Golang在BFE的应用

百度运维部 陶春华

taochunhua@baidu.com

2016年4月

个人简介

- 陶春华，运维部，Baidu Front End团队
  - 2010年，天津大学，计算机专业博士
- 2013年7月，加入百度
  - 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委员会成员

内容提要

- 后台程序开发的需求和难点
  - C, Python and Go对比
- 采用Go语言重构BFE
  - 背景和技术路线
  - GC问题
  - 协议一致性
  - 分布式架构

后台程序开发的需求(1)

- 性能
  - C/C++, Java
  - Python, Ruby
- 并发性
  - Process, Thread, Event(编程难度)
- 开发效率
  - 语言的描述效率：代码量
  - 语言的简洁、易用
  - 库支持

后台程序开发的需求(2)

- 大型程序的组织
  - 数据封装能力
  - Namespace
- 可测试能力
  - 单测，覆盖度测量
- 错误检查能力
  - 编译
  - 程序异常的trouble shooting

后台程序开发的需求(3)

- 上线和运维
  - 对运行环境的依赖
  - 对库（动态库）的依赖

后台程序编程的难点

- 内存的管理
  - C程序中很大比例的Bug和内容有关
- 分布式/高并发的处理
  - 10年前还是一个很hot的话题；目前也还没有普遍掌握
  - CPU资源的调度：Process/Thread/Event
  - 数据的封装和互斥访问；
  - 并行运算逻辑的同步

C vs Python (1)

- 性能：
  - 相差10倍以上
  - Python: 解释执行，动态类型
- 并发性能
  - C：直接用系统的机制
  - Python: 自己实现的thread, 只能使用一个CPU
- 开发效率
  - 相差5-10倍
  - 内存的处理是一个难点
  - dict/map, list

C vs Python (2)

- 大型程序的组织
  - C: 无namespace
- 可测试能力
  - C、python都有*Unit测试框架
- 错误检查能力
  - C: 编译，core dump
    调试成本
  - Python:
    - 无编译，可用pylint做检查，易出低级错误
    - 错误，exception

C vs Python (3)

- 上线和运维
  - C: 可编译为独立可执行程序(包括依赖的库)
  - Python: 需要python运行环境，及依赖的库

Golang (1)

- 性能
  - 和C接近
- 并发性
  - Go routine: 屏蔽底层的机制，充分利用cpu资源
  - 多线程模型：容易思考
- 开发效率
  - 描述能力和python接近
  - 较丰富的库（系统库，第三方库）

Golang (2)

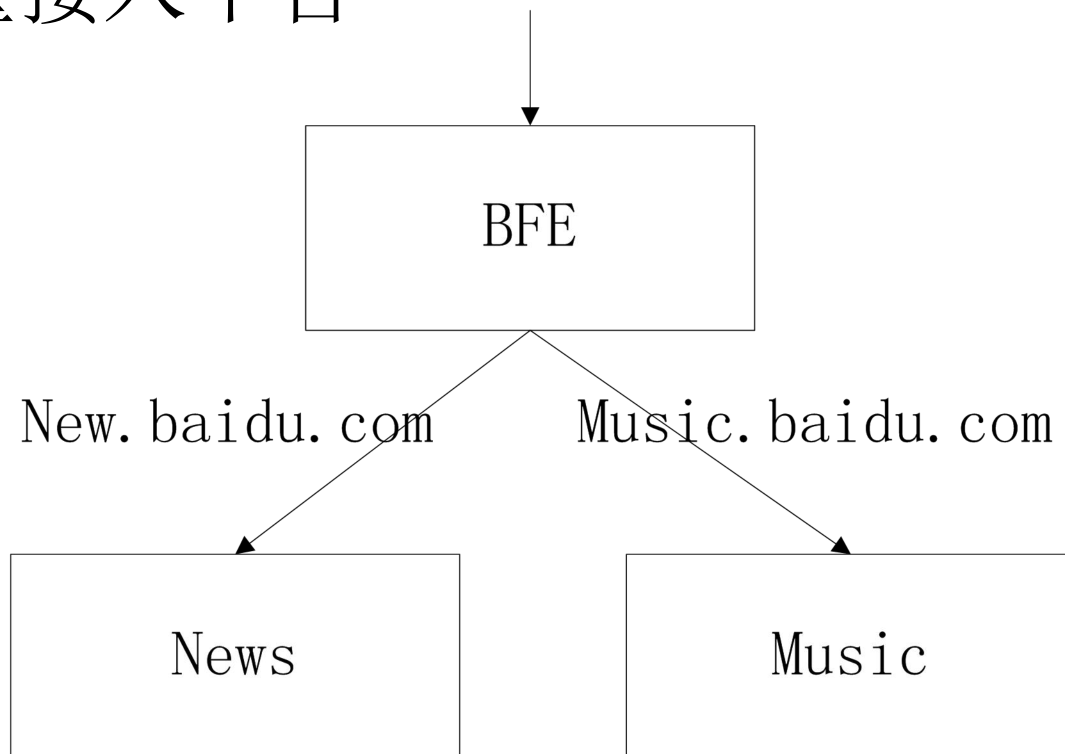
- 大型程序的组织
  - Package
  - 数据访问的限制(首字母大小写的区别)
- 可测试能力
  - 内置的单测和覆盖检查工具，易于做TDD
  - go test
- 错误检查能力
  - 严格的编译阶段检查：强类型，文件包含，…
  - Panic，便于定位问题

Golang(3)

- 上线和运维
  - 可编译为独立可执行程序

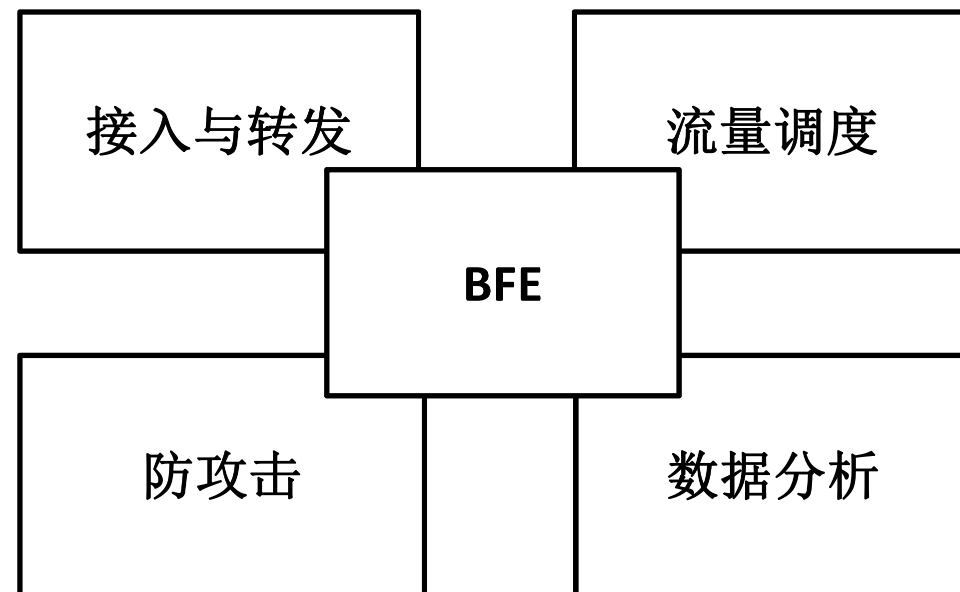
BFE(Baidu Front End)

- 百度统一前端
  - 七层流量接入平台



BFE(Baidu Front End)

- 主要服务
  - 接入转发
  - 防攻击、流量调度、数据分析
- 业务现状
  - 覆盖大部分重要产品
  - 日请求量千亿级别



为什么重写BFE

- 现存问题
  - 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
  - 配置管理方式落后
    - 为单产品线设计，无法支持平台化要求
    - 配置变更(修改、重载、验证)能力差
  - 变更和稳定性的矛盾
    - 程序出core

技术选型：Go vs Nginx

- 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，性能在可接受范围内
  - 通过算法设计和架构设计来弥补

几个问题

- GC优化
- http协议栈
- 分布式架构

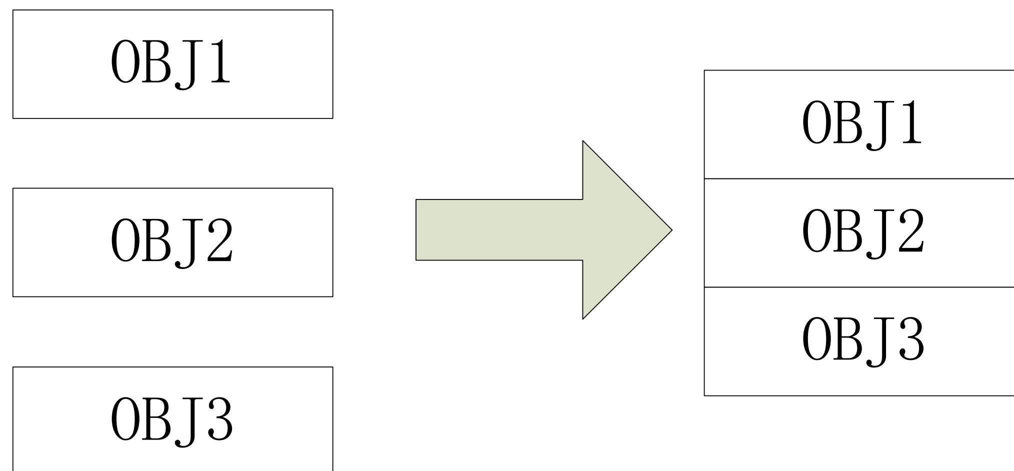
GC带来的问题

- 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
    - 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（go1.3）
  - Mark and Sweep：大量时间时间用于扫描对象
- 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做接口使用
  - 对象复用 (对象池)
  - 深度优化系统结构和算法

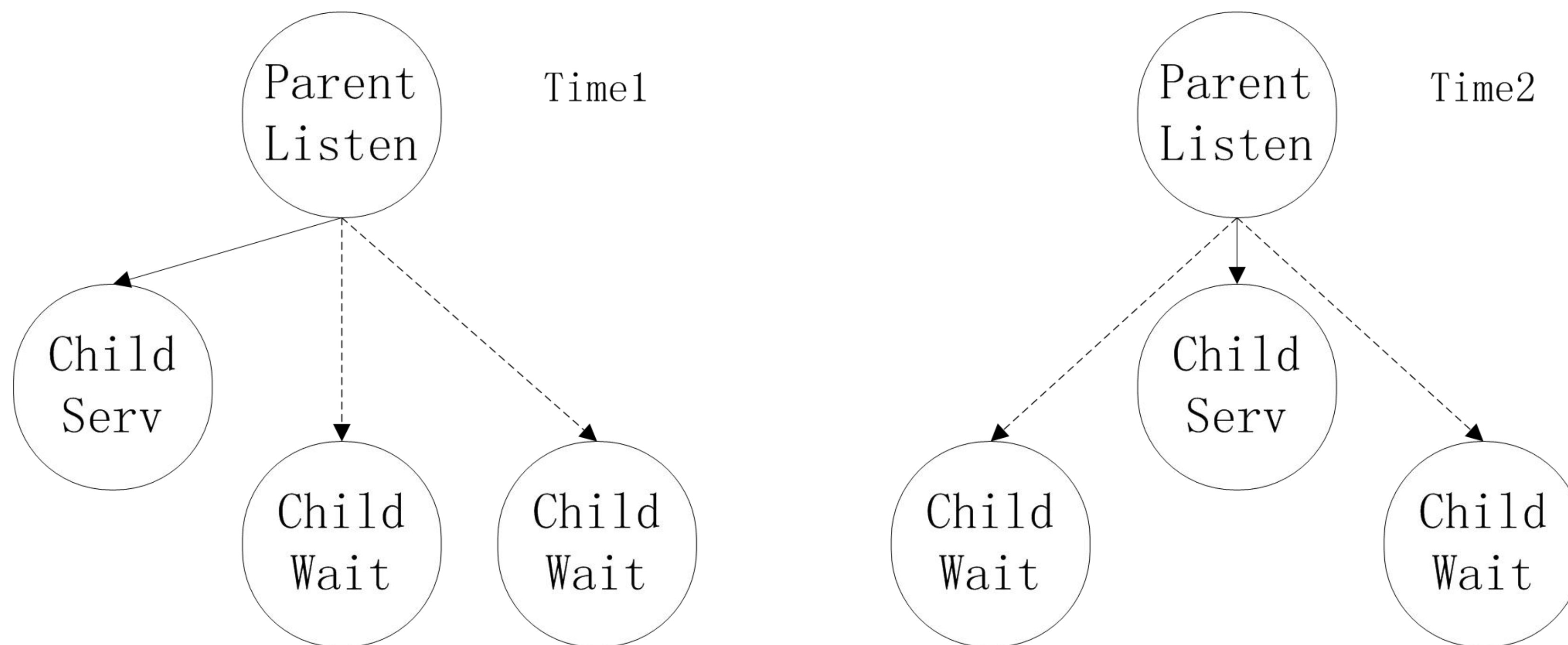
通过Array减少引用计数



困境

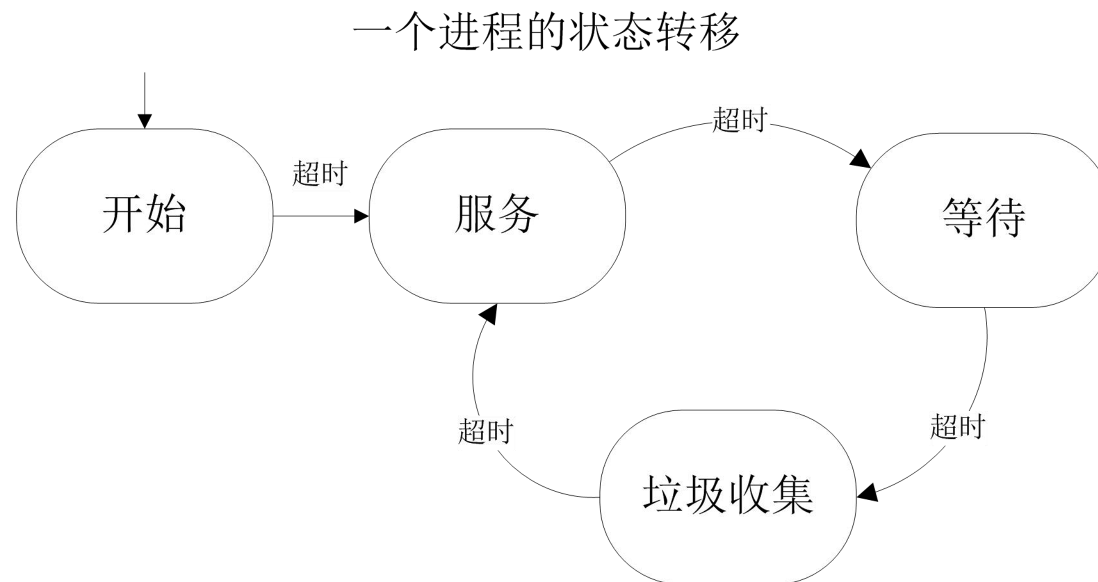
- 减小对象数的困境
  - 常态下需要保持几十万的连接 => 几十个ms
  - 修改golang网络库，重写基本数据结构
- 不使用让go管理内存
  - 通过Cgo手工维护，很危险 (go中调用c代码)
  - 不能解决问题：大量go对象难以避免

车轮大战！

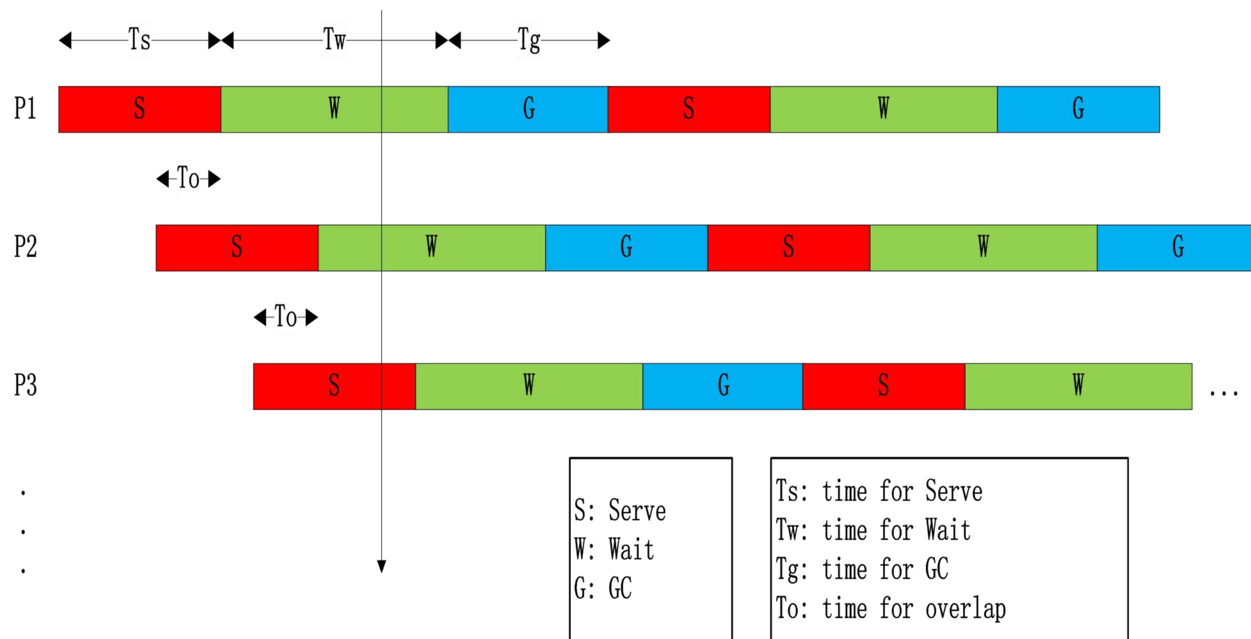


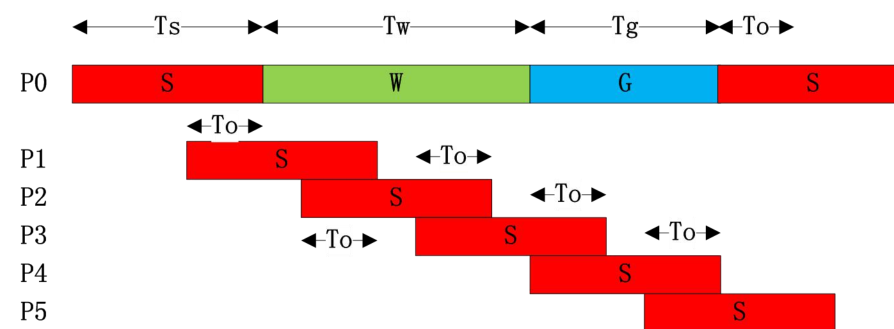
轮转GC方案

- 基本思路
  - 关闭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





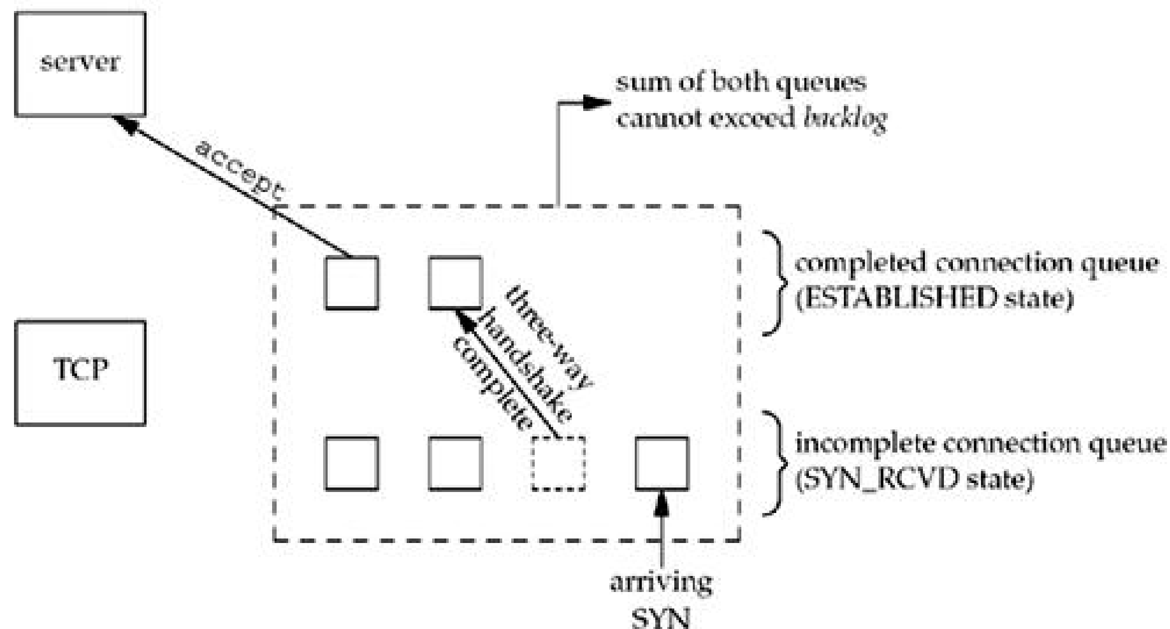
进程数的计算： $N = 1 + \left\lceil \frac{Tw + Tg + To}{Ts - To} \right\rceil$

技术细节

- 本质上：多个进程监听同一个端口
  - 高版本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之内)
  - 大文件请求
    - 对gc造成的延迟(几十ms)不敏感


```
- Past: {
    BucketSize: 1,
    BucketNum: 10,
    Count: 139660,
    Sum: 44606461,
    Ave: 319,
  - Counters: [
        137584,
        1714,
        189,
        58,
        38,
        27,
        19,
        5,
        5,
        6,
        15
    ]
}
```

多说一句Go 1.5/1.6：没有银弹

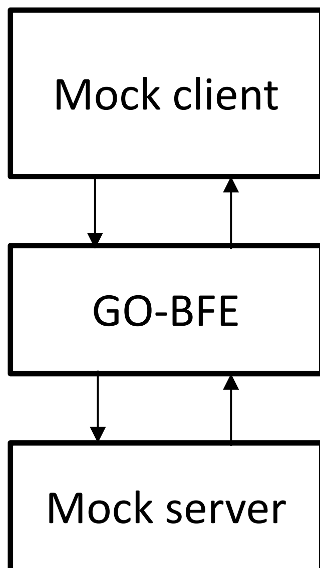
- Stop-The-World(STW)缩短了，决定因素也变了
  - Time spent looping over **goroutines**
  - Time spent looping over **malloc spans**
- 实际运行，还是有几十个ms的STW时间
  - GO-BFE的场景和服务模式，大量的goroutine必然存在
- 需要根据线上运行实际情况来做选型

协议一致性问题

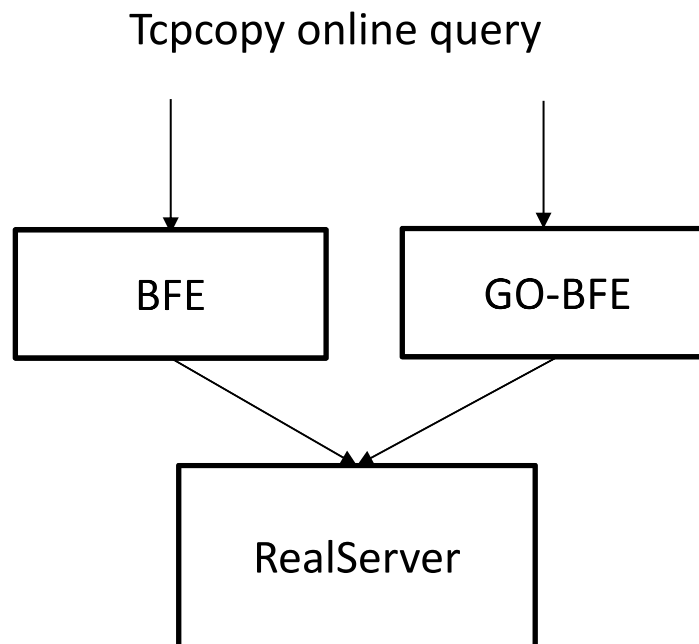
- GO-BFE 参考了Go的http库
- 基于Go的http实现是否完善，符合rfc标准
  - 没有大规模的应用的例子
- 需要一些方法来验证
  - 网络协议一致性测试是难点

协议一致性

- Macaroon框架



- Tcpcopy线上引流对比

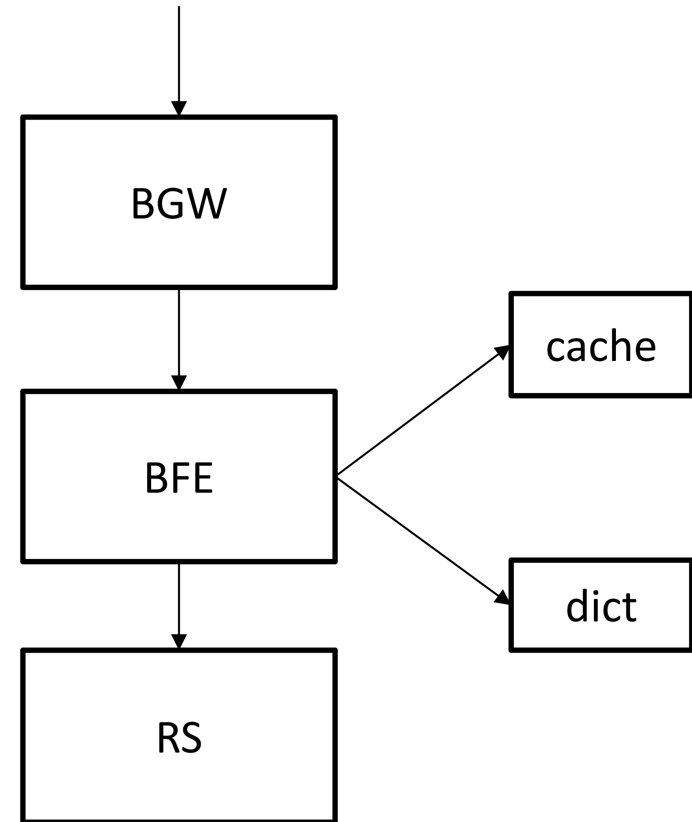


一个例子

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

分布式架构

- BFE程序结构：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



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