Hystrix 初体验 (一): 原理

作者：flowaters
原文链接：https://ld246.com/article/1527577919301
来源网站：链滴

简介

Hystrix是一个用来隔离延迟和错误的库，即熔断器。

具体来讲，当远程系统、服务、第三方库不可用时，可以将其隔离，防止系统整体雪崩。

由于系统迟早会出现不可用情况，其可以提高系统的弹性。

如果想了解更多，请参考底部的防雪崩利器：熔断器 Hystrix 的原理与使用。

能做什么

- 处理突发的延时和故障：防止雪崩，做到优雅的降级。
- 实时检测：实时监控系统运行情况和配置变化情况，秒级做出响应。
- 并发性：并发执行请求和缓存结果。

第一个例子

$ git clone https://github.com/Netflix/Hystrix.git

在Hystrix/hystrix-examples/src/main/java/com/netflix/hystrix/examples/demo下有两个例子HystrixCommandDemo和HystrixCommandAsyncDemo

HystrixCommandDemo的

运行效果如下:

```
Request => GetUserAccountCommand[SUCCESS][40ms], GetPaymentInformationCommand[
UCCESS][14ms], GetOrderCommand[SUCCESS][216ms], GetUserAccountCommand[SUCCESS,
RESPONSE_FROM_CACHE][0ms]x2, CreditCardCommand[SUCCESS][889ms]
Request => GetUserAccountCommand[SUCCESS][39ms], GetPaymentInformationCommand[
UCCESS][27ms], GetOrderCommand[SUCCESS][234ms], GetUserAccountCommand[SUCCESS,
RESPONSE_FROM_CACHE][0ms]x2, CreditCardCommand[SUCCESS][920ms]
Request => GetUserAccountCommand[SUCCESS][41ms], GetPaymentInformationCommand[
UCCESS][12ms], GetOrderCommand[SUCCESS][226ms], GetUserAccountCommand[SUCCESS,
RESPONSE_FROM_CACHE][0ms]x2, CreditCardCommand[SUCCESS][1040ms]
Request => GetUserAccountCommand[SUCCESS][42ms], GetPaymentInformationCommand[
UCCESS][25ms], GetOrderCommand[SUCCESS][234ms], GetUserAccountCommand[SUCCESS,
RESPONSE_FROM_CACHE][0ms]x2, CreditCardCommand[SUCCESS][1302ms]
Request => GetUserAccountCommand[SUCCESS][36ms], GetPaymentInformationCommand[
UCCESS][12ms], GetOrderCommand[SUCCESS][229ms], GetUserAccountCommand[SUCCESS,
RESPONSE_FROM_CACHE][0ms]x2, CreditCardCommand[SUCCESS][1361ms]
Request => GetUserAccountCommand[SUCCESS][43ms], GetPaymentInformationCommand[
UCCESS][12ms], GetOrderCommand[SUCCESS][214ms], GetUserAccountCommand[SUCCESS,
RESPONSE_FROM_CACHE][0ms]x2, CreditCardCommand[SUCCESS][1494ms]
Request => GetUserAccountCommand[FAILURE, FALLBACK_SUCCESS][10ms], GetPaymentInf
rmationCommand[SUCCESS][8ms], GetOrderCommand[SUCCESS][101ms], GetUserAccountC
mmand[FAILURE, FALLBACK_SUCCESS, RESPONSE_FROM_CACHE][0ms]x2, CreditCardComma
d[SUCCESS][1150ms]
Request => GetUserAccountCommand[SUCCESS][3ms], GetPaymentInformationCommand[S
CCESS][19ms], GetOrderCommand[SUCCESS][71ms], GetUserAccountCommand[SUCCESS, RE
PONSE_FROM_CACHE][0ms]x2, CreditCardCommand[SUCCESS][831ms]
```

```
Request => GetUserAccountCommand[SUCCESS][6ms], GetPaymentInformationCommand[S
CCESS][24ms], GetOrderCommand[SUCCESS][112ms], GetUserAccountCommand[SUCCESS, R
SPONSE_FROM_CACHE][0ms]x2, CreditCardCommand[SUCCESS][1015ms]
Request => GetUserAccountCommand[SUCCESS][11ms], GetPaymentInformationCommand[
UCCESS][26ms], GetOrderCommand[SUCCESS][232ms], GetUserAccountCommand[SUCCESS,
RESPONSE_FROM_CACHE][0ms]x2, CreditCardCommand[SUCCESS][971ms]
Request => GetUserAccountCommand[SUCCESS][4ms], GetPaymentInformationCommand[S
CCESS][9ms], GetOrderCommand[SUCCESS][214ms], GetUserAccountCommand[SUCCESS, RE
PONSE_FROM_CACHE][0ms]x2, CreditCardCommand[SUCCESS][1325ms]
Request => GetUserAccountCommand[SUCCESS][7ms], GetPaymentInformationCommand[S
CCESS][19ms], GetOrderCommand[SUCCESS][91ms], GetUserAccountCommand[SUCCESS, RE
PONSE_FROM_CACHE][0ms]x2, CreditCardCommand[SUCCESS][1480ms]
Request => GetUserAccountCommand[SUCCESS][10ms], GetPaymentInformationCommand[
UCCESS][15ms], GetOrderCommand[SUCCESS][237ms], GetUserAccountCommand[SUCCESS,
RESPONSE_FROM_CACHE][0ms]x2, CreditCardCommand[SUCCESS][1412ms]

##########################################################################
##############
# CreditCardCommand: Requests: 13 Errors: 0 (0%)   Mean: 0 75th: 0 90th: 0 99th: 0
# GetOrderCommand: Requests: 16 Errors: 0 (0%)   Mean: 0 75th: 0 90th: 0 99th: 0
# GetUserAccountCommand: Requests: 16 Errors: 1 (6%)   Mean: 0 75th: 0 90th: 0 99th: 0
# GetPaymentInformationCommand: Requests: 16 Errors: 0 (0%)   Mean: 0 75th: 0 90th: 0 99th
 0
##########################################################################
##############
```

HystrixCommandAsyncDemo的运行效果如下：

```
Request => GetUserAccountCommand[TIMEOUT, FALLBACK_SUCCESS][80ms], GetOrderCo
mand[SUCCESS][209ms], GetPaymentInformationCommand[SUCCESS][13ms], GetUserAccoun
Command[TIMEOUT, FALLBACK_SUCCESS, RESPONSE_FROM_CACHE][0ms]x2, CreditCardCo
mand[SUCCESS][1269ms]
Request => GetUserAccountCommand[SUCCESS][5ms], GetOrderCommand[SUCCESS][183ms
, GetPaymentInformationCommand[SUCCESS][23ms], GetUserAccountCommand[SUCCESS, R
SPONSE_FROM_CACHE][0ms]x2, CreditCardCommand[SUCCESS][1443ms]
Request => GetUserAccountCommand[SUCCESS][11ms], GetOrderCommand[SUCCESS][91ms
, GetPaymentInformationCommand[SUCCESS][24ms], GetUserAccountCommand[SUCCESS, R
SPONSE_FROM_CACHE][0ms]x2, CreditCardCommand[SUCCESS][980ms]

##########################################################################
##############
# CreditCardCommand: Requests: 3 Errors: 0 (0%)   Mean: 0 75th: 0 90th: 0 99th: 0
# GetOrderCommand: Requests: 4 Errors: 0 (0%)   Mean: 0 75th: 0 90th: 0 99th: 0
# GetUserAccountCommand: Requests: 4 Errors: 1 (25%)   Mean: 0 75th: 0 90th: 0 99th: 0
# GetPaymentInformationCommand: Requests: 4 Errors: 0 (0%)   Mean: 0 75th: 0 90th: 0 99th:
0
##########################################################################
##############
```

这两个示例中，包含打印状态和执行Hystrix请求两部分。请求包含了四种情形：

- HystrixCommand.execute
- HystrixCommand.queue

- HystrixCommand.observe
- HystrixCommand.toObservable

具体可以看代码。

同时，通过ConfigurationManager来设置了超时时间等信息，如下：

```
        ConfigurationManager.getConfigInstance().setProperty("hystrix.threadpool.default.coreSi
e", 8);
        ConfigurationManager.getConfigInstance().setProperty("hystrix.command.CreditCardCo
mand.execution.isolation.thread.timeoutInMilliseconds", 3000);
        ConfigurationManager.getConfigInstance().setProperty("hystrix.command.GetUserAccou
tCommand.execution.isolation.thread.timeoutInMilliseconds", 50);
        ConfigurationManager.getConfigInstance().setProperty("hystrix.command.default.metrics
rollingPercentile.numBuckets", 60);
```

原理

在有了直观的认识之后，来看下其工作原理。

官方的原理介绍非常的详细，点击可见其流程图。

流程图中分为9个步骤，分别介绍如下：

构建对象

HystrixCommand

返回一个单一的响应

```
HystrixCommand command = new HystrixCommand(arg1, arg2);
```

HystrixObservableCommand

返回一个观测对象

```
HystrixObservableCommand command = new HystrixObservableCommand(arg1, arg2);
```

执行命令

类	方法	
HystrixCommand	execute()	阻塞执行, 返回response 异常 queue().get()
HystrixCommand	queue()	异步执行，返回future对象 oObservable().toBlocking().toFuture()
HystrixObservableCommand	observe()	返回订阅观测量
HystrixObservableCommand	toObservable()	返回观测量

订阅后执行

```
K             value   = command.execute();
Future<K>     fValue  = command.queue();
Observable<K> ohValue = command.observe();         //hot observable
Observable<K> ocValue = command.toObservable();    //cold observable
```

结果缓存

可以设置是否缓存请求的结果。

如果设置了缓存并且缓存存在，则会立即返回缓存中的结果。

断路器状态

执行请求前，会先检查断路器状态。如果断路器是打开状态(open/tripped)，则不会执行请求，而是接到第8步，得到反馈。

是否已满

在执行请求前，再检查此请求相关的线程池是否已满，或者信号量是否已满。

如果满了，同样直接执行第8步，得到反馈。

构造方法

到了这里，才真正开始执行自己的方法。

先执行 HystrixCommand.run()或者HystrixObservableCommand.construct()

如果run()或者construct()方法超时了，则抛出TimeoutException异常，执行第8步，得到反馈；同丢弃掉run或者construct()方法的返回值。

注，Hystrix是无法强制线程停止工作的，最多是触发一个InterruptedException异常。如果其内部程不响应InterruptedException，则还是会继续工作，虽然client已经返回了超时异常。

这种机制可能把线程池打满，同时系统load还有可能不高。

大多数的http库是不响应InterruptedException异常的，需要自己设置合理的read/write超时时间。

如果命令执行正常结束，没有抛异常。Hystrix在记录日志和更新状态后，会返回正常执行结果。

对于run()，返回带有响应的Observable，同时发送onCompleted事件。

对于construct()，返回和construct()的返回相同的Observable。

计算断路器状态

Hystrix会记录成功次数，失败次数，拒绝次数和超时次数。

通过这些统计信息，来切换断路器的状态。

得到反馈

当有请求失败时，Hystrix就会执行fallback方法。

对于HystrixCommand，执行HystrixCommand.getFallback()

对于HystrixObservableCommand，执行HystrixObservableCommand.resumeWithFallback()

如果没有实现fallback方法，或者fallback方法抛异常了，Hystrix同样返回一个没有内容的Observable象，同时发送onError事件。

最佳实践是实现一个不会失败的fallback方法。

方法	结果
execute()	抛异常
queue()	返回Future对象，调用其get()时抛异常
observe()	返回Observable对象，订阅其时会触onError()方法来快速失败
toObservable()	同observe()

返回成功结果

具体见图片

● HystrixCommand的execute和queue方法，以及HystrixObservableCommand的observe()和tobserve()方法，最后都会转化成HystrixObservableCommand的toObserve()方法。

@adrianb11制作了一个动画版本的调用说明

使用

断路器配置

含义	参数
容量	HystrixCommandProperties.circuitBreakeRequestVolumeThreshold()
错误率	HystrixCommandProperties.circuitBreaerErrorThresholdPercentage()
冷却时间	HystrixCommandProperties.circuitBrakerSleepWindowInMilliseconds()

在冷却时间后，仅将一个新请求放到后端。成功后转为CLOSE，否则维持OPEN。

线程池方式

优点

线程池方式可以提供完备的隔离保护，适合于经常变化的场景。

缺点

线程池方式增加了复杂度，每一个命令的执行都要经历排队、调度、上下文切换的过程。

性能影响

Netflix选择接受这些复杂度带来的额外性能损失，来得到系统的稳定性。

在单机350QPS的测试场景下，

平均是没有额外时间消耗；

90th是3ms

99th是9ms

信号量方式

启用

将execution.isolation.strategy设置为SEMAPHORE

fallback

直接由tomcat线程来执行fallback操作

请求合并(Request Collapsing)

如果多个请求，会同时访问后端的同一个服务时，那么可以做一个batch操作，将对后端的多并发访，变成单一请求访问。

- 优点：减少网络消耗
- 缺点：增加延时
- 不适用场景：并发量很小
- 适用场景：网络IO是瓶颈时

请求缓存(Request Caching)

如果开启了请求缓存，不同线程请求同一数据时，仅会请求后端数据一次，然后不同线程会返回同样请求结果。

其中的缓存主键(cache key)需要用户自己定义。

参考

- 防雪崩利器：熔断器 Hystrix 的原理与使用

- Netflix Hystrix github
- Hystrix Home
- Hystrix How To Use
- Hystrix How it Works