1.1、Hystrix 内容补充

1. @SpringCloudApplication=@EnableDiscoveryClient+@SpringBootApplication+@EnableCircuitBreaker

1.2、服务熔断

//使用@HystrixCommand 进行熔断控制

```
    @HystrixCommand(
            //熔断的细节属性设置
            commandProperties={
             //每个属性都是一个HystrixProperty
            @HystrixProperty(name="execution.isolation.thread.timeoutInMilliseconds",value =
"2000")
    })
```

2s 秒中一过就熔断,

1.3、服务降级

1)服务提供者处理超时，熔断，返回错误信息

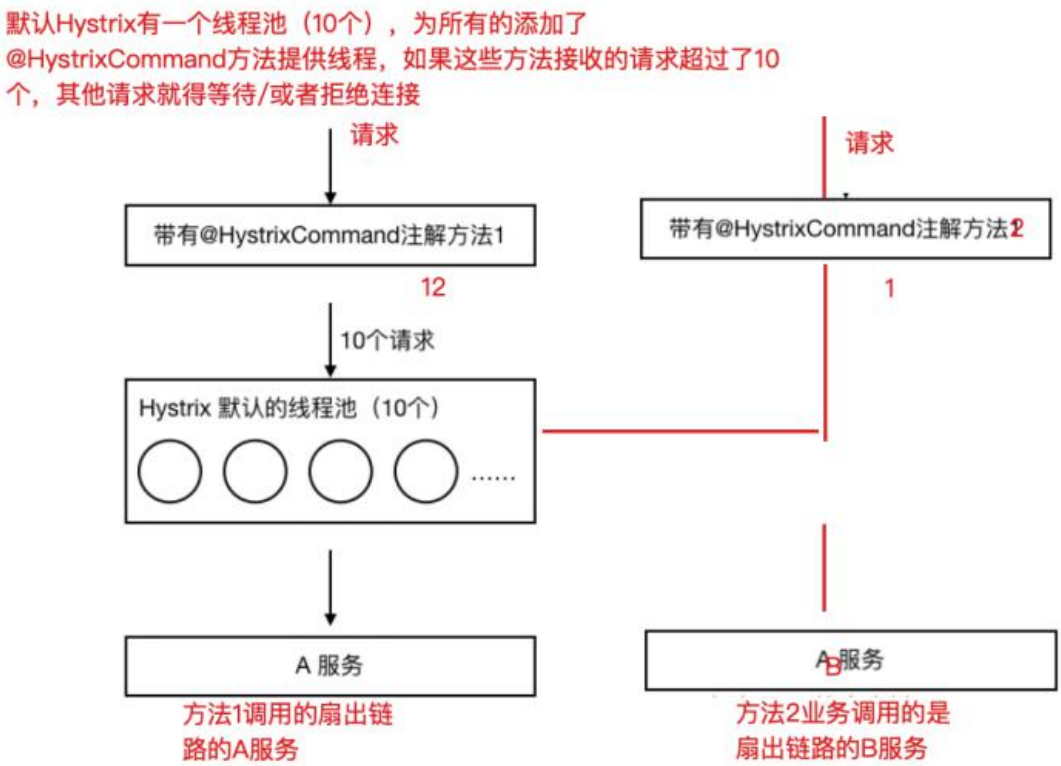
2)有可能服务提供者出现异常直接抛出异常信息

以上信息，都会返回到消费者这里，很多时候消费者服务不希望把收到的异常/错误信息再向上抛到它的上游去，因为这些信息对用户来说并不友好。

//使用@HystrixCommand 进行熔断控制--->之后开始走服务降级

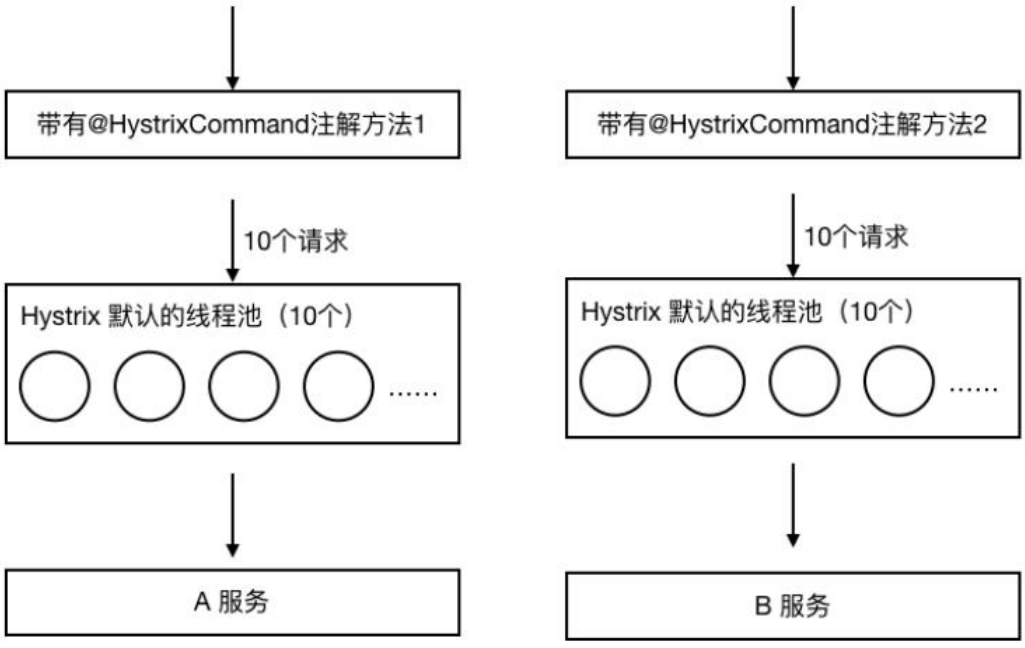
```
    @HystrixCommand(
            //熔断的细节属性设置
            commandProperties={
             //每个属性都是一个HystrixProperty
            @HystrixProperty(name="execution.isolation.thread.timeoutInMilliseconds",value = "2000")
            },
            fallbackMethod = "myFallBack" //指定服务降级方法名称，要求服务降级方法必须在当前类中，和当前
方法保持一致
    )
```

1.4、仓壁模式（线程池隔离策略）



如果不进行任何设置，所有熔断方法使用一个 Hystrix 线程池（10 个线程），那么这样的话会导致问题，这个问题并不是扇出链路微服务不可用导致的，而是我们的线程机制导致的，如果方法 A 的请求把 10 个线程都用了，方法 2 请求处理的时候压根都没法去访问 B，因为没有线程可用，并不是 B 服务不可用。

所以这里引入了 Hystrix 仓壁模式：



当然你也可以为每个标记@HystrixCommand 注解的方法的线程池中分配 1000 个线程也是可以的。

为了避免问题服务请求过多导致正常服务无法访问，Hystrix 不是采用增加线程数，而是单独的为每一个控制方法创建一个线程池的方式，这种模式叫**做“舱壁模式"，也是线程隔离的**手段。

这里可以借助于 jps 命令和 jstack 命令验证

Jps 可以查看进程信息

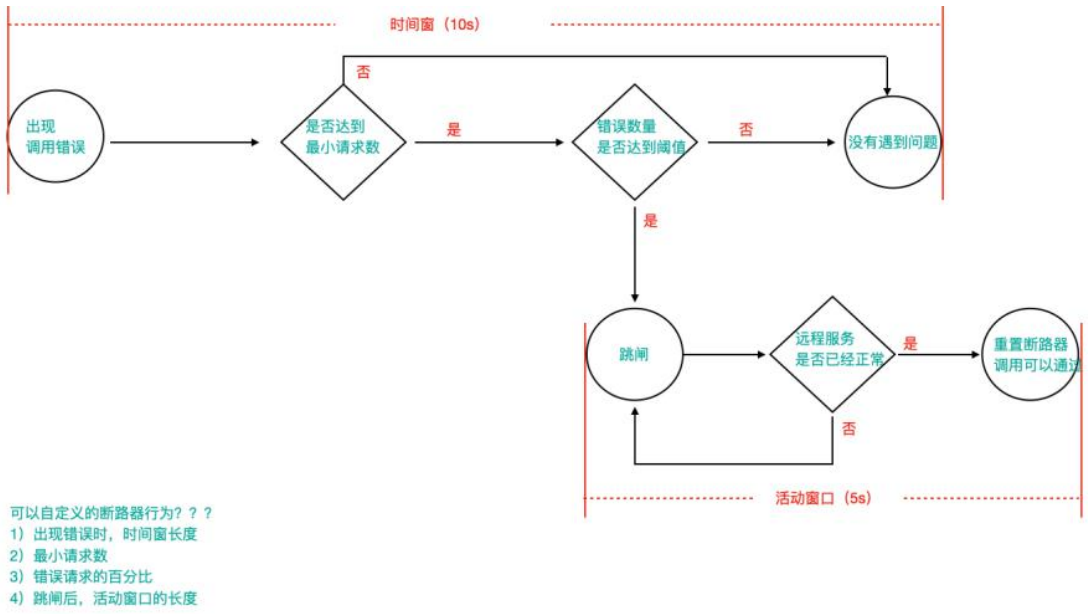
Jstack 可以查看进程中的线程信息：jstack　pid | findstr hystrix

还要借助于 postman 批量发送请求：

怎么配置？

```
// 使用@HystrixCommand注解进行熔断控制
@HystrixCommand(
        // 线程池标识，要保持唯一，不唯一的话就共用了
        threadPoolKey = "findResumeOpenStateTimeout",
        // 线程池细节属性配置
        threadPoolProperties = {
                @HystrixProperty(name="coreSize",value = "1"),// 线程数
                @HystrixProperty(name="maxQueueSize",value="20") // 等待队列长度
        },
        // commandProperties熔断的一些细节属性配置
        commandProperties = {
                // 每一个属性都是一个HystrixProperty
                @HystrixProperty(name="execution.isolation.thread.timeoutInMilliseconds",value="2000")
                }
 )
```

1.5、Hystrix 工作流程



1）当调用出现问题时，开启一个时间窗（10s）

2）在这个时间窗内，统计调用次数是否达到最小请求数？

```
如果没有达到，则重置统计信息，回到第1步
如果达到了，则统计失败的请求数占所有请求数的百分比，是否达到阈值？
  如果达到，则跳闸（不再请求对应服务）
  如果没有达到，则重置统计信息，回到第1步
```

3）如果跳闸，则会开启一个活动窗口（默认 5s），每隔 5s，Hystrix 会让一个请求通过,到达那个问题服务，看 是否调用成功，如果成功,重置断路器回到第 1 步，如果失败，回到第 3 步:

```
/**
 * 8秒钟内，请求次数达到2个，并且失败率在50%以上，就跳闸
 * 跳闸后活动窗口设置为3s
 */
@HystrixCommand(
        commandProperties = {
                //统计时间窗口长度
                @HystrixProperty(name ="metrics.rollingStats.timeInMilliseconds",value = "8000"),
                //统计时间窗口内的最小请求数
                @HystrixProperty(name ="circuitBreaker.requestVolumeThreshold",value = "2"),
                //统计时间窗口内错误数量百分比阈值
                @HystrixProperty(name ="circuitBreaker.errorThresholdPercentage",value = "50"),
                //活动窗口长度
                @HystrixProperty(name ="circuitBreaker.sleepWindowInMilliseconds",value = "3000")
        }
)
```

我们上述通过注解进行的配置也可以配置在配置文件中:

配置熔断策略:

```
hystrix:
    command:
      default:
        circuitBreaker:
 # 强制打开熔断器，如果该属性设置为true，强制断路器进入打开状态，将会拒
绝所有的请求。 默认false
          forceOpen: false
          # 触发熔断错误比例阈值，默认值50%
          errorThresholdPercentage: 50
          # 熔断后休眠时长，默认值5秒
          sleepWindowInMilliseconds: 3000
          # 熔断触发最小请求次数，默认值是20
          requestVolumeThreshold: 2
      execution:
        isolation:
          thread:
            # 熔断超时设置，默认为1秒
            timeoutInMilliseconds: 2000
```

基于 springboot 的健康检查观察跳闸状态（自动投递微服务暴露健康检查细节）

```
# springboot中暴露健康检查等断点接口
management:
  endpoints:
    web:
      exposure:
        include: "*"
  # 暴露健康接口的细节
  endpoint:
    health:
      show-details: always
```

访问健康检查接口：http://localhost:8090/actuator/health

hystrix 正常工作状态:

跳闸状态

活动窗口内自我修复

1.6、超时注意点

针对超时这一点，当前有两个超时时间设置（Feign/hystrix），熔断的时候是根据这两个时间的最小值来进行的，即处理时长超过最短的那个超时时间了就熔断进入回退降级逻辑。

1.7、Feign 对请求压缩和响应压缩的支持

Feign 支持对请求和响应进行GZIP 压缩，以减少通信过程中的性能损耗。通过下面的参数 即可开启请求与响应的压缩功能:

```
feign:
  compression:
    request:   # 开启请求压缩
      enabled: true
      mime-types: text/html,application/xml,application/json # 设置压缩的数据类型，此处也是默认值
      min-request-size: 2048 # 设置触发压缩的大小下限，此处也是默认值
    response:   # 开启响应压缩
      enabled: true
```

1.8、Ribbon 知识点补充

```
#针对的被调用方微服务名称,不加就是全局生效
atguigu-service-resume:
  ribbon:
    #请求连接超时时间
    ConnectTimeout: 2000
    #请求处理超时时间
    ##############################################Feign超时时长设置
    ReadTimeout: 3000
    #对所有操作都进行重试
    OkToRetryOnAllOperations: true
    ####根据如上配置，当访问到故障请求的时候，它会再尝试访问一次当前实例
    （次数由MaxAutoRetries配置），
    ####如果不行，就换一个实例进行访问，如果还不行，再换一次实例访问（更换次数由MaxAutoRetriesNextServer配
置），
    ####如果依然不行，返回失败信息。
    MaxAutoRetries: 0 #对当前选中实例重试次数，不包括第一次调用
    MaxAutoRetriesNextServer: 0 #切换实例的重试次数
    NFLoadBalancerRuleClassName: com.netflix.loadbalancer.RoundRobinRule #负载策略调整
```