PProf

- 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

是什么

pprof 是用于可视化和分析性能分析数据的工具

pprof 以 profile.proto 读取分析样本的集合，并生成报告以可视化并帮助分析数据（支持文本和图形报告）

profile.proto 是一个 Protocol Buffer v3 的描述文件，它描述了一组 callstack 和 symbolization 信息， 作用是表示统计分析的一组采样的调用栈，是很常见的 stacktrace 配置文件格式

可以做什么

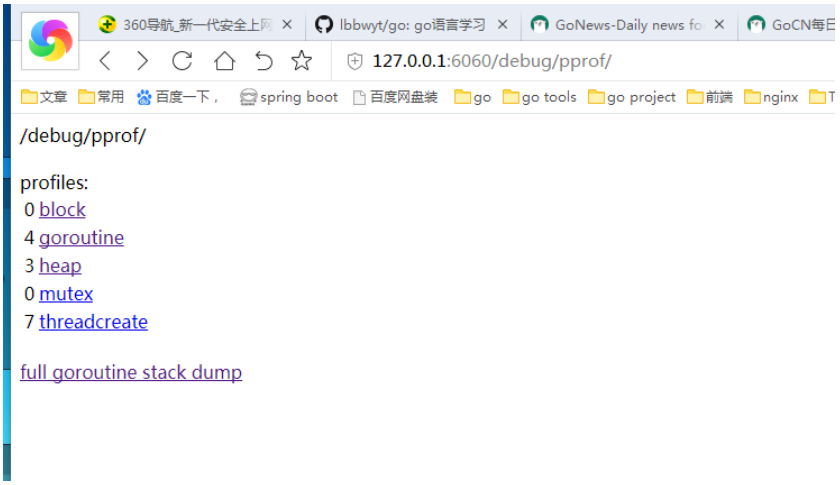
- CPU Profiling：CPU 分析，按照一定的频率采集所监听的应用程序 CPU（含寄存器）的使用情况，可确定应用程序在主动消耗 CPU 周期时花费时间的位置
- Memory Profiling：内存分析，在应用程序进行堆分配时记录堆栈跟踪，用于监视当前和历史内存使用情况，以及检查内存泄漏
- Block Profiling：阻塞分析，记录 goroutine 阻塞等待同步（包括定时器通道）的位置
- Mutex Profiling：互斥锁分析，报告互斥锁的竞争情况

```
package main

import (
  "log"
  "net/http"
  "strings"
  _ "net/http/pprof"
)
```

```go
func main() {
  go func() {
    for {
      log.Println(Add("https://github.com"))
    }
  }()

  http.ListenAndServe("0.0.0.0:6060", nil)


}
//缓存字符串
var datas []string
//add
func Add(str string) string {
  datas := append(datas, str)
  return strings.Join(datas, "##")
}
```

运行这个文件，你的 HTTP 服务会多出一个 /debug/pprof 的 endpoint 可用于观察应用程序运行的情况

查看页面地址：访问http://127.0.0.1:6060/debug/pprof/

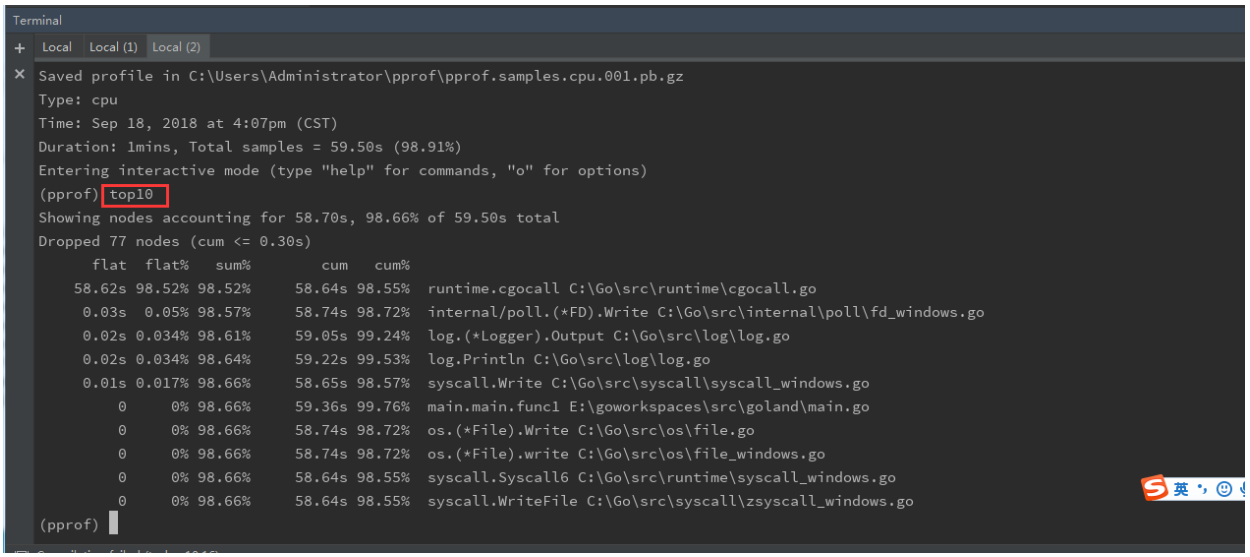


通过交互式终端使用

go tool pprof http://localhost:6060/debug/pprof/profile?seconds=60

在terminal中执行上述命令，需等待 60 秒（可调整 seconds 的值），pprof 会进行 CPU Profiling（cpu性能分析）。结束后将默认进入 pprof 的交互式命令模式，可以对分析的结果进行查看或导出。

top 10 命令：（具体可执行pprof help查看查看命令说明）



- 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

最后一列为函数名称

（3）go tool pprof http://localhost:6060/debug/pprof/heap

（3）go tool pprof http://localhost:6060/debug/pprof/block

（4）go tool pprof http://localhost:6060/debug/pprof/mutex

PProf 可视化界面

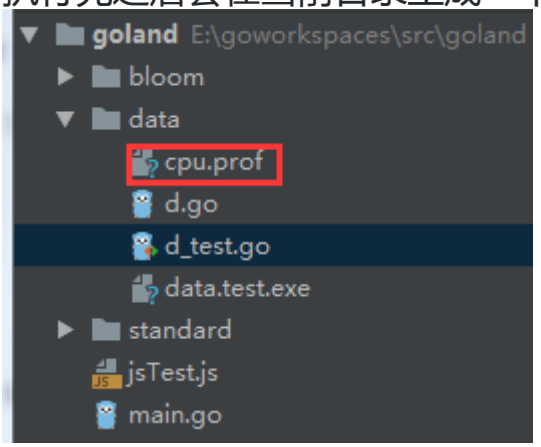
新建 data/d_test.go，文件内容：

```
package data

import "testing"

const url = "https://github.com/lbbwyt/go"

func TestAdd(t *testing.T) {
  s:=Add(url)
  if s == "" {
    t.Errorf("test add error")
  }
}
//必须循环 b.N 次 。 这个数字 b.N 会在运行中调整，以便最终达到合适的时间消耗。方便计算出合理的数据。
//压力测试用例必须遵循如下格式，其中XXX可以是任意字母数字的组合，但是首字母不能是小写字母
//func BenchmarkXXX(b *testing.B) { ... }
//go test不会默认执行压力测试的函数，如果要执行压力测试需要带上参数-test.bench，
//语法:-test.bench="test_name_regex",例如go test -test.bench=".*"表示测试全部的压力测试函数
//文件名也必须以_test.go结尾
func BenchmarkAdd(b *testing.B) {
  for i:=0;i<b.N;i++ {
    Add(url)
  }
}
```

执行上述压力测试用例：go test -bench=. -cpuprofile=cpu.prof

```
Terminal
+ Microsoft Windows [版本 6.1.7601]
× 版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

  E:\goworkspaces\src\goland\data>go test -bench=. -cpuprofile=cpu.prof
  goos: windows
  goarch: amd64
  pkg: goland/data
  BenchmarkAdd-4          10000000               213 ns/op
  PASS
  ok      goland/data     2.800s
```

执行完之后会在当前目录生成一个cpu.pprof的文件

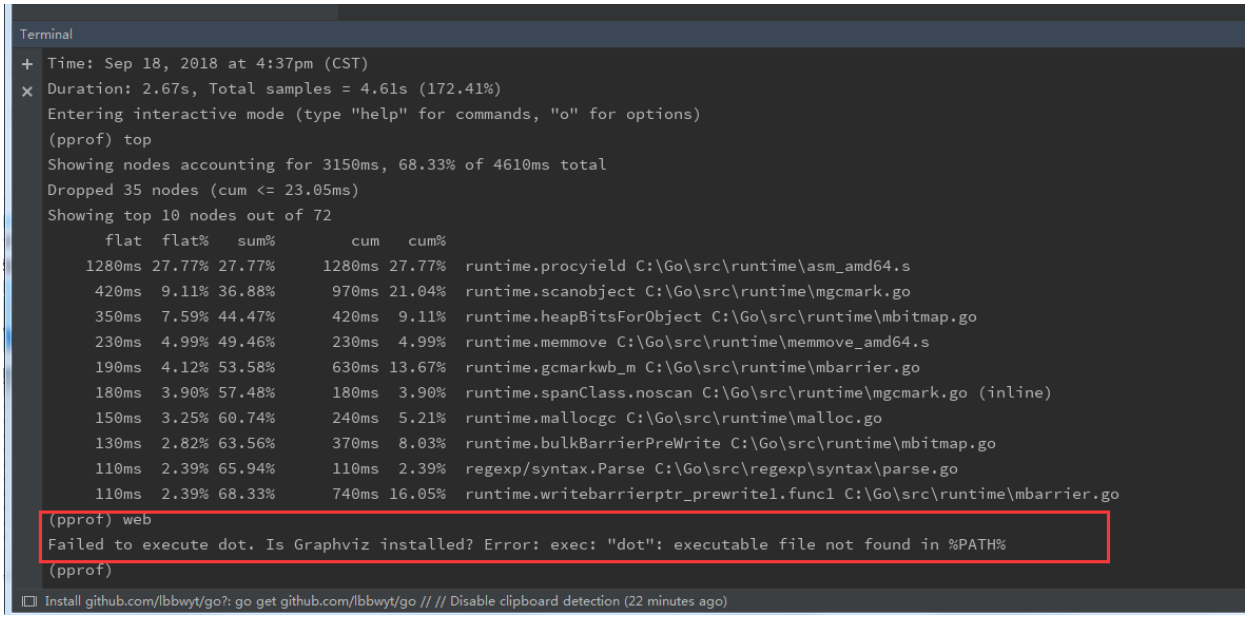


启动 PProf 可视化界面

go tool pprof cpu.prof

```
Terminal
+ E:\goworkspaces\src\goland\data>go tool pprof cpu.prof
× Type: cpu
  Time: Sep 18, 2018 at 4:37pm (CST)
  Duration: 2.67s, Total samples = 4.61s (172.41%)
  Entering interactive mode (type "help" for commands, "o" for options)
  (pprof) top
  Showing nodes accounting for 3150ms, 68.33% of 4610ms total
  Dropped 35 nodes (cum <= 23.05ms)
  Showing top 10 nodes out of 72
        flat  flat%   sum%        cum   cum%
     1280ms 27.77% 27.77%     1280ms 27.77%  runtime.procyield C:\Go\src\runtime\asm_amd64.s
      420ms  9.11% 36.88%      970ms 21.04%  runtime.scanobject C:\Go\src\runtime\mgcmark.go
      350ms  7.59% 44.47%      420ms  9.11%  runtime.heapBitsForObject C:\Go\src\runtime\mbitmap.go
      230ms  4.99% 49.46%      230ms  4.99%  runtime.memmove C:\Go\src\runtime\memmove_amd64.s
      190ms  4.12% 53.58%      630ms 13.67%  runtime.gcmarkwb_m C:\Go\src\runtime\mbarrier.go
      180ms  3.90% 57.48%      180ms  3.90%  runtime.spanClass.noscan C:\Go\src\runtime\mgcmark.go (inline)
      150ms  3.25% 60.74%      240ms  5.21%  runtime.mallocgc C:\Go\src\runtime\malloc.go
      130ms  2.82% 63.56%      370ms  8.03%  runtime.bulkBarrierPreWrite C:\Go\src\runtime\mbitmap.go
      110ms  2.39% 65.94%      110ms  2.39%  regexp/syntax.Parse C:\Go\src\regexp\syntax\parse.go
      110ms  2.39% 68.33%      740ms 16.05%  runtime.writebarrierptr_prewrite1.func1 C:\Go\src\runtime\mbarrier.go
  (pprof)
```

如需浏览器下查看，需执行web命令；
如果出现如下提示，就是提示你要安装 Graphviz了 （请右拐谷歌）


Terminal
Time: Sep 18, 2018 at 4:37pm (CST)
Duration: 2.67s, Total samples = 4.61s (172.41%)
Entering interactive mode (type "help" for commands, "o" for options)
(pprof) top
Showing nodes accounting for 3150ms, 68.33% of 4610ms total
Dropped 35 nodes (cum <= 23.05ms)
Showing top 10 nodes out of 72
flat flat% sum% cum cum%
1280ms 27.77% 27.77% 1280ms 27.77% runtime.procyield C:\Go\src\runtime\asm_amd64.s
420ms 9.11% 36.88% 970ms 21.04% runtime.scanobject C:\Go\src\runtime\mgcmark.go
350ms 7.59% 44.47% 420ms 9.11% runtime.heapBitsForObject C:\Go\src\runtime\mbitmap.go
230ms 4.99% 49.46% 230ms 4.99% runtime.memmove C:\Go\src\runtime\memmove_amd64.s
190ms 4.12% 53.58% 630ms 13.67% runtime.gcmarkwb_m C:\Go\src\runtime\mbarrier.go
180ms 3.90% 57.48% 180ms 3.90% runtime.spanClass.noscan C:\Go\src\runtime\mgcmark.go (inline)
150ms 3.25% 60.74% 240ms 5.21% runtime.mallocgc C:\Go\src\runtime\malloc.go
130ms 2.82% 63.56% 370ms 8.03% runtime.bulkBarrierPreWrite C:\Go\src\runtime\mbitmap.go
110ms 2.39% 65.94% 110ms 2.39% regexp/syntax.Parse C:\Go\src\regexp\syntax\parse.go
110ms 2.39% 68.33% 740ms 16.05% runtime.writebarrierptr_prewrite1.func1 C:\Go\src\runtime\mbarrier.go
(pprof) web
Failed to execute dot. Is Graphviz installed? Error: exec: "dot": executable file not found in %PATH%
(pprof)
Install github.com/lbbwyt/go?: go get github.com/lbbwyt/go // // Disable clipboard detection (22 minutes ago)