Go在百度BFE的应用

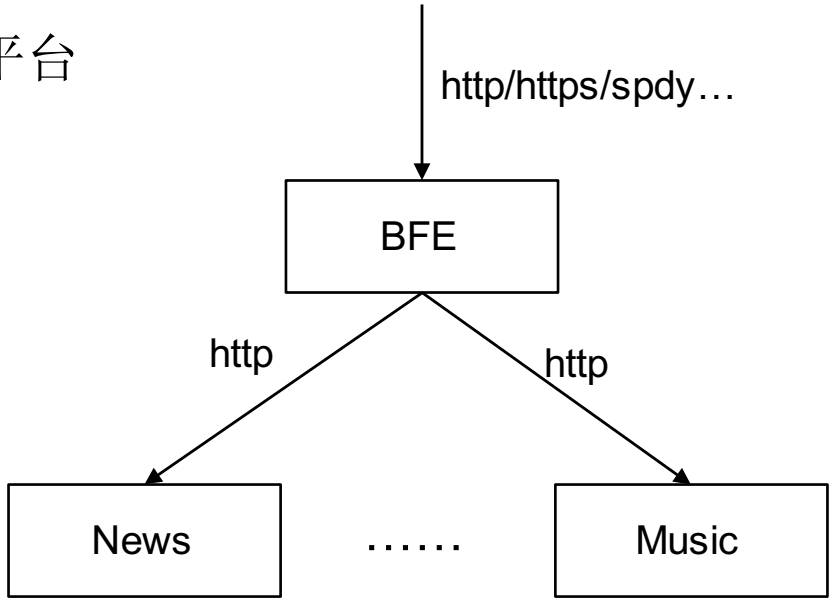
百度运维部 李炳毅
libingyi@baidu.com
2016年4月

个人简介

- 李炳毅，运维部，Baidu FrontEnd（BFE）团队成员
- 2010年7月，加入百度
  - 工作方向集中在流量接入、安全、防攻击等方面
  - 使用Go开发的项目
    - 7层流量代理Go-BFE
    - 应用层防火墙WAF
- 百度Golang委员会委员

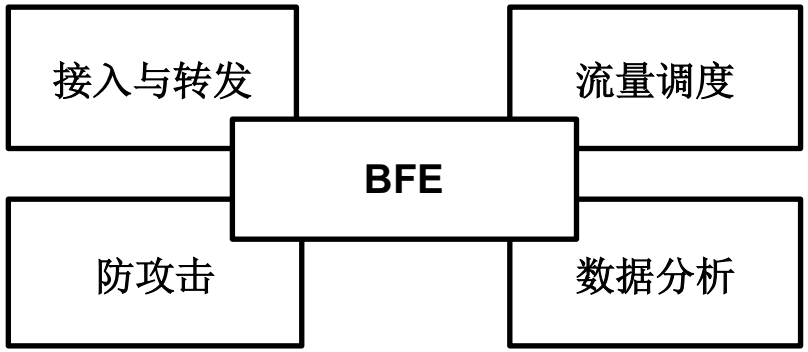
BFE（Baidu FrontEnd）

- 百度统一前端
  - 七层流量接入平台



BFE（Baidu FrontEnd）

- 主要服务
  - 接入和转发
  - 防攻击、流量调度、数据分析
- 服务现状
  - 覆盖大部分重要产品线
  - 日请求量：千亿级别



为什么重写BFE

- BFE的发展
  - Transmit -> Utr（2008）-> C-BFE（2012）-> Go-BFE（2014）
- 现有问题
  - 修改成本高
    - 事件驱动的编程模型：编码和调试难度大
    - C语言本身的难度和开发效率
  - 配置管理方式和落后
    - 为单产品线设计，无法支持平台化能力
    - 配置变更（修改、重载和验证）的能力差
  - 变更和稳定性矛盾突出

技术选型

- 学习成本
- 开发成本
  - 并发编程模型：同步（Go）VS 异步（Nginx）
  - 内存管理
  - 语言描述能力
- 性能
  - 在BFE的场景下，性能在可接受的范围内
  - 通过算法设计和架构来弥补

几个问题

- GC优化
- http协议栈
- 分布式架构
- 好用的工具链

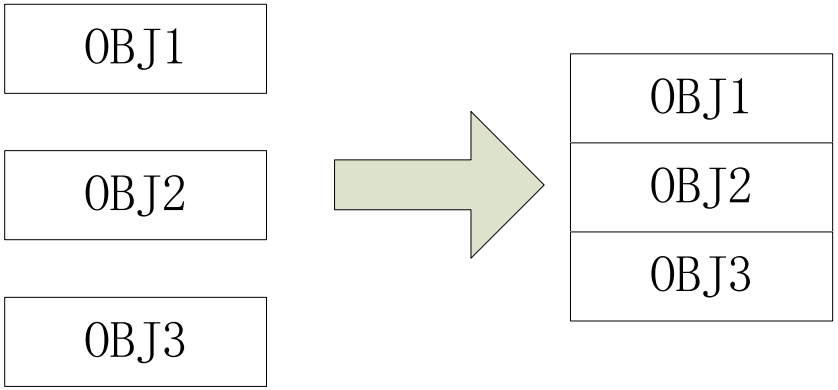
GC带来的问题

- GC是个好东西，但也有问题
- 难以避免的延迟（几十到几百ms）
  - 经验公式：10万对象1ms扫描时间（Go1.3）
    - 1个tcp连接，约10个对象 -> 1万连接，1ms GC延迟
- GO-BFE的实时需求
  - 请求的处理延迟平均1ms以内，最大10ms
- 实测
  - 100万连接，400ms GC延迟

GC优化思路

- Go的gc算法
  - Mark and Sweep：大量时间用于扫描对象
- 常规手段的核心：减少对象数
  - 小对象合并成大对象
    - 利用Array来合并一组对象（内部对象计数为1）
  - 把数据放到c代码里面，通过cgo做接口调用
  - 对象复用（对象池）
  - 深度优化系统结构和算法

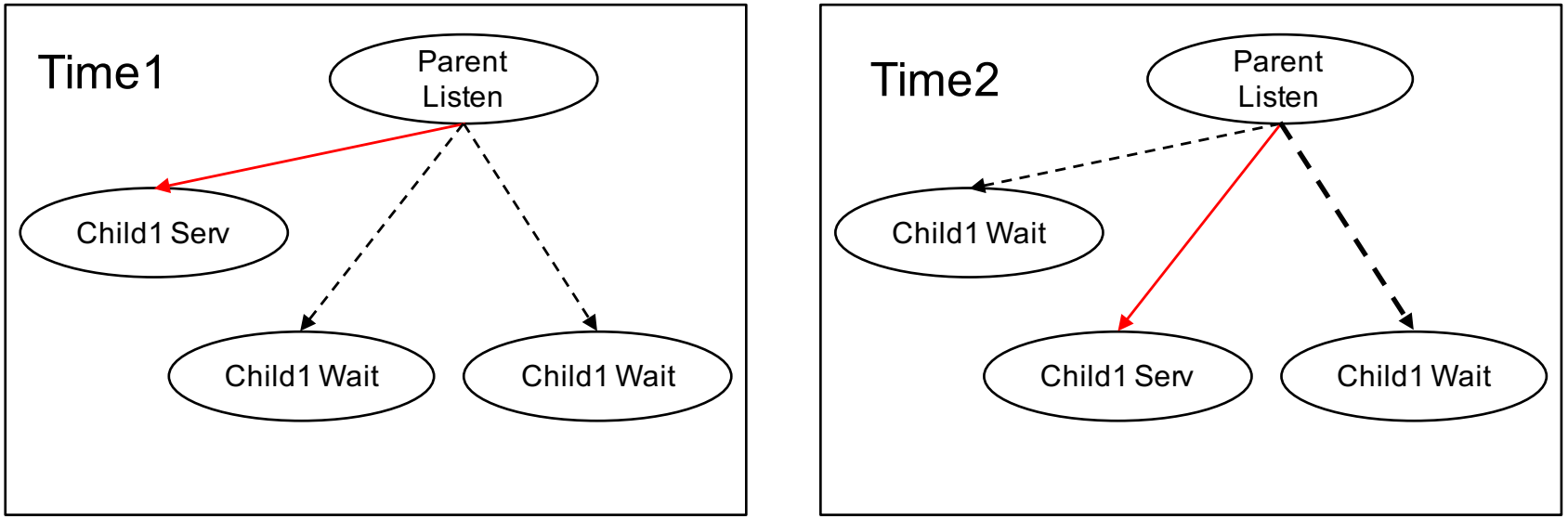
通过Array减少引用技术



困境

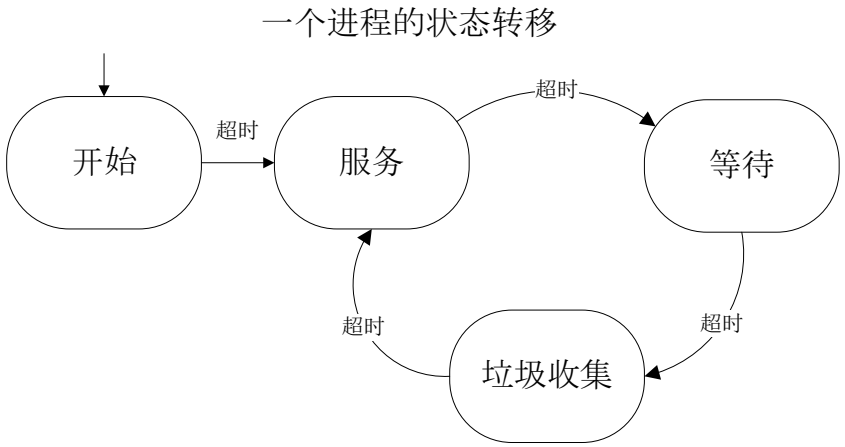
- 减小对象数的困境
  - 常态下需要保持几十万的连接 => 几十ms
  - 修改golang网络库，重写基本数据结构
- 不让go管理内存
  - 通过cgo手工维护，很危险
  - 不能解决问题：大量go对象难以避免

车轮大战

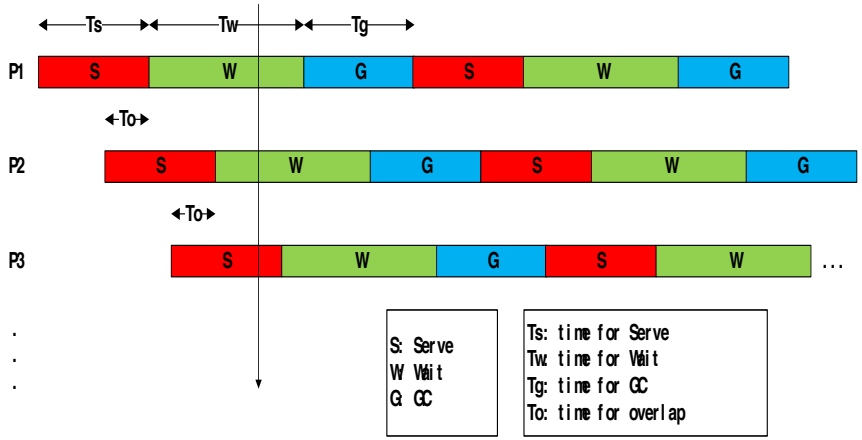

Time1
Parent
Listen
Child1 Serv
Child1 Wait
Child1 Wait
Time2
Parent
Listen
Child1 Wait
Child1 Serv
Child1 Wait

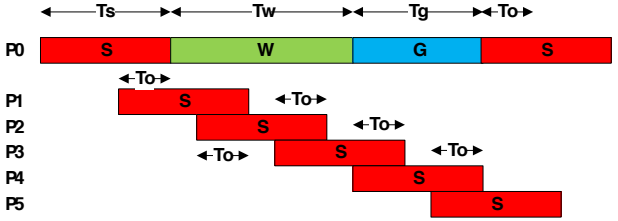
轮转GC方案

- 基本思路
  - 关闭自动GC
  - 多进程轮流工作
- 单进程状态
  - 服务态
  - 等待态
  - 垃圾回收状态



GC优化 - 多进程配合





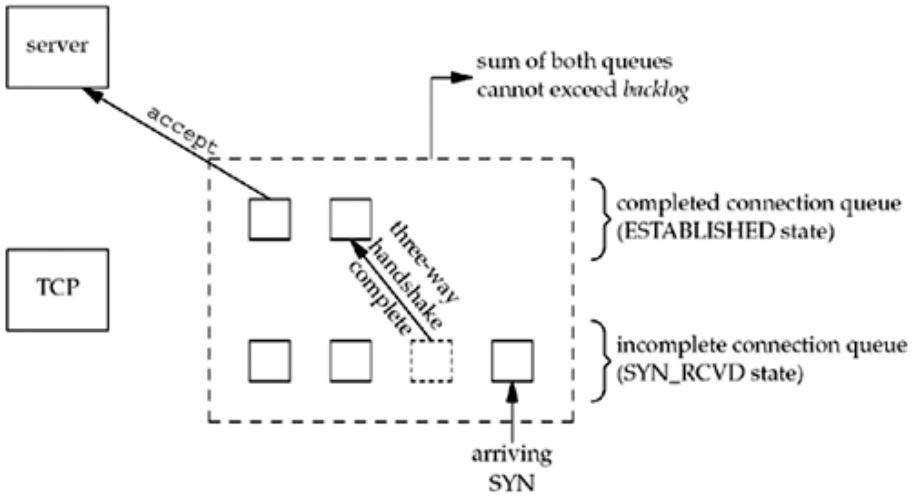
进程数的计算：$N = 1 + \left\lceil \frac{Tw + Tg + To}{Ts - To} \right\rceil$

技术细节

- 本质上：多个进程监听同一个端口
  - 搞版本linux直接支持
  - 低版本linux方案
    - 父进程Listen
    - 子进程Accept

```
#include <sys/socket.h>

#int listen (int sockfd, int backlog);
```



技术细节

- 服务态
  - 调用Accept，获取新的请求
- 等待态
  - 不调用Accept，已经连接的client，可以继续收发
  - 等待这些已有的连接关闭
- 垃圾收集态
  - 主动调用GC

GC优化 - 补充分析

- HTTP场景
  - 短连接
  - 长连接
    - 平均连接上的请求是3个
    - 90%（20s以内）、98%（50s以内）
  - 大文件请求
    - 对GC造成的延迟（几十ms）不敏感

多说一句Go 1.5：没有银弹

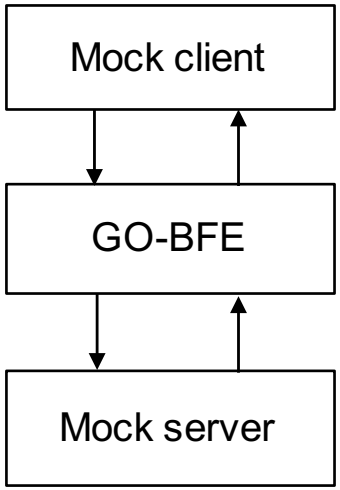
- Stop-The-World（STW）缩短了，决定因素也变了
  - Time spent looping over goroutines
  - Time spent looping over malloc spans
- 实际运行，还是有几十ms的STW
  - GO-BFE的场景和服务模式，大量的goroutine必然存在
- 需要根据业务实际情况做选型

协议一致性问题

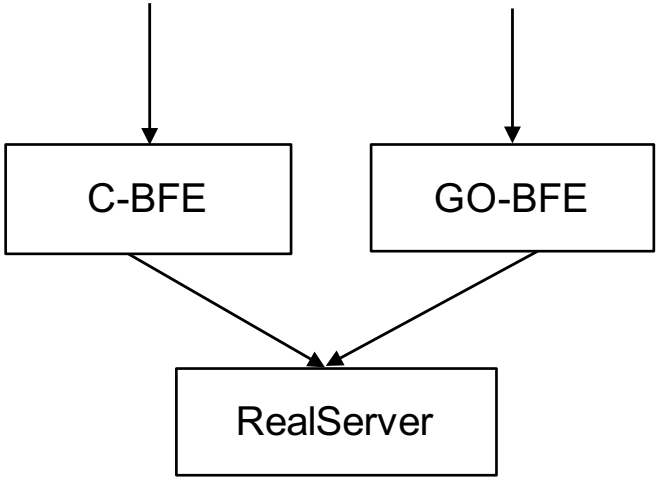
- GO-BFE参考了Go的http库
- 基于Go的http实现是否完善，符合rfc标准
  - 没有大规模应用的例子
- 需要一些方法来验证
  - 网络协议一致性测试是难点

协议一致性

- Macaroon



- Tcpcopy线上引流对比

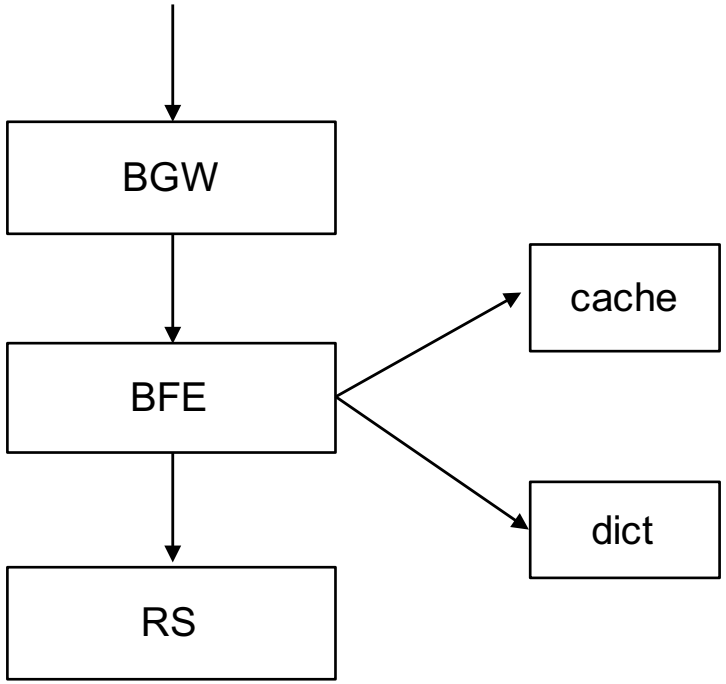


一个例子

- url encode case
  - http://xxx.baidu.com/item/JELIM+PLASTIC+SURGERY+%26+AESTHETIC
  - http://xxx.baidu.com/item/JELIM+PLASTIC+SURGERY+&+AESTHETIC

分布式架构

- BFE程序架构：core + 众多功能模块
  - 分流
  - cache
  - dict
- 问题
  - 变更频率
  - 启停速度
  - 功能单一，各自扩展
- 同步/异步，开发效率4:1

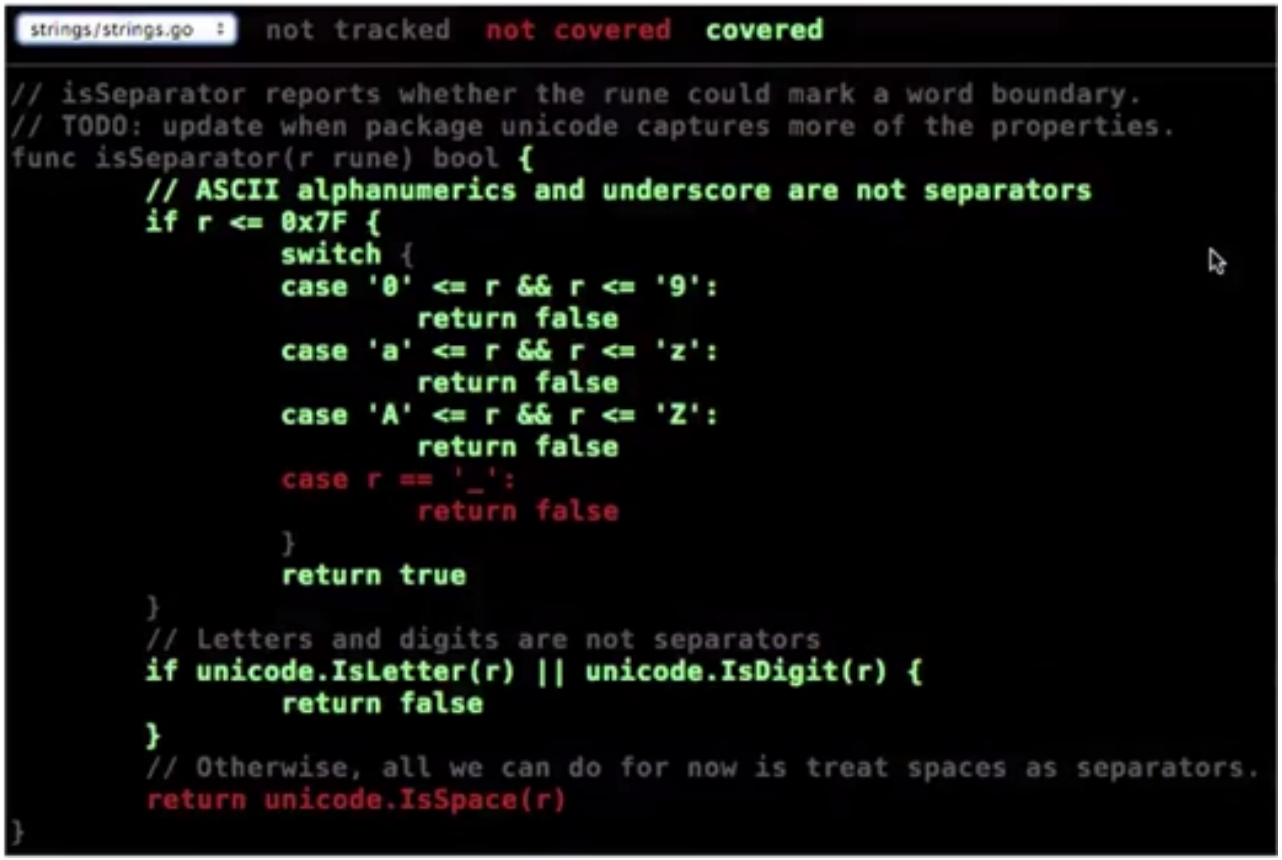


Go工具链的一些分享

- 测试
- 程序性能调优
- 服务内部状态暴露

测试

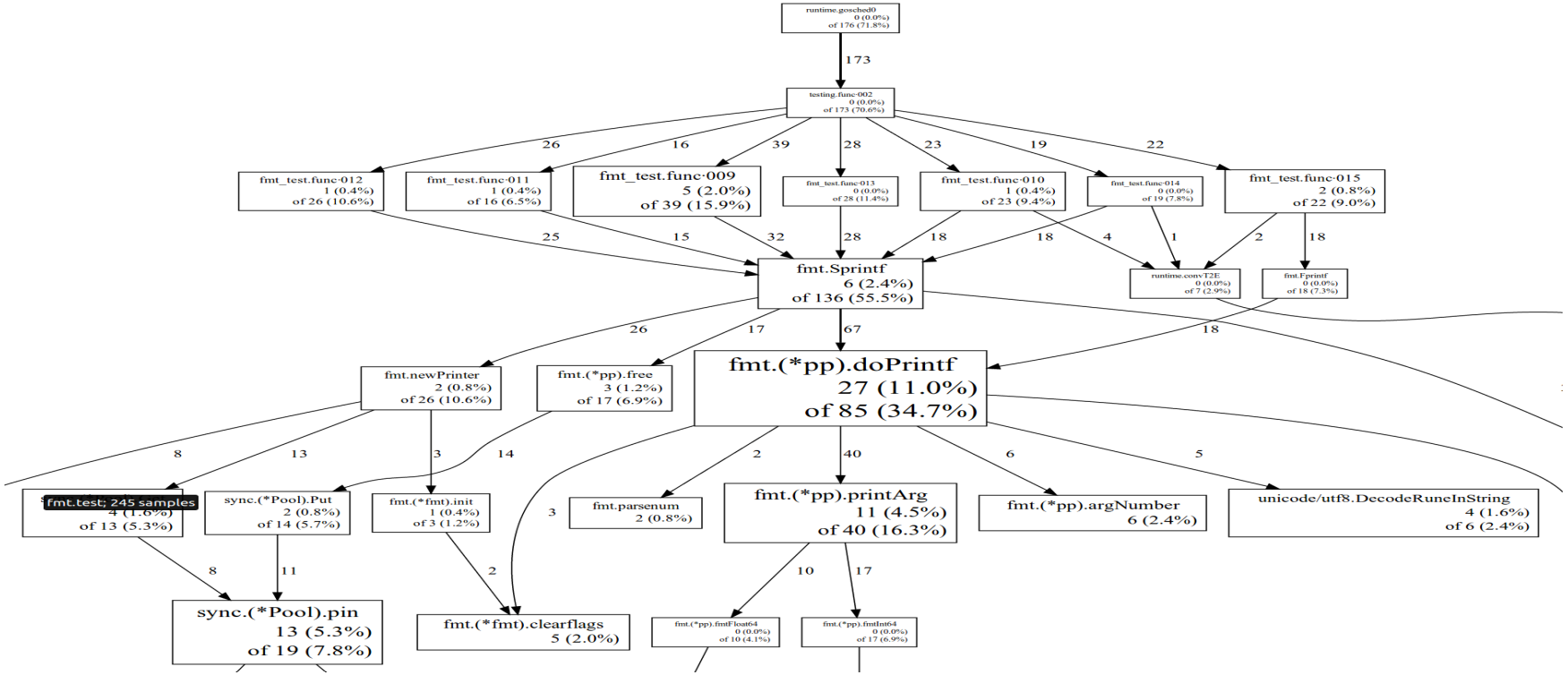
- Go自带测试框架
- quic/slow
- ut/benchmark
- race detection
- Html coverage



调优(pprof)

- 易用
  - 一行代码：import “net/http/pprof”
  - 不需要编译不同版本
- 全面
  - Cpu/mem、blocking、goroutine、gc
- 可读
  - Call graph
  - 定位到代码行

调优(pprof)


runtime.gosched0
0 (0.0%)
of 176 (71.8%)
173
testing.func·002
0 (0.0%)
of 173 (70.6%)
26
16
39
28
23
19
22
fmt_test.func·012
1 (0.4%)
of 26 (10.6%)
fmt_test.func·011
1 (0.4%)
of 16 (6.5%)
fmt_test.func·009
5 (2.0%)
of 39 (15.9%)
fmt_test.func·013
0 (0.0%)
of 28 (11.4%)
fmt_test.func·010
1 (0.4%)
of 23 (9.4%)
fmt_test.func·014
0 (0.0%)
of 19 (7.8%)
fmt_test.func·015
2 (0.8%)
of 22 (9.0%)
25
15
32
28
18
18
4
1
2
18
fmt.Sprintf
6 (2.4%)
of 136 (55.5%)
runtime.convT2E
0 (0.0%)
of 7 (2.9%)
fmt.Fprintf
0 (0.0%)
of 18 (7.3%)
26
17
67
18
fmt.newPrinter
2 (0.8%)
of 26 (10.6%)
fmt.(*pp).free
3 (1.2%)
of 17 (6.9%)
fmt.(*pp).doPrintf
27 (11.0%)
of 85 (34.7%)
8
13
3
14
2
40
6
5
fmt.test; 245 samples
4 (1.6%)
of 13 (5.3%)
sync.(*Pool).Put
2 (0.8%)
of 14 (5.7%)
fmt.(*fmt).init
1 (0.4%)
of 3 (1.2%)
3
fmt.parsenum
2 (0.8%)
fmt.(*pp).printArg
11 (4.5%)
of 40 (16.3%)
fmt.(*pp).argNumber
6 (2.4%)
unicode/utf8.DecodeRuneInString
4 (1.6%)
of 6 (2.4%)
8
11
2
10
17
sync.(*Pool).pin
13 (5.3%)
of 19 (7.8%)
fmt.(*fmt).clearflags
5 (2.0%)
fmt.(*pp).fmtFloat64
0 (0.0%)
of 10 (4.1%)
fmt.(*pp).fmtInt64
0 (0.0%)
of 17 (6.9%)

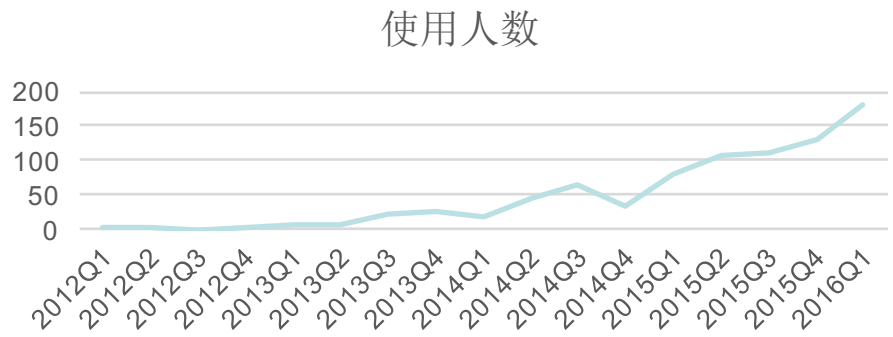
运营的考虑- 程序内部状态暴露

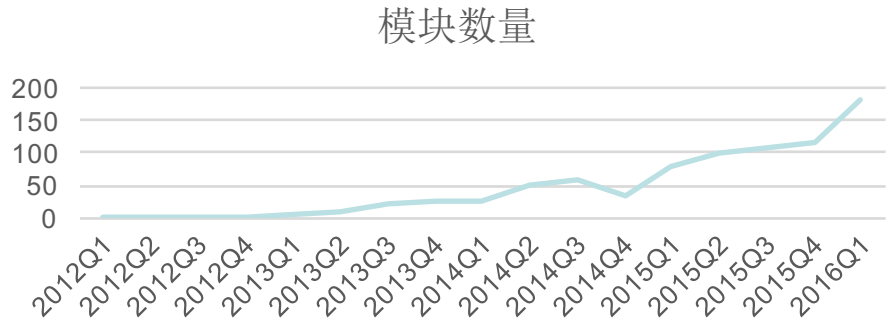
- 内置http server
- 多重格式输出：json、noah
- 指标
  - 数量多：4000+
  - 类型丰富：
    - 计数器、分布漏桶、累计量、切片量、文本、状态
  - Feature核心指标
  - 用户特征指标

baidu推广Go，我们做了什么

- Done
  - 成立Baidu Golang编程规范委员会
  - 发布Baidu Golang编程规范
  - 组织Golang Good Cooder考试
- Doing
  - Baidu Golang Lib

Go在Baidu的应用





总结

- Go可以用于高并发、低延迟的程序开发
- Go极大的提升了开发效率

Thanks

Q & A