答疑文档【Go 技术专家 1 期】

提问注意事项：

1. 提具体的问题，问题越具体，越能得到准确的答案。
2. 用疑问句提问，而不是仅仅用陈述句描述你碰到的情况，也不要用反义疑问句提问。

问题发出之前，检查一下是否补充了足够的背景信息，以便别人能清楚地理解你所提出的问题。

第二次直播答疑

下面补充问题

问题一

问题二

第一次直播答疑

答疑时间：4 月 20 日 周四 晚 20:00，回放第二个工作日上传

直播答疑地址：#腾讯会议：876-6359-3845

绘图.pdf

问题 1：技术总监面，最困难的是什么？

- 如果你面技术岗 => 优先回答硬技术问题
  - 带一点沟通协调内容的硬技术问题。
    - 把系统核心接口的响应时间 99 线都降低到 200ms
      - 措施 ABCD
      - 推动下游接口优化性能
      - 推动运维团队优化 MySQL/Redis/Kafka 等依赖的性能
    - 全方面提高可观测性
      - 技术选型、环境准备
      - 推动别的团队接入可观测性
- 如果你面偏管理的岗位 => 可以考虑回答团队管理、项目管理、沟通方面的内容
- 正常为了应付总监面，HR 面，你都要准备一个纯软实力的案例，而后看总监他的倾向，选择是否使用
  - 体现你 owner 意识
  - 体现你推进项目的
  - 体现你沟通能力的
  - 体现你技术影响力的

问题 2：项目一定要有数据吗？

- 是的。而且数据不要太难看。
- 让 DeepSeek 给你编数据

问题 3：GO 分库分表用什么框架

- 代理类的都可以用：独立部署的代理类的都可以用
  - shardingsphere
  - mycat
- vitess 也可以试试
- kindshard
- 手搓：如果你不是要造一个通用的分库分表解决方案，搓起来还是很快的。

- 什么时候分表、什么时候分库？（第四周会讨论这个问题）
  - TPS => 决定分库，而且物理分库（分集群）
    - 分多少库 => 至少要按 TPS/QPS 来算
  - 分表 => 读写分离都瓶颈了，数据量太大了 => 分表

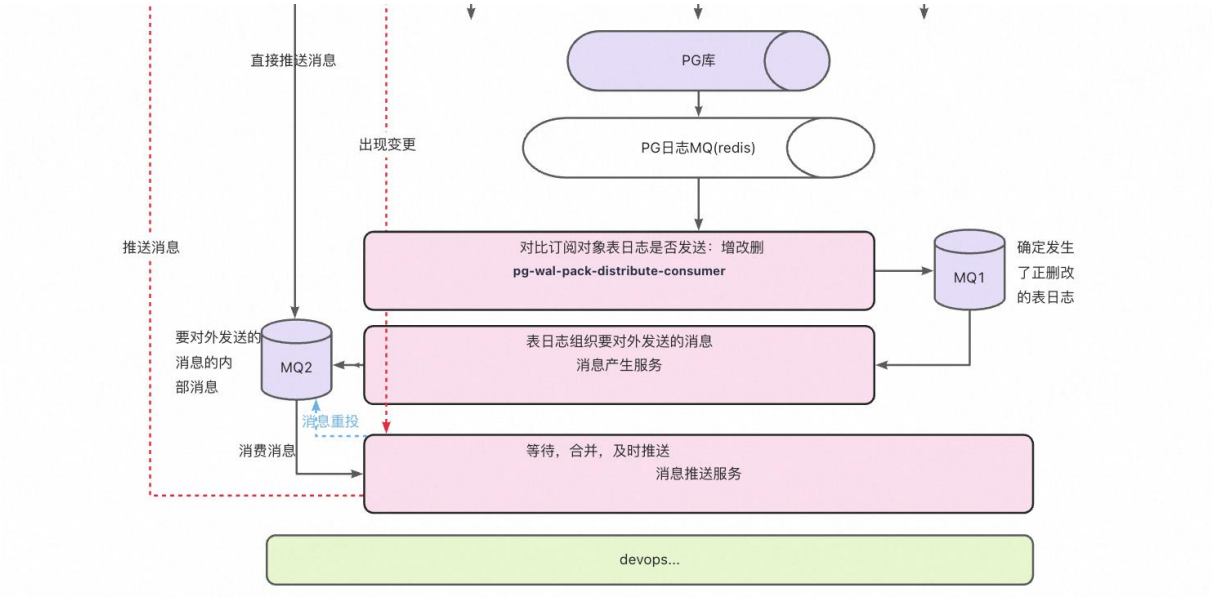
    - MySQL 8.0 测试结果-阿里云帮助中心
    - 市面上的标准：2000w 就要分表（我个人存疑），但是这个标准来算：
        - 分完之后，你要确保你的表不超过 1000w，留出 1000w 来应付数据增长
            - 如果要是数据增长很快，应该分完之后，不超过 500w
        - 分完之后，五年内表不会超过 1500w，也就是不会触发再次分库分表
            - 你要是配合数据归档，比如说 MySQL 只保留两年数据，两年数据绝对不会导致单表超过 1500w，你就一劳永逸了
        - 它没考虑 TPS，也没考虑 QPS，还没考虑宽表问题
    - 逻辑分库有效果，但是效果不佳。
        - 比如说你只有一个主从集群，然后你在上面创建 user_db_0，user_db_1，有效果，但是效果不佳
- 云服务商的基准测试：你记住一些关键产品的数据，防着面试官装逼打你脸
    - 性能测试
    - 可用性 SLA

使用 go 如果想设计一个微内核架构的系统，其他业务方通过开发插件的方式使用这个系统，插件机制有啥好的实践没？

- 插件不是一个好的实践
    - 版本问题就特别麻烦，你的主体的依赖的各个版本，要和插件的依赖的各个版本保持完全一致，不然就各种问题
- 提供插件模板，固定 go.mod；或者要对 go.mod 文件进行管理，确保各个依赖版本和系统主体保持一致
- 提供 .so 的编译环境 => 提供固定的编译 docker
- 炫技 > 实践意义的
    - 除非不能接受 gRPC 的性能损耗，不然优先考虑用微服务来接入不同的实现
- Redis 怎么做分布式重入锁？
    - 最关键的就是 key 对应一个哈希结构（也可以用两个 key），有 value 和 cnt 两个字段，cnt 是加锁次数。
        - key => int64 的数字，然后高 32 位是你的值，用来区分加锁人的值，低 32 位就是你的加锁次数（lua 脚本操作）
    - 释放锁，就是 cnt -1，减到 0 就是彻底释放了，删掉这个 key
    - 使用 lua 脚本
    - 不要支持重入锁 => **不要给你的同事写垃圾代码的机会**。

问题 4:

实现一个消息推送方案：开放给第三方应用订阅系统内的比如订单、回款单等单据的增改删消息；



MQ1 内部消息：表日志

MQ2 推送消息：有两种消息

消息 1：{

"list":[{"biz_type":"服务.模块.动作..","data":{},"biz_time":"业务消息时间戳"}],

"metadata":{

"timestamp":1742871619,

"client_id":"企业 ID",

"version":"消息版本（待定）",

"source":{"service":"建议定义比较唯一区别消息的什么功能的，也方便查询异常消息。必填"},

"trace":{"trace_id":"跟踪 ID"},

"retry_policy":{"retry_count":1,"max_retries":3},

"security":{"method":"AES-256-GCM"},

"callback":{"processor":1,"url":"xxxx","method":"POST","headers":{"xxx":"value"},

"custome":{"用户定义回传字段":"用户定义回传值"},"timeout":3}}

}

消息 2：{

"message_id":"消息唯一 ID",

"data":{

"custome":{"用户定义字段":"用户定义字段值"},

"biz":[{"biz_type":"","biz_time":1234343,"biz_data":{"业务消息，消息版本版本号整个 data 会被加密发送"}]

},

"data_version":"消息版本号",

"security_verison":"加密版本"

}

一般 MQ2 基本都是【消息 1】，但是有些发送失败的【消息 2】重投入队列，这些【消息 2】不会被”等待（等 5 秒或者 100 条）“，会直接发送就是，我们只等待合并【消息 1】

合并消息逻辑：根据【消息 1】的 client_id 认为相同企业的消息进行合并 list 为【消息 2】的 data。

由于 MQ1 也会存在一定被合并消息的情况，但是消费者处理很快时，还是会存在消息被一条一条生成【消息 1】，所以我也想在【消息推送服务】上进行”等待“，减少对第三方接口的调用。

这里有疑问：

1、【消息推送服务】的”等待“是否有必要？

a) 没有瓶颈问题，没多少必要的

b) 按照 client_id 进行分组，它不一定效果好

i) 如果 client_id 相同的很容易产生一批数据，那么效果就会好

ii) 提高效果，【消息 1】 按照 client_id 来选分区

2、”等待“一般如何实现这个？（我这边暂时想到共享内存和定时器）

```
func (b *BatchHandler[T]) ConsumeClaimV1(session
sarama.ConsumerGroupSession, claim sarama.ConsumerGroupClaim)
error {
    msgs := claim.Messages()
    const batchSize = 10
    for {
        batchMap := mapx.NewMultiBuiltinMap[string,
*sarama.ConsumerMessage](10)
        //batch := make([]*sarama.ConsumerMessage, 0, batchSize)
        ts := make([]T, 0, batchSize)
        ctx, cancel := context.WithTimeout(context.Background(),
time.Second)
        var done = false
        for i := 0; i < batchSize && !done; i++ {
            select {
            case <-ctx.Done():
                // 要注意超时控制
                // 超时了
                done = true
            case msg, ok := <-msgs:
                if !ok {
                    cancel()
                    return nil
                }
                // 解决有序性，开 channel，同一个业务发到同一个 channel
                //batch = append(batch, msg)
                _ = batchMap.Put(msg.ClientId, msg)
                vals, _ := batchMap.Get(msg.ClientID)
                if len(vals) >= batchSize {
                    // 发出去，然后删掉这个 key
                }
            }
        }
        cancel()
        // 凑够了一批，然后你就处理
        err := b.fn(batch, ts)
        if err != nil {
            b.l.Error("处理消息失败",
                // 把真个 msgs 都记录下来
                logger.Error(err))
        }
        for _, msg := range batch {
```

问题 5：

1. 调用【三方接口】和自己的【数据库事务】，有没有什么最简单的方式保证原子性呢？我现在的做法是【数据库事务】先执行，最后执行调用【三方接口】。但是有时候数据库事务需要三方接口的返回值，就很难受了，想问一下有没有什么好的 9 办法。
    a. 没有，想 P 吃；
    b. 首先，规避在数据库事务中调用三方接口，不然就是长事务问题。
        i. 如果数据库事务需要三方接口的数据，就只能先调用三方接口

问题 6：

1. 我现在主要是在做一个支付项目，业务复杂度其实很高（换汇，换汇加点，手续费，结算对账）尤其是结算和对账，还挺复杂的，但是技术上的复杂度还好，我们现在还是单体架构，公司算是中型公司把，想问下之后应该去搞一些什么对自己的帮助最大呢？大明老师之前也是支付公司把，想听听你的意见。
    a. 分布式事务解决方案：基于事件驱动的 SAGA 的框架
        i. 你要抽象出来一个统一的解决支付相关场景的这种分布式事务解决方案
    b. 数据一致性的问题
    c. 对账（业务复杂度）：但是面相关的业务会有优势
    d. 支付：高可用，换汇应该没有高并发

问题 7: 工作中，经常遇到运营人员要导出 execl 数据， 一般导出半年，一年的数据，最少半年数据， 一般订单半年数据几十万，导出 execl 我想做成一页页的导出，像下载一样慢慢累积，前端说实现不了，现在是后端先生成 execl 文件，前端拿到文件地址再下载，这种方式占有内存比较大.

- 分批导出数据到内存
- 分批写入到文件/上传到对象存储
    - 对象存储是支持分片上传的
    - 先 w 取 1000 条（放入到了 excel 中），上传到对象存储
- 分批写入到本地文件，调用对象存储接口统一上传