“舱壁模式”对于熟悉Docker的读者一定不陌生，Docker通过“舱壁模式”实现进程的隔离，使得容器与容器之间不会互相影响。而Hystrix则使用该模式实现线程池的隔离，它会为每一个Hystrix命令创建一个独立的线程池，这样就算某个在Hystrix命令包装下的依赖服务出现延迟过高的情况，也只是对该依赖服务的调用产生影响，而不会拖慢其他的服务。

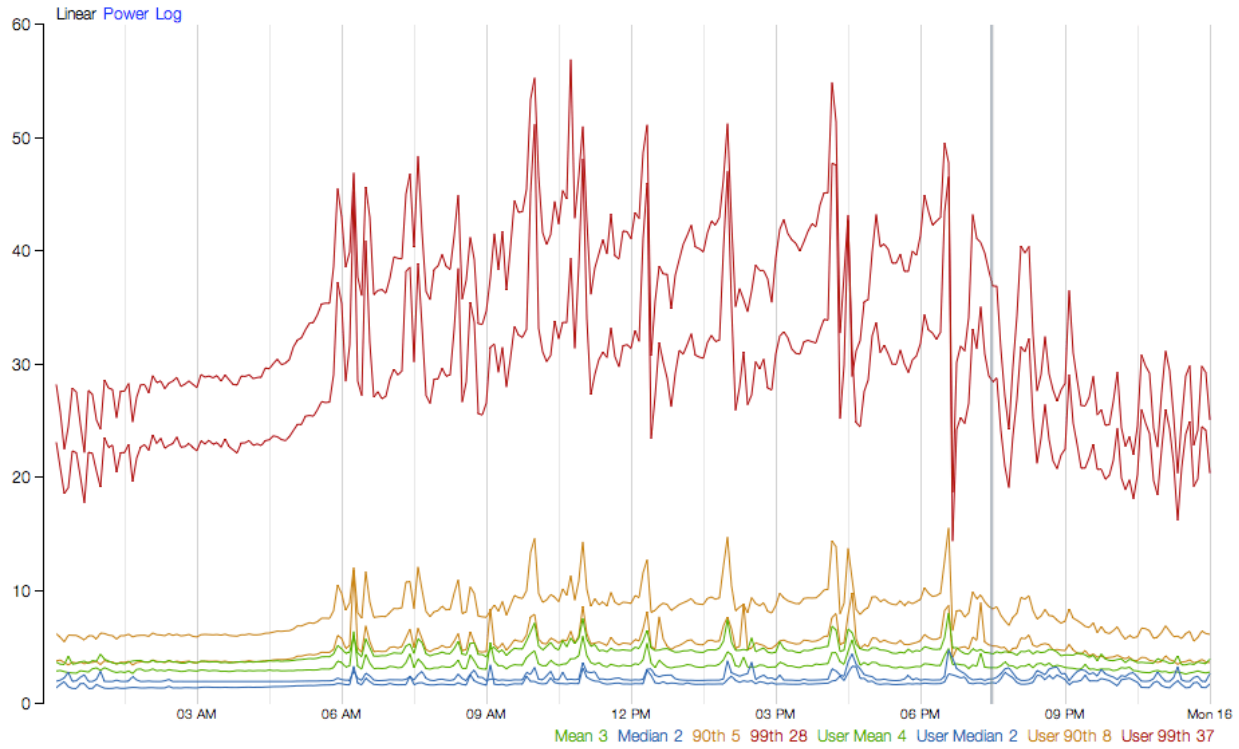
通过对依赖服务的线程池隔离实现，可以带来如下优势：

- 应用自身得到完全的保护，不会受不可控的依赖服务影响。即便给依赖服务分配的线程池被填满，也不会影响应用自身的额其余部分。
- 可以有效的降低接入新服务的风险。如果新服务接入后运行不稳定或存在问题，完全不会影响到应用其他的请求。
- 当依赖的服务从失效恢复正常后，它的线程池会被清理并且能够马上恢复健康的服务，相比之下容器级别的清理恢复速度要慢得多。
- 当依赖的服务出现配置错误的时候，线程池会快速的反应出此问题（通过失败次数、延迟、超时、拒绝等指标的增加情况）。同时，我们可以在不影响应用功能的情况下通过实时的动态属性刷新（后续会通过Spring Cloud Config与Spring Cloud Bus的联合使用来介绍）来处理它。
- 当依赖的服务因实现机制调整等原因造成其性能出现很大变化的时候，此时线程池的监控指标信息会反映出这样的变化。同时，我们也可以通过实时动态刷新自身应用对依赖服务的阈值进行调整以适应依赖方的改变。
- 除了上面通过线程池隔离服务发挥的优点之外，每个专有线程池都提供了内置的并发实现，可以利用它为同步的依赖服务构建异步的访问。

总之，通过对依赖服务实现线程池隔离，让我们的应用更加健壮，不会因为个别依赖服务出现问题而引起非相关服务的异常。同时，也使得我们的应用变得更加灵活，可以在不停止服务的情况下，配合动态配置刷新实现性能配置上的调整。

虽然线程池隔离的方案带了如此多的好处，但是很多使用者可能会担心为每一个依赖服务都分配一个线程池是否会过多地增加系统的负载和开销。对于这一点，使用者不用过于担心，因为这些顾虑也是大部分工程师们会考虑到的，Netflix在设计Hystrix的时候，认为线程池上的开销相对于隔离所带来的好处是无法比拟的。同时，Netflix也针对线程池的开销做了相关的测试，以证明和打消Hystrix实现对性能影响的顾虑。

下图是Netflix Hystrix官方提供的一个Hystrix命令的性能监控，该命令以每秒60个请求的速度（QPS）向一个单服务实例进行访问，该服务实例每秒运行的线程数峰值为350个。



从图中的统计我们可以看到，使用线程池隔离与不使用线程池隔离的耗时差异如下表所示：

比较情况	未使用线程池隔离	使用了线程池隔离	耗时差距
中位数	2ms	2ms	2ms
90百分位	5ms	8ms	3ms
99百分位	28ms	37ms	9ms

在99%的情况下，使用线程池隔离的延迟有9ms，对于大多数需求来说这样的消耗是微乎其微的，更何况为系统在稳定性和灵活性上所带来的巨大提升。虽然对于大部分的请求我们可以忽略线程池的额外开销，而对于小部分延迟本身就非常小的请求（可能只需要1ms），那么9ms的延迟开销还是非常昂贵的。实际上Hystrix也为此设计了另外的一个解决方案：信号量。

Hystrix中除了使用线程池之外，还可以使用信号量来控制单个依赖服务的并发度，信号量的开销要远比线程池的开销小得多，但是它不能设置超时和实现异步访问。所以，只有在依赖服务是足够可靠的情况下才使用信号量。在HystrixCommand和HystrixObservableCommand中2处支持信号量的使用：

- 命令执行：如果隔离策略参数execution.isolation.strategy设置为SEMAPHORE，Hystrix会使用信号量替代线程池来控制依赖服务的并发控制。
- 降级逻辑：当Hystrix尝试降级逻辑时候，它会在调用线程中使用信号量。

信号量的默认值为10，我们也可以通过动态刷新配置的方式来控制并发线程的数量。对于信号量大小的估算方法与线程池并发度的估算类似。仅访问内存数据的请求一般耗时在1ms以

内，性能可以达到5000rps，这样级别的请求我们可以将信号量设置为1或者2，我们可以按此标准并根据实际请求耗时来设置信号量。

如何使用

在上一节的示例中，我们使用了@HystrixCommand来将某个函数包装成了Hystrix命令，这里除了定义服务降级之外，Hystrix框架就会自动的为这个函数实现调用的隔离。所以，依赖隔离、服务降级在使用时候都是一体化实现的，这样利用Hystrix来实现服务容错保护在编程模型上就非常方便的，并且考虑更为全面。除了依赖隔离、服务降级之外，还有一个重要元素：断路器。我们将在下一节做详细的介绍，这三个重要利器构成了Hystrix实现服务容错保护的强力组合拳。