题目标签

- 题目难度：一般
- 知识点标签：feign超时，定时任务超时
- 课程时长：20分钟

题目描述

面试中的问题：定时任务和feign的超时如何优化呢？

案情回顾

业务定时器应用半夜经常会触发熔断异常的告警邮件

根据邮件提示的类找到归纳以下表格

编号	报错方法	接口所属应用	所属定时任务类
A	VipTradeReportFeignService#getShopTradeReportByDate	pinka-mod-stats	ShopOrderSturctureTask
B	VipMemberStatsFeignService#statMemberRecord	pinka-mod-stats	MemberStatTask
C	VipPartnerWalletFeignService.handlePartnerWithdraw	pinka-mod-customer	PartnerWithdrawCheckTask
D	VipWeixinBabyActivityFeignService.getBabyActivityNoticePage	pinka-mod-weixin	VipWeixinBabyNoticeTask

~~以上AD都是在一个分布式定时器事件处理应用(pinka-mod-scheduler)中对外的feign微服务调用产生的，相当于4类任务，每类都会调1次或多次外部feign微服务接口，而其中的AD~~接口发生了问题

其中A和B都是如下形式的异常

```
com.netflix.hystrix.exception.HystrixTimeoutException
at
com.netflix.hystrix.AbstractCommand$HystrixObservableTimeoutOperator$1$1.run(Abs
tractCommand.java:1154)
at
com.netflix.hystrix.strategy.concurrency.HystrixContextRunnable$1.call(HystrixCo
ntextRunnable.java:45)
at
com.netflix.hystrix.strategy.concurrency.HystrixContextRunnable$1.call(HystrixCo
ntextRunnable.java:41)
...
```

而C和D都是如下形式的异常

```
feign.RetryableException: 10.13.32.111:56000 failed to respond executing POST
http://pinka-mod-customer/vip/partner/wallet/handlePartnerWithdraw
at feign.FeignException.errorExecuting(FeignException.java:67)
at
feign.SynchronousMethodHandler.executeAndDecode(SynchronousMethodHandler.java:10
4)
at feign.SynchronousMethodHandler.invoke(SynchronousMethodHandler.java:76)
at
feign.hystrix.HystrixInvocationHandler$1.run(HystrixInvocationHandler.java:114)
...
Caused by: org.apache.http.NoHttpResponseException: 10.13.32.111:56000 failed to
respond
at
org.apache.http.impl.conn.DefaultHttpResponseParser.parseHead(DefaultHttpRespons
eParser.java:141)
at
org.apache.http.impl.conn.DefaultHttpResponseParser.parseHead(DefaultHttpRespons
eParser.java:56)
at
org.apache.http.impl.io.AbstractMessageParser.parse(AbstractMessageParser.java:2
59)
...
```

追查

HystrixTimeoutException超时异常

A与B的异常几乎每天都发生，且提示很明显，是Hystrix中设置了超时时间（目前为10s），并且执行超时导致的。为何会超时？去接口实现发现是有for循环场景的耗时逻辑

```
ApiResult<Boolean> getShopTradeReportByDate(@RequestBody VipShopOrderStructureDto
                                                    vipShopOrderStructureDto){
    ApiResult<Boolean> apiResult=ApiResult.build();
    try{
        List<VipShopOrderStrucAccountDto> vipShopOrderStrucAccountDtos=new ArrayList<~>();
        List<VipShopOrderStructureDto> vipShopOrderStructureDtos=
                    vipShopTradeReportService.queryShopOrderStructureDetails(vipShopOrderStructureDto);
        Map<Long, VipSearchKeywordsDto> idsMap = Maps.newHashMap();
        idsMap = vipSearchKeywordsService.getIdMap();
        if(CollectionUtils.isNotEmpty(vipShopOrderStructureDtos)){
            for(VipShopOrderStructureDto structureDto:vipShopOrderStructureDtos){
                VipShopOrderStrucAccountDto vipShopOrderStrucAccountDto=new VipShopOrderStrucAccountDto();
                vipShopOrderStrucAccountDto.setTradeDate(vipShopOrderStructureDto.getSummaryDate());
```

通过Kibana日志系统查历史执行耗时，也可以发现都基本>13s，所以这类异常基本确因

Time	text
September 22nd 2020, 02:00:14.334	[d822fa5e5540475b8f7c5702d75ced89]根据日期获取商户订单数据|执行结束|结果: {"code":"0","entry":true,"success":true} |耗时is13292msec
September 22nd 2020, 02:00:01.042	[d822fa5e5540475b8f7c5702d75ced89]根据日期获取商户订单数据|执行开始|参数: [{"activityQueryType":-1,"cmQueryType":-1,"firstIndustryType":-1,"secondIndustryType":-1,"summaryDate":"2020-09-21"}]
September 21st 2020, 02:00:17.209	[61ee7b6f16de4c6f95e139d9794c911f]根据日期获取商户订单数据|执行结束|结果: {"code":"0","entry":true,"success":true} |耗时is15313msec
September 21st 2020, 02:00:01.895	[61ee7b6f16de4c6f95e139d9794c911f]根据日期获取商户订单数据|执行开始|参数: [{"activityQueryType":-1,"cmQueryType":-1,"firstIndustryType":-1,"secondIndustryType":-1,"summaryDate":"2020-09-20"}]
September 20th 2020, 02:00:16.723	[2e7d7e83efc24853a2e727f486ec6987]根据日期获取商户订单数据|执行结束|结果: {"code":"0","entry":true,"success":true} |耗时is15966msec
September 20th 2020, 02:00:00.756	[2e7d7e83efc24853a2e727f486ec6987]根据日期获取商户订单数据|执行开始|参数: [{"activityQueryType":-1,"cmQueryType":-1,"firstIndustryType":-1,"secondIndustryType":-1,"summaryDate":"2020-09-19"}]
September 19th 2020, 02:00:16.495	[1ca6dde6b5544e9c965447e72a891622]根据日期获取商户订单数据|执行结束|结果: {"code":"0","entry":true,"success":true} |耗时is15035msec
September 19th 2020, 02:00:01.459	[1ca6dde6b5544e9c965447e72a891622]根据日期获取商户订单数据|执行开始|参数: [{"activityQueryType":-1,"cmQueryType":-1,"firstIndustryType":-1,"secondIndustryType":-1,"summaryDate":"2020-09-18"}]
September 18th 2020, 02:00:14.816	[82685aae7efa406682de03ce610a5adc]根据日期获取商户订单数据|执行结束|结果: {"code":"0","entry":true,"success":true} |耗时is13540msec

解决与思考

这其实是一个很典型场景，定时器任务执行并且处理逻辑是在另外一个微服务中，而处理逻辑属于复杂耗时，怎么办？

A. 增加超时时间，这是个粗暴的思路，因为设长了可能导致更大的问题，因为超时本来就是为了fastfail，设20s那之后可能还会遇到要30s甚至更久的场景。所以这个方案不能用在所有调用的公共默认超时时间上；

但是可以考虑用在某些接口上，比如VipTradeReportFeignService#getShopTradeReportByDate接口评估正常耗时就是要15s以上，那就单独为其设置。相关配置方式：

```
#默认公共超时
hystrix.command.default.execution.isolation.thread.timeoutInMilliseconds=10000
#单独为某个feign接口设置超时
hystrix.command."FeignService#sayHello(String)".execution.isolation.thread.timeoutInMilliseconds=15000
```

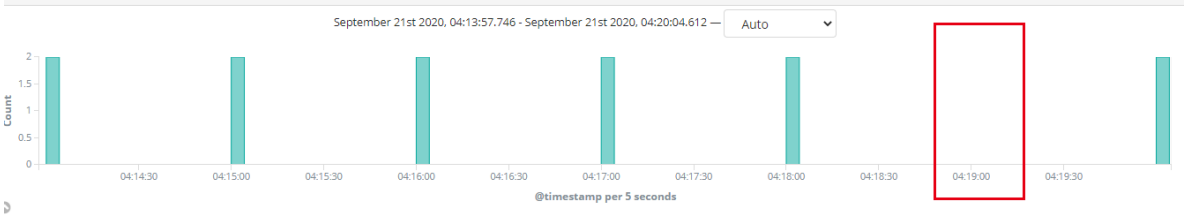
B. 优化接口提供方的逻辑执行时间。比如上述VipTradeReportFeignService#getShopTradeReportByDate中的for循环，能否移到接口调用方，相当于接口提供方每次只执行for循环的1次操作。说白了就是确保接口返回要在超时时间内，这也符合微服务接口的设计原则。

C. 还有种思路是接口处理异步化，即接口提供方立刻返回，自己再用异步线程去处理最终逻辑。但是单纯这样会导致任务执行不可靠，即接口返回成功不代表真实一定执行成功了，如果此时接口提供方重启或异常导致耗时的异步逻辑执行一半就中断了，反而无法利用分布式定时任务调度的机制去重试执行等。所以使用此思路时，接口立刻返回但不能立刻将任务也作为成功执行完毕，需要配合一些异步通知机制，即接口提供方真实成功结束耗时操作，通知给接口调用方，接口调用方再将任务作为成功返回上报。

feign.RetryableException failed to respond executing 异常

这是C和D的异常，是随机低频告警。看字面意思是接口请求无响应，再结合邮件中的“熔断”字眼就自然推测是接口提供应用的问题了（事后证明被“熔断”字眼坑了）。所以去追查接口所属应用pinka-mod-customer在告警前后的监控指标，发现tcp连接、CPU、内存、网络流量表现都没什么异常状况。另外如果是熔断，那此接口必然得是调用失败多次呀，而每次定时任务对该接口调用却只有一次。

这时候查看接口提供方Controller层日志，发现告警时刻确实提供方没进入controller处理。



由此推测，提供方应用本身并无问题。而查看调用方应用日志和性能指标，在那个时刻也无异常情况，还在不断向其他应用调用产生日志。再结合这个异常日志，推测原因是**由于调用方与提供方某次调用的网络闪断导致的**（所以是随机低频）。

但为何会开启“熔断”，这个还无法解释。此时去追查邮件告警的代码源头，告警本质是通过重写了openfeign官方的HystrixCommand创建逻辑中的getFallback方法实现的，即进入fallback逻辑就会发邮件

```
HystrixCommand<Object> hystrixCommand = new HystrixCommand<Object>(setterMethodMap.get(method)) {
  @Override
  protected Object run() throws Exception {
    try {
      //判断对应的方法是否进入强制熔断
      ForceFusingKit.checkForceFusing(method);

      MethodHandler handler = HystrixInvocationHandler.this.dispatch.get(method);
      return handler.invoke(args);
    } catch (ForceFusingException e){
      logger.info("{}进入强制熔断",ForceFusingKit.getMethodFullNameFromCache(method));
      throw e;
    } catch (Exception e) {
      throw e;
    } catch (Throwable t) {
      throw (Error) t;
    }
  }

  @Override
  protected Object getFallback() {
    sendExceptionEmail(new ExceptionInfo(getExecutionException(),method));

    // 方法全路径名
    String methodFullName = Joiner.on("#").join(method.getDeclaringClass().getName(), method.getName());
```

此时真相大白了，其实只是进了fallback降级，并不代表开启熔断，比如在HystrixCommand的run中抛出异常会进fallback，run执行超时会进fallback，熔断也会进fallback。即A~D这些异常，虽然邮件写的是熔断，但其实都没开启熔断，而只是进了fallback降级！

所以feign.RetryableException failed to respond executing这个其实只是一次偶然的调用失败进了fallback而已，并没之前猜想的那么复杂。

解决与思考

邮件告警逻辑自然是要修改，区分熔断和降级。如果要判断熔断，可以用如下方法

```
protected Object getFallback() {
        if (this.isCircuitBreakerOpen()) {
          // 熔断告警方式
          sendExceptionEmail(...);
        }else{
          // 非熔断降级告警，如果无需告警也可不写
          sendExceptionEmail(...);
        }
                ....
}
```