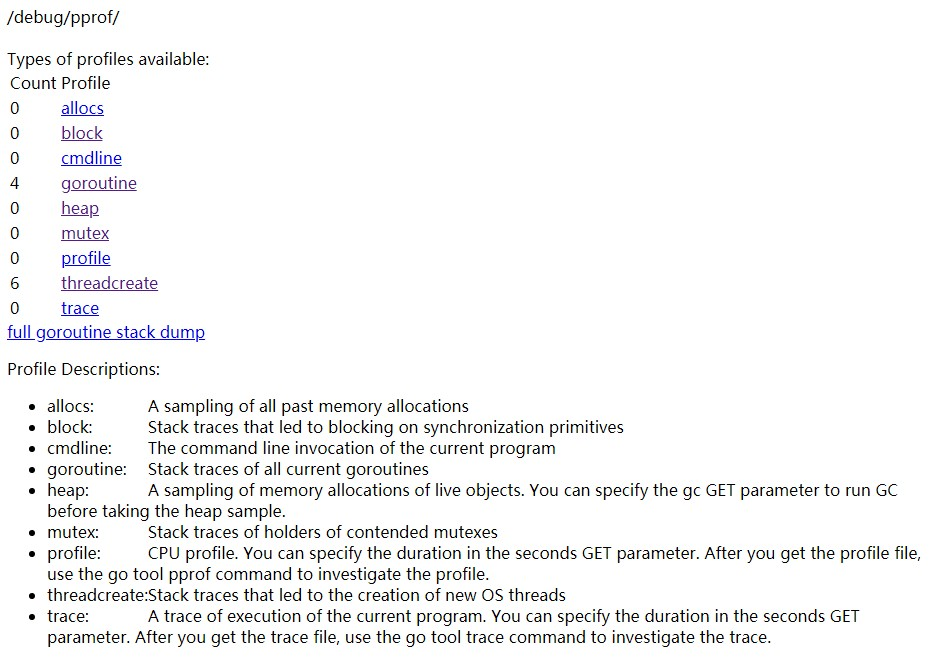
# 如何使用pprof

## 能做什么

### Pprof index



* allocs 内存分配情况的采样信息
* blocks 阻塞操作情况的采样信息
* cmdline 显示程序启动命令及参数
* goroutine 当前所有协程的堆栈信息
* heap 堆上内存使用情况的采样信息，默认每分配512K字节时取样一次，无需触发
* mutex 锁争用情况的采样信息
* profile CPU 占用情况的采样信息, 默认以100 Hz的的频率对CPU使用情况进行取样,需触发
* threadcreate 系统线程创建情况的采样信息
* trace 程序运行跟踪信息

### Heap 参数详解

* Alloc uint64 golang语言框架堆空间分配的字节数
* TotalAlloc uint64 从服务开始运行至今分配器为分配的堆空间总 和，只有增加，释放的时候不减少
* Sys uint64 服务现在系统使用的内存
* Lookups uint64 被runtime监视的指针数
* Mallocs uint64 服务malloc heap objects的次数
* Frees uint64 服务回收的heap objects的次数HeapAlloc uint64 服务分配的堆内存字节

数

* HeapSys uint64 系统分配的作为运行栈的内存
* HeapIdle uint64 申请但是未分配的堆内存或者回收了的堆内存（空闲）字节数
* HeapInuse uint64 正在使用的堆内存字节数
* HeapReleased uint64 返回给OS的堆内存，类似C/C++中的free。
* HeapObjects uint64 堆内存块申请的量
* StackInuse uint64 正在使用的栈字节数
* StackSys uint64 系统分配的作为运行栈的内存
* MSpanInuse uint64 用于测试用的结构体使用的字节数
* MSpanSys uint64 系统为测试用的结构体分配的字节数
* MCacheInuse uint64 mcache结构体申请的字节数(不会被视为垃圾回收)
* MCacheSys uint64 操作系统申请的堆空间用于mcache的字节数
* BuckHashSys uint64 用于剖析桶散列表的堆空间
* GCSys uint64 垃圾回收标记元信息使用的内存
* OtherSys uint64 golang系统架构占用的额外空间
* NextGC uint64 垃圾回收器检视的内存大小
* LastGC uint64 垃圾回收器最后一次执行时间。
* PauseTotalNs uint64 垃圾回收或者其他信息收集导致服务暂停的次数。
* PauseNs [256]uint64 一个循环队列，记录最近垃圾回收系统中断的时间
* PauseEnd [256]uint64 一个循环队列，记录最近垃圾回收系统中断的时间开始点。
* NumForcedGC uint32 服务调用runtime.GC()强制使用垃圾回收的次数。
* GCCPUFraction float64 垃圾回收占用服务CPU工作的时间总和。如果有100个goroutine，垃圾回收的时间为1S,那么就占用了100S。
* BySize 内存分配器使用情况

## 如何导入

runtime/pprof: 采集程序（非 Server）的运行数据进行分析

net/http/pprof：采集 HTTP Server 的运行时数据进行分析

## 使用

CPU使用情况

go tool pprof http://localhost:8000/debug/pprof/profile?seconds=30

内存使用情况

go tool pprof <http://localhost:8000/debug/pprof/>heap

-inuse\_space：分析应用程序的常驻内存占用情况

-alloc\_objects：分析应用程序的内存临时分配情况

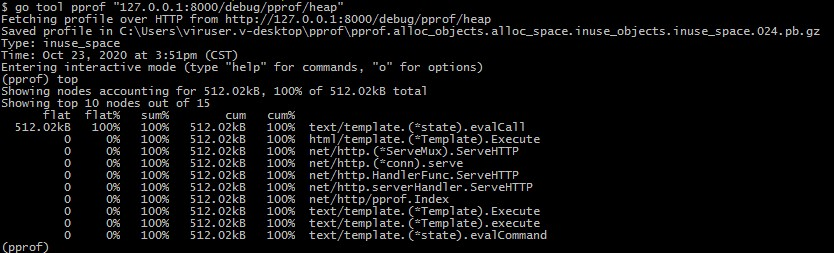
协程阻塞情况

go tool pprof http://localhost:8000/debug/pprof/block

## 常用交互命令行

* help可以查看使用说明
* **top** 可以查看前10分配情况
* **list** 展示源码及相应损耗
* **web** 使用浏览器视图展开
* tree 以树状显示
* png 以图片格式输出
* svg 生成浏览器可以识别的svg文件

### Top 参数详解



* flat：给定函数上运行耗时
* flat%：同上的 CPU 运行耗时总比例
* sum%：给定函数累积使用 CPU 总比例
* cum：当前函数加上它之上的调用运行总耗时
* cum%：同上的 CPU 运行耗时总比例

## 可视化界面

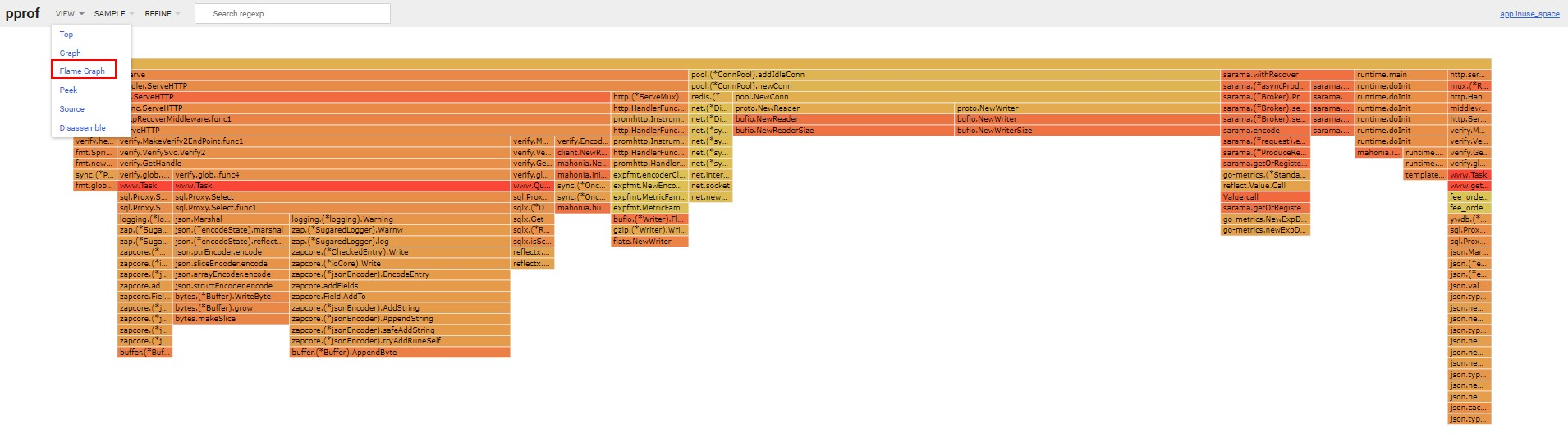
可视化界面依赖graphviz。解压附件压缩包，将bin文件夹添加到Path环境变量即可。

测试方式：打开cmd，输入dot –version 查看是否有版本信息返回

进入可视化界面方式

1. 在交互界面使用web命令
2. go tool pprof –http=:1234 <http://127.0.0.1:8000/debug/pprof/heap>

开发测试环境调试建立使用-http方式进行，原因是其中含有火焰图，便于分析GC耗时，及对象生成个数。建议每个标签都浏览器



# 二、常见技巧与实例分析

## 编程技巧

1. 小对象合并成结构体一次分配，减少内存分配次数
2. buffer缓存区内容一次分配足够大小空间，并适当复用
3. slice和map采make创建时，预估大小指定容量
4. 避免频繁创建临时对象
5. 减少[]byte与string转换，字符串拼接优先考虑bytes.Buffer
6. 传递struct尽量使用指针，减少复制和内存占用消耗（尤其对于赋值给interface，会分配到堆上，额外增加gc消耗）
7. 能在初始化中做的事就不要放到每次调用的时候做

## 定位技巧

**耗时问题相关**

若 gc 相关函数占用异常，可重点排查对象数量

解决速度问题（CPU占用）时，关注对象数量（ --inuse/alloc\_objects ）指标

解决内存占用问题时，关注分配空间（ --inuse/alloc\_space ）指标

通过，

inuse 代表当前时刻的内存情况，alloc 代表从从程序启动到当前时刻累计的内存情况.

**内存泄漏分析**

在两个标准点进行请求

curl “127.0.0.1:8000/debug/pprof/heap” > 1

curl “127.0.0.1:8000/debug/pprof/heap” > 2

比较内存变化

go tool pprof –base 1 2

**缓存对象分析**

从交互界面，或视图定位到具体哪个函数资源消耗过大时，可能函数本身较长，不便于定位到具体的细节，可在交互界面上通过 list 函数名称的方式，查看内存消耗较大的代码内容，如：

