## What Is Hystrix?

在分布式环境中，许多服务依赖性中的一些不可避免地会失败。 Hystrix是这样一个库，通过添加延迟容忍和容错逻辑，可帮助您控制这些分布式服务之间的交互。 Hystrix通过隔离服务之间的访问点，停止它们之间的级联故障以及提供回滚你选项来实现这一点，所有这些都可以提高系统的整体弹性。

#### History of Hystrix

Hystrix从Netflix API团队在2011年开始的弹性工程工作演变而来。2012年，Hystrix继续发展和成熟，Netflix内的许多团队都采用了它。 今天，Netflix每天都会通过Hystrix执行数百亿线程隔离和数千亿次信号隔离调用。 这导致了正常运行时间和弹性的显着改善。

以下链接提供了有关Hystrix的更多上下文以及它试图解决的挑战：

* [“Making Netflix API More Resilient”](http://techblog.netflix.com/2011/12/making-netflix-api-more-resilient.html)
* [“Fault Tolerance in a High Volume, Distributed System”](http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html)
* [“Performance and Fault Tolerance for the Netflix API”](https://speakerdeck.com/benjchristensen/performance-and-fault-tolerance-for-the-netflix-api-august-2012)
* [“Application Resilience in a Service-oriented Architecture”](http://programming.oreilly.com/2013/06/application-resilience-in-a-service-oriented-architecture.html)
* [“Application Resilience Engineering & Operations at Netflix”] (<https://speakerdeck.com/benjchristensen/application-resilience-engineering-and-operations-at-netflix>)

## What Is Hystrix For?

通过第三方客户端库（通常通过网络）访问依赖关系，从而对延迟和故障进行保护和控制。

停止复杂分布式系统中的级联故障。

失败迅速并迅速恢复。

可能的情况下回退并优雅地降级。

启用近实时监控，警报和操作控制。

## What Problem Does Hystrix Solve?

复杂分布式体系结构中的应用程序具有许多依赖关系，每个依赖关系在某些时候都不可避免地失败。 如果主机应用程序不与这些外部故障隔离，则有可能被拆除。

例如，对于依赖30个服务的应用程序，每个服务的正常运行时间为99.99％，以下是您所期望的：

99.9930 = 99.7％的正常运行时间

10亿次请求中的0.3％= 3,000,000次失败

即使所有依赖关系都具有出色的正常运行时间，也可以保持2+小时的停机时间/月

现实通常更糟糕。

即使所有依赖性都能很好地发挥作用，如果您不设计整个系统的弹性，那么几十项服务中的每一项甚至0.01％的停机时间的总体影响等同于每月停机时间可能达到的小时数。

当一切服务都很健壮时，请求流程可能如下所示：



当许多后端系统中的一个出现延迟时，它可能会阻止整个用户请求：



对于高流量场景，单个后端依赖出现延迟可能导致所有服务器上的所有资源在数秒内达到饱和状态。

应用程序中的每个点都可以通过网络或可能导致网络请求的客户端库进行传播，这是潜在失败的根源。 比失败更糟糕的是，这些应用程序还可能导致服务之间的延迟增加，从而备份队列，线程和其他系统资源，从而导致整个系统发生更多级联故障。



当通过第三方客户端执行网络访问时，这些问题会加剧 - 实施细节隐藏且随时可能更改的“黑匣子”，并且每个客户端库的网络或资源配置都不同，并且通常难以监控和 更改。

更糟糕的是，传递依赖性执行潜在的昂贵或容易出错的网络调用，而不会被应用程序明确调用。

网络连接失败或降级。 服务和服务器失败或变得缓慢。 新的库或服务部署会改变行为或性能特征。 客户端库存在bug。

所有这些都表示需要进行隔离和管理的故障和延迟，以便单个失败的依赖关系无法拖垮整个应用程序或系统。

## What Design Principles Underlie Hystrix?

Hystrix works by:

防止任何单个依赖项使用容器（如Tomcat）所有的用户线程。

切断负载并快速失败而不是排队。

尽可能提供回退以保护用户免受故障。

使用隔离技术（例如隔板，泳道和断路器模式）来限制任何一种依赖的影响。

通过接近实时的指标，监控和警报来优化发现时间

通过配置更改的低延迟传播优化恢复时间，并支持Hystrix大多数方面的动态属性更改，这使您可以使用低延迟反馈环进行实时操作修改。

防止整个依赖客户端执行失败，而不仅仅是网络流量。

## How Does Hystrix Accomplish Its Goals?