之前几讲，我们用实际的业务背景给了一些可用性的问题

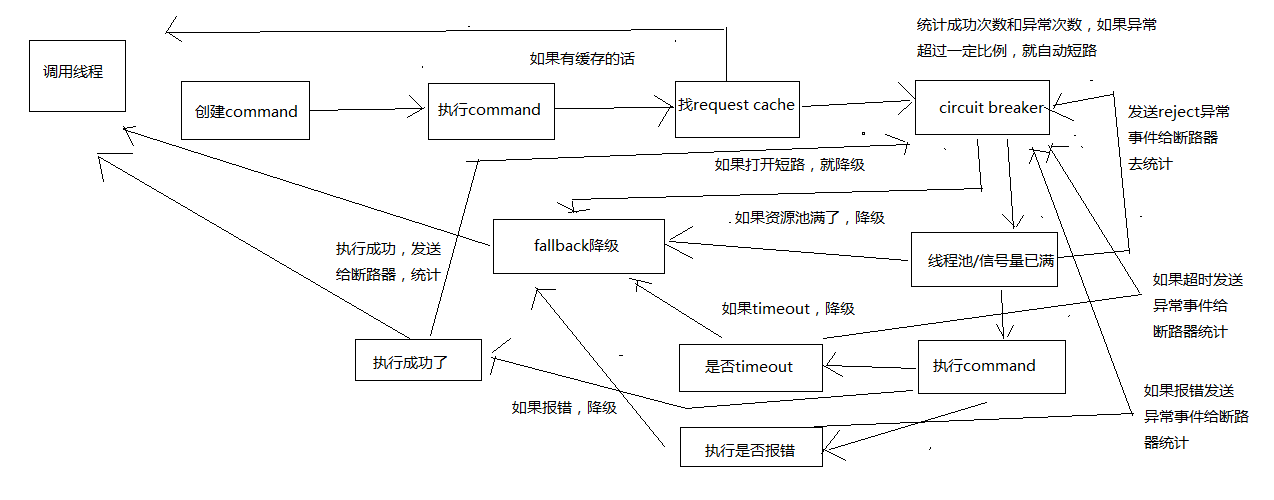
然后借着那些最最基础的可用性的问题，然后讲解了hystrix最基本的支持高可用的技术，资源隔离+限流

创建command，执行这个command，配置这个command对应的group和线程池，以及线程池/信号量的容量和大小

我们要去讲解一下，你开始执行这个command，调用了这个command的execute()方法以后，hystrix内部的底层的执行流程和步骤以及原理是什么呢？

在讲解这个流程的过程中，我们会带出来hystrix其他的一些核心以及重要的功能

画图分析整个8大步骤的流程，然后再对每个步骤进行细致的讲解



1、构建一个HystrixCommand或者HystrixObservableCommand

一个HystrixCommand或一个HystrixObservableCommand对象，代表了对某个依赖服务发起的一次请求或者调用

构造的时候，可以在构造函数中传入任何需要的参数

HystrixCommand主要用于仅仅会返回一个结果的调用

HystrixObservableCommand主要用于可能会返回多条结果的调用

HystrixCommand command = new HystrixCommand(arg1, arg2);

HystrixObservableCommand command = new HystrixObservableCommand(arg1, arg2);

2、调用command的执行方法

执行Command就可以发起一次对依赖服务的调用

要执行Command，需要在4个方法中选择其中的一个：execute()，queue()，observe()，toObservable()

其中execute()和queue()仅仅对HystrixCommand适用

execute()：调用后直接block住，属于同步调用，直到依赖服务返回单条结果，或者抛出异常

queue()：返回一个Future，属于异步调用，后面可以通过Future获取单条结果

observe()：订阅一个Observable对象，Observable代表的是依赖服务返回的结果，获取到一个那个代表结果的Observable对象的拷贝对象

toObservable()：返回一个Observable对象，如果我们订阅这个对象，就会执行command并且获取返回结果

K value = command.execute();

Future<K> fValue = command.queue();

Observable<K> ohValue = command.observe();

Observable<K> ocValue = command.toObservable();

execute()实际上会调用queue().get().queue()，接着会调用toObservable().toBlocking().toFuture()

也就是说，无论是哪种执行command的方式，最终都是依赖toObservable()去执行的

3、检查是否开启缓存

从这一步开始，进入我们的底层的运行原理啦，了解hysrix的一些更加高级的功能和特性

如果这个command开启了请求缓存，request cache，而且这个调用的结果在缓存中存在，那么直接从缓存中返回结果

4、检查是否开启了短路器

检查这个command对应的依赖服务是否开启了短路器

如果断路器被打开了，那么hystrix就不会执行这个command，而是直接去执行fallback降级机制

5、检查线程池/队列/semaphore是否已经满了

如果command对应的线程池/队列/semaphore已经满了，那么也不会执行command，而是直接去调用fallback降级机制

6、执行command

调用HystrixObservableCommand.construct()或HystrixCommand.run()来实际执行这个command

HystrixCommand.run()是返回一个单条结果，或者抛出一个异常

HystrixObservableCommand.construct()是返回一个Observable对象，可以获取多条结果

如果HystrixCommand.run()或HystrixObservableCommand.construct()的执行，超过了timeout时长的话，那么command所在的线程就会抛出一个TimeoutException

如果timeout了，也会去执行fallback降级机制，而且就不会管run()或construct()返回的值了

这里要注意的一点是，我们是不可能终止掉一个调用严重延迟的依赖服务的线程的，只能说给你抛出来一个TimeoutException，但是还是可能会因为严重延迟的调用线程占满整个线程池的

即使这个时候新来的流量都被限流了。。。

如果没有timeout的话，那么就会拿到一些调用依赖服务获取到的结果，然后hystrix会做一些logging记录和metric统计

7、短路健康检查

Hystrix会将每一个依赖服务的调用成功，失败，拒绝，超时，等事件，都会发送给circuit breaker断路器

短路器就会对调用成功/失败/拒绝/超时等事件的次数进行统计

短路器会根据这些统计次数来决定，是否要进行短路，如果打开了短路器，那么在一段时间内就会直接短路，然后如果在之后第一次检查发现调用成功了，就关闭断路器

8、调用fallback降级机制

在以下几种情况中，hystrix会调用fallback降级机制：run()或construct()抛出一个异常，短路器打开，线程池/队列/semaphore满了，command执行超时了

一般在降级机制中，都建议给出一些默认的返回值，比如静态的一些代码逻辑，或者从内存中的缓存中提取一些数据，尽量在这里不要再进行网络请求了

即使在降级中，一定要进行网络调用，也应该将那个调用放在一个HystrixCommand中，进行隔离

在HystrixCommand中，上线getFallback()方法，可以提供降级机制

在HystirxObservableCommand中，实现一个resumeWithFallback()方法，返回一个Observable对象，可以提供降级结果

如果fallback返回了结果，那么hystrix就会返回这个结果

对于HystrixCommand，会返回一个Observable对象，其中会发返回对应的结果

对于HystrixObservableCommand，会返回一个原始的Observable对象

如果没有实现fallback，或者是fallback抛出了异常，Hystrix会返回一个Observable，但是不会返回任何数据

不同的command执行方式，其fallback为空或者异常时的返回结果不同

对于execute()，直接抛出异常

对于queue()，返回一个Future，调用get()时抛出异常

对于observe()，返回一个Observable对象，但是调用subscribe()方法订阅它时，理解抛出调用者的onError方法

对于toObservable()，返回一个Observable对象，但是调用subscribe()方法订阅它时，理解抛出调用者的onError方法

9、不同的执行方式

execute()，获取一个Future.get()，然后拿到单个结果

queue()，返回一个Future

observer()，立即订阅Observable，然后启动8大执行步骤，返回一个拷贝的Observable，订阅时理解回调给你结果

toObservable()，返回一个原始的Observable，必须手动订阅才会去执行8大步骤

《亿级流量电商详情页系统的大型高并发与高可用缓存架构实战》

1、亿级流量的电商网站的商品详情页系统架构

2、大型的企业级缓存架构，支撑高并发与高可用

3、几十万QPS的高并发+99.99%高可用+1T以上的海量数据+绝对数据安全的redis集群架构

4、高并发场景下的数据库+缓存双写一致性保障方案

5、大缓存的维度化拆分方案

6、基于双层nginx部署架构的缓存命中率提升方案

7、基于kafka+spring boot+ehcache+redis+nginx+lua的多级缓存架构

8、基于zookeeper的缓存并发更新安全保障方案

9、基于storm+zookeeper的大规模缓存预热解决方案

10、基于storm+zookeeper+nginx+lua的热点缓存自动降级与恢复解决方案

11、基于hystrix的高可用缓存服务架构

12、hystrix的进阶高可用架构方案、架构性能优化以及监控运维

13、基于hystrix的大规模缓存雪崩解决方案

14、高并发场景下的缓存穿透解决方案

15、高并发场景下的缓存失效解决方案