使用 Hystrix 实现自动降级与依赖隔离

2017-07-18 ImportNew

（点击上方公众号，可快速关注）

> 来源：高广超，
> www.jianshu.com/p/138f92aa83dc
> 如有好文章投稿，请点击 → 这里了解详情

这篇文章是记录了自己的一次集成Hystrix的经验，原本写在公司内部wiki里，所以里面有一些内容为了避免重复，直接引用了其他同事的wiki，而发布到外网，这部分就不能直接引用了，因此可能不会太完整，后续会补充进去。

1.背景

目前对于一些非核心操作，如增减库存后保存操作日志 发送异步消息时（具体业务流程），一旦出现MQ服务异常时，会导致接口响应超时，因此可以考虑对非核心操作引入服务降级、服务隔离。

2.Hystrix说明

官方文档 [https://github.com/Netflix/Hystrix/wiki]

hystrix是netflix开源的一个容灾框架，解决当外部依赖故障时拖垮业务系统、甚至引起雪崩的问题。

2.1为什么需要Hystrix?

在大中型分布式系统中，通常系统很多依赖(HTTP,hession,Netty,Dubbo等)，在高并发访问下,这些依赖的稳定性与否对系统的影响非常大,但是依赖有很多不可控问题:如网络连接缓慢，资源繁忙，暂时不可用，服务脱机等。

当依赖阻塞时,大多数服务器的线程池就出现阻塞(BLOCK),影响整个线上服务的稳定性，在复杂的分布式架构的应用程序有很多的依赖，都会不可避免地在某些时候失败。高并发的依赖失败时如果没有隔离措施，当前应用服务就有被拖垮的风险。

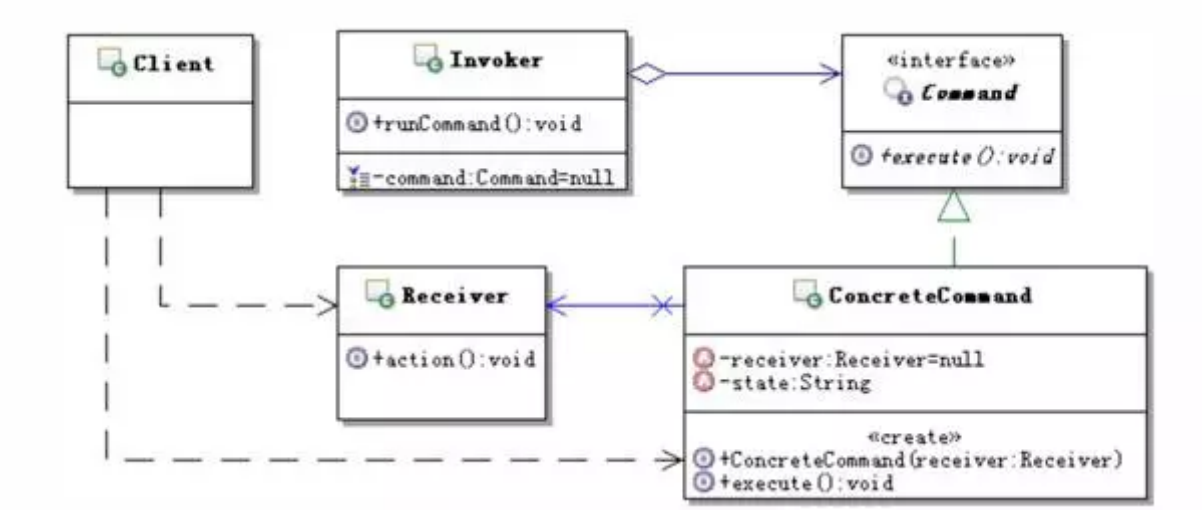
> 例如:一个依赖30个SOA服务的系统,每个服务99.99%可用。
> 99.99%的30次方 ≈ 99.7%
> 0.3% 意味着一亿次请求 会有 3,000,00次失败
> 换算成时间大约每月有2个小时服务不稳定.

> 随着服务依赖数量的变多，服务不稳定的概率会成指数性提高.

解决问题方案:对依赖做隔离。

2.2Hystrix设计理念

想要知道如何使用，必须先明白其核心设计理念，Hystrix基于命令模式，通过UML图先直观的认识一下这一设计模式。



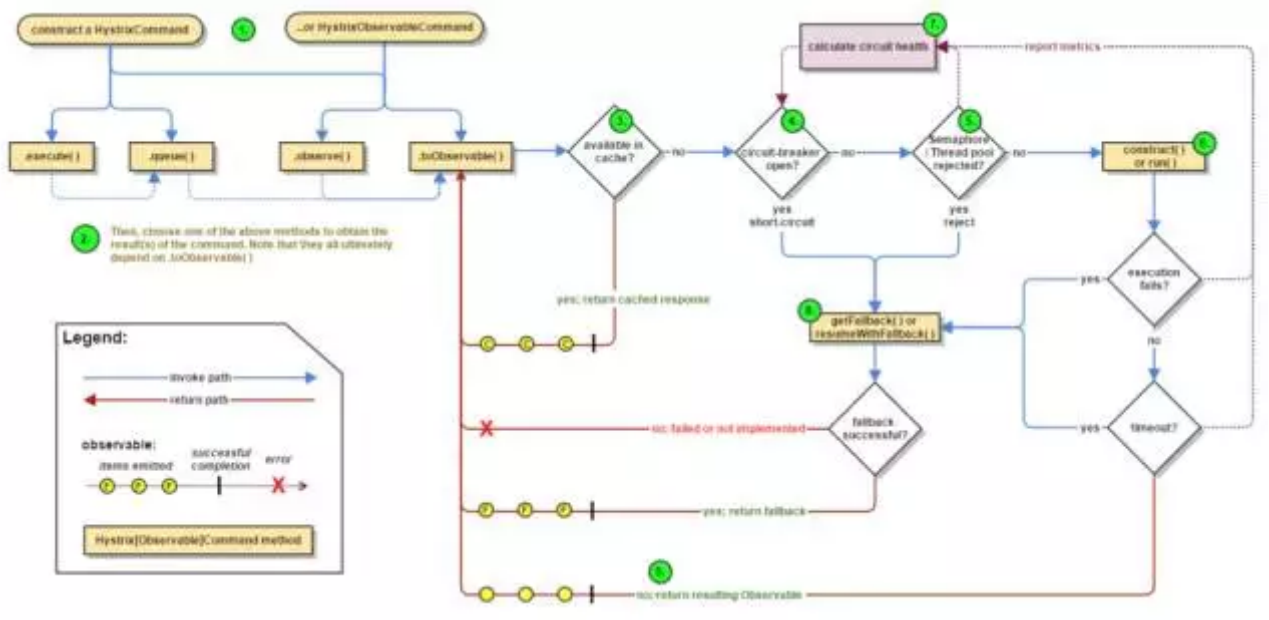
可见，Command是在Receiver和Invoker之间添加的中间层，Command实现了对Receiver的封装。那么Hystrix的应用场景如何与上图对应呢?

API既可以是Invoker又可以是reciever，通过继承Hystrix核心类HystrixCommand来封装这些API（例如，远程接口调用，数据库查询之类可能会产生延时的操作）。就可以为API提供弹性保护了。

2.3 Hystrix如何解决依赖隔离

- Hystrix使用命令模式HystrixCommand(Command)包装依赖调用逻辑，每个命令在单独线程中/信号授权下执行。
- 可配置依赖调用超时时间,超时时间一般设为比99.5%平均时间略高即可.当调用超时时，直接返回或执行fallback逻辑。
- 为每个依赖提供一个小的线程池（或信号），如果线程池已满调用将被立即拒绝，默认不采用排队.加速失败判定时间。
- 依赖调用结果分:成功，失败（抛出异常），超时，线程拒绝，短路。请求失败(异常，拒绝，超时，短路)时执行fallback(降级)逻辑。
- 提供熔断器组件,可以自动运行或手动调用,停止当前依赖一段时间(10秒)，熔断器默认错误率阈值为50%,超过将自动运行。
- 提供近实时依赖的统计和监控。

2.4Hystrix流程结构解析



流程说明:

1:每次调用创建一个新的HystrixCommand,把依赖调用封装在run()方法中.

2:执行execute()/queue做同步或异步调用.

3:判断熔断器(circuit-breaker)是否打开,如果打开跳到步骤8,进行降级策略,如果关闭进入步骤.

4:判断线程池/队列/信号量是否跑满，如果跑满进入降级步骤8,否则继续后续步骤.

5:调用HystrixCommand的run方法.运行依赖逻辑

5a:依赖逻辑调用超时,进入步骤8.

6:判断逻辑是否调用成功

6a:返回成功调用结果

6b:调用出错，进入步骤8.

7:计算熔断器状态,所有的运行状态(成功, 失败, 拒绝,超时)上报给熔断器，用于统计从而判断熔断器状态.

8:getFallback()降级逻辑.

以下四种情况将触发getFallback调用:

(1):run()方法抛出非HystrixBadRequestException异常。

(2):run()方法调用超时

(3):熔断器开启拦截调用

(4):线程池/队列/信号量是否跑满

8a:没有实现getFallback的Command将直接抛出异常

8b:fallback降级逻辑调用成功直接返回

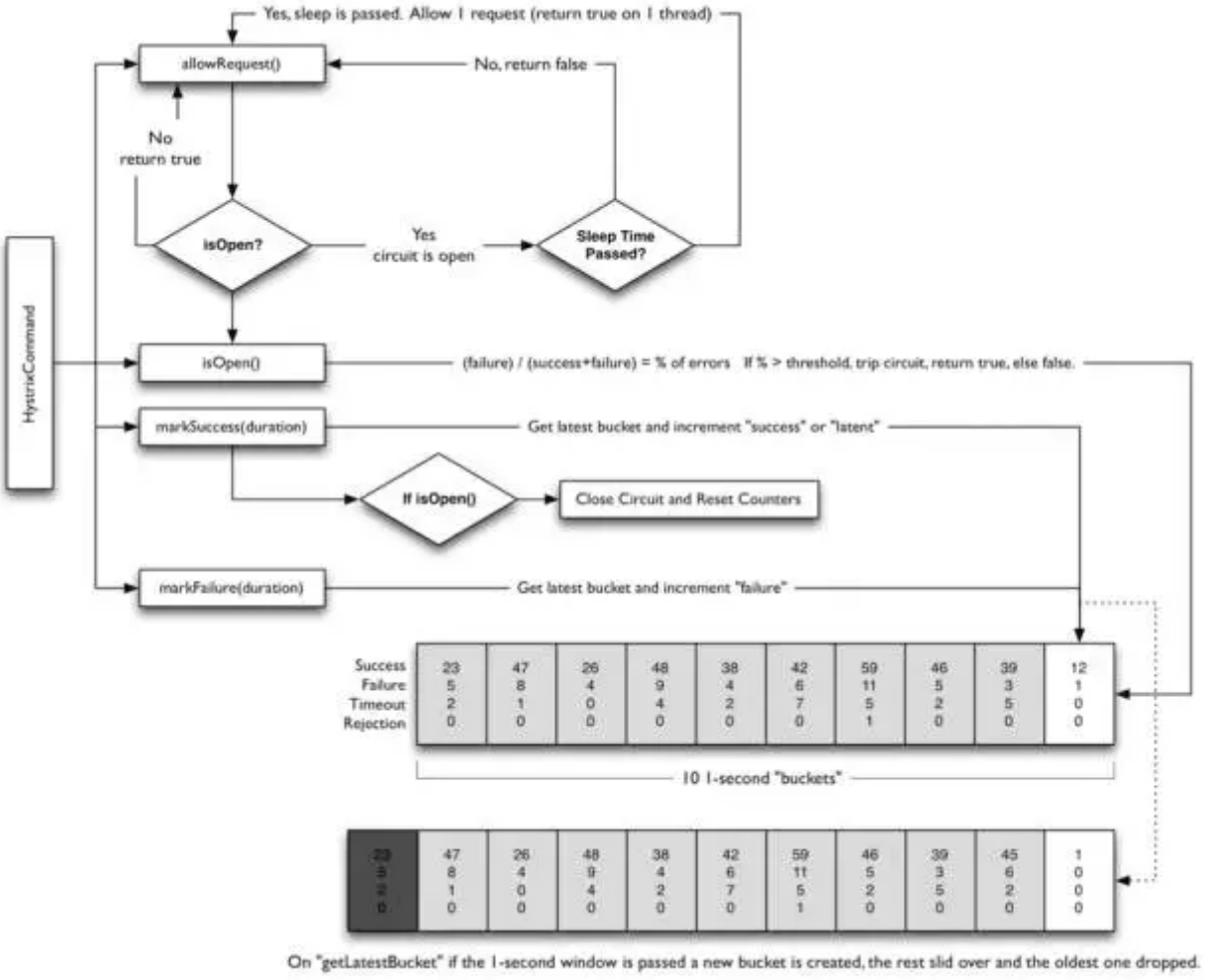
8c:降级逻辑调用失败抛出异常

9:返回执行成功结果

2.5 熔断器:Circuit Breaker

每个熔断器默认维护10个bucket,每秒一个bucket,每个bucket记录成功,失败,超时,拒绝的状态。

默认错误超过50%且10秒内超过20个请求进行中断拦截。



2.6 Hystrix隔离分析

Hystrix隔离方式采用线程/信号的方式,通过隔离限制依赖的并发量和阻塞扩散。

(1)线程隔离

把执行依赖代码的线程与请求线程(如:jetty线程)分离，请求线程可以自由控制离开的时间(异步过程)。

通过线程池大小可以控制并发量，当线程池饱和时可以提前拒绝服务,防止依赖问题扩散。

线上建议线程池不要设置过大，否则大量堵塞线程有可能会拖慢服务器。

(2)线程隔离的优缺点

- 线程隔离的优点:

[1]:使用线程可以完全隔离第三方代码,请求线程可以快速放回。
[2]:当一个失败的依赖再次变成可用时，线程池将清理，并立即恢复可用，而不是一个长时间的恢复。
[3]:可以完全模拟异步调用，方便异步编程。

- 线程隔离的缺点:

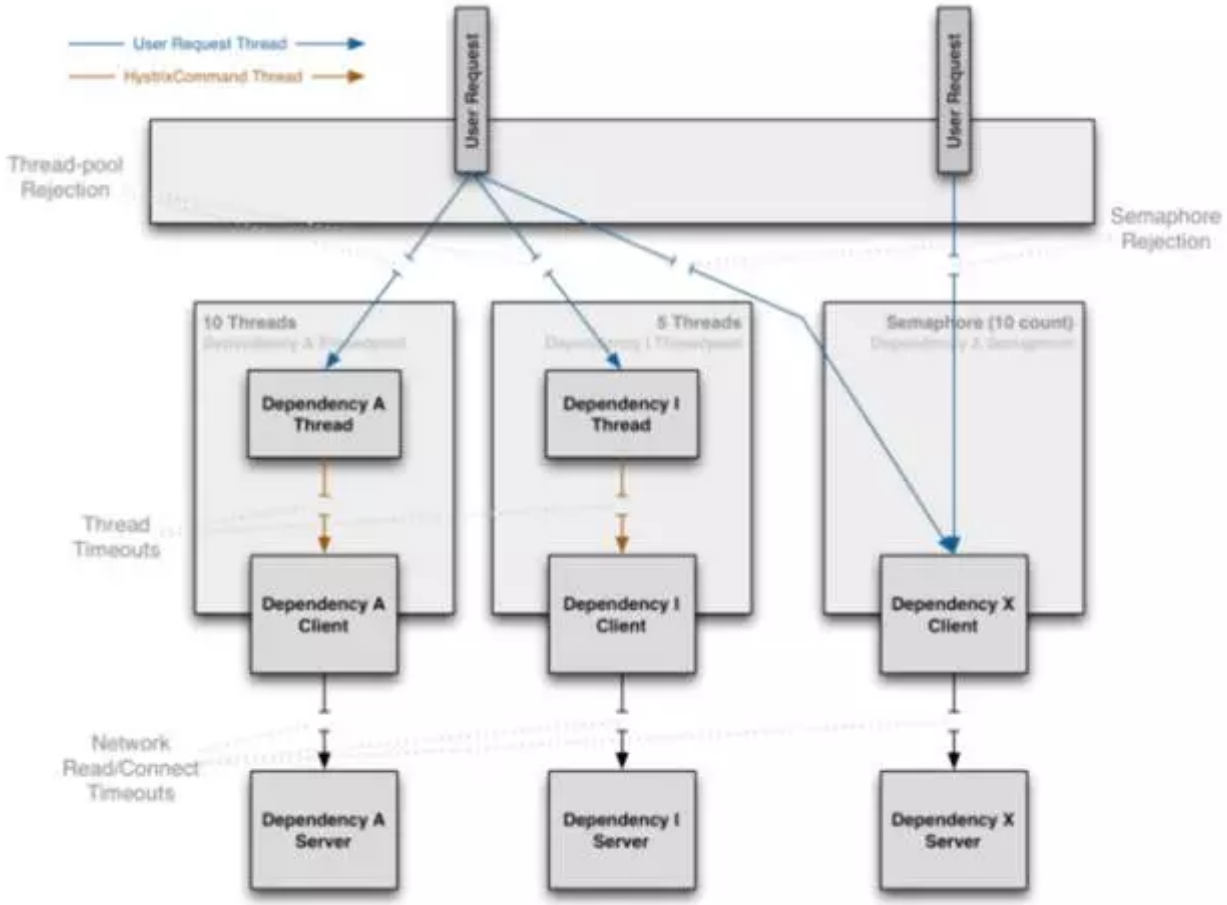
[1]:线程池的主要缺点是它增加了cpu，因为每个命令的执行涉及到排队(默认使用SynchronousQueue避免排队)，调度和上下文切换。
[2]:对使用ThreadLocal等依赖线程状态的代码增加复杂性，需要手动传递和清理线程状态。

- NOTE: Netflix公司内部认为线程隔离开销足够小，不会造成重大的成本或性能的影响。
- Netflix 内部API 每天100亿的HystrixCommand依赖请求使用线程隔，每个应用大约40多个线程池，每个线程池大约5-20个线程。

(3)信号隔离

- 信号隔离也可以用于限制并发访问，防止阻塞扩散, 与线程隔离最大不同在于执行依赖代码的线程依然是请求线程（该线程需要通过信号申请），
- 如果客户端是可信的且可以快速返回，可以使用信号隔离替换线程隔离,降低开销.
- 信号量的大小可以动态调整, 线程池大小不可以。

线程隔离与信号隔离区别如下图:



3.接入方式

本文会重点介绍基于服务化项目（thrift服务化项目）的接入方式。

3.1添加hystrix依赖

关于版本问题：由于不同版本Compile Dependencies不同，在使用过程中可以针对具体情况修改版本，具体依赖关系http://mvnrepository.com/artifact/com.netflix.hystrix/hystrix-javanica

```
<hystrix-version>1.4.22</hystrix-version>


<dependency>
    <groupId>com.netflix.hystrix</groupId>
    <artifactId>hystrix-core</artifactId>
    <version>${hystrix-version}</version>
</dependency>
<dependency>
    <groupId>com.netflix.hystrix</groupId>
    <artifactId>hystrix-metrics-event-stream</artifactId>
    <version>${hystrix-version}</version>
```

```
</dependency>
<dependency>
    <groupId>com.netflix.hystrix</groupId>
    <artifactId>hystrix-javanica</artifactId>
    <version>${hystrix-version}</version>
</dependency>
<dependency>
    <groupId>com.netflix.hystrix</groupId>
    <artifactId>hystrix-servo-metrics-publisher</artifactId>
    <version>${hystrix-version}</version>
</dependency>
<dependency>
    <groupId>com.meituan.service.us</groupId>
    <artifactId>hystrix-collector</artifactId>
    <version>1.0-SNAPSHOT</version>
</dependency>
```

3.2引入Hystrix Aspect

application-context.xml文件中：

```
<aop:aspectj-autoproxy/>
<bean id="hystrixAspect" class="com.netflix.hystrix.contrib.javanica.aop.aspectj.HystrixCommandAspect"></bean>
<context:component-scan base-package="com.***.***"/>
<context:annotation-config/>
```

注意：

1）hystrixAspect的这两行配置一定要和下面的context:component-scan放在同一个文件；
2）Hystrix依赖的一些jar需要解决冲突问题，例如guava为15.0版本。

3.3统计数据

需要注册plugin，直接从plugin中获取统计数据。

新增初始化Bean：

```
import com.meituan.service.us.collector.notifier.CustomEventNotifier;
import com.netflix.hystrix.contrib.servopublisher.HystrixServoMetricsPublisher;
```

```
import com.netflix.hystrix.strategy.HystrixPlugins;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.InitializingBean;

/**
 * Created by gaoguangchao on 16/7/1.
 */
public class HystrixMetricsInitializingBean {
    private static final Logger LOGGER = LoggerFactory.getLogger(HystrixMetricsInitializingBean.class);

    public void init() throws Exception {
        LOGGER.info("HystrixMetrics starting...");
        HystrixPlugins.getInstance().registerEventNotifier(CustomEventNotifier.getInstance());
        HystrixPlugins.getInstance().registerMetricsPublisher(HystrixServoMetricsPublisher.getInstance());
    }
}
```

application-context.xml文件中：

```
<bean id="hystrixMetricsInitializingBean" class="com.***.HystrixMetricsInitializingBean" init-method="init"/>
```

3.4添加注解

本文使用同步执行方式，因此注解及方法实现都为同步方式，如果有异步执行、反应执行的需求，可以参考：官方注解说明[https://github.com/Netflix/Hystrix/tree/master/hystrix-contrib/hystrix-javanica]

```
@HystrixCommand(groupKey = "productStockOpLog", commandKey = "addProductStockOpLog", fallbackMethod =
"addProductStockOpLogFallback",
        commandProperties = {
                @HystrixProperty(name = "execution.isolation.thread.timeoutInMilliseconds", value = "400"),//指定多久超时，单位毫秒。超时进fallback
                @HystrixProperty(name = "circuitBreaker.requestVolumeThreshold", value = "10"),//判断熔断的最少请求数，默认是10；只有在一个统计窗口内处理的请求数量达到这个阈值，才会进行熔断与否的判断
                @HystrixProperty(name = "circuitBreaker.errorThresholdPercentage", value = "10"),//判断熔断的阈值，默认值50，表示在一个统计窗口内有50%的请求处理失败，会触发熔断
        }
)
```

```
public void addProductStockOpLog(Long sku_id, Object old_value, Object new_value) throws Exception {
    if (new_value != null && !new_value.equals(old_value)) {
        doAddOpLog(null, null, sku_id, null, ProductOpType.PRODUCT_STOCK, old_value != null ? String.valueOf(old_value) :
null, String.valueOf(new_value), 0, "C端", null);
    }
}

public void addProductStockOpLogFallback(Long sku_id, Object old_value, Object new_value) throws Exception {
    LOGGER.warn("发送商品库存变更消息失败,进入Fallback,skuId:{},oldValue:{},newValue:{}", sku_id, old_value, new_value);
}
```

示例：

```
@HystrixCommand(groupKey="UserGroup", commandKey = "GetUserByIdCommand",
            commandProperties = {
                @HystrixProperty(name = "execution.isolation.thread.timeoutInMilliseconds", value = "100"),//指定多久超时，单位
毫秒。超时进fallback
                @HystrixProperty(name = "circuitBreaker.requestVolumeThreshold", value = "10"),//判断熔断的最少请求数，默认
是10；只有在一个统计窗口内处理的请求数量达到这个阈值，才会进行熔断与否的判断
                @HystrixProperty(name = "circuitBreaker.errorThresholdPercentage", value = "10"),//判断熔断的阈值，默认值
50，表示在一个统计窗口内有50%的请求处理失败，会触发熔断
            },
            threadPoolProperties = {
                    @HystrixProperty(name = "coreSize", value = "30"),
                    @HystrixProperty(name = "maxQueueSize", value = "101"),
                    @HystrixProperty(name = "keepAliveTimeMinutes", value = "2"),
                    @HystrixProperty(name = "queueSizeRejectionThreshold", value = "15"),
                    @HystrixProperty(name = "metrics.rollingStats.numBuckets", value = "12"),
                    @HystrixProperty(name = "metrics.rollingStats.timeInMilliseconds", value = "1440")
        })
```

说明：

hystrix函数必须为public，fallback函数可以为private。两者需要返回值和参数相同 详情。

hystrix函数需要放在一个service中，并且，在类本身的其他函数中调用hystrix函数，是无法达到监控的目的的。

3.5参数配置

参数说明	值	备注
groupKey	productStockOpLog	group标识，一个group使用一个线程池
commandKey	addProductStockOpLog	command标识
fallbackMethod	addProductStockOpLogFallback	fallback方法，两者需要返回值和参数相同
超时时间设置	400ms	执行策略，在THREAD模式下，达到超时时间，可以中断 For most circuits, you should try to set their timeout values close to the 99.5th percentile of a normal healthy system so they will cut off bad requests and not let them take up system resources or affect user behavior.
统计窗口（10s）内最少请求数	10	熔断策略
熔断多少秒后去尝试请求	5s	熔断策略，默认值
熔断阀值	10%	熔断策略：一个统计窗口内有10%的请求处理失败，会触发熔断
线程池coreSize	10	默认值（推荐值）.在当前项目中，需要做依赖隔离的方法为发送一条MQ消息，发送MQ消息方法的TP99耗时在1ms以下，近2周单机QPS最高值在18左右，经过灰度验证了午高峰后（当日QPS>上周末QPS），ActiveThreads<=2，rejected=0，经过压测后得出结论：线程池大小为10足以应对2000QPS，前提发送MQ消息时耗时正常（该部分为实际项目中的情况，在此不做详述）
线程池maxQueueSize	-1	即线程池队列为SynchronousQueue

4.参数说明

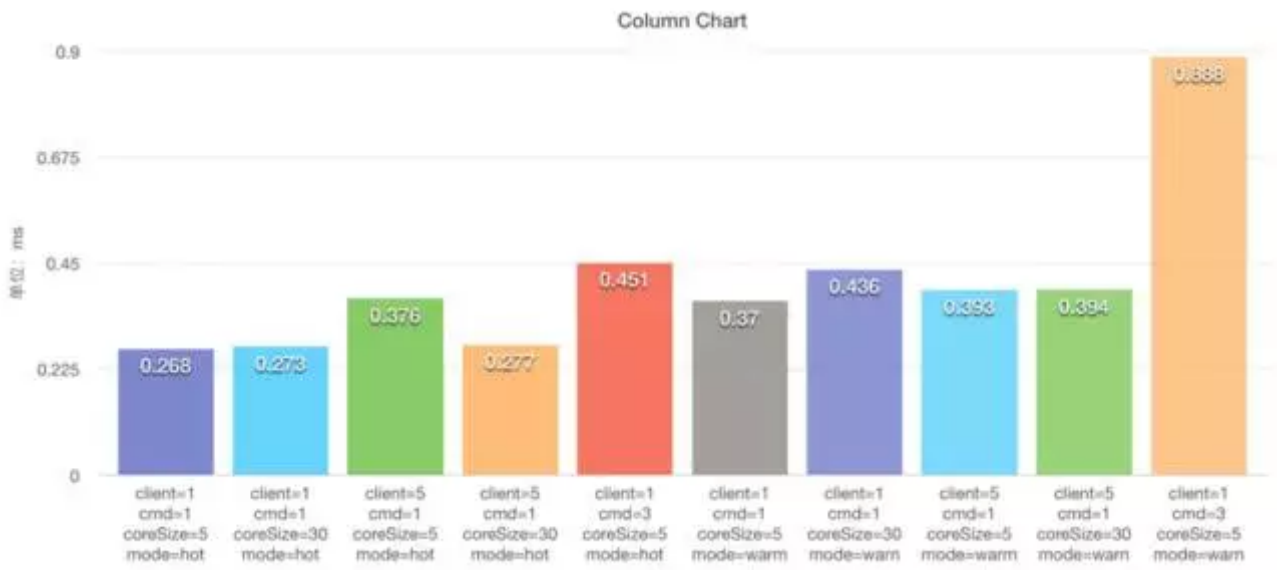
其他参数可参见 https://github.com/Netflix/Hystrix/wiki/Con

分类	参数	作用	默认值	备注
基本参数	groupKey	表示所属的group，一个group共用线程池	getClass().getSimpleName();	
基本参数	commandKey		当前执行方法名	
Execution （控制HystrixCommand.run()的执行策略）	execution.isolation.strategy	隔离策略，有THREAD和SEMAPHORE THREAD	以下几种可以使用SEMAPHORE模式： 只想控制并发度 外部的方法已经做了线程隔离 调用的是本地方法或者可靠度非常高、耗时特别小的方法（如memdis）	
Execution	execution.isolation.thread.timeoutInMilliseconds	超时时间	1000ms	默认值：1000 在THREAD模式下，达到超时时间，可以中断 在SEMAPHORE模式下，会等待执行完成后，再去判断是否超时 设置标准： 有retry，99meantime+avg meantime 没有retry，99.5meantime
Execution	execution.timeout.enabled	是否打开超时	true	
Execution	execution.isolation.thread.interruptOnTimeout	是否打开超时线程中断	true	THREAD模式有效
Execution	execution.isolation.semaphore.maxConcurrentRequests	信号量最大并发度	10	SEMAPHORE模式有效
Fallback （设置当fallback降级发生时的策略）	fallback.isolation.semaphore.maxConcurrentRequests	fallback最大并发度	10	
Fallback	fallback.enabled	fallback是否可用	true	
Circuit Breaker （配置熔断的策略）	circuitBreaker.enabled	是否开启熔断	true	
Circuit Breaker	circuitBreaker.requestVolumeThreshold	一个统计窗口内熔断触发的最小个数/10s	20	
Circuit Breaker	circuitBreaker.sleepWindowInMilliseconds	熔断多少秒后去尝试请求	5000ms	
Circuit Breaker	circuitBreaker.errorThresholdPercentage	失败率达到多少百分比后熔断	50	主要根据依赖重要性进行调整
Circuit Breaker	circuitBreaker.forceOpen	是否强制开启熔断		
Circuit Breaker	circuitBreaker.forceClosed	是否强制关闭熔断		如果是强依赖，应该设置为true
Metrics （设置关于HystrixCommand执行需要的统计信息）	metrics.rollingStats.timeInMilliseconds	设置统计滚动窗口的长度，以毫秒为单位。用于监控和熔断器。	10000	滚动窗口被分隔成桶(bucket)，并且进行滚动。 例如这个属性设置10s(10000)，一个桶是1s。
Metrics	metrics.rollingStats.numBuckets 设置统计窗口的桶数量	10	metrics.rollingStats.timeInMilliseconds必须能被这个值整除	
Metrics	metrics.rollingPercentile.enabled	设置执行时间是否被跟踪，并且计算各个百分比，50%,90%等的时间	true	
Metrics	metrics.rollingPercentile.timeInMilliseconds	设置执行时间在滚动窗口中保留时间，用来计算百分比	60000ms	

Metrics	metrics.rollingPercentile.numBuckets	设置rollingPercentile窗口的桶数量	6	metrics.rollingPercentile.timeInMilliseconds必须能被这个值整除
Metrics	metrics.rollingPercentile.bucketSize	此属性设置每个桶保存的执行时间的最大值。	100	如果设置为100，但是有500次求情，则只会计算最近的100次
Metrics	metrics.healthSnapshot.intervalInMilliseconds	采样时间间隔	500	
Request Context (设置HystrixCommand使用的HystrixRequestContext相关的属性)	requestCache.enabled	设置是否缓存请求，request-scope内缓存	true	
Request Context	requestLog.enabled	设置HystrixCommand执行和事件是否打印到HystrixRequestLog中		
ThreadPool Properties(配置HystrixCommand使用的线程池的属性)	coreSize	设置线程池的core size,这是最大的并发执行数量。	10	设置标准：coreSize = requests per second at peak when healthy × 99th percentile latency in seconds + some breathing room 大多数情况下默认的10个线程都是值得建议的
ThreadPool Properties	maxQueueSize	最大队列长度，设置BlockingQueue的最大长度	-1	默认值：-1 如果使用正数，队列将从SynchronousQueue改为LinkedBlockingQueue
ThreadPool Properties	queueSizeRejectionThreshold	设置拒绝请求的临界值	5	此属性不适用于maxQueueSize = - 1时 设置设个值的原因是maxQueueSize值运行时不能改变，我们可以通过修改这个变量动态修改允许排队的长度
ThreadPool Properties	keepAliveTimeMinutes	设置keep-live时间	1分钟	这个一般用不到因为默认corePoolSize和maxPoolSize是一样的。

5.性能测试

5.1测试情况



去除Cold状态的第一个异常点后，1-10测试场景的Hystrix平均耗时如上图所示， 可以得出结论：

1. 单个HystrixCommand的额外耗时基本稳定处于0.3ms左右，和线程池大小无关，和client数量无关；
2. hystrix的额外耗时和执行的HystrixCommand数量有关系，随着command数量增多，耗时增加，但是增量较小，没有比例关系；
3. App刚启动时，第一个请求耗时300+ms，随后请求的耗时降低至1ms以下；刚启动的一小段时间内耗时略大于Hot状态时耗时，总体不超过