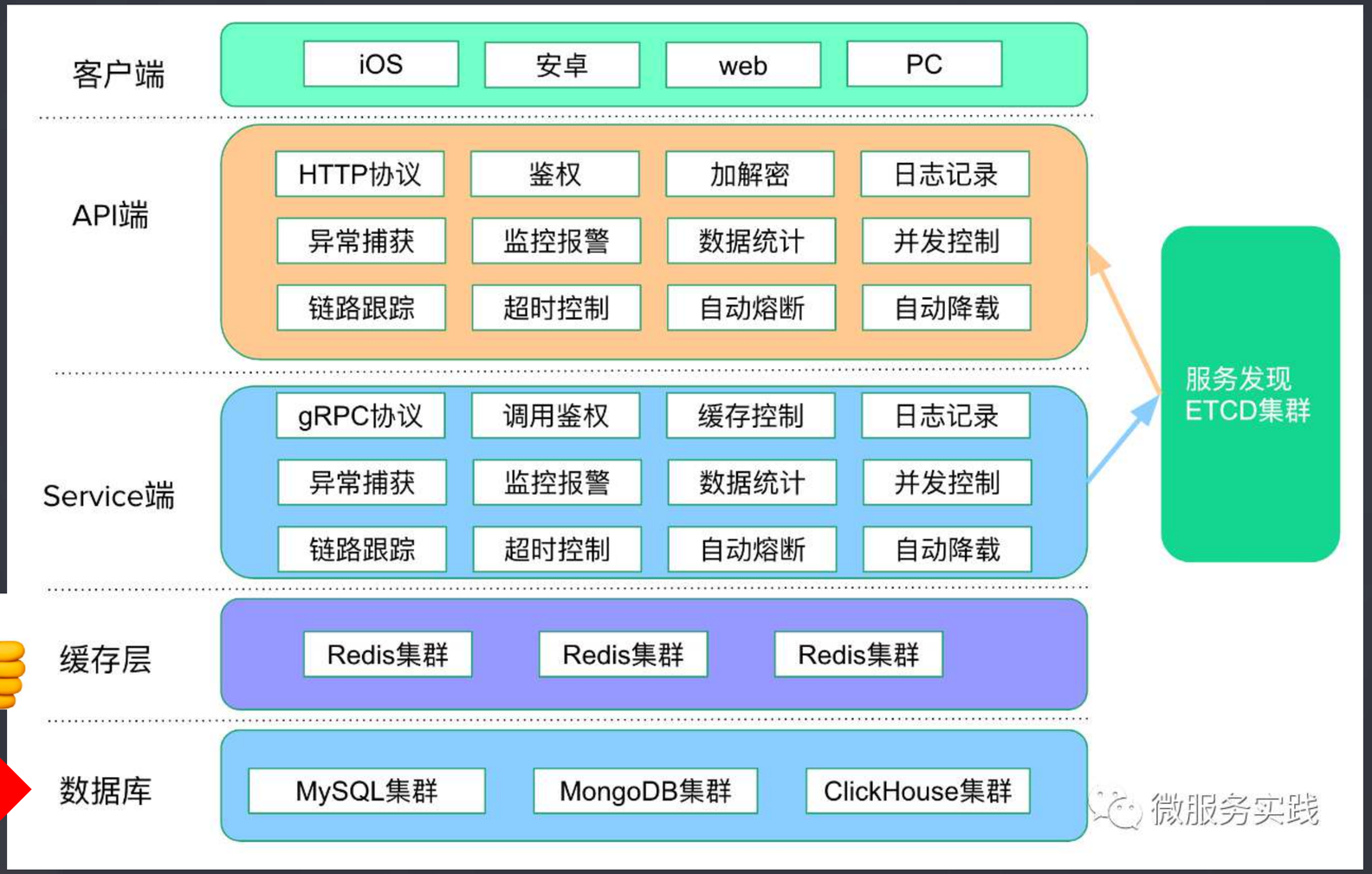
go-zero分布式缓存最佳实践

万俊峰Kevin

万俊峰Kevin

- go-zero作者
- 好未来技术委员会资深专家
- 晓黑板研发负责人
- 14年研发管理经验
- 20年开发和架构经验

高并发下的系统瓶颈?

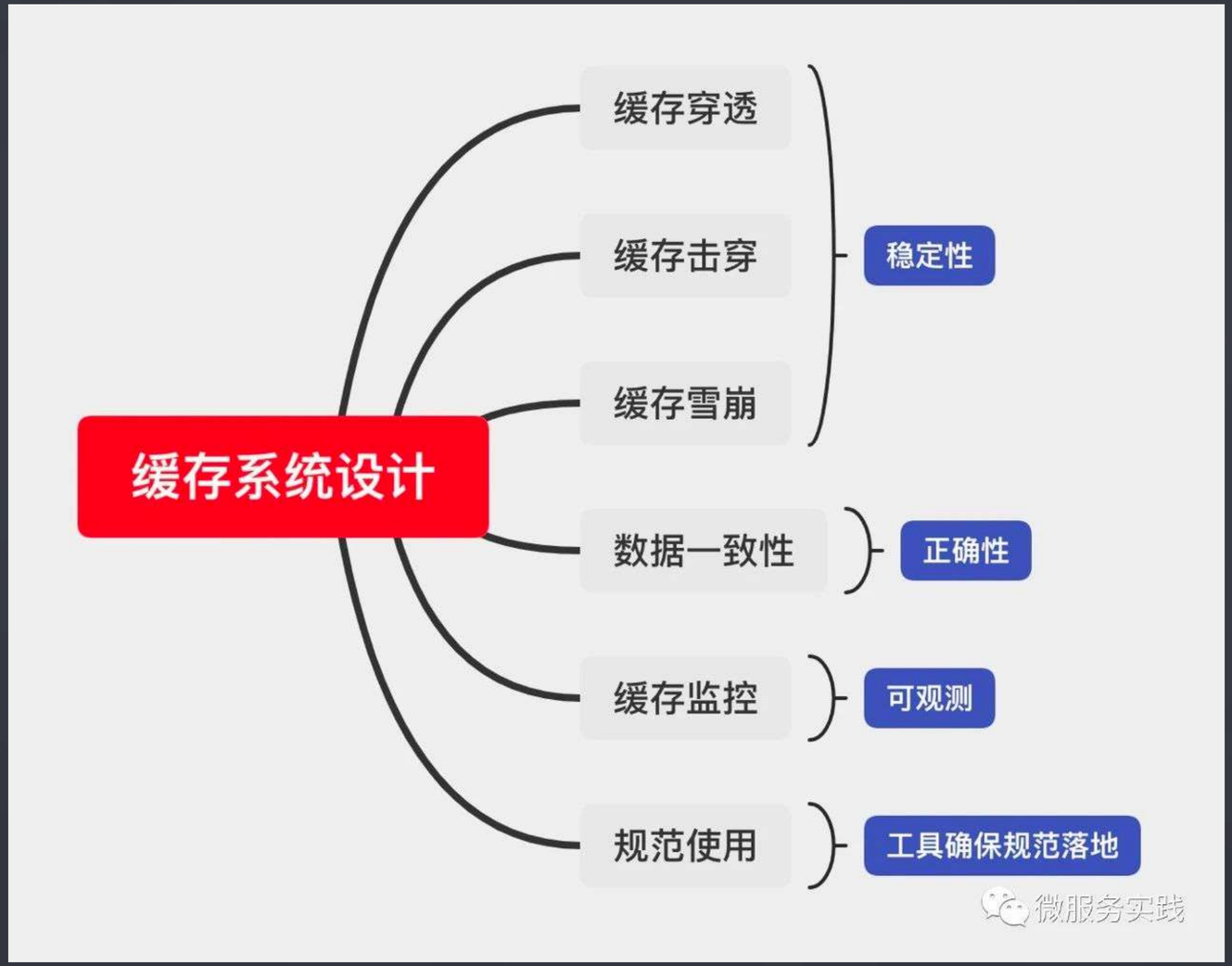

客户端
iOS
安卓
web
PC
API端
HTTP协议
鉴权
加解密
日志记录
异常捕获
监控报警
数据统计
并发控制
链路跟踪
超时控制
自动熔断
自动降载
Service端
gRPC协议
调用鉴权
缓存控制
日志记录
异常捕获
监控报警
数据统计
并发控制
链路跟踪
超时控制
自动熔断
自动降载
服务发现
ETCD集群
缓存层
Redis集群
Redis集群
Redis集群
数据库
MySQL集群
MongoDB集群
ClickHouse集群
微服务实践

大纲

- 缓存系统常见问题
- 单行查询缓存与自动管理
- 多行查询缓存机制
- 分布式缓存系统设计
- 缓存代码自动化实践

缓存系统常见问题

缓存系统常见问题


缓存系统设计
缓存穿透
缓存击穿
缓存雪崩
稳定性
数据一致性
正确性
缓存监控
可观测
规范使用
工具确保规范落地
微服务实践

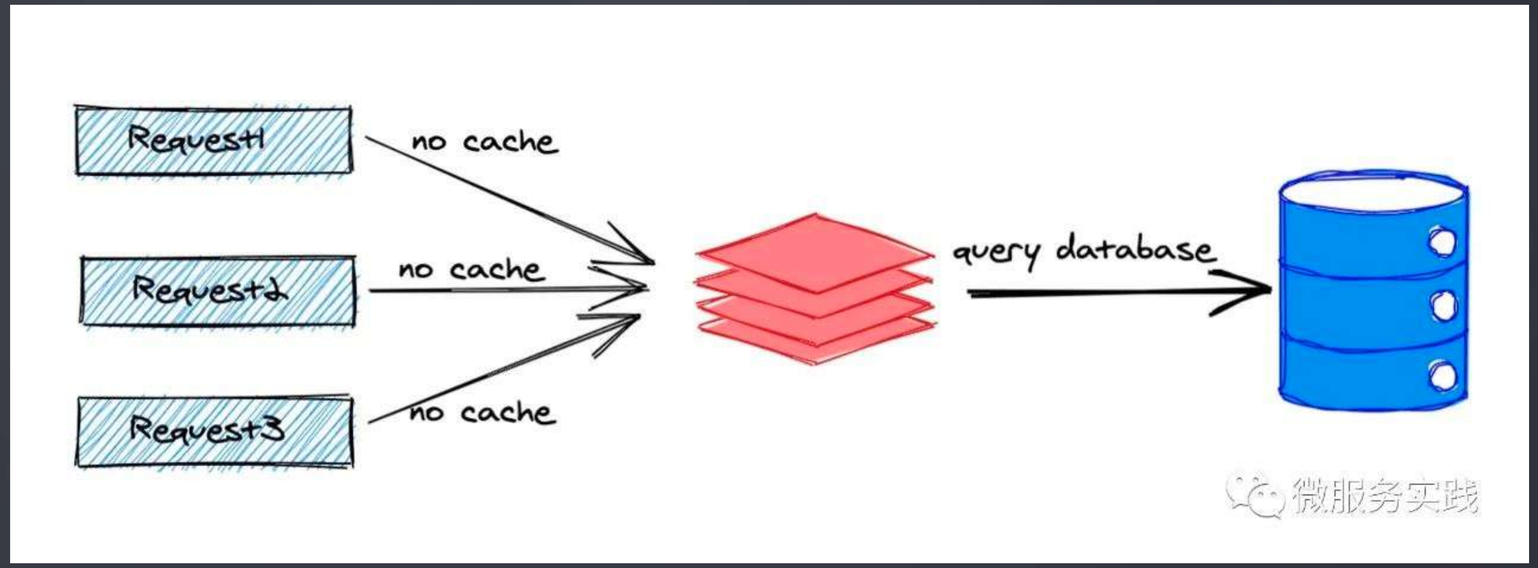
缓存穿透

问题原因：

- 不存在的数据请求
- 恶意请求

解决方法：

- 缓存占位符
- 短缓存，比如一分钟



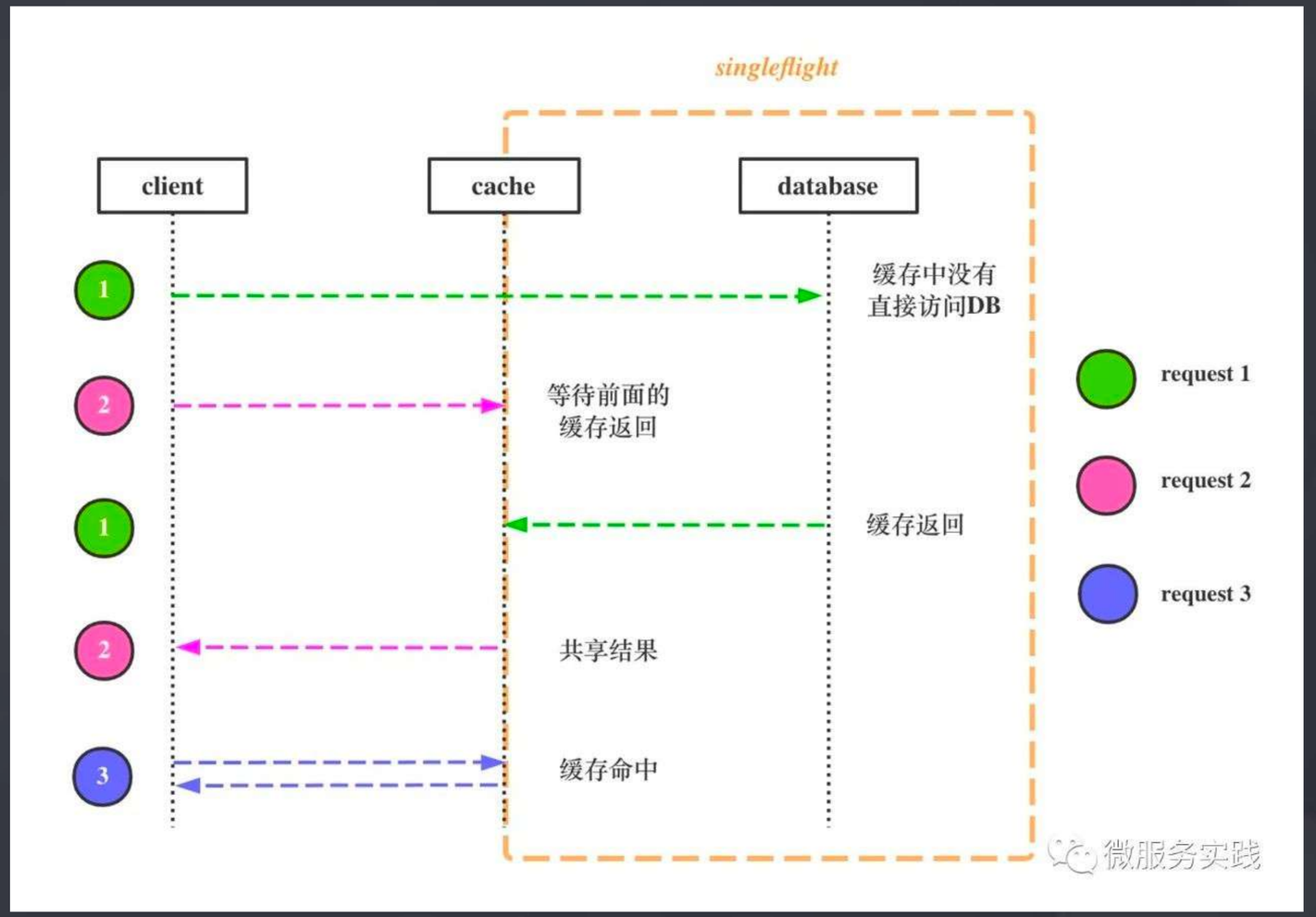
缓存击穿

问题原因：

- 热点数据过期
- 大量并发请求

解决方法：

- 唯一DB请求，共享结果
- 分布式锁



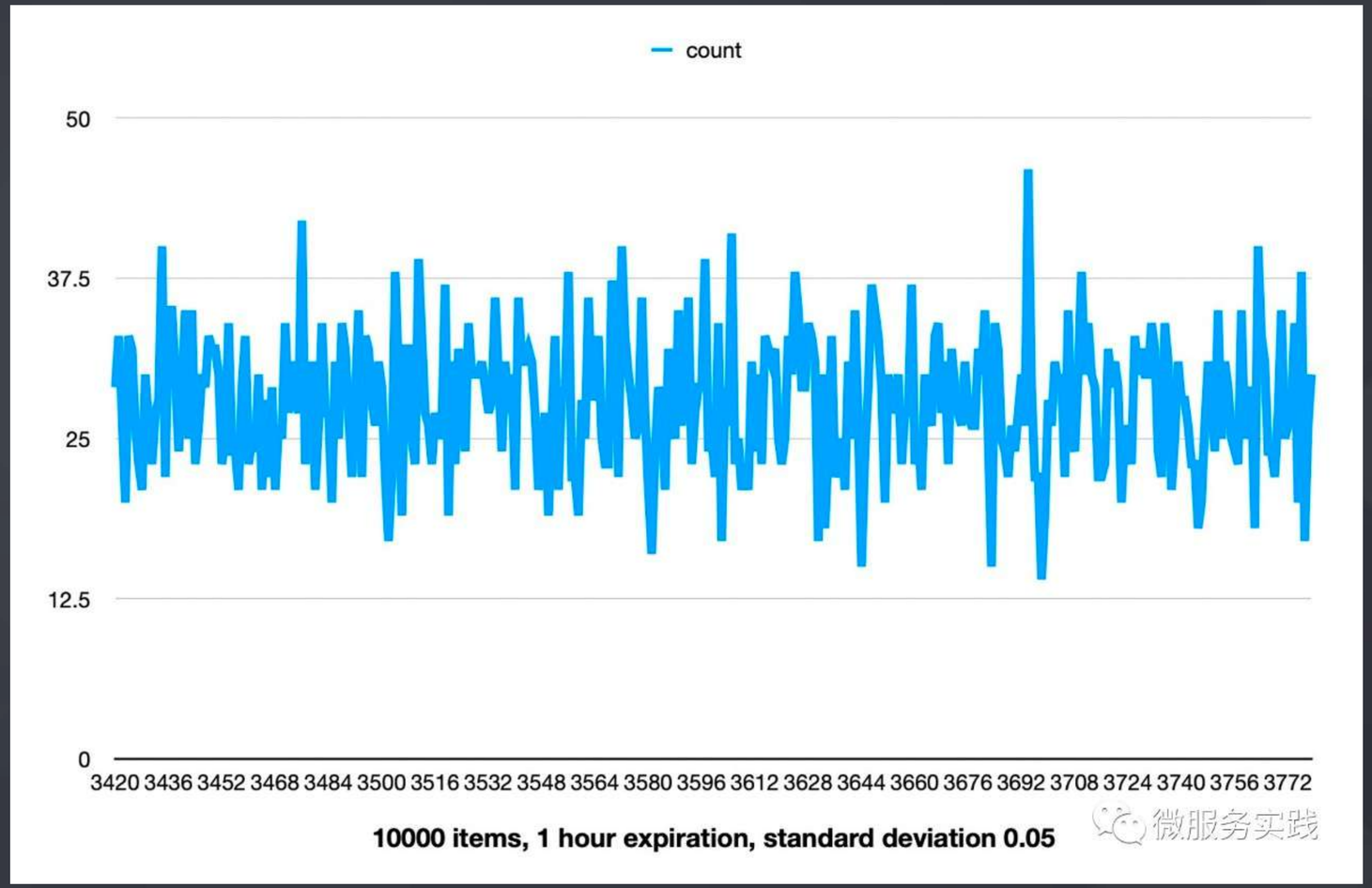
缓存雪崩

问题原因：

- 相同的缓存时间
- 缓存同时过期
- 大量请求落到DB

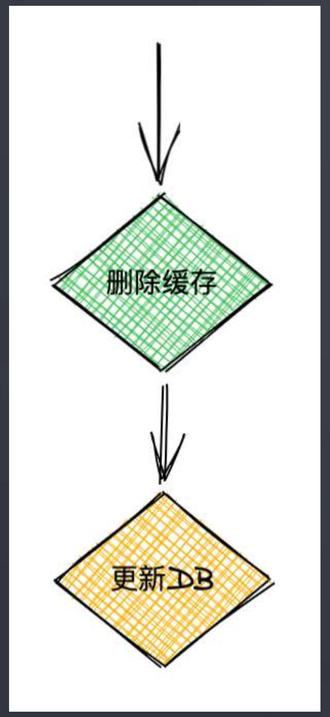
解决方法：

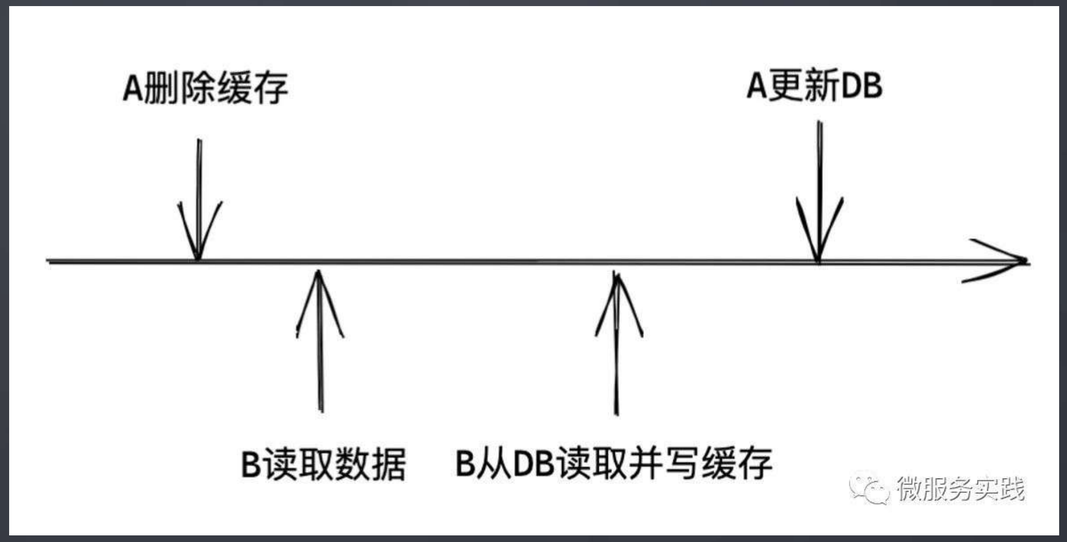
- 缓存过期时间加上随机偏差
- 分布式缓存，防止单节点故障



缓存一致性

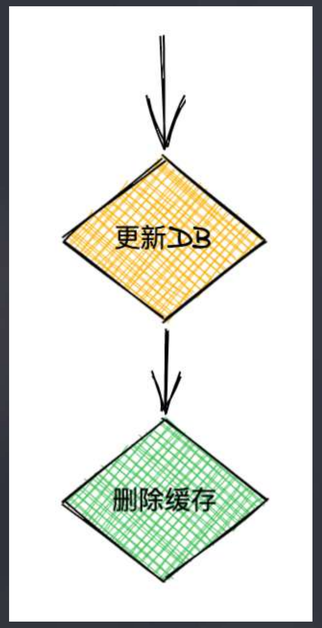
先删除缓存，再更新数据库

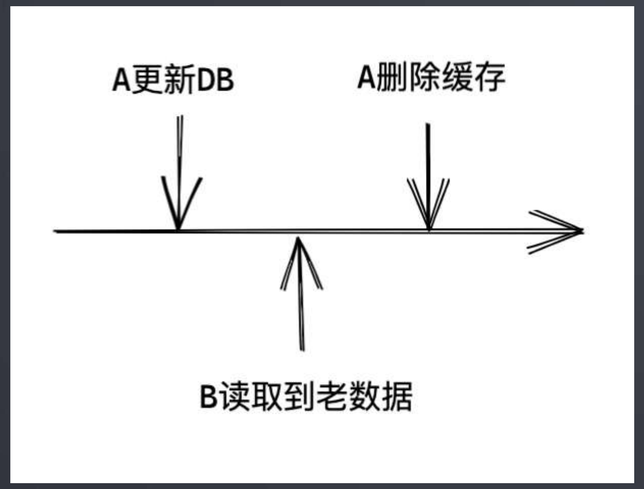


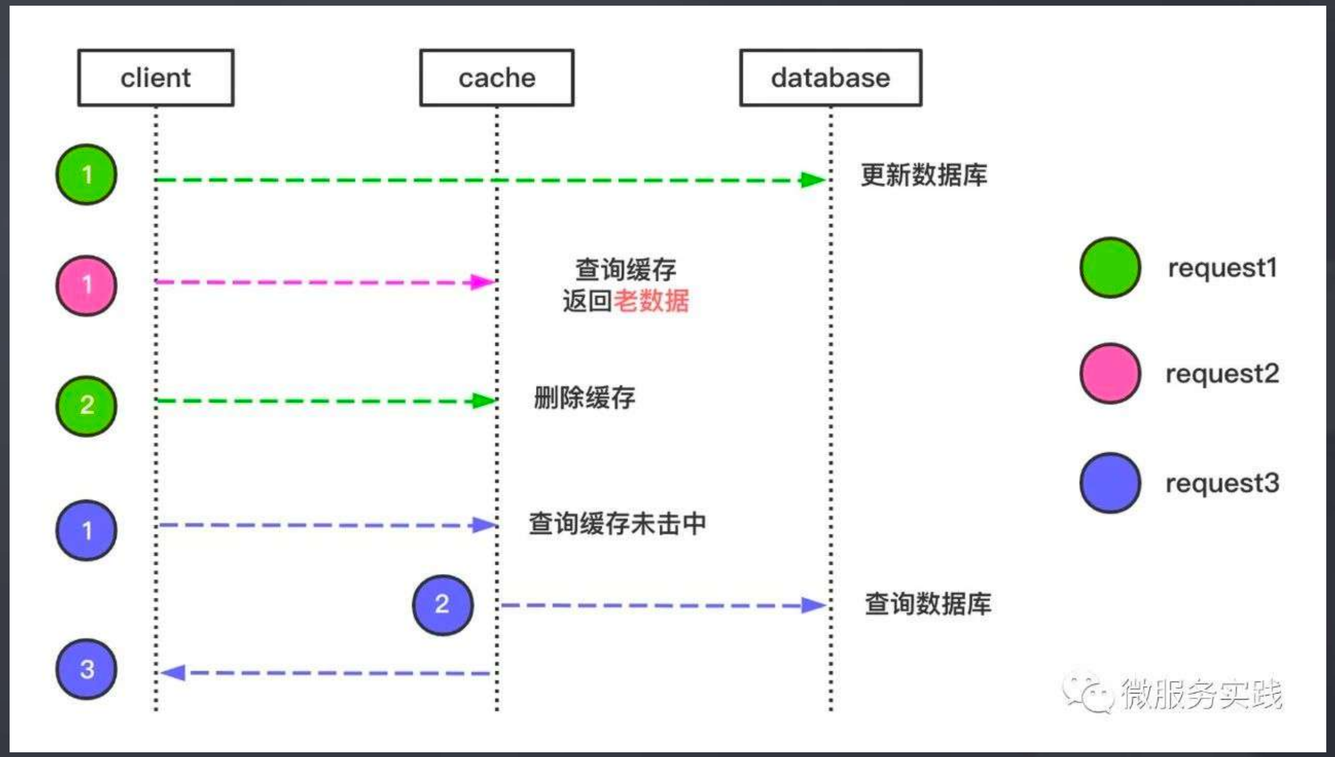


缓存一致性

先更新数据库，再删除缓存

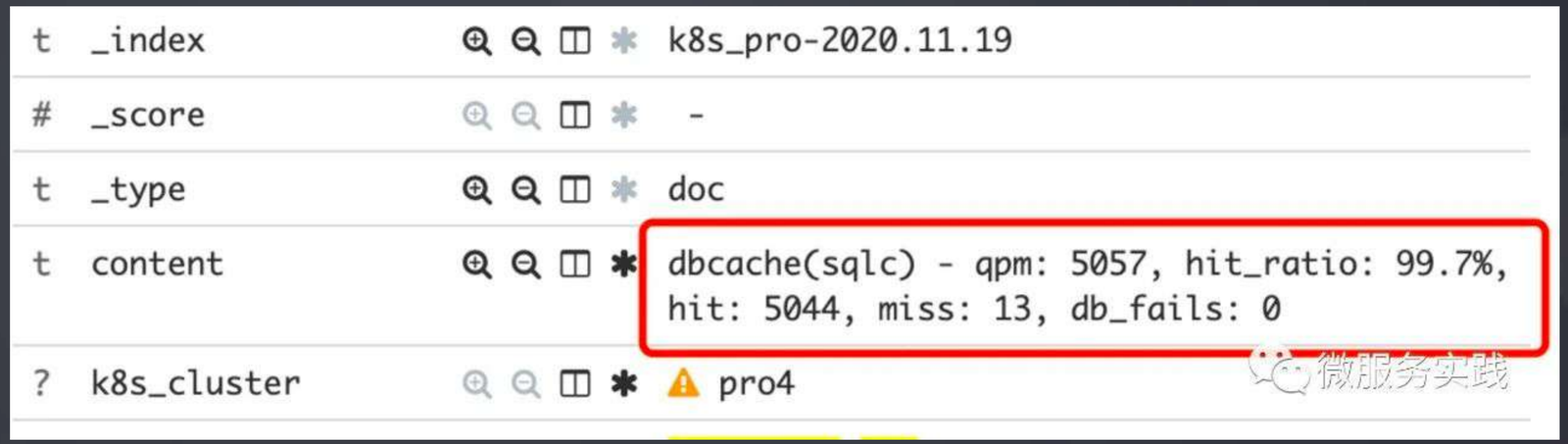






缓存监控

- 缓存是否必需
- 过期时间是否合适
- 命中率如何
- 是否需要进一步调优



如何让业务规范缓存的使用?

问题:

- 缓存知识点繁多
- 千人千面的缓存代码
- 很难写对

解决方法:

- 框架集成
- 工具一键生成

- 在 `rpc/model` 目录下执行如下命令生成CRUD+cache代码, `-c` 表示使用 `redis cache`

```
goctl model mysql ddl -c -src book.sql -dir .
```

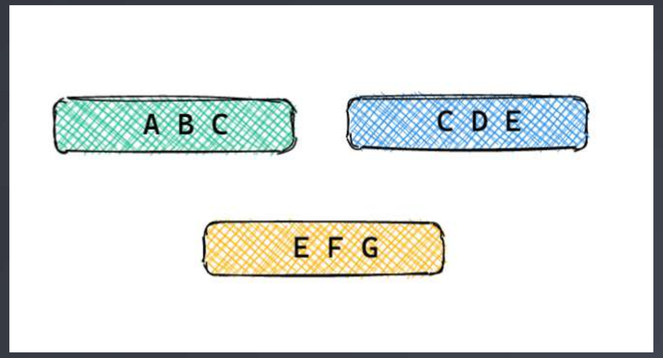
也可以用 `datasource` 命令代替 `ddl` 来指定数据库链接直接从schema生成

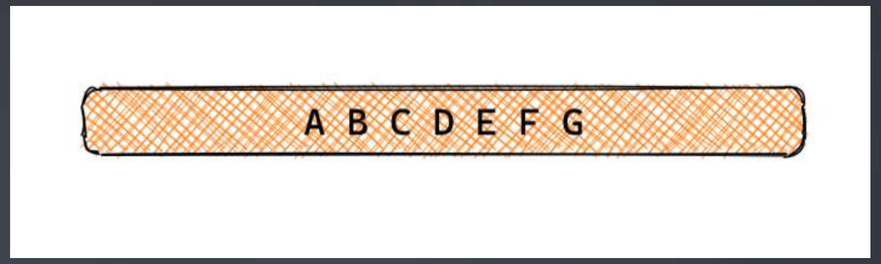
生成后的文件结构如下:

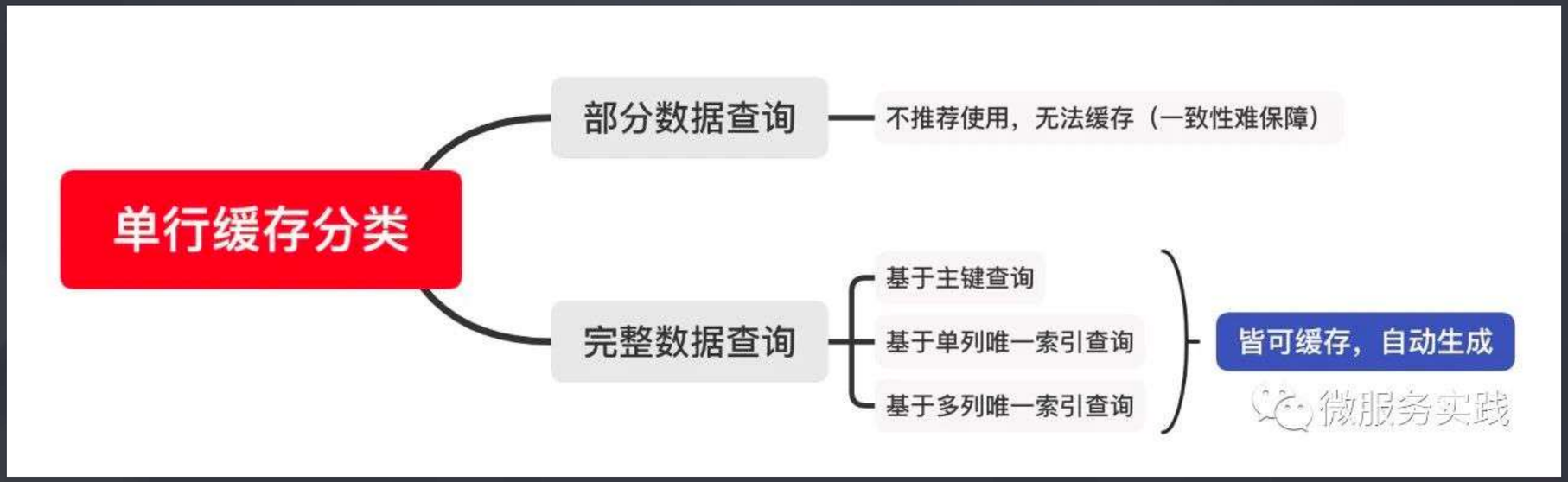
```
rpc/model
├── book.sql
├── bookstoremodel.go       // CRUD+cache代码
└── vars.go                 // 定义常量和变量
```

单行查询的缓存与自动管理

单行查询的缓存

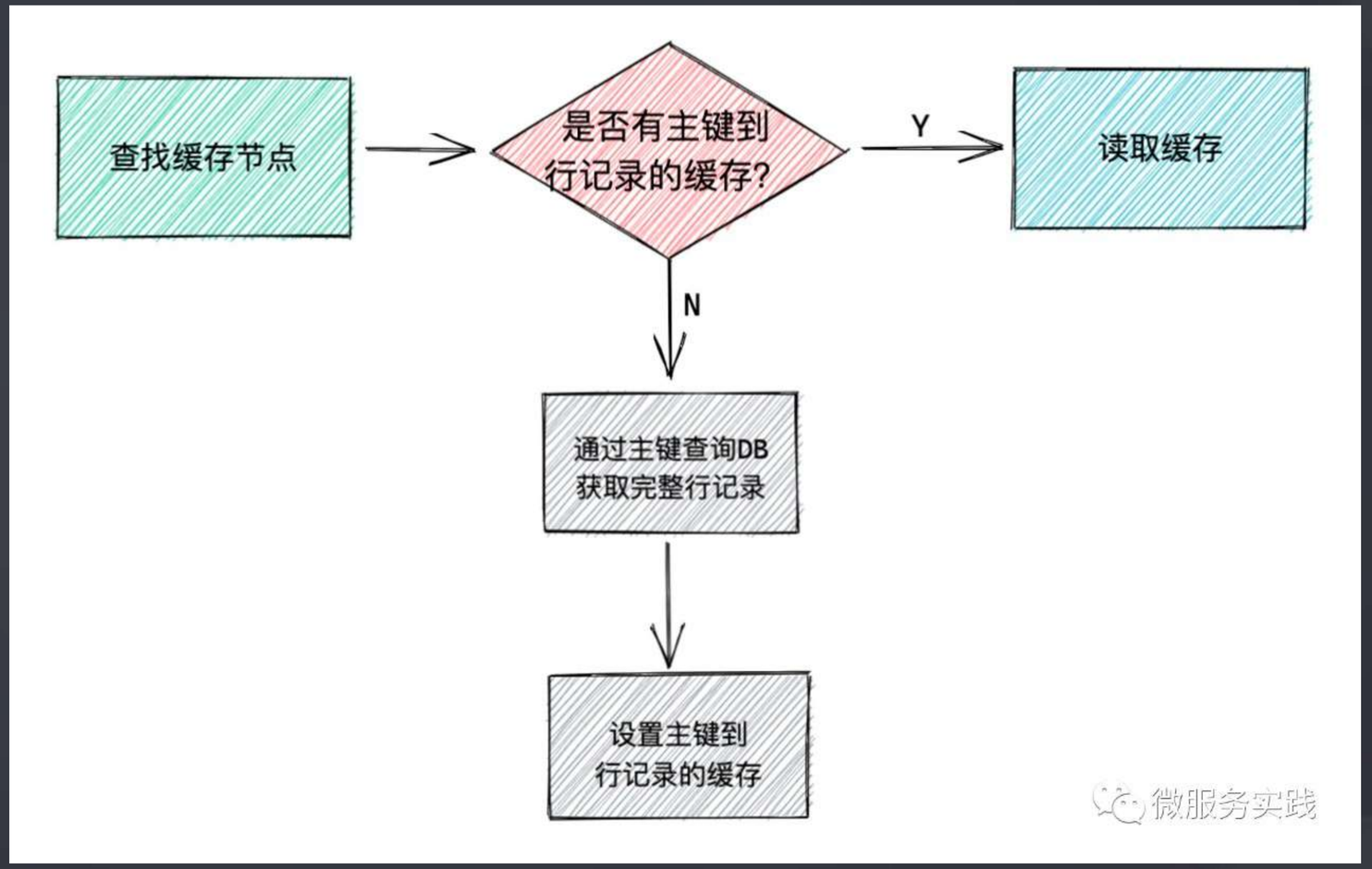

A B C
C D E
E F G


A B C D E F G


单行缓存分类
部分数据查询
不推荐使用，无法缓存（一致性难保障）
完整数据查询
基于主键查询
基于单列唯一索引查询
基于多列唯一索引查询
皆可缓存，自动生成
微服务实践

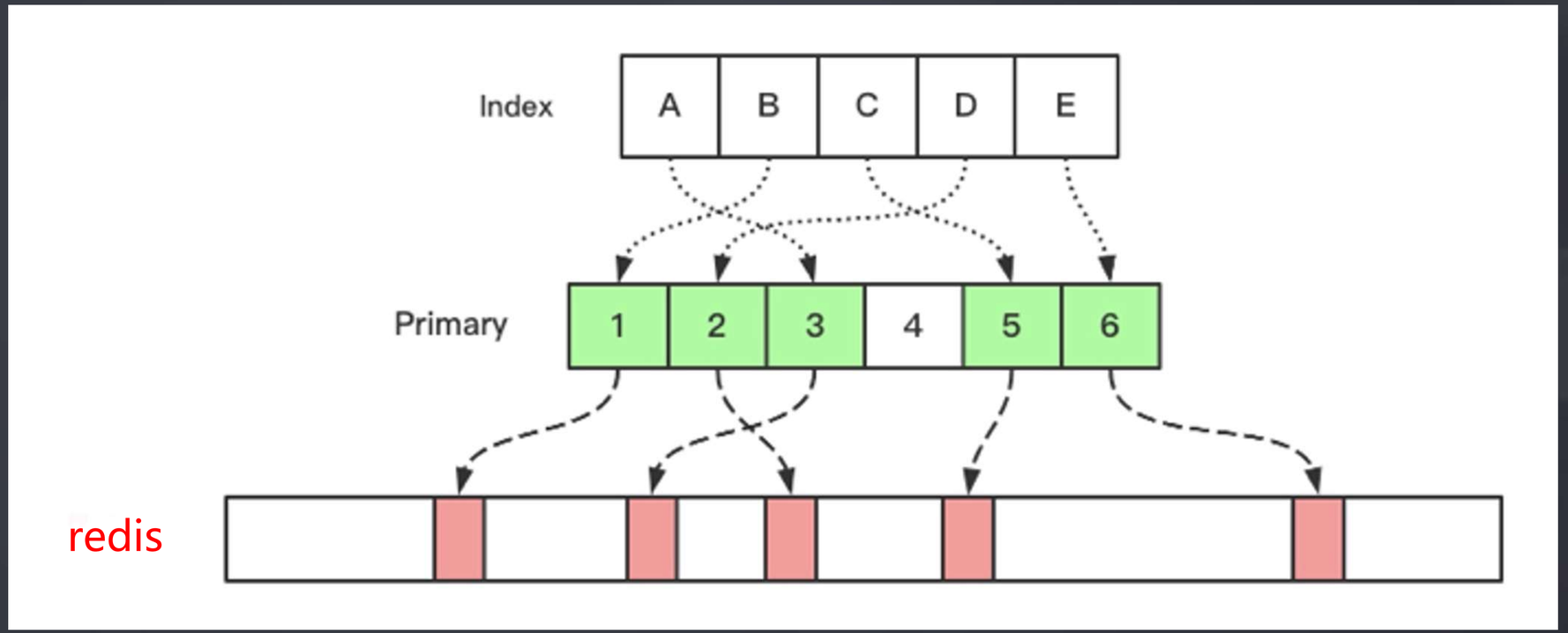
基于主键的单行缓存

- 基于主键缓存
- 返回单行完整数据
- 自动生成代码



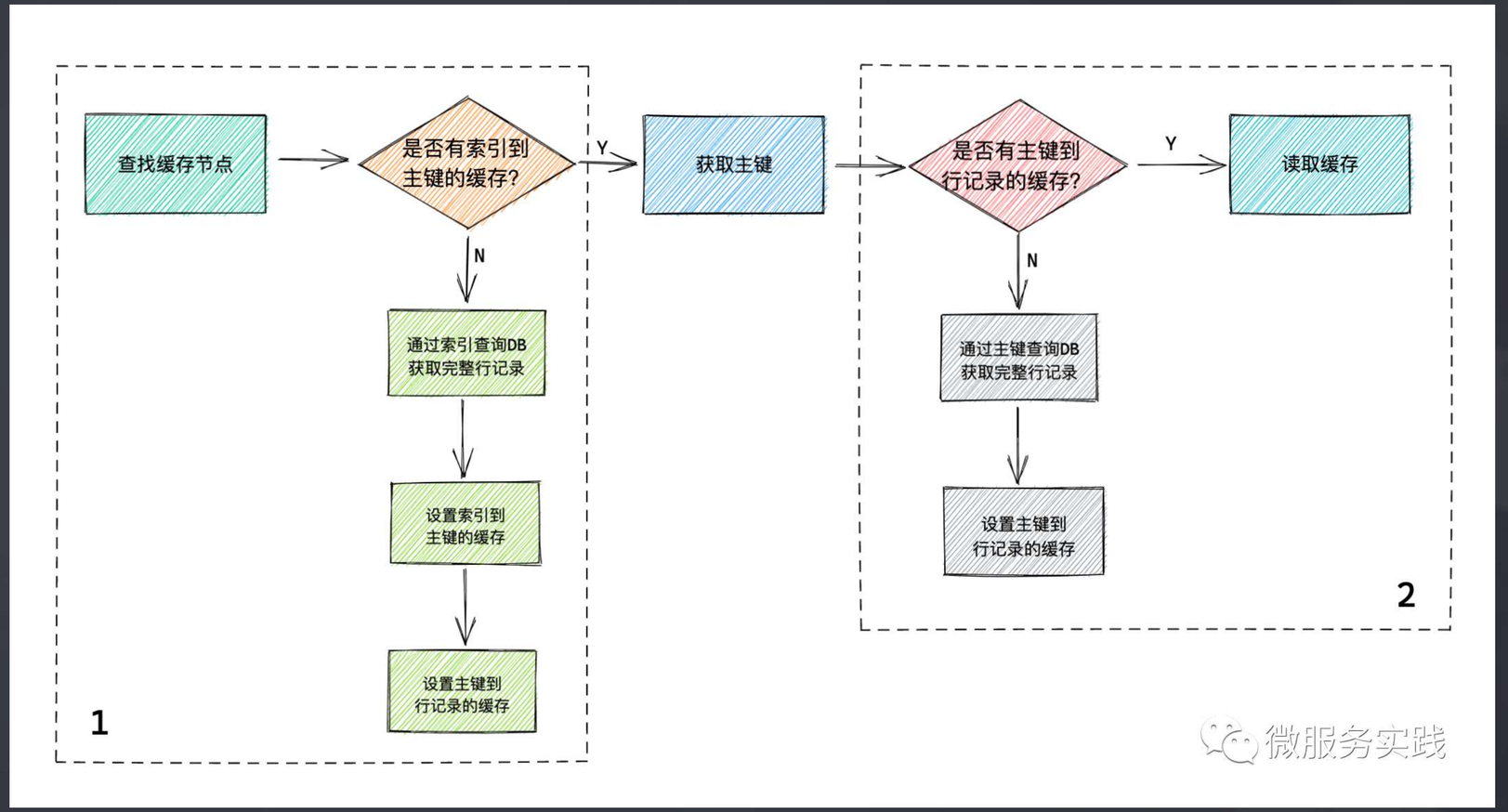
基于唯一索引的单行缓存设计

- 单列唯一索引
- 多列唯一索引
- 主键：完整数据



基于唯一索引的单行缓存逻辑

- 唯一索引： 主键
- 主键： 完整行数据
- 返回单行完整数据



单行缓存自动管理伪代码

- 自动防止数据不一致
- 自动防击穿
- 自动防穿透
- 自动防雪崩
- 自动统计

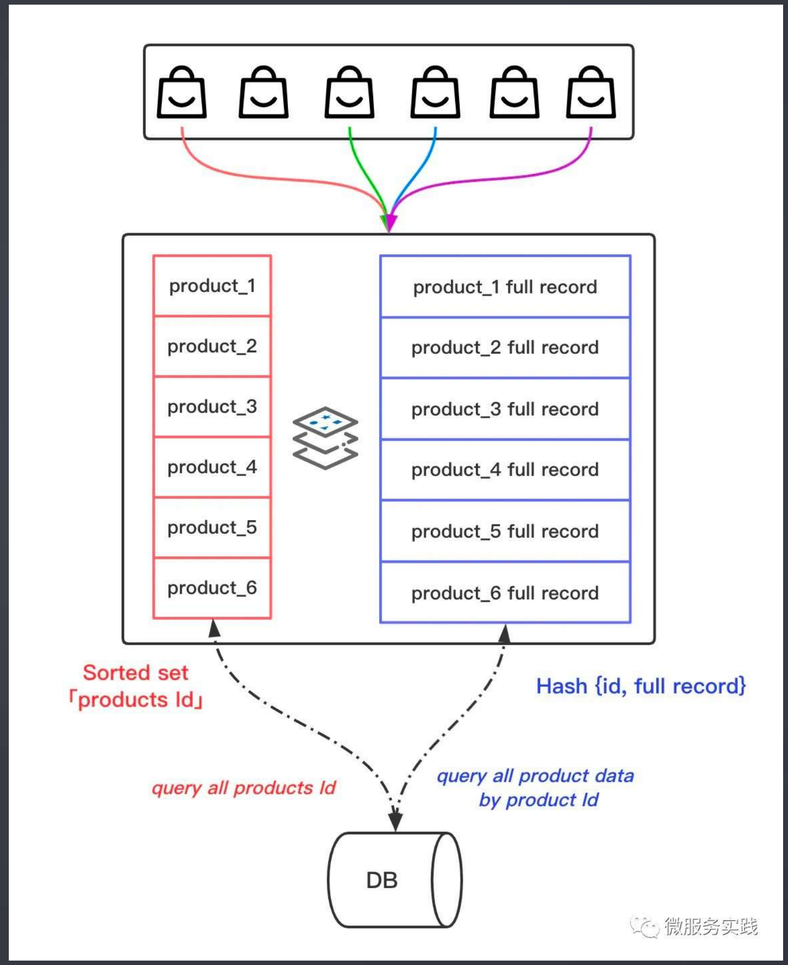
```
func QueryRow(...) error {
    singleflight {
        read cache
        if placeholder {
            return not found
        }

        if has cache {
            return val
        }

        query db
        if not found {
            set cache with not found placeholder
            return not found
        }

        set cache
        return val
    }
}
```

完整实现代码见 go-zero/core/stores 下的 cache, sqlc, mongoc 子包

多行查询的缓存机制

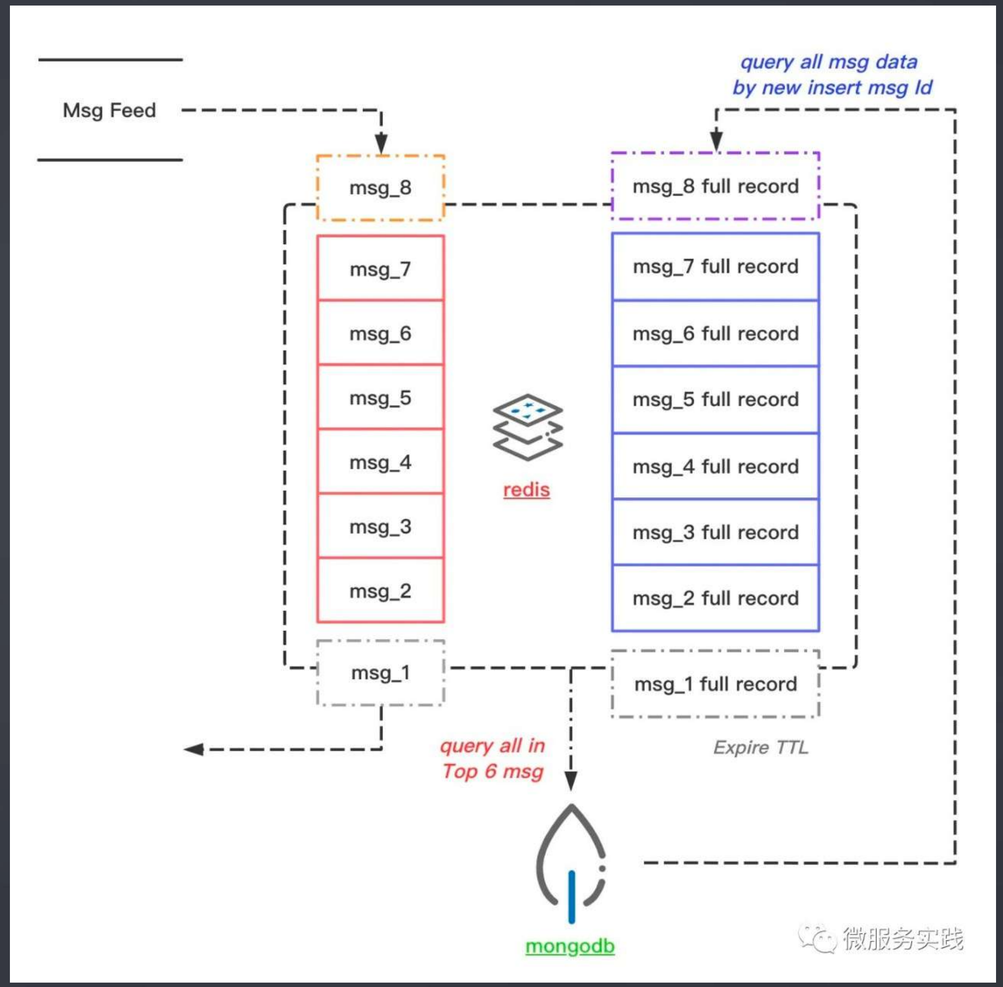
不易变的列表

- 比如店铺商品
- ID放在redis的sorted set
- 单列DB查询+缓存
- 修改友好，单条缓存删除
- mapreduce并发请求



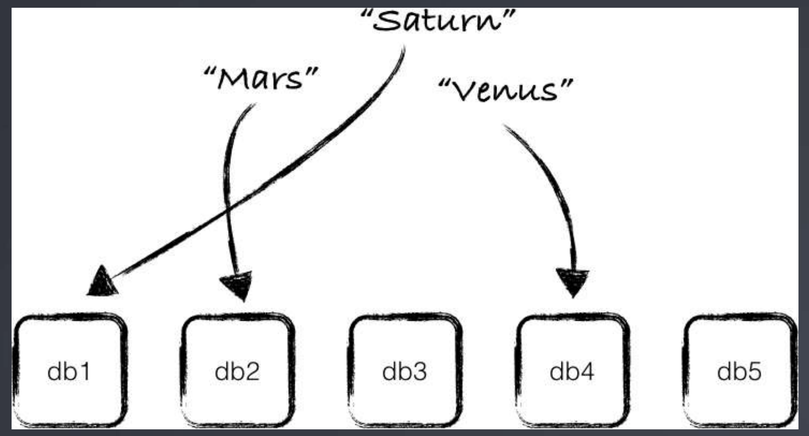
易变且分页的列表

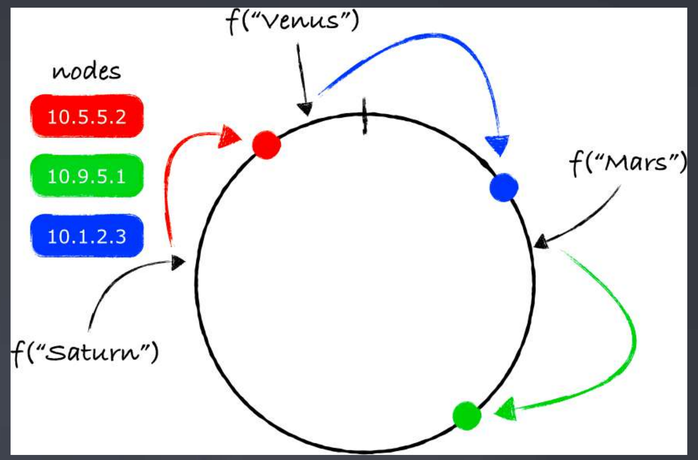
- 比如信息流时间线
- ID放在redis的sorted set
- 前面加，后面减
- 单列DB查询+缓存
- mapreduce并发请求

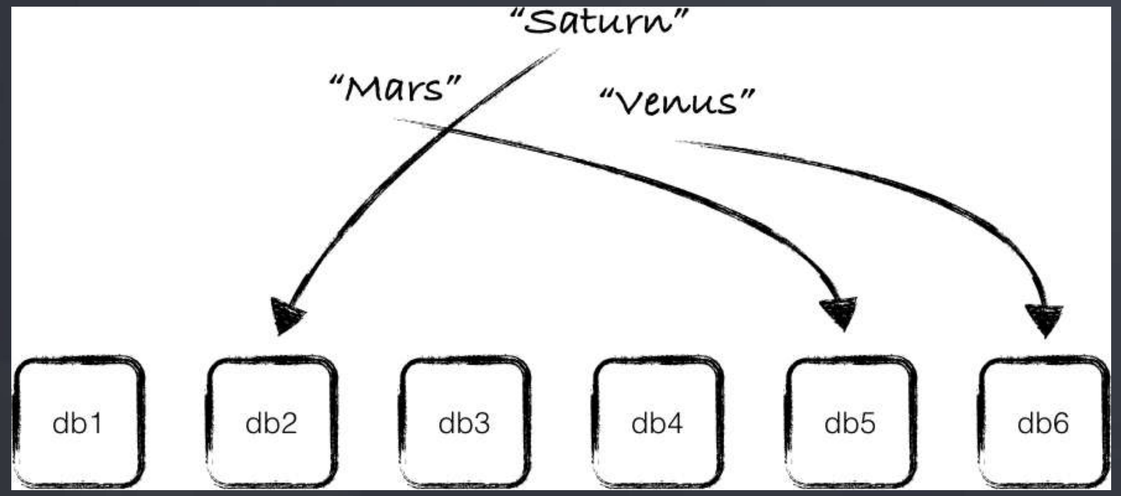


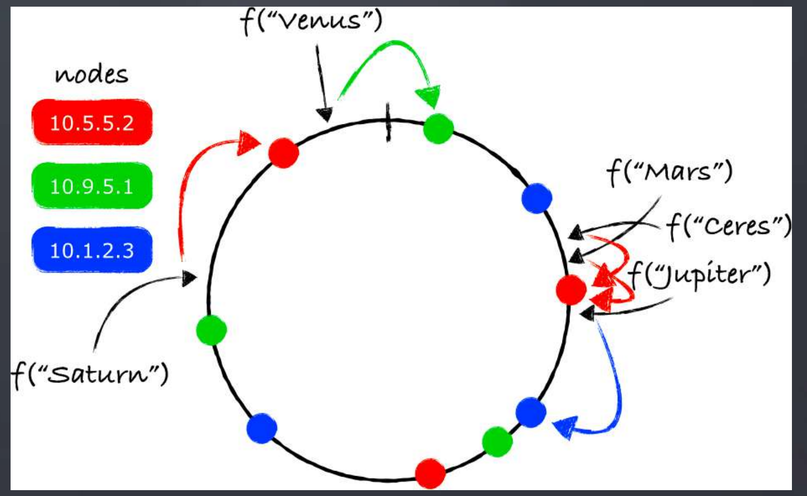
分布式缓存系统设计

缓存分发算法









https://blog.carlosgaldino.com/consistent-hashing.html

缓存集群成本控制

- 数据量 vs QPS
- 数据量小、QPS高 => 多实例和分片
- Case study
  - 1TB redis cluster
  - 128GB redis clusters (16GB*8)

缓存代码自动化实践

代码正确性 – 传统写法 vs goctl生成

- 怎么存放缓存数据
- 考虑缓存穿透、击穿、雪崩
- 如何保证数据一致性
- 如何监控缓存使用情况
- 手撸代码越多，bug越多

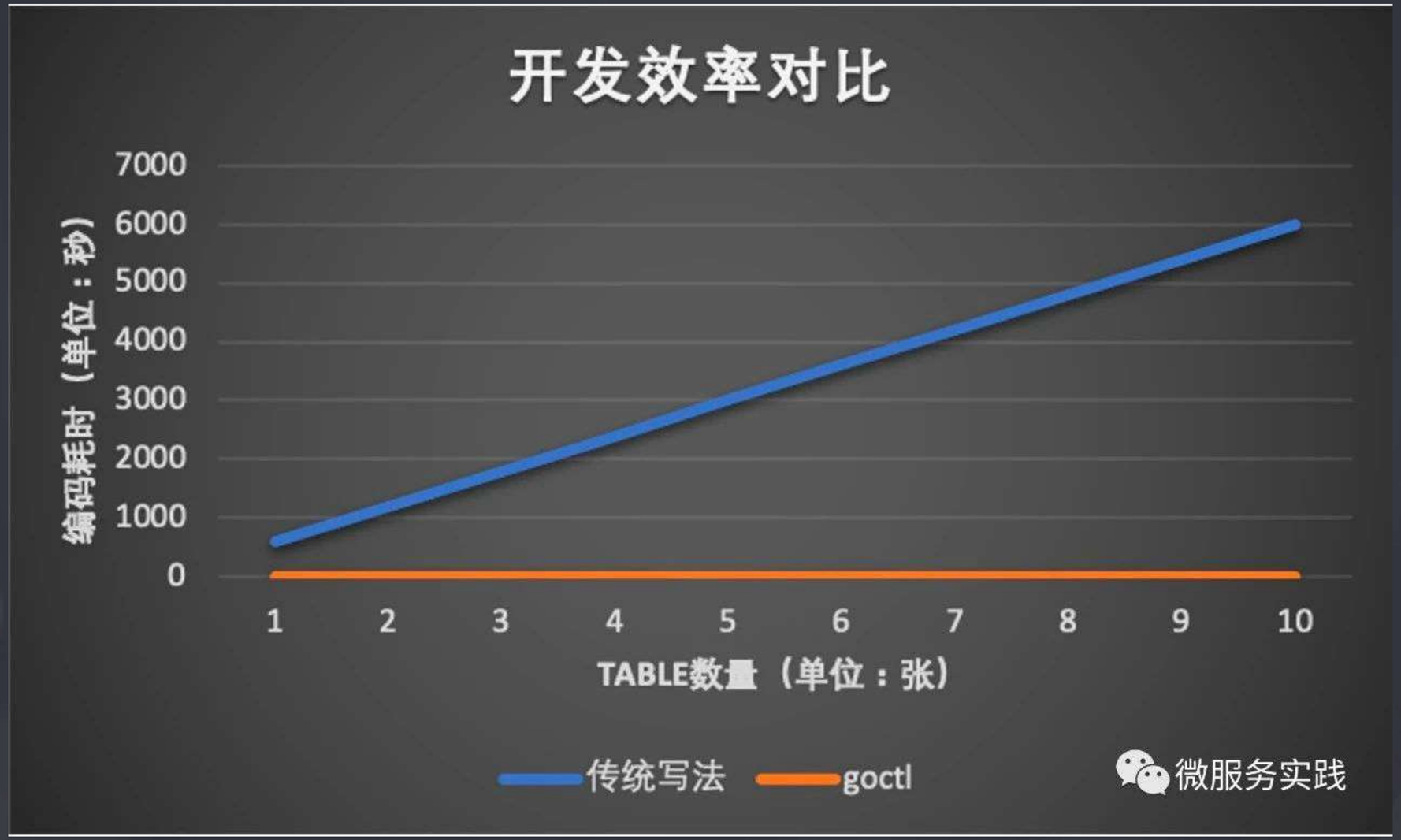
- sqlc，mongoc自动管理缓存
- Nothing else!

开发效率 – 传统写法 vs goctl生成

- 10分钟一个表的CRUD
- 30分钟一个表的缓存代码
- table –> 结构体
- code review, test

- ddl（sql文件）一键生成
- 同一DB下多表一次生成
- 自动生成sql，自动检测问题

开发效率 – 传统写法 vs goctl生成



懂原理，用工具

觉得不错？ star 鼓励一下！

https://github.com/tal-tech/go-zero