简历

个人技能

- 熟练掌握大规模、**高可用、高性能**、【高并发】分布式架构
- 熟练掌握 GO 语言，内置类型原理，GO 运行时原理，GO 并发库源码，
  - 熟练掌握 GO **并发编程**，网络编程
    - 用并发工具库、TaskPool 作为证据
      - 并发队列
      - 优化版的读写锁：
        - 写优先：一旦来了一个写请求，后续读请求都不能加锁
        - 读优先：不管你来不来写请求，我都可以一直加读锁
        - 按比例计算：来了一个写请求，然后你计算一个随机值；阻塞的写请求，越多，越可能阻塞读请求。
      - 按 key 加锁： key -> RWMutex, key -> Mutex
        - 实现方案：sync.Map + sync.Mutex，控制住 key 的数量
    - 网络编程可以考虑尝试设计一些简单的协议
  - 熟练掌握 **GO 性能优化**
    - 内存优化，准备案例：内存逃逸优化，Buffer Pool（对象池） 优化，字段对齐，Arena 优化
    - 并发优化，准备案例：读写锁取代写锁，原子操作取代读写锁（用原子操作来优化 double-check），无锁化，singleflight 模式，全局竞争转局部竞争
    - GO 性能 BUG 排查
- 数据库，Innodb 引擎原理，熟练掌握查询优化、锁优化
  - 查询优化案例：
    - 索引优化：使用覆盖索引，调整索引列的顺序，
    - 排序优化：利用索引排序
    - 分页优化：禁止跳页
    - 改写 SQL：
    - HAVING 优化：把 HAVING 中的条件尽可能放到 WHERE
    - JOIN 查询优化，ON：ON 的条件命中索引

        - 子查询优化：
    - 锁优化：
        - 案例一：使用乐观锁取代悲观锁
        - 案例二：修改查询，命中索引，避免表锁
        - 死锁案例：如何发现、定位、解决
    - 数据库本体优化：调整 MySQL 参数，InnoDB 引擎参数，所在操作系统参数
    - 读写分离：
    - 分库分表：
        - 主键问题：主键生成策略，高性能发号器
            - 批量获取 ID
            - 提前获取 ID
            - singleflight 获取
        - 容量预估与扩容问题
        - 数据迁移：完整方案
            - 数据校验和修复（并发、数据一致性问题）
            - 性能问题
        - 分布式事务
        - 性能优化：
            - 案例一：优化分页查询，禁用跳页
- Redis，高可用高性能原理
    - 一致性问题，但凡用了缓存，你就要想好你怎么解决一致性问题。
    - 与众不同的缓存方案
        - 案例一：Redis 崩溃，启用本地缓存
        - 案例二：降级状态下，缓存未命中直接返回
        - 案例三：一致性哈希负载均衡算法 + 本地缓存 + Redis 缓存 + 数据库
        - 案例四：不同业务的 Redis 集群互为备份
    - 熟练掌握缓存模式，解决缓存穿透（DB 本身就没有数据）、击穿和雪崩：
    - 高可用 Redis 分布式锁：续约机制，重试机制，性能优化
        - 本地 singleflight 优化全局竞争
        - 分布式锁本地转交
        - 去分布式锁

- 适合面试的其它方案：
  - 灵活的淘汰策略
  - 灵活的过期时间
- 消息队列，Kafka 高可用高性能原理
  - 解决顺序消息。
    - 案例一：解决顺序消息消息积压问题
  - 解决消息积压问题：
    - 案例一：顺序消息消息积压问题
    - 案例二：消费者调用批量接口批量消费
    - 案例三：异步批量消费
  - 解决重复消费：布隆过滤器 + Redis + 唯一索引的高并发幂等方案
    - 如果 Redis 是高可用的，可以不用唯一索引。
  - 延时消息：
    - 方案一：分区/topic 设置固定延迟时间，比如说 1，3，5
    - 方案二：借助 MySQL 实现任意延迟消息
      - 交替表
      - 分区表
      - 分库分表
  - 消息回查（事务消息）：

项目经历

高可用的短信服务

短信服务是公司的核心服务，大部分的关键业务都需要使用短信服务，因此短信服务的性能和可用性对整个系统性能和可用性有关键影响。

主要职责：优化短信服务的性能，提高可用性。

1. 提高可观测性，丰富监控和告警，做到快速发现问题，并且通知相关业务方，同时可观测性数据也是优化性能、提高可用性的基础。（吹一下效果……什么一秒钟快速发现问题）
2. 提高短信服务的可用性，引入了自动重试、同步转异步，复杂的 failover 机制和客户端限流机制。可用性提高到了三个九。（这里可以举例子，某云的短信服务出问题的时候，我这边自动切换到了腾讯云，整个过程没有感知）保障短信 100% 发送，彻底解决了短信丢失问题
3. 优化性能，引入异步发送（指提供了 MQ 的接口）、回调机制，以及单个转批量发送（这里

要稍微解释一下单个转批量的意思）机制，性能瓶颈从 500/s 提高到了 4000/s

4. （这一条你记住了多少就补充多少）引入了管理平台和资源审批机制（模板管理，短信资源管理，权限控制等）、业务鉴权机制，可以细粒度控制不同业务的短信使用量、QPS 等，同时提供了友好的界面给业务方查询短信相关数据；提供界面，允许运营人员配置一定的查询条件，筛选目标用户发送短信

插件库

提供了性能优秀、功能丰富的数据结构、并发数据结构，以及 GIN，GORM，saramax，redis 客户端等插件，是全公司的核心插件库。

- 并发工具类：延迟队列，并发阻塞队列，协程池等
- GIN 插件：高并发的全新 Session 插件，可观测性插件，Web 端治理插件（项目介绍的时候补充熔断限流降级鉴权等），泛型包装类
- GORM：可观测性插件、限流
- sarama：可观测性插件，Consumer 装饰器（重试、可观测性、批量消费、异步消费）
- redis：可观测性插件，以及各种缓存模式实现，分布式锁实现

1. GO 泛型工具库：

面试要点：

   a. 并发队列
   b. 基于 key 的加锁机制
   c. TaskPool，或者协程池
   d. 复杂数据结构：红黑树、跳表等
   e. 连接池：TCP 连接池
   f. 优化版的读写锁：
      i. 写优先：一旦来了一个写请求，后续读请求都不能加锁
      ii. 读优先：不管你来不来写请求，我都可以一直加读锁
      iii. 按比例计算：来了一个写请求，然后你计算一个随机值；阻塞的写请求，越多，越可能阻塞读请求。

2. 缓存中间件：

   a. 实现了本地缓存
      i. 红黑树的本地缓存
      ii. **Arena 的本地缓存**

        iii. 控制内存
    b. 实现了各种缓存模式
    c. 统一解决了缓存穿透、击穿和雪崩
    d. 支持 Redis 容错方案
        i. 本地缓存取代 Redis 容错
        ii. Redis 互为备份
    e. 缓存可观测性：缓存命中率
    f. 紧密结合服务治理方案：熔断限流降级的时候，缓存可以做一些事情
        i. 降级的时候，只查询缓存
        ii. 熔断的时候，缓存直接返回默认值
3. 微服务治理框架，基于 GRPC 的微服务框架
    a. 负载均衡算法
    b. 熔断限流降级
    c. 重试
4. Kafka 扩展支持
    a. 延时消息
    b. 消息回查机制
    c. 异步批量消费封装：进一步结合 TaskPool 来控制 goroutine 数量
5. 数据迁移方案

面试方案

高可用面试方案

- 全部第三方读启用高可用方案：
    - redis cluster
    - Kafka 集群
    - MySQL 读写分离
- 微服务治理：
    - 熔断

    - 限流
    - 降级
    - 链路超时控制
    - 重试 + 幂等
    - 同步转异步/消息队列解耦
    - 隔离
    - 负载均衡
- 容错：
    - redis 崩溃
    - Kafka 崩溃
    - MySQL 崩溃
    - ... 任何第三方崩溃

高性能面试方案

- 前端优化性能：
    - CDN
    - 调整静态资源缓存过期时间
- JWT token 保存频繁访问数据：例如 username 等，避免频繁访问
- 分布式锁优化
    - singleflight 优化分布式锁
    - 本地转交
    - 去除分布式锁
- 客户端缓存权限信息：客户端调用权限服务拿到权限信息，直接缓存到本地，监听权限服务的权限变更消息
- 缓存优化：
    - 哈希负载均衡 + 本地缓存 + Redis 缓存
    - 缓存预热
    - 业务专属淘汰策略
    - 调整过期时间
    - 启用压缩（调整压缩算法）
    - 使用缓存模式
        - singleflight

- MySQL 调优：
  - 本体调优（去问你的运维团队）
    - 调整隔离级别
    - 调整刷盘时机
    - 调整 buffer pool
    - 调整操作系统
  - 查询优化：
    - 改写 SQL
    - 优化索引
    - 优化排序（分页）
  - 锁优化
  - 读写分离
  - 乐观锁
  - 分库分表
- Kafka 调优：
  - 优化发送者性能：调大批次，启用压缩（更换压缩算法）
  - 优化消费者性能：批量消费（调用批量接口）、异步消费
  - 优化 Kafka（去问你的运维团队）：调整 Kafka 所在操作系统，JVM 调优（垃圾回收调优）
- 语言层面优化
  - GO GC 优化
  - GO 并发优化
- 业务流程
- 架构优化

有竞争力的项目贡献

- 非功能性：
  - 高并发
  - 高可用
  - 高性能
  - 可观测性
  - 扩展性

- 交付质量
  - 代码质量
  - 测试覆盖率
- 研发效率
  - 各种优化流程
  - 各种辅助工具
- 管理方面
  - 人才培养
  - 成本节省