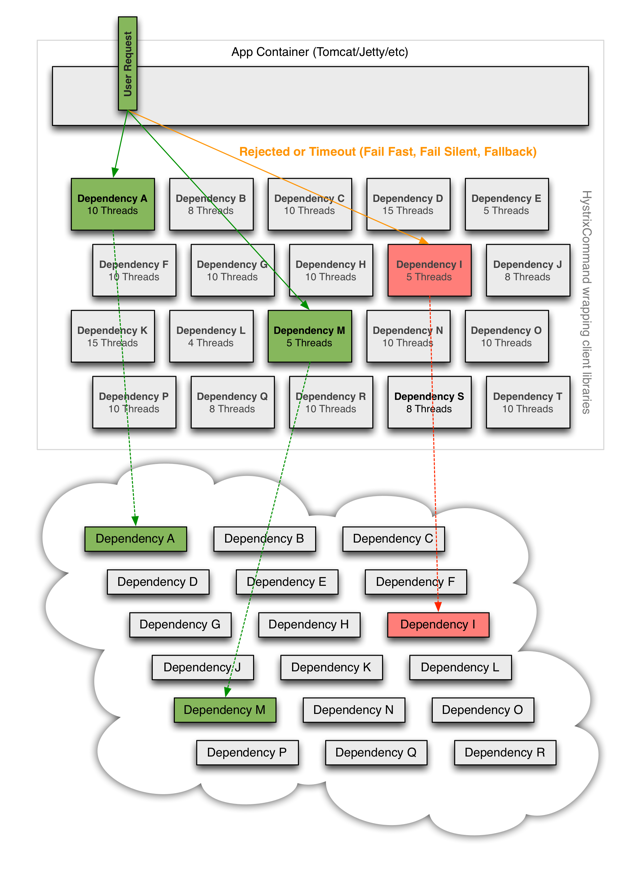
* 当请求失败时执行回退逻辑，被拒绝，超时或短路。

Performing fallback logic when a request fails, is rejected, times-out, or short-circuits.

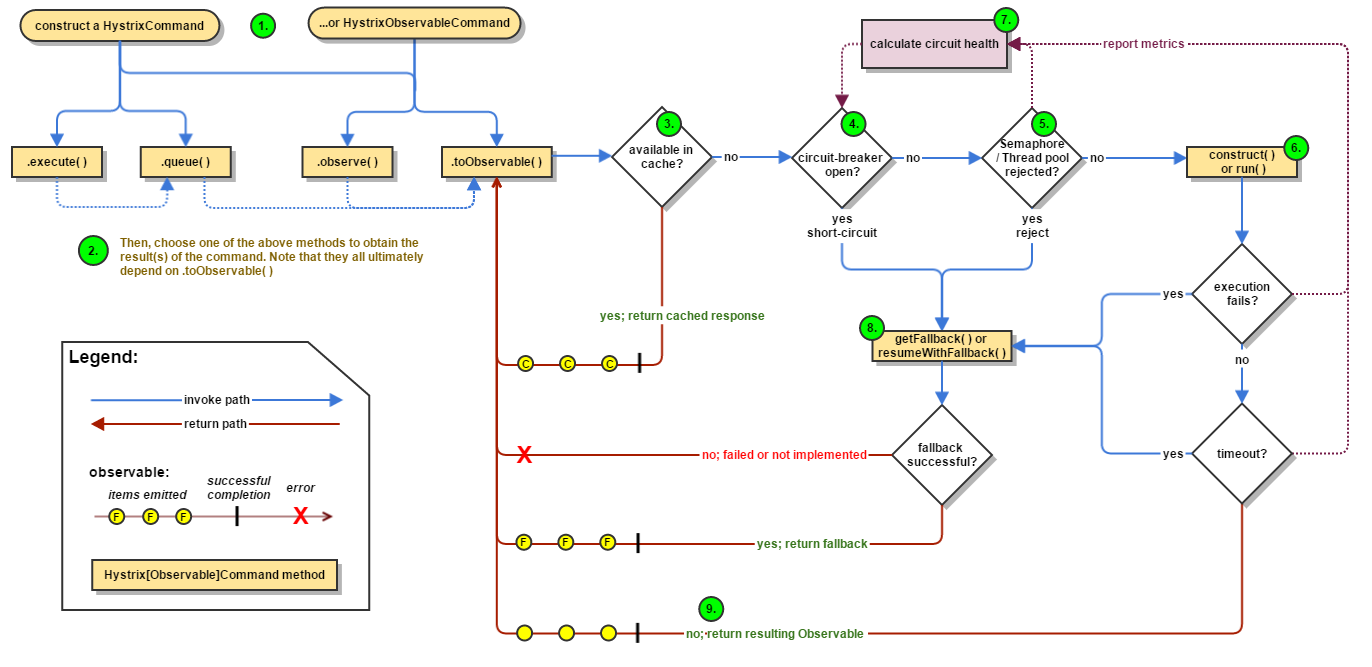
* 近乎实时地监控指标和配置更改。

Monitoring metrics and configuration changes in near real-time.

# Hystrix依赖隔离



# Hystrix工作原理

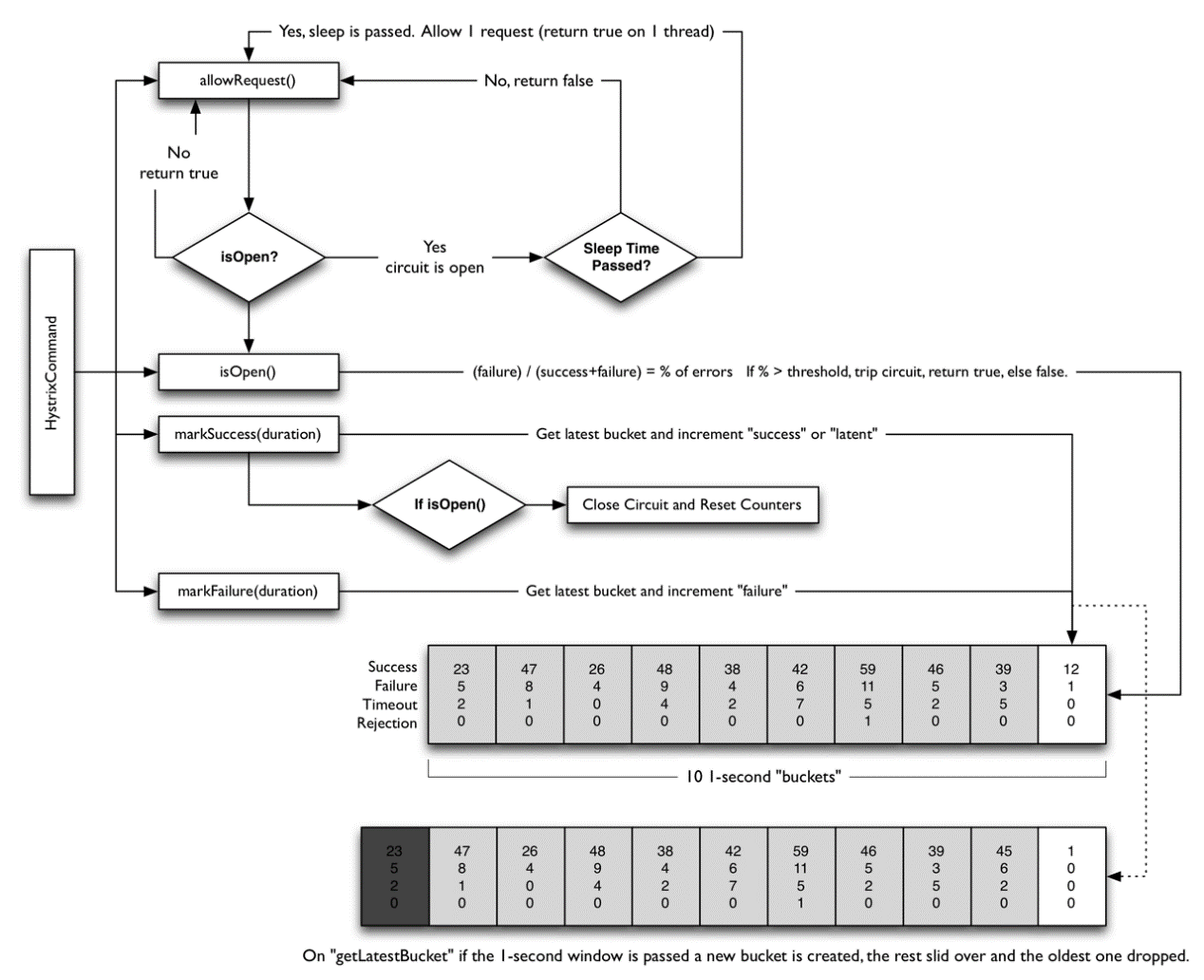


<https://raw.githubusercontent.com/wiki/Netflix/Hystrix/images/hystrix-command-flow-chart.png>

1. [Construct a HystrixCommand or HystrixObservableCommand Object](https://github.com/Netflix/Hystrix/wiki/How-it-Works#flow1)
2. [Execute the Command](https://github.com/Netflix/Hystrix/wiki/How-it-Works#flow2)
3. Is the Response Cached?
4. [Is the Circuit Open?](https://github.com/Netflix/Hystrix/wiki/How-it-Works#flow4)
5. [Is the Thread Pool/Queue/Semaphore Full?](https://github.com/Netflix/Hystrix/wiki/How-it-Works#flow5)
6. [HystrixObservableCommand.construct() or HystrixCommand.run()](https://github.com/Netflix/Hystrix/wiki/How-it-Works#flow6)
7. [Calculate Circuit Health](https://github.com/Netflix/Hystrix/wiki/How-it-Works#flow7)
8. [Get the Fallback](https://github.com/Netflix/Hystrix/wiki/How-it-Works#flow8)
9. [Return the Successful Response](https://github.com/Netflix/Hystrix/wiki/How-it-Works#flow9)

<https://github.com/Netflix/Hystrix/wiki/How-it-Works>

# Hystrix断路器内核



<https://raw.githubusercontent.com/wiki/Netflix/Hystrix/images/circuit-breaker-1280.png>

断路器打开或关闭状态判定

1. 假设电路上的并发流量达到某个阈值HystrixCommandProperties.circuitBreakerRequestVolumeThreshold()
2. 并假设错误百分比超过阈值错误百分比HystrixCommandProperties.circuitBreakerErrorThresholdPercentage()
3. 然后断路器从转换CLOSED到OPEN。
4. 当它打开时，它会短路所有针对该断路器的请求。
5. 经过一段时间HystrixCommandProperties.circuitBreakerSleepWindowInMilliseconds()后，下一个请求将通过（这是HALF-OPEN状态）。如果请求失败，则断路器返回OPEN睡眠窗口持续时间的状态。如果请求成功，则断路器转换为1 CLOSED并且逻辑**1**再次接管。

# 演示

**student-springboot** 基础服务提供者

#### school-springboot 调用 student 服务

通过postman调用school微服务

http://localhost:8088/getSchoolDetails1/abcschool

修改io.spring2go.hystrixlab.schoolservice.delegate.StudentServiceDelegate中StudentServiceDelegate方法上的标注：

@HystrixCommand(fallbackMethod = "callStudentService\_Fallback",

commandProperties = {

@HystrixProperty(name = "circuitBreaker.forceClosed", value = "true"),

@HystrixProperty(name = "execution.isolation.thread.timeoutInMilliseconds", value = "4000")

},

threadPoolKey = "studentServiceThreadPool",

threadPoolProperties = {

@HystrixProperty(name = "coreSize", value = "5"),

@HystrixProperty(name = "maxQueueSize", value = "5")

})

查看hystrix stream和dashboard

hystrix stream:

http://localhost:8088/hystrix.stream

hystrix dashboard

http://localhost:8088/hystrix

# 结尾

配置中心 spring cloud config / Apollo @RefreshScope

网关 spring cloud gateway 80 20 原则

# 相关产品

<https://github.com/Netflix/Hystrix/> Star 16K

• Alibaba Sentinel(Java，5.8K stars)

https://github.com/alibaba/Sentinel中文文档

• Polly(C#，5.6k stars) • https://github.com/App-vNext/Polly

• Hystrix-go(golang, 1.6k stars) • https://github.com/afex/hystrix-go

• Failsafe(Java，2.6k stars) • https://github.com/jhalterman/failsafe

• Resilience4j(Java，3.1k stars) • https://github.com/resilience4j/resilience4j