课程总结

概述

Go 进阶训练营已经结束了。在这 3 个多月的学习中，我有了非常多的收获。

我将课程的内容分为四个部分：一是在工作中困扰我的问题的解决方案：包括 异常处理、并发编程、工程化实践。学完这些章节后，我随即将其实践和落地到了工作里，对项目代码进行了深度优化；二是特定场景下的技术方案：包括评论系统架构设计，播放历史架构设计。这些章节虽然与我的工作内容不是很接近，但是毛老师在课程中提到的很多设计方案，架构思维都令我醍醐灌顶；三是介绍了中间件，包括缓存和数据库中间件、网络编程中间件、日志，指标，链路追踪中间件。这些中间件过去于我而言只是一套基础设施，我知其然但不知其所以然。课程学习了之后，我了解到了业界的主流架构，将毛老师介绍的技术栈对比了公司目前采用的技术栈，分析出哪些可以用到工作中，哪些已经落地，哪些为什么没有被我们所采用。在之后相关的技术选型时，我有了更多的思路和选择。最后是一些高于我实际工作的内容，包括分布式架构，消息队列，和 Runtime。这几个章节的内容说实话是我目前的工作覆盖不到的，尤其是多活，dns相关内容。但是通过这部分的学习，我的眼界得到了拓展。下面我会逐个详细的介绍下，我是如何将学习内容与工作相结合的。

微服务

在学习这个章节以前，由于目前接触的服务架构仍是毛老师所说的巨石架构，所以对微服务的了解不是很深入。学习了微服务的概念以后，我开始反思工作中的服务设计，对旧的巨石架构进行了简单的拆分。比如目

前我们的支付业务和账号业务耦合较深，完全可以分拆为两套服务；而账号服务本身也过于庞大，完全可以再进一步细分，比如权限校验微服务，账号属性微服务等等。还有我们的前端与后端之间，并不存在毛老师所说的 BFF 层，前端需要展示的所有数据仍然是通过许多个网络调用完成的，速度上有非常大的优化空间。由于工作重心不在此，所以不能很好的实践这些理念，但毛老师所讲的内容确实给我带来了新意，让我理解到业界有更好的架构设计。关于服务发现和多集群多租户，工作中其实已经开始用到了，只是公司采用的方案和毛老师所说不太一致，但核心的理念是一样的。我们使用 consul 完成服务注册和发现已经健康检查，部署了多个环境，多个分支的服务，并在请求中使用特殊标识实现了类似多租户的功能，便于测试和灰度发布。

异常处理

这个章节是直接吸引我参加训练营的原因。在编写 go 代码的时候，处理 error 确实是一个困扰我的问题。以前的处理方式会产生大量的冗余日志，日志结构也很混乱，排查效率收到了很大的影响。在了解到课程中对这块有介绍时，我便报名了课程。毛老师介绍了多个 error 的处理方式，对比了各自的优劣，最后总结了一套完整的实践方案。我学习完之后，立马在工作代码中进行了实践：设计基础的 sentinel error，牢牢实践“能解决的异常就降级，不能解决的才上抛” 等等。改良完之后，整个代码的异常处理变得十分清晰而简洁。

并发编程

并发是 go 体系下非常基础又与其他语言相差很大的部分。我个人觉得 go 与其他语言差别最大的地方就在于此。我本人是 java 转 go 的，在初次接触到 go 的并发设计时着实被他的设计惊艳到了。go 的并发写着很轻松，但是调试起来会有些不太直观，有些问题可能需要稍微绕一下才能想到。毛老师不仅介绍了基础的使用方法，还深入到了源码甚至编译器环节，从语法到内存，深入分析了各种竞争的情况，抛出了很多过去我不理解或者不熟悉的概念，比如 happen-before，重排等等。而后面提到的 errgroup 也被我实践到了工作中，参照了 kratos 的设计，将作业里的服务启动与信号处理实践到业务代码里，解决了这一部分的问题。

工程化实践

在学习这一章节前，我曾被专门安排过相关的工作：为小组提供 go 工程化的最佳实践。毛老师提到的一些文章我也有参考过，所以整章学习起来非常的亲切，感悟更深。但老师毕竟是老师，考虑的很多问题都是我没有考虑到或者有更好的解决方案的。老师以 kratos 为例，介绍了 kratos 的代码工程化的思路。我对比了我自己探索的最佳实践，发现在 /internal 下的分层考虑还不够深入，而 api 目录，gRPC 和 wire 的使用也是我不曾了解到的。这一部分的工作我有在业务代码中实践，但由于掌握的还不够完善，没有能落地到生产环境。后续我也会加强 gRPC 和 wire 的学习，结合这章的内容，对业务代码进行改造。

可用性设计

这一章节毛老师提出的所有概念，都是与工作息息相关的。我对这些业务可用性概念虽有有一定的理解，但脑海里并没有一个完整而清晰的理论图谱和解决方案。这一章学习下来，我反思我工作中很多方面都有可优化的空间，比如服务降级这一块我遇到的一些服务都是比较欠缺的，某些地方甚至都不存在降级的设计，坏了就是坏了。但是实际上由于业务逻辑比较简单，服务的流量也比较稳定，在过去的日子中确实还没有产生过大范围的问题。理论和实践的差距不禁让我反思，学习理论的时候一定要考虑周全，要学习考虑最深刻，可用性最好，兼容性最强的设计；但实际设计和开发服务时要评估对服务的可用性要求以及开发周期，不一定所有的理论和异常处理都要落地。

评论架构和播放历史架构设计

毛老师在这两节内容里，结合了 Bilibili 的业务场景，介绍了如何去设计这两块核心功能的架构。这两部分的内容对我来说是既实际又抽象的，实际是因为他是两个业务模块的实际架构设计，抽象是因为这两个业务场景离我当下的工作比较远，没有做过类似的设计和思考。这两章的学习让我体会到，大厂在面对大体量的核心业务的时候是如何考虑业务场景，如何设计架构保证服务可用性的。并且对比了公司项目中的类似的设计思路，重新审视了使用到缓存的逻辑，修复了一些潜在的问题。

缓存，数据库

这一章的内容是存储，但是都是比较高级的分布式存储。我了解到的项目中还没有使用到分布式存储的业务。我回顾了公司支付项目中的本地存储和缓存的使用，对比了本地存储保证数据一致性和分布式存储保证数据一致性的差别，大致构想了如果要做分布式存储的话需要优化和改动的地方。

网络编程

网络编程的内容学习起来非常的亲切，因为我负责了公司里长链接消息相关的工作。Goim 的设计中与我们自研的 TCP 有非常多的共同之处，比如字段类型和长度的设计，网络传输中大小端的考虑等等。唯一 ID 的设计也是我负责的业务中很关键的一环，我们会有一个专门的 uuid 生成服务，提供到所有要使用的业务，以一个中心化的 ID 发信器保证 ID 的唯一性。

日志，指标，链路追踪

这一块是偏于基础架构的内容。我们不同的服务采用了不同的解决方案。日志方面，比较古老的项目可能仍然打印到特定的日志机上，较新的业务已经都迁移到 ELK 上了，还有部分日志是会上传到 Clickhouse 上做聚合。链路追踪方面我们目前使用的是 skywalking，思路也是类似的，使用 traceid 保存请求 ID，并顺着请求链路传递下去。后续查看的时候可以在 ES 或者 Clickhouse 里进行完整链路的查询。指标方面，过去我们会因为开启 pprof 会造成一定的开销而没有选择在线上环境开启。听了毛老师的课后，我这边也与领导进行了沟通，为线上服务开启了 profiling。此外我们也还在尝试接入 epbf 的工具，引入系统探针捕获底层连接的数据，完成连接数据的统计和问题的排查。

分布式架构

这一块的内容比较高于我目前的工作内容了，DNS，CDN 都是运维同事在维护。而多活方面，目前我负责项目下还没有多活架构，只是简单国内外单独部署，代码也是不一样的，数据是共享的，存放在国内，国外通过专线访问。这一章学习后可以说是开拓了眼界，以后对相关的工作有了初步的积累。

消息队列，服务发现

消息队列方面，我们项目中使用了多个消息队列。毛老师主要讲的 kafka 被应用在了埋点数据的收集上。过去 kafka 对我来说是比较黑盒的概念，没有深入去了解过。学习课程后我对 kafka 的架构有了一定的了解。我们业务中还用到了 NSQ 的消息队列。对比 kafka，NSQ 是 go 实现的，相对轻量很多，消息数据落盘的设计也不太相同。两者各有优劣，我们将他们灵活的运用在工作里。

Runtime

这一部分的内容是 Go 的底层原理的介绍。在使用 Go 工作了一段时间后我也自己研究过相关的内容。在听到毛老师讲的内容，不仅重新拣回了以前学习过但不常回顾导致有些遗忘的知识，还听到了很多新的知识。这一部分的内容比较深入，平时工作也不太容易使用到，后续会经常再拿出来反复学习，增加印象和理解。

实践

毕业项目的要求很多都已经落到实际项目里，部分要求由于改动较大未能发布到生产环境。因为安全和保密原因无法贴上代码，这里简述一下落地思路。

项目是某产品的账号服务功能。

微服务架构（BFF、Service、Admin、Job、Task 分模块）

1. BFF

   旧的架构没有 BFF 的概念，所有数据需要前端发起多个网络调用，获取响应自行渲染。改造后为账号系统提供了 BFF 层，整合了账号的属性，关注的动态信息，广告位 Banner，账号有权限访问的子功能等等。并且会区分账号的类型，为不同的账号展示不同的数据。

2. Service

   Service 为直接提供到前端的服务，包括账号属性的查询和修改，统一权限校验服务等。

3. Admin

   Admin 由于公司有统一的平台，所有的管理接口都向这个平台提供服务，包括账号属性的查询和修改，账号历史变更记录，账号注销，活跃记录查询等。

4. Job

   Job 处理流式任务，比如账号合并时账号数据的同步等。

5. Task

   Task 处理定时任务，比如定期的对用户的行为数据进行整合和上报等。

API 设计（包括 API 定义、错误码规范、Error 的使用）

1. API 定义

   过去定义 API 使用专门的内部文档系统，代码修改了，文档还得一并修改。现在使用 protobuf 定义接口的文档，并有 api 目录暴露这些文档，需要对接的一方直接查看 api 中的 proto 文件即可。修改的话也直接修改 proto 文件生成新的代码。

2. 错误码规范

   由于服务体量较小，公司内部也没有统一的错误码定义平台。所以自行维护整个账号的错误码。使用 7 位数的设计，前 3 位是 HTTP 状态码，请求的响应严格遵循 HTTP 状态码的定义。只有请求成功了才是 200 开头，如果是接口参数异常则是 400 开头，权限校验失败则是 403 开头等。后 4 位是详情错误码，比如 0000 是成功，0100 是参数格式错误等。所有的错误码定义到一个 errcode 文件下。

3. Error 使用

   对一些经常出现的错误定义 sentinel error，比如权限校验失败，账号不存在等。然后每一层遇到错误时，如果错误可以处理或者可以接受，则打印日志并将错误降级。如果不可接受，使用 wrap 包装错误，添加引发错误出现的场景和其他的补充信息并上抛。到最外层进行日志的打印和响应返回。

gRPC 的使用

1. 对原本基于 Gin 的 HTTP 服务进行 gRPC 改造。不光是账号系统，支付系统，短信邮件系统也需要进行对应的改造。

Go 项目工程化（项目结构、DI、代码分层、ORM 框架）

项目结构借鉴 kratos 以及毛老师所教学的案例，包括 /api 存放接口 proto 文件。/cmd 存放服务入口代码。/internal 存放项目内部代码，不同的子项目代码隔离在不同的文件夹下，/biz 目录下定义 repo 接口，/data 目录下实现 /biz 下的 repo 接口，/service 编排 /biz 下的各种对象。/pkg 存放可跨项目使用的公共代码。ORM 使用 ent，填写使用到了结构体，自动生成一些数据操作代码。

并发的使用（errgroup 的并行链路请求）

最直接的使用方式是同时启动服务和信号处理逻辑。其他情况比如我在主逻辑以外需要开启一个新的 goroutine 用来进行心跳检测时也可以使用 errgroup 同时启动主逻辑和心跳检测逻辑，某一个错误时，通知另一个 goroutine 停止工作。

微服务中间件的使用（ELK、Opentracing、Prometheus、Kafka）

使用容器化启动服务，服务的日志直接打印到服务器本地，并添加 log max 限制，然后启动 fluentd 不断捕获本地日志上传到 es 或者 clickhouse 上，通过 kibana 查看日志。

缓存的使用优化（一致性处理、Pipeline 优化）

1. 很多场景下会遇到缓存一致性问题，比如登陆 token 的更新和校验等等，采用毛老师介绍的 “set cache 优先度大于回填（setnx） cache” 的方案保证缓存一致性的问题。

2. 对于 pipeline 优化，目前由于使用缓存处理的场景相对简单，不使用 pipeline 也能很好的保证数据一致性。所以暂未落地 pipeline。

最后

总的来说，这门课程的学习下来收获非常的多，很多已经应用到了自己的工作，产出不少的结果。此外还有很多我没有消化的内容，需要日后反复琢磨。最后，感谢极客时间，毛剑老师推出的课程！感谢大明老师每周的拓展直播！感谢班班的后勤服务以及助教老师的作业批改和问题解答！