文档：

https://pkg.go.dev/net/http/pprof

golang程序有哪些内容需要调优

1. cpu：程序对cpu的使用情况 - 使用时长，占比等
2. 内存：程序对cpu的使用情况 - 使用时长，占比，内存泄露等。如果在往里分，程序堆、栈使用情况
3. I/O：IO的使用情况 - 哪个程序IO占用时间比较长
4. goroutine：go的协程使用情况，调用链的情况
5. goroutine leak：goroutine泄露检查。什么是泄露：随着时间的增加goroutine的数量不断增加
6. go dead lock：死锁的检测分析
7. data race detector：数据竞争分析，其实也与死锁分析有关

可以做什么

1. 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
2. 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
3. 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
4. Mutex Profiling：互斥锁分析，报告互斥锁的竞争情况

需要注意什么

1. 对线上环境做数据采集是应当选择流量较少的时间段避免影响生产环境
2. 可以通过测试用例获取性能指标数据，或者通过测试环境模拟生产环境获取数据
3. 如果在完全无流量的时间段获取指标数据，所获取到的数据可能毫无意义

三种数据采集方式

web网页采集

程序入口代码调整

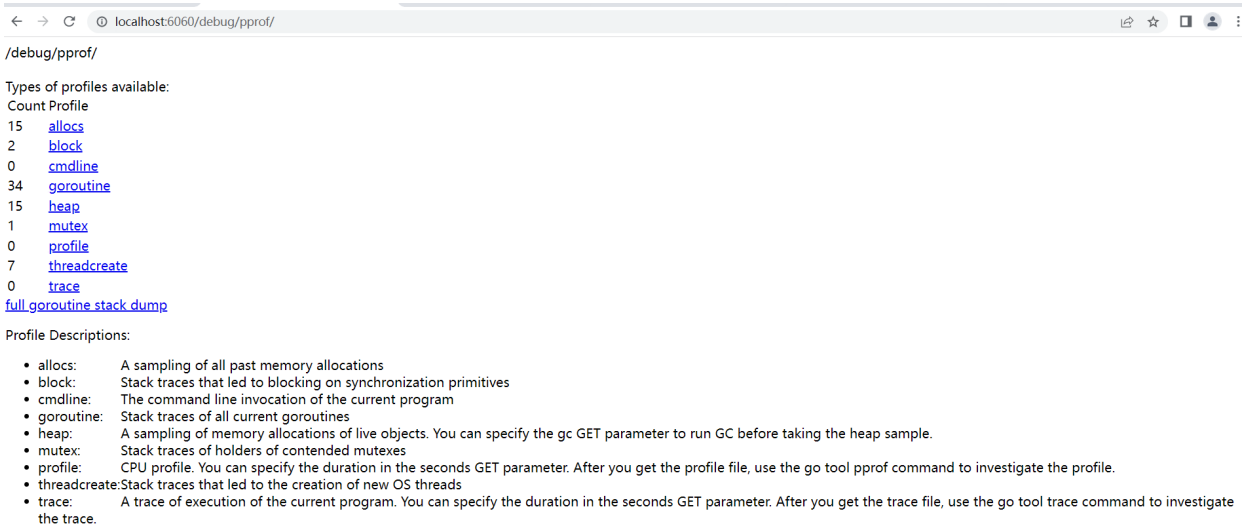
```
//导入依赖
import (
    "net/http"
    _ "net/http/pprof"
)
```

```
// 启动一个httpserver，用于数据导出
go func() {
   log.Println(http.ListenAndServe(":6060", nil))
}()
```

```
//入口函数添加以下代码
runtime.SetBlockProfileRate(1) // 开启对阻塞操作的跟踪，block
runtime.SetMutexProfileFraction(1) // 开启对锁调用的跟踪，mutex
```

采集信息访问

```
http://localhost:6060/debug/pprof/
```



1. allocs：查看过去所有内存分配的样本
2. block：查看导致同步原语阻塞的堆栈跟踪
3. cmdline：当前程序的命令行的完整调用路径
4. goroutine：查看当前所有运行的 goroutines 堆栈跟踪
5. heap：查看活动对象的内存分配情况，与allocs
6. mutex：查看互斥锁的竞争持有者的堆栈跟踪
7. profile：默认进行 30s 的 CPU Profiling，得到一个分析用的 profile 文件。浏览器访问下载文件
8. threadcreate：查看创建新OS线程的堆栈跟踪
9. trace：程序运行跟踪信息。浏览器访问下载文件

数据下载

```
//访问格式
http://localhost:6060/debug/pprof/xxx
//如：
http://localhost:6060/debug/pprof/allocs
http://localhost:6060/debug/pprof/block
http://localhost:6060/debug/pprof/goroutine
http://localhost:6060/debug/pprof/heap
http://localhost:6060/debug/pprof/mutex
http://localhost:6060/debug/pprof/profile
http://localhost:6060/debug/pprof/threadcreate
http://localhost:6060/debug/pprof/trace
```

通过基准测试采集数据

```
# 执行基准测试
go test -run ^$ -bench . .\data_test\ -blockprofile block.out -cpuprofile cpu.out -
memprofile mem.out  -mutexprofile mutex.out -trace trace.out -outputdir ./testout
```

通过硬编码采集数据

cpu、内存、trace 数据采集

```
import(
    "runtime/pprof"
    "runtime/trace"
)
```

```
cpufile, err := os.OpenFile("code_collection/out/cpu.out", os.O_CREATE|os.O_WRONLY,
0644)
    if err != nil {
        log.Fatal(err)
    }
    err = pprof.StartCPUProfile(cpufile)
    if err != nil {
        log.Fatal(err)
    }
    defer pprof.StopCPUProfile()

    memfile, err := os.OpenFile("code_collection/out/mem.out",
os.O_CREATE|os.O_WRONLY, 0644)
    if err != nil {
        log.Fatal(err)
    }
    pprof.WriteHeapProfile(memfile)

    tracefile, err := os.OpenFile("code_collection/out/trace.out",
os.O_CREATE|os.O_WRONLY, 0644)
    if err != nil {
        log.Fatal(err)
```

```
    }
    err = trace.Start(tracefile)
    if err != nil {
        log.Fatal(err)
    }
    defer trace.Stop()

// 调用需要分析的代码
...
```

数据分析

指标

1. Flat：函数自身运行资源消耗
2. Flat%：函数自身资源耗比例
3. Sum%：指的就是每一行的flat%与上面所有行的flat%总和
4. Cum：当前函数加上它所有调用栈的运行总消耗
5. Cum%：当前函数加上它所有调用栈的运行总消耗比例

注意：以上指标，对应不同的资源单位不一样，比如：内存、CPU、耗时、协程数等

命令行分析数据

1. 命令行操作命令 go tool pprof ， source 既可以是本地文件也可以是网络地址
2. 例如：

```
# 分析内存分配情况
go tool pprof http://localhost:6060/debug/pprof/allocs
# 分析阻塞情况
go tool pprof http://localhost:6060/debug/pprof/block
# 分析协程情况
go tool pprof http://localhost:6060/debug/pprof/goroutine
# 分析活动对象内存分配情况
go tool pprof http://localhost:6060/debug/pprof/heap
# 分析锁情况
go tool pprof http://localhost:6060/debug/pprof/mutex
# 分析CPU使用情况
go tool pprof http://localhost:6060/debug/pprof/profile
# 指定获取多长时间的CPU数据
go tool pprof http://localhost:6060/debug/pprof/profile?seconds=30
```

3. 可通过 go tool pprof --help 查看更多用法
4. 进入命令行交互模式后，常用的三个命令 top、list、web

top： 按Flat排序，默认取前10条。top5 表示取前5条

```
(pprof) top
Showing nodes accounting for 6.46s, 99.23% of 6.51s total
Dropped 21 nodes (cum <= 0.03s)
      flat  flat%   sum%        cum   cum%
     6.41s 98.46% 98.46%      6.46s 99.23%  github.com/wolfogre/go-pprof-practice/animal/felidae/tiger.(*Tiger).Eat
     0.05s  0.77% 99.23%      0.05s  0.77%  runtime.asyncPreempt
         0     0% 99.23%      6.46s 99.23%  github.com/wolfogre/go-pprof-practice/animal/felidae/tiger.(*Tiger).Live
         0     0% 99.23%      6.48s 99.54%  main.main
         0     0% 99.23%      6.48s 99.54%  runtime.main
(pprof)
```

list：输出匹配regexp的带注释的源代码

```
(pprof) list tiger
Total: 6.51s
ROUTINE ======================== github.com/wolfogre/go-pprof-practice/animal/felidae/tiger.(*Tiger).Eat in E:\Work\Code\go\go-pprof-practice-master\animal\felidae\tiger\tiger.go
     6.41s      6.46s (flat, cum) 99.23% of Total
         .          .     19:}
         .          .     20:
         .          .     21:func (t *Tiger) Eat() {
         .          .     22:   log.Println(t.Name(), "eat")
         .          .     23:   loop := 10000000000
     6.41s      6.46s     24:   for i := 0; i < loop; i++ {
         .          .     25:           // do nothing
         .          .     26:   }
         .          .     27:}
         .          .     28:
         .          .     29:func (t *Tiger) Drink() {
ROUTINE ======================== github.com/wolfogre/go-pprof-practice/animal/felidae/tiger.(*Tiger).Live in E:\Work\Code\go\go-pprof-practice-master\animal\felidae\tiger\tiger.go
         0      6.46s (flat, cum) 99.23% of Total
         .          .      8:func (t *Tiger) Name() string {
         .          .      9:   return "tiger"
         .          .     10:}
         .          .     11:
         .          .     12:func (t *Tiger) Live() {
         .      6.46s     13:   t.Eat()
         .          .     14:   t.Drink()
         .          .     15:   t.Shit()
         .          .     16:   t.Pee()
         .          .     17:   t.Climb()
         .          .     18:   t.Sneak()
(pprof)
```

web：将会生成一张svg格式的图片，默认使用浏览器打开

web数据分析

安装插件

地址：https://graphviz.gitlab.io/download/

linux：

- Ubuntu packages*

```
sudo apt install graphviz
```

- Fedora project*

```
sudo yum install graphviz
```

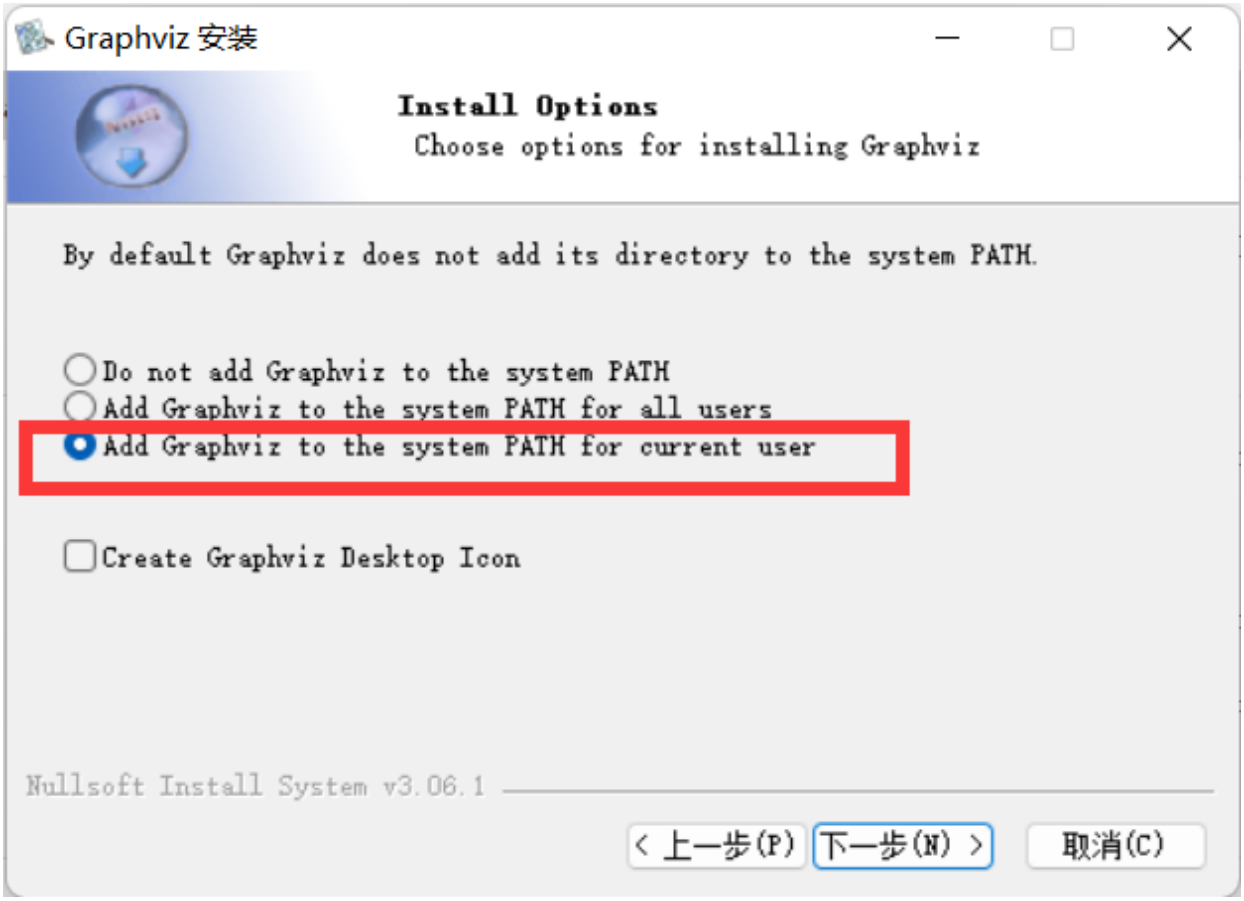
- Debian packages*

```
sudo apt install graphviz
```

- Stable and development rpms for Redhat Enterprise, or CentOS systems* available but are out of date.

```
sudo yum install graphviz
```

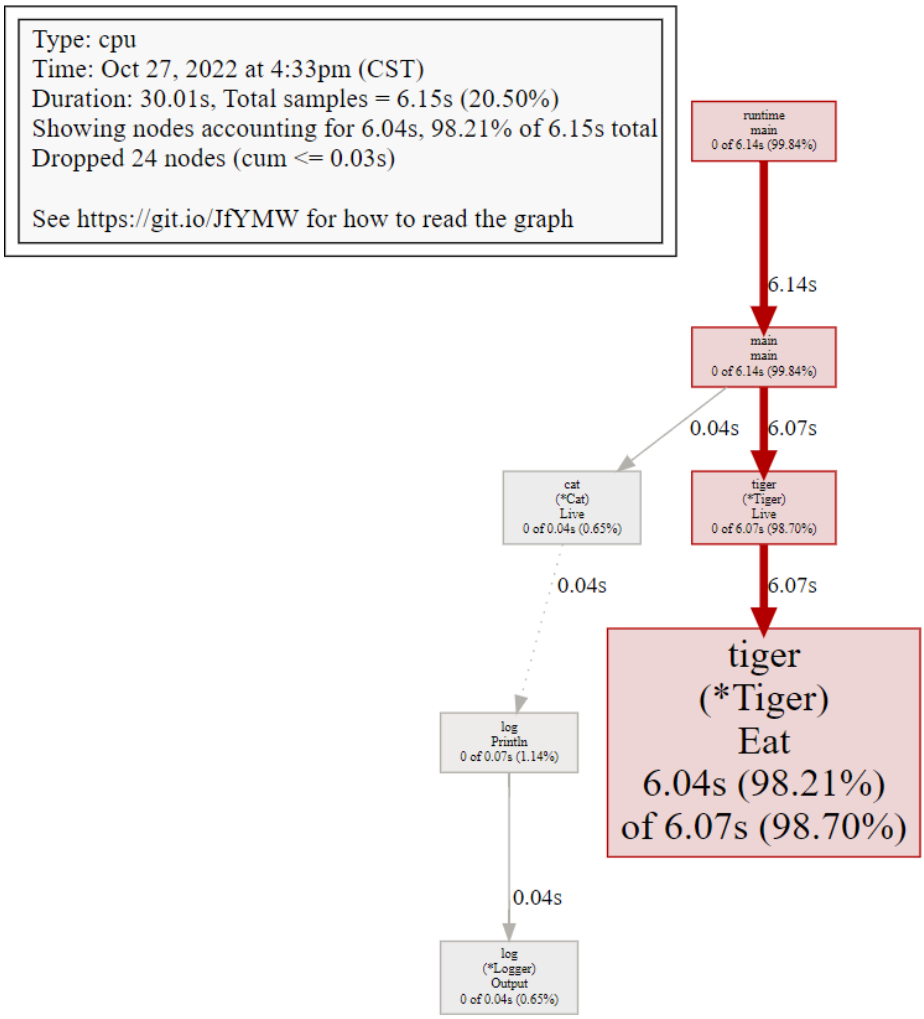
windows：



通过浏览器打开svg查看

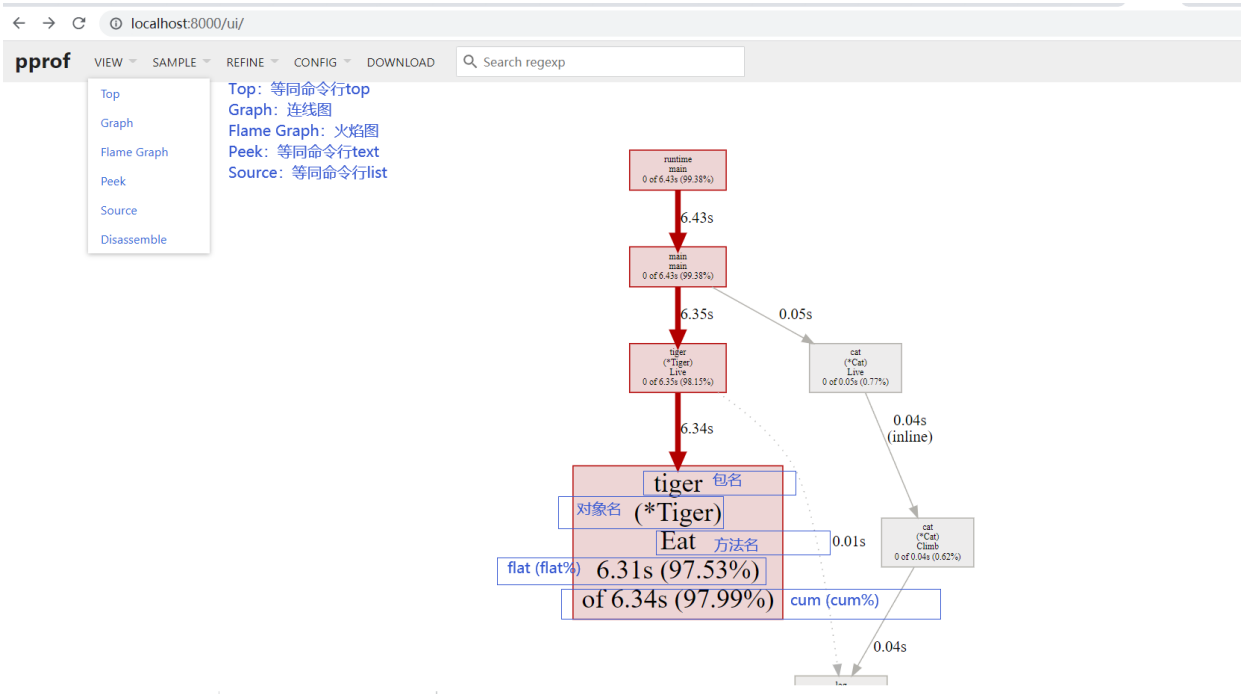
```
go tool pprof -web http://127.0.0.1:6060/debug/pprof/profile
```

```
PS C:\Users\86188> go tool pprof  http://localhost:6060/debug/pprof/profile
Fetching profile over HTTP from http://localhost:6060/debug/pprof/profile
Saved profile in C:\Users\86188\pprof\pprof.samples.cpu.006.pb.gz
Type: cpu
Time: Oct 27, 2022 at 4:33pm (CST)
Duration: 30.01s, Total samples = 6.15s (20.50%)
Entering interactive mode (type "help" for commands, "o" for options)
(pprof) web
(pprof)
```



通过web网页查看

```
go tool pprof -http=:8000 http://127.0.0.1:6060/debug/pprof/profile
```



按指定格式导出

```
# 以pdf格式导出到当前文件夹report文件
go tool pprof -pdf -output=report http://127.0.0.1:6060/debug/pprof/profile
```