main.go

**func** init() {

fmt.Println("我是init函数")

fmt.Println("在main函数执行之前的init函数")

}

**func** main() {

fmt.Println("我是main函数")

}

init函数在这个文件执行之前，默认执行

===============================================================================

    //添加记录CPU使用率的文件--------------------

    cpuf, err1 := os.Create("cpu\_profile")

    if err1 != nil {

        log.Fatal(err1)

    }

    pprof.StartCPUProfile(cpuf)

    defer pprof.StopCPUProfile()

    //添加记录CPU使用率的文件--------------------

添加记录cpu时间代码

===============================================================================

go分析程序性能

server是程序 cpu\_profile采样文件

go tool pprof server cpu\_profile

top

peek 函数名

list 函数名

exit 退出

go tool pprof <http://192.168.5.99:2014/debug/pprof/profile>

go tool pprof [http://192.168.5.99:2014/debug/pprof/heap](https://link.jianshu.com/?t=http://localhost:6060/debug/pprof/heap" \t "C:\\Users\\chc\\AppData\\Local\\Temp\\_blank)  
go tool pprof [http://192.168.5.99:2014/debug/pprof/block](https://link.jianshu.com/?t=http://localhost:6060/debug/pprof/block" \t "C:\\Users\\chc\\AppData\\Local\\Temp\\_blank)

web 生成文件

import (\_ "net/http/pprof")

func main(){

go func() {

log.Println(http.ListenAndServe("localhost:8000", nil))

}()}

通过浏览器 [http://127.0.0.1:8000/debug/pprof/](http://127.0.0.1:8000/debug/pprof/" \t "C:\\Users\\chc\\AppData\\Local\\Temp\\_blank) 查看堆栈和routine等信息  
也可以通过命令行调试，新开shell，输入  
go tool pprof [http://127.0.0.1:8000/debug/pprof/profile](http://127.0.0.1:8000/debug/pprof/profile" \t "C:\\Users\\chc\\AppData\\Local\\Temp\\_blank)  
等待采样，然后再输入  
web  
此时可通过浏览器查看graphviz生成的svg图片。

func main() {

cpuf, err := os.Create("cpu\_profile")

if err != nil {

log.Fatal(err)

}

pprof.StartCPUProfile(cpuf)

defer pprof.StopCPUProfile()

ctx, \_ := context.WithTimeout(context.Background(), time.Second\*5)

test(ctx)

time.Sleep(time.Second \* 3)

memf, err := os.Create("mem\_profile")

if err != nil {

log.Fatal("could not create memory profile: ", err)

}

if err := pprof.WriteHeapProfile(memf); err != nil {

log.Fatal("could not write memory profile: ", err)

}

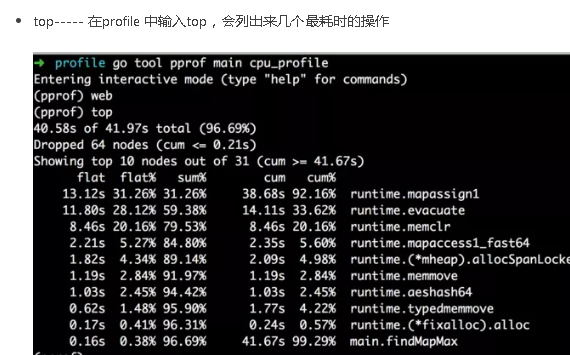
memf.Close()}

会生成两个profile，一个是cpu的，一个是内存的。进入proflie 方法

go tool pprof main profile  
main 代表的是二进制文件，也就是编译出来的可执行文件  
profile 就是上文中生成的profile，可以是cpu\_profile, 也可以是mem\_profile

对于cpu\_profile 来说，代码开始的时候就可以开始统计了  
mem\_profile 部分代码如果写在代码开始的位置是统计不出来的，需要找到一个比较好的位置

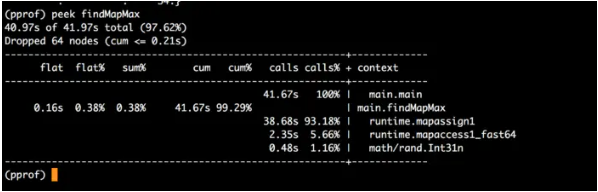
top----- 在profile 中输入top，会列出来几个最耗时的操作



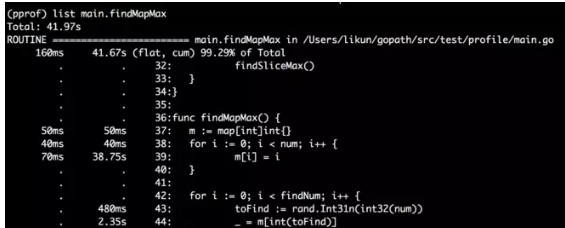
1）list main.main



peek findMapMax (因为根据1可以看出来消耗都在 findMapMax)



1. list main.findMapMax (根据2可以看出来名字是 main.findMapMax)



## pprof运行原理and解释and调优

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | (pprof) top10  Total: 2525 samples  298 11.8% 11.8% 345 13.7% runtime.mapaccess1\_fast64  268 10.6% 22.4% 2124 84.1% main.FindLoops  251 9.9% 32.4% 451 17.9% scanblock  178 7.0% 39.4% 351 13.9% hash\_insert  131 5.2% 44.6% 158 6.3% sweepspan  119 4.7% 49.3% 350 13.9% main.DFS  96 3.8% 53.1% 98 3.9% flushptrbuf  95 3.8% 56.9% 95 3.8% runtime.aeshash64  95 3.8% 60.6% 101 4.0% runtime.settype\_flush  88 3.5% 64.1% 988 39.1% runtime.mallocgc |

pprof模块通过每秒大概100次的对runtime 中的 stack 进行取样来进行统计的。下面来解释一下报表为啥是上面这个样子。  
首先 Total 2525 程序大概运行了25s+  
———-这一部分是针对单个函数的统计  
col1： 在取样中作为栈顶的次数  
col2： 作为堆顶的百分比，以第一行为例统计关系：298/2525 约等于 11.8% 就好理解了  
col3： 排名结果的累加，都是这个位置的数的上面加左面获取的结果，有了这个就可以大概看出来几个热点占用的总比例，非常方便  
———-这一部分是对整个堆栈的统计。与上面的区别是不考虑是否在堆栈顶部。  
col4： 在sample堆栈中出现的次数，不管是waiting还是return只要出现就计入统计。  
col5： 出现次数百分比，与左边报表左边类似。  
col6： 略

查看进程信息

ps -aux|grep server

USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND

USER: 行程拥有者

PID: pid

%CPU: 占用的 CPU 使用率

%MEM: 占用的记忆体使用率

VSZ: 占用的虚拟记忆体大小

RSS: 占用的记忆体大小

TTY: 终端的次要装置号码 (minor device number of tty)

STAT: 该行程的状态:

D: 无法中断的休眠状态 (通常 IO 的进程)

R: 正在执行中

S: 静止状态

T: 暂停执行

Z: 不存在但暂时无法消除

W: 没有足够的记忆体分页可分配

<: 高优先序的行程

N: 低优先序的行程

L: 有记忆体分页分配并锁在记忆体内 (实时系统或捱A I/O)

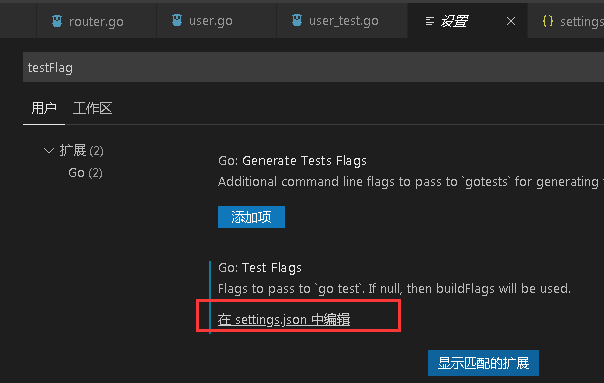
START: 行程开始时间

TIME: 执行的时间

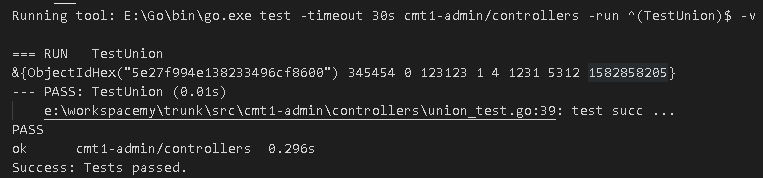
COMMAND:所执行的指令

在vscode里面的testing，显示打印信息。

"go.testFlags": ["-v"]







# **[golang error信息转字符串 x := fmt.Sprintf("%s", err)](https://www.cnblogs.com/liujianzuo888/p/7486092.html)**

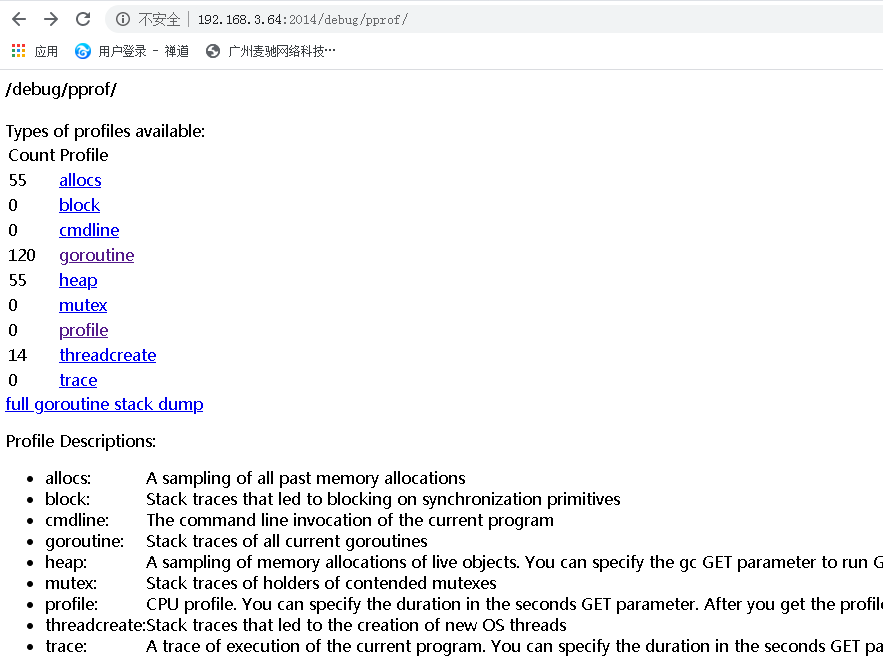
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | \_, \_, ch, err := m.ZkConn.ChildrenW(node)  if err != nil {      x := fmt.Sprintf("%s", err)      if strings.Contains(x, "node does not exist"){          log.Error(fmt.Sprintf("node %s error: %s", m.tcpMonTaskNode, x))          return      }      panic(err)  } |

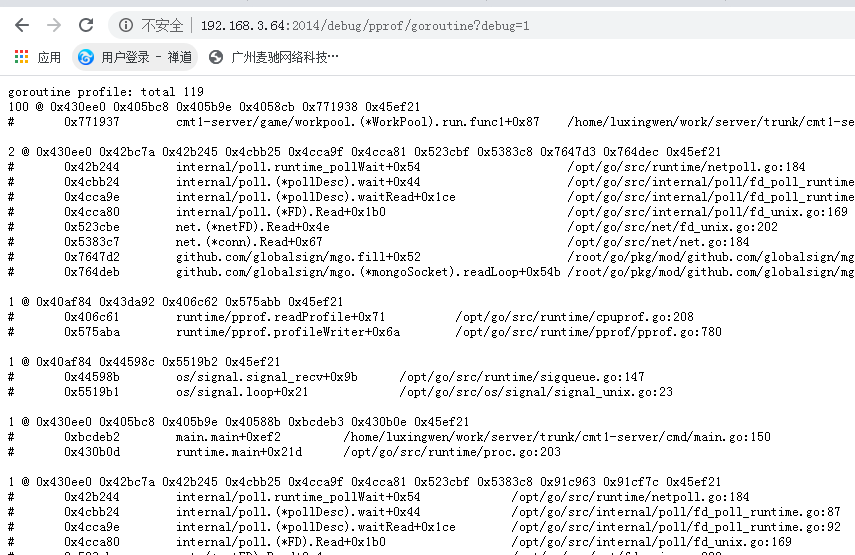
[http://127.0.0.1:8080/debug/pprof](https://link.jianshu.com/?t=http://127.0.0.1:8080/debug/pprof" \t "C:\\Users\\chc\\AppData\\Local\\Temp\\_blank) 查看整体信息  
[http://127.0.0.1:8080/debug/pprof/profile](https://link.jianshu.com/?t=http://127.0.0.1:8080/debug/pprof/profile" \t "C:\\Users\\chc\\AppData\\Local\\Temp\\_blank) 可以将cpu profile下载下来观察分析

从terminal进入profile，进行细致分析  
go tool pprof [http://localhost:6060/debug/pprof/profile](https://link.jianshu.com/?t=http://localhost:6060/debug/pprof/profile" \t "C:\\Users\\chc\\AppData\\Local\\Temp\\_blank)  
go tool pprof [http://localhost:6060/debug/pprof/heap](https://link.jianshu.com/?t=http://localhost:6060/debug/pprof/heap" \t "C:\\Users\\chc\\AppData\\Local\\Temp\\_blank)  
go tool pprof [http://localhost:6060/debug/pprof/block](https://link.jianshu.com/?t=http://localhost:6060/debug/pprof/block" \t "C:\\Users\\chc\\AppData\\Local\\Temp\\_blank)

<http://192.168.3.64:2014/debug/pprof/>

[http://192.168.5.99:2014/debug/pprof/](http://192.168.3.64:2014/debug/pprof/)





[可视化 pprof 的工具 · GopherChina](https://gocn.vip/topics/10144" \t "_blank)

[GopherChina专注于为 Go语言开发者的技术提升，用户数量、活跃度和内容热度均居国内首位。及时推送官方最新资讯、精选优质语言学习文档、传递一手深工程实践经验，并且会定期举办线下meetup、峰](https://gocn.vip/topics/10144" \t "_blank)

[](https://gocn.vip/topics/10144" \t "_blank)

[https://gocn.vip/topics/10144](https://gocn.vip/topics/10144" \t "_blank)