晓黑板Go微服务框架

Kevin@晓黑板

晓黑板服务之前世 - *单体服务的困局*

- 乱！
- Java + MongoDB
- 无人可以驾驭
- 很难新增功能
- 最致命的是频繁持久的宕机

~~单体架构~~  微服务架构

开着飞机换引擎

- ***最重要的是决心！***
- 连我6人Go小团队

怎么换？

- Proxy分流验证
- 数据一致性保障，双向同步

架构设计的几点思考

- 架构设计不能脱离业务
- 能够快速定位服务问题
- PHP? Java or Golang?
- 开源 or 自研？

当时考虑的一些设计原则

- 保持简单，第一法则
- 高可用
- 高并发
- 易扩展
- 弹性设计
- 封装微服务复杂度，业务开发友好
- 做一件事只提供一种方式

Go微服务框架的实现

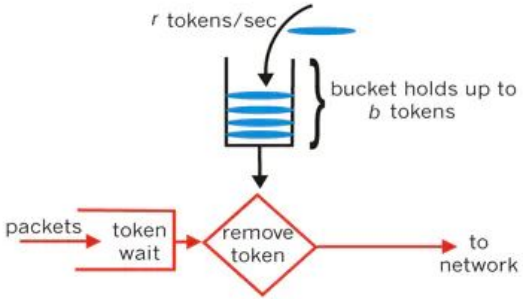
微服务重要组件

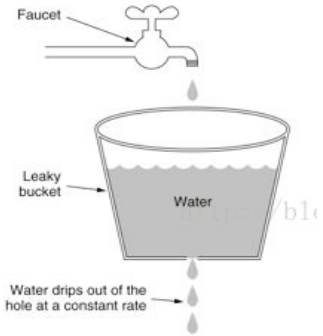
- 限流
- 自适应降载
- 自适应熔断
- 负载均衡
- 超时控制
- 缓存
- 可观测性

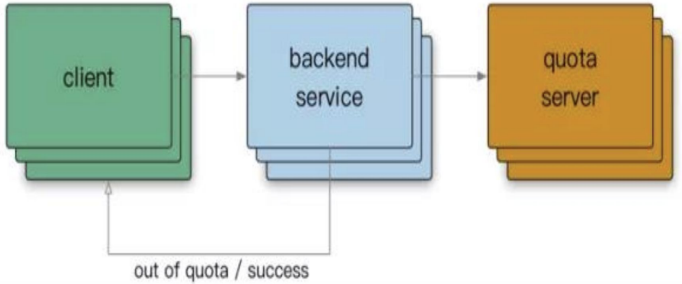
限流

进程内限流

- 控制并发请求量
- 简单高效
- 可配置，有默认值
- 有效防止突发恶意流量
- 第一道防护（WAF等除外）





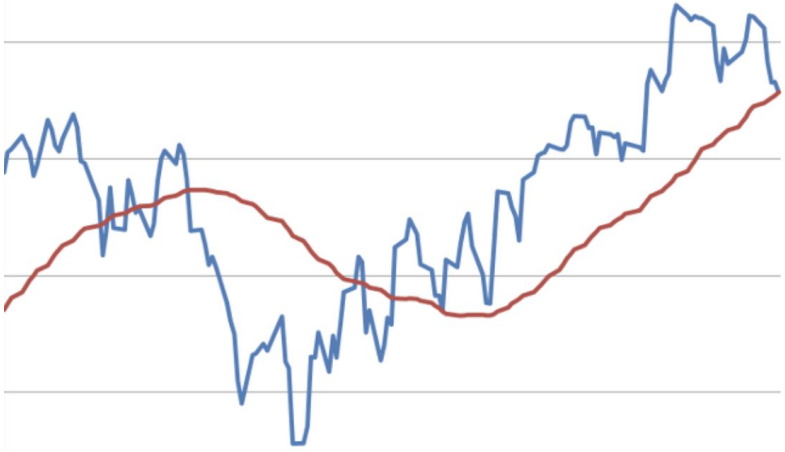


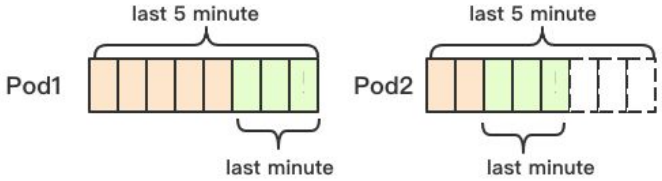
分布式限流

- 基于redis/lua
- 令牌桶
- 漏桶
- 广泛使用，短信、推送等

自适应分级降载

$$min(InFlight, MovingAvg(InFlight)) > MaxPass \times AvgRT$$



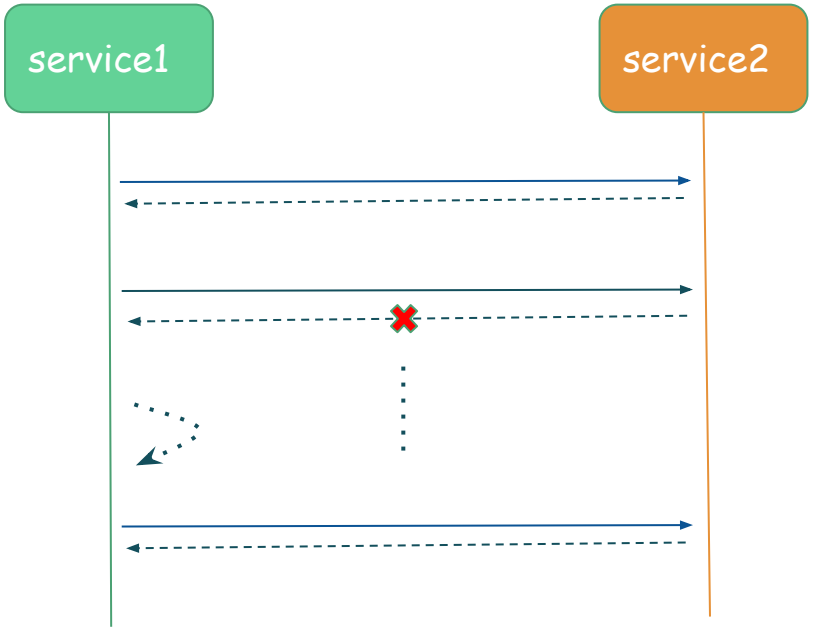


基于优先级进行分级降载保护

- CPU>90%开始拒绝低优先级请求
- CPU>95%开始拒绝高优先级请求
- K8S的HPA 80%触发
- http/rpc框架内建
- 基于滑动窗口，防止毛刺
- 有冷却时间，防止抖动
- 实践检验，配合K8S弹性伸缩
- 第三道防护

自适应熔断

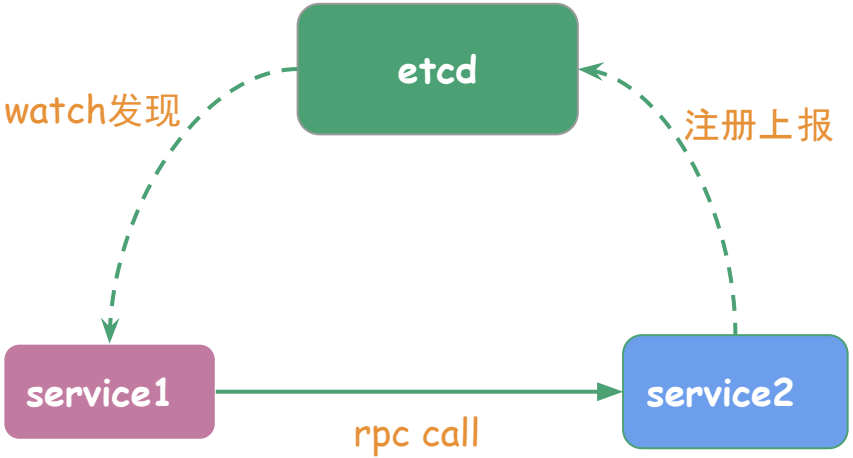
$$dropRatio = max(0, \frac{(requests - protection) - K \times accepts}{requests + 1})$$



路径级别的自适应熔断

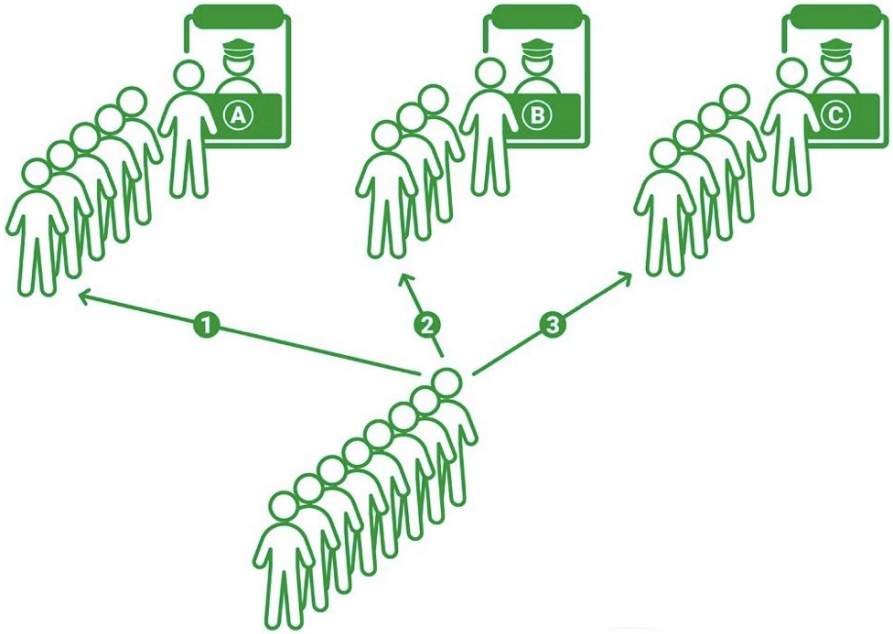
- 自动触发，自动恢复
- http/rpc框架内建
- Google SRE算法
- 基于滑动窗口（10秒/40窗口）
- 支持自定义触发条件
- 支持自定义fallback
- 第二道防护

负载均衡



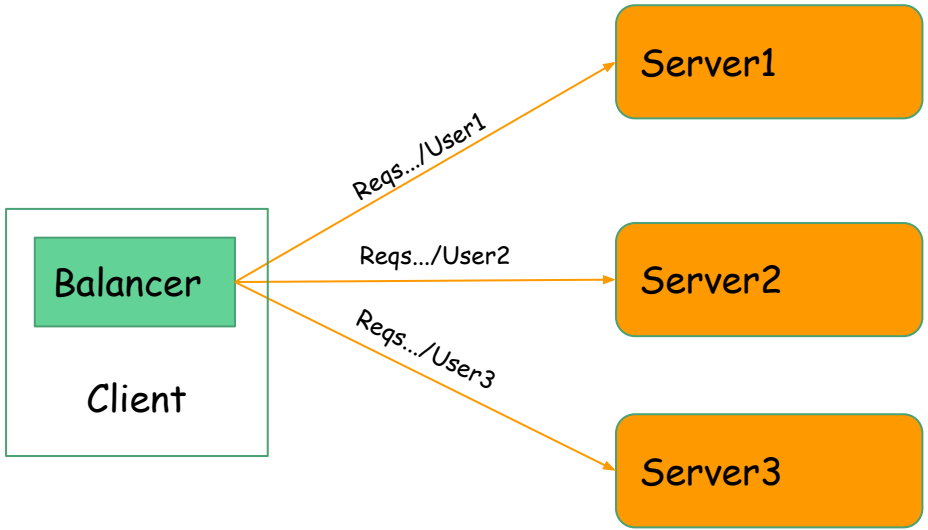
负载均衡基于服务发现

- 服务提供方注册上报
- 服务调用方watch发现
- 进行服务调用



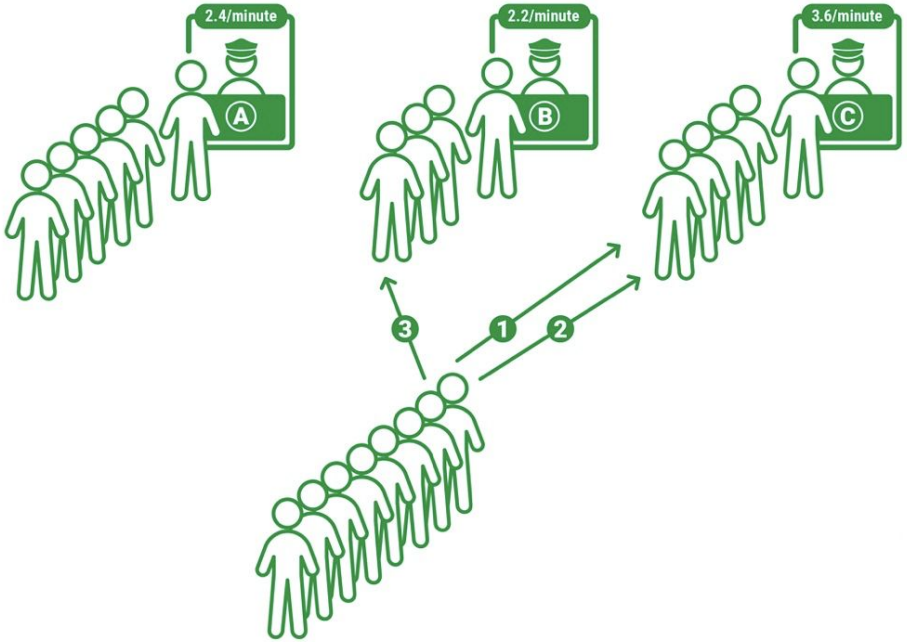
Round Robin

- 简单
- 要求后端性能均等
- 响应时间变化较大时，容易导致负载不平衡



(batch)一致性分发

- 基于hash ring
- 便于基于实体增加进程内缓存
- 支持最小化放量，预热
- 少量高热点请求容易导致后端负载不均衡



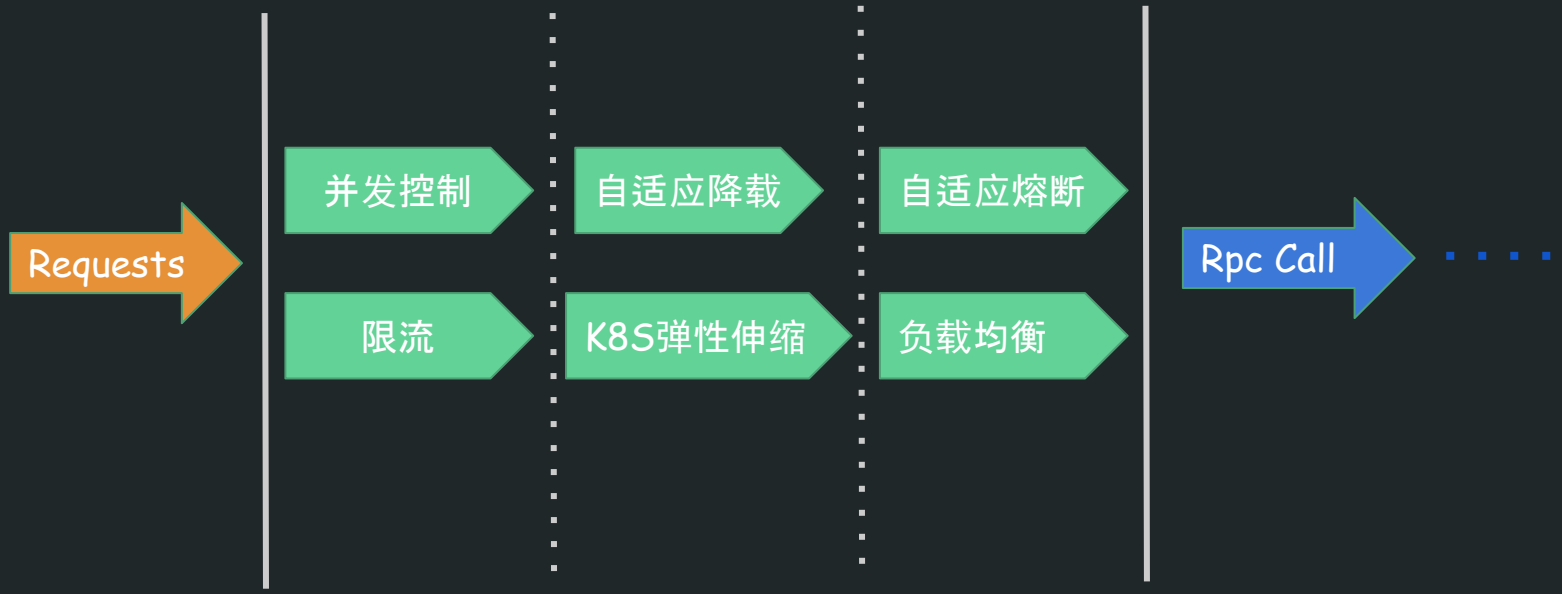
Power of Two Choices（将实现）

- 当前请求数
- 处理时长，指数加权移动平均

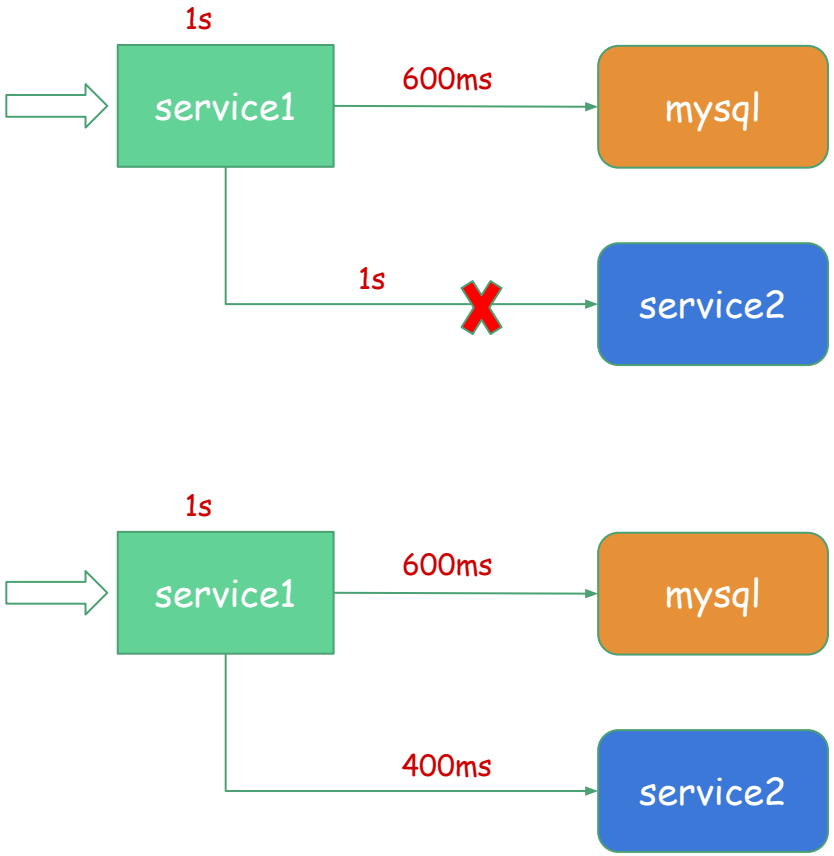
参考自Nginx & Envoy & Finagle：

- https://www.nginx.com/blog/nginx-power-of-two-choices-load-balancing-algorithm/

多重防护，保障高可用

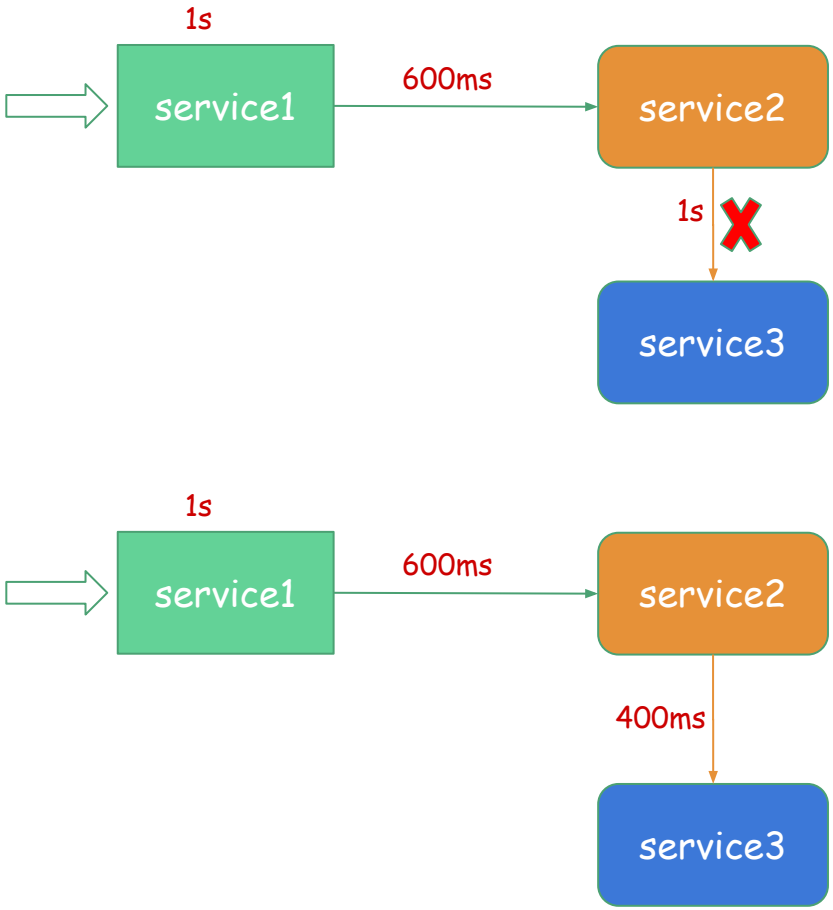


超时控制



进程内超时控制

- Fail Fast，少做无用功
- 有默认值，业务开发无需关注
- 注意和客户端协同



服务间超时控制

- Fail Fast，少做无用功
- 通过context自动传递
- 忽略服务间调用网络损耗(一般几ms)，降低复杂度

重试？ *不自动重试！*

重试机制注意事项

- 指数退避
- 流量quota
- 超时相关性

http请求自动解析校验

```
type createRequest {
    name    string `path:"name"`
    age     int    `form:"age,default=18,range=[18:100)"`
    role    string `json:"role,options=teacher|parent"`
    address string `json:"address,optional"`
}

service user-api {
    @doc(
        summary: create user
    )
    @server(
        handler: CreateUserHandler
    )
    post /api/users/create(createRequest)
}      You, a few seconds ago • Uncommitted changes
```

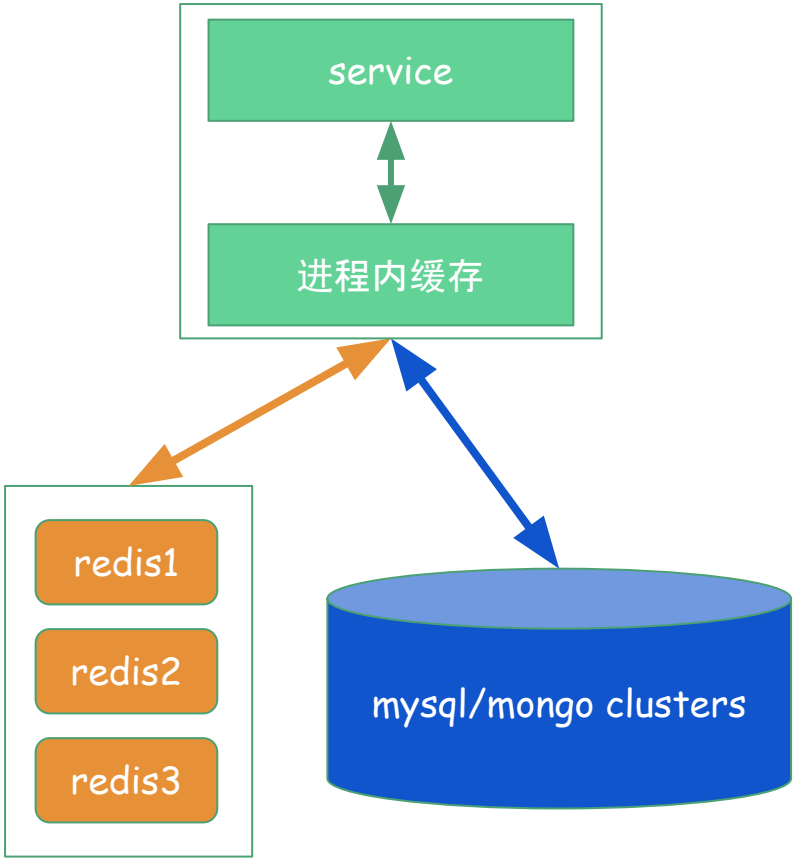
支持的特性

- path, form, json
- default, optional, options, range

解析校验

- httpx.Parse(...)
- 参数错误自动返回400
- 配合goctl使用，效果更佳

缓存



缓存设计三要点

- 缓存穿透，不存在的数据
- 缓存击穿，热点key过期
- 缓存雪崩，大量缓存同时过期

我们的解决方法

- 缓存穿透
  - 即使查不到，也自动缓存，短过期时间，1分钟
- 缓存击穿
  - 确保一个进程同时只拿一次数据，并共享结果
- 缓存雪崩
  - 针对缓存过期时间设置随机偏差，确保分散过期
- 缓存基于非主键的查询
  - 转换为主键做缓存
  - 获取复杂，查询->主键->缓存
  - 过期复杂，顺序重要
- 分布式缓存
  - 多虚拟节点一致性hash，避免升降级后过多cache miss
- 很难全做对，是不是？
  - 我们全部自动化了，goctl自动生成CRUD+cache代码，省力不出错
  - 自带sql慢查询记录和缓存命中率统计

可观测性

- 链路跟踪
- Logging
- Metrics
- 监控报警

traceid	spanid	app	start_time	end_time	flag
B3E484E78DEA	0	打卡服务	2020-07-01 10:00:00:0000	2020-07-01 10:00:00:0070	server
B3E484E78DEA	0.1	打卡服务	2020-07-01 10:00:00:0005	2020-07-01 10:00:00:0045	client
B3E484E78DEA	0.1	班级服务	2020-07-01 10:00:00:0015	2020-07-01 10:00:00:0040	server
B3E484E78DEA	0.1.1	班级服务	2020-07-01 10:00:00:0020	2020-07-01 10:00:00:0035	client
B3E484E78DEA	0.1.1	用户服务	2020-07-01 10:00:00:0025	2020-07-01 10:00:00:0030	server
B3E484E78DEA	0.2	打卡服务	2020-07-01 10:00:00:0050	2020-07-01 10:00:00:65	client
B3E484E78DEA	0.2	通知服务	2020-07-01 10:00:00:0055	2020-07-01 10:00:00:60	server

链路追踪

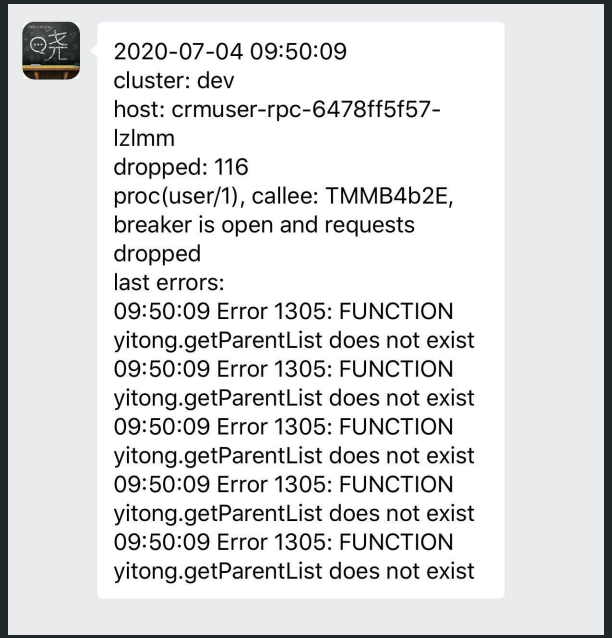
- 框架内建，context传递
- Trace id，贯穿整个调用链
- Span id，有层级和时序关系
- 记录起止时间
- 记录调用关系，client/server

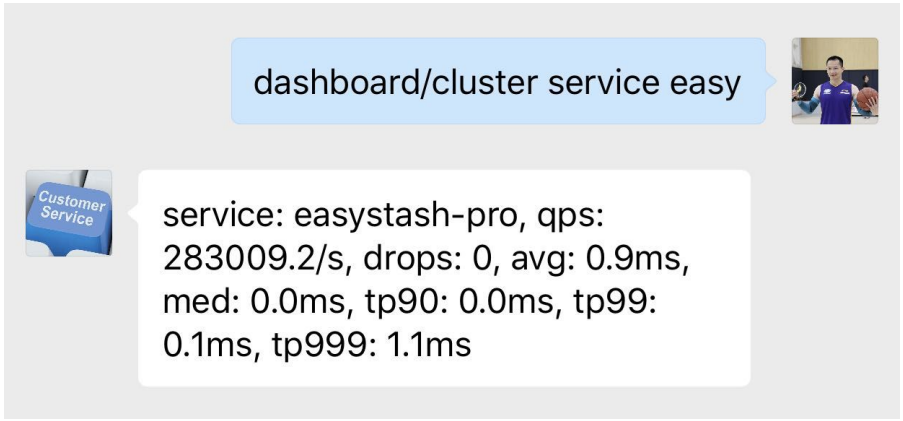
Logging

- 自动rotate
- 多模式支持，console, file, volume
- 自动压缩
- 自动删除过期日志

监控报警

- 自动聚合汇报异常，比如http code 5xx
- 自动控制频率并汇总异常





数据上报

- 上报到自建dashboard服务
- 上报到prometheus

A lot more...

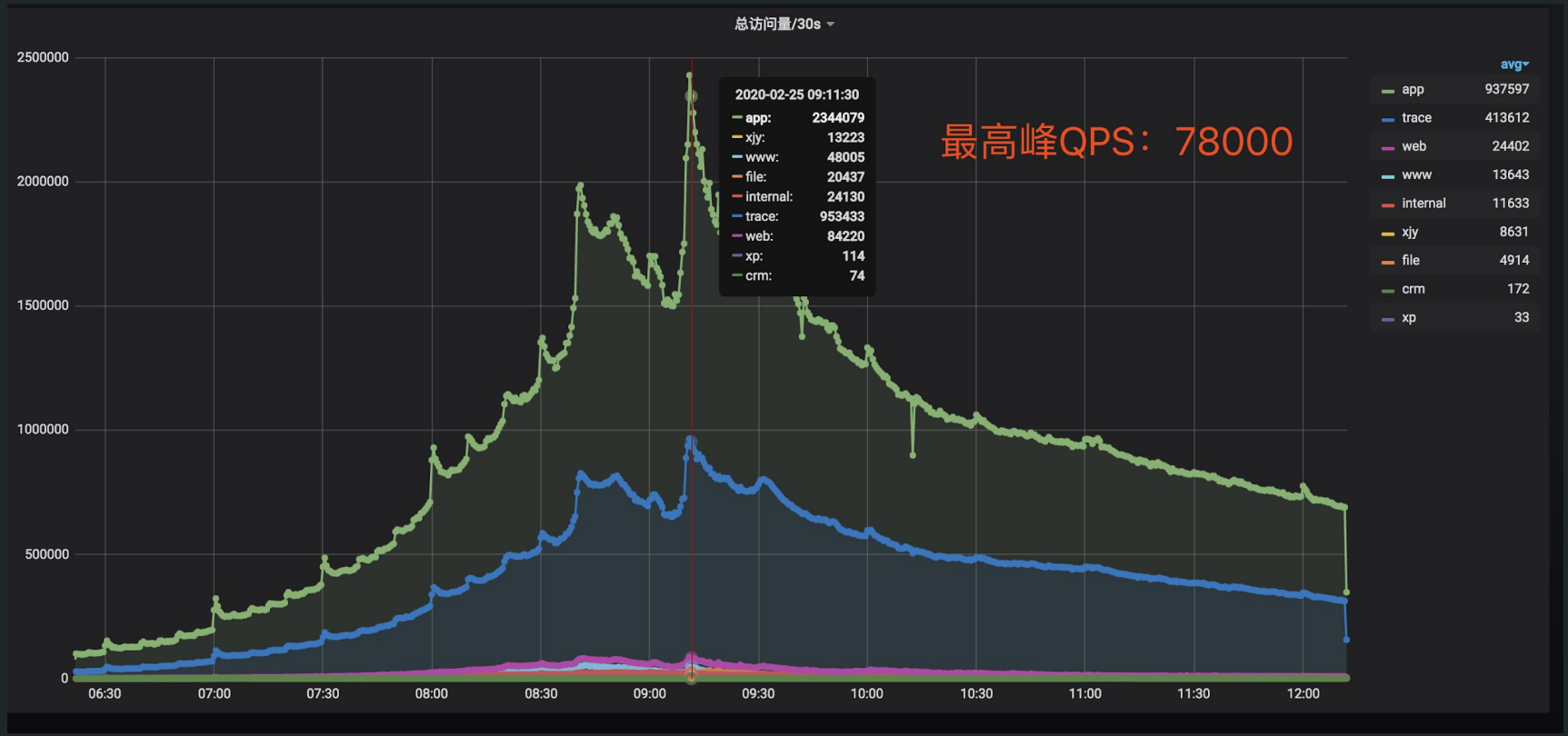
- 基于JWT的自动鉴权
- MapReduce
- Graceful shutdown
- 并发控制工具箱
- 资源控制工具箱, 比如多个线程同时创建同一个数据库链接
- 分布式高可用延迟任务框架
- 极简Kafka Pub/Sub框架
- Logstash 5倍性能的easystash框架
- Find more from code...

新框架的收益

我们获得了什么收益？

- 18年9月上线至今再未出现大规模宕机
- 稳定支撑了多次大规模用户增长和尖峰流量
- 疫情期间峰值630万日活，且流量非常集中
- 开发效率大幅提升
- 新特性快速满足，源于自研
- 开发了基于自研框架的工具链，极大提升开发效率
- 团队得到了很好的成长
- 形成了良好的工程师文化

疫情下的入口流量图

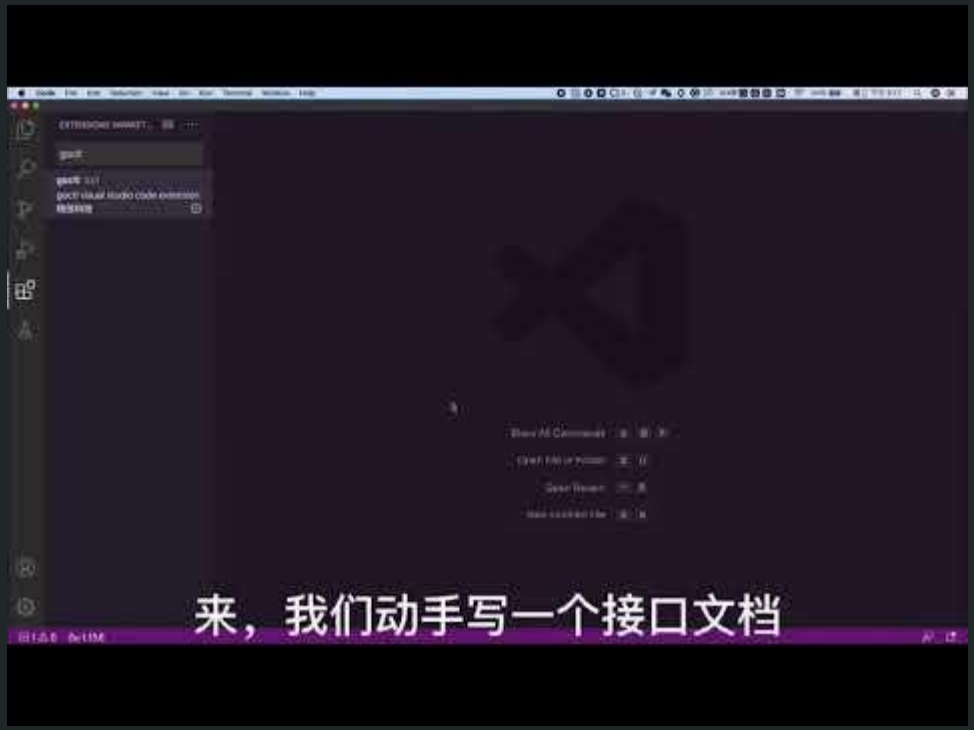

总访问量/30s
2500000
2000000
1500000
1000000
500000
0
06:30
07:00
07:30
08:00
08:30
09:00
09:30
10:00
10:30
11:00
11:30
12:00
2020-02-25 09:11:30
app: 2344079
xjy: 13223
www: 48005
file: 20437
internal: 24130
trace: 953433
web: 84220
xp: 114
crm: 74
最高峰QPS：78000
avg
app 937597
trace 413612
web 24402
www 13643
internal 11633
xjy 8631
file 4914
crm 172
xp 33

工具大于约定和文档

goctl工具

- 极简API描述语法
- 极大简化前后端代码编写
- 减少沟通，避免出错
- 自动生成Golang后端Restful API、RPC代码、iOS, Android, TypeScript, Dart, JS代码，并可直接运行
- 自动生成CRUD+cache代码
- 自动生成docker, k8s部署文件

goctl代码生成演示



谢谢！

简单，是终极的复杂！