



链滴

# golang pprof 调试 goroutine

作者: [happyue](#)

原文链接: <https://ld246.com/article/1585904405240>

来源网站: [链滴](#)

许可协议: [署名-相同方式共享 4.0 国际 \(CC BY-SA 4.0\)](#)

## 一 给项目加上pprof

### 1 简单的http服务直接使用 `_ "net/http/pprof"`

```
package main

import (
    // 略
    "net/http/pprof" // 会自动注册 handler 到 http server, 方便通过 http 接口获取程序运行采
    报告
    // 略
)

func main() {
    // 略

    runtime.GOMAXPROCS(1) // 限制 CPU 使用数, 避免过载
    runtime.SetMutexProfileFraction(1) // 开启对锁调用的跟踪
    runtime.SetBlockProfileRate(1) // 开启对阻塞操作的跟踪

    go func() {
        // 启动一个 http server, 注意 pprof 相关的 handler 已经自动注册过了
        if err := http.ListenAndServe(":8023", nil); err != nil {
            log.Fatal(err)
        }
        os.Exit(0)
    }()

    // 略
}
```

### 2 [gowork](#)项目使用gin框架的, pprof使用 `"github.com/DeanThompson/ginpprof"`

```
package main

import (
    "github.com/gin-gonic/gin"
    "github.com/DeanThompson/ginpprof"
)

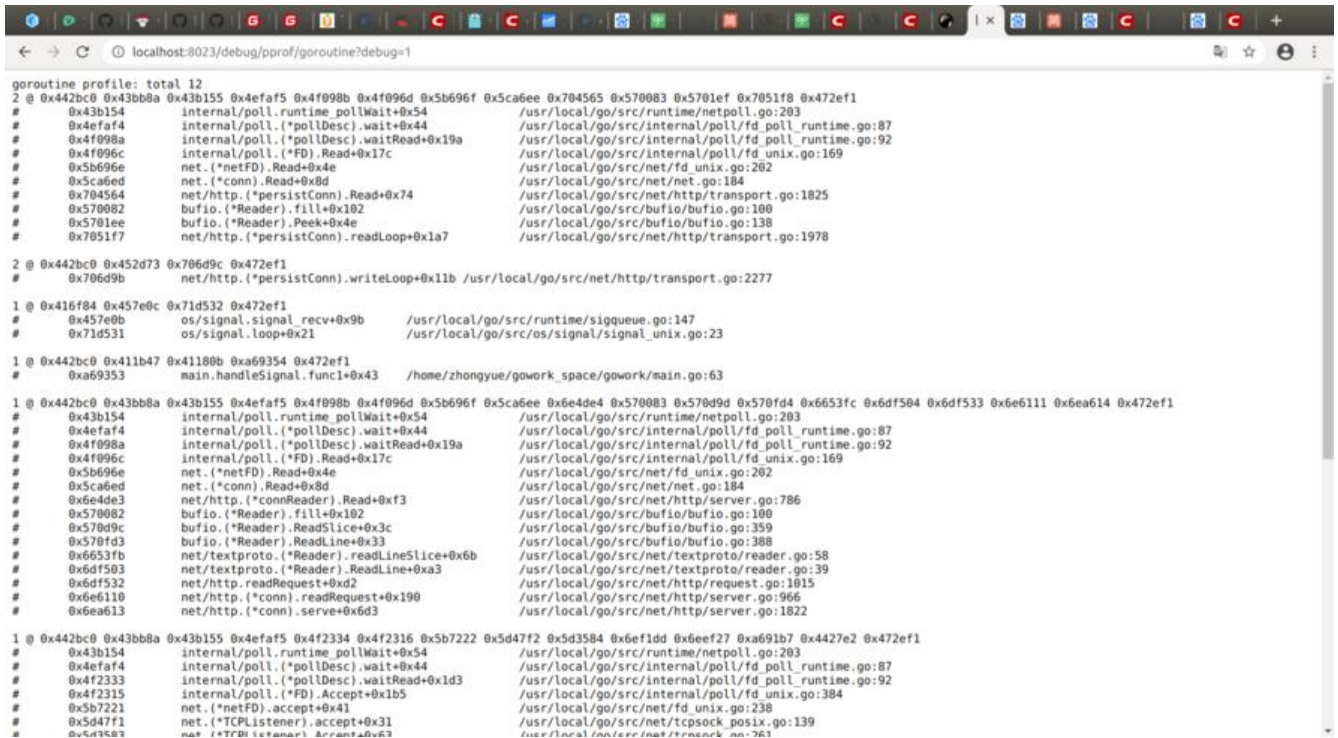
func main() {
    router := gin.Default()

    router.GET("/ping", func(c *gin.Context) {
        c.String(200, "pong")
    })

    // automatically add routers for net/http/pprof
    // e.g. /debug/pprof, /debug/pprof/heap, etc.
    ginpprof.Wrapper(router)

    router.Run(":8023")
}
```

2 运行项目，可以在<http://localhost:8023/debug/pprof/goroutine?debug=1>看到输出的信息。



```
goroutine profile: total 12
2 @ 0x442bc0 0x43bb8a 0x43b155 0x4efaf5 0x4f098b 0x4f096d 0x5b696f 0x5ca6ee 0x704565 0x570083 0x5701ef 0x7051f8 0x472ef1
# 0x43b154 internal/poll.runtime_pollWait+0x54 /usr/local/go/src/runtime/netpoll.go:203
# 0x4efaf4 internal/poll.(*pollDesc).wait+0x44 /usr/local/go/src/internal/poll/fd_poll_runtime.go:87
# 0x4f098a internal/poll.(*pollDesc).waitRead+0x19a /usr/local/go/src/internal/poll/fd_poll_runtime.go:92
# 0x4f096c internal/poll.(*FD).Read+0x17c /usr/local/go/src/internal/poll/fd_unix.go:169
# 0x5b696e net.(*netFD).Read+0x4e /usr/local/go/src/net/fd_unix.go:202
# 0x5ca6ed net.(*conn).Read+0x8d /usr/local/go/src/net/net.go:184
# 0x704564 net/http.(*persistConn).Read+0x74 /usr/local/go/src/net/http/transport.go:1825
# 0x570082 bufio.(*Reader).fill+0x102 /usr/local/go/src/bufio/bufio.go:100
# 0x5701ee bufio.(*Reader).Peek+0x4e /usr/local/go/src/bufio/bufio.go:138
# 0x7051f7 net/http.(*persistConn).readLoop+0x1a7 /usr/local/go/src/net/http/transport.go:1978

2 @ 0x442bc0 0x452d73 0x706d9c 0x472ef1
# 0x706d9b net/http.(*persistConn).writeLoop+0x11b /usr/local/go/src/net/http/transport.go:2277

1 @ 0x416f84 0x457e0c 0x71d532 0x472ef1
# 0x457e0b os/signal.signal_recv+0x9b /usr/local/go/src/runtime/sigqueue.go:147
# 0x71d531 os/signal.loop+0x21 /usr/local/go/src/os/signal/signal_unix.go:23

1 @ 0x442bc0 0x411b47 0x41180b 0xa69354 0x472ef1
# 0xa69353 main.handleSignal.func1+0x43 /home/zhongyue/gowork_space/gowork/main.go:63

1 @ 0x442bc0 0x43bb8a 0x43b155 0x4efaf5 0x4f098b 0x4f096d 0x5b696f 0x5ca6ee 0x6e4de4 0x570083 0x570d9d 0x570fd4 0x6653fc 0x6df504 0x6df533 0x6e6111 0x6e614 0x472ef1
# 0x43b154 internal/poll.runtime_pollWait+0x54 /usr/local/go/src/runtime/netpoll.go:203
# 0x4efaf4 internal/poll.(*pollDesc).wait+0x44 /usr/local/go/src/internal/poll/fd_poll_runtime.go:87
# 0x4f098a internal/poll.(*pollDesc).waitRead+0x19a /usr/local/go/src/internal/poll/fd_poll_runtime.go:92
# 0x4f096c internal/poll.(*FD).Read+0x17c /usr/local/go/src/internal/poll/fd_unix.go:169
# 0x5b696e net.(*netFD).Read+0x4e /usr/local/go/src/net/fd_unix.go:202
# 0x5ca6ed net.(*conn).Read+0x8d /usr/local/go/src/net/net.go:184
# 0x6e4de3 net/http.(*connReader).Read+0xf3 /usr/local/go/src/net/http/server.go:786
# 0x570082 bufio.(*Reader).fill+0x102 /usr/local/go/src/bufio/bufio.go:100
# 0x570d9c bufio.(*Reader).ReadSlice+0x3c /usr/local/go/src/bufio/bufio.go:359
# 0x570fd3 bufio.(*Reader).ReadLine+0x33 /usr/local/go/src/bufio/bufio.go:388
# 0x6653fb net/textproto.(*Reader).readLineSlice+0x6b /usr/local/go/src/net/textproto/reader.go:58
# 0x6df503 net/textproto.(*Reader).ReadLine+0xa3 /usr/local/go/src/net/textproto/reader.go:39
# 0x6df532 net/http.readRequest+0xd2 /usr/local/go/src/net/http/request.go:1015
# 0x6e6110 net/http.(*conn).readRequest+0x190 /usr/local/go/src/net/http/server.go:966
# 0x6e6e13 net/http.(*conn).serve+0x6d3 /usr/local/go/src/net/http/server.go:1822

1 @ 0x442bc0 0x43bb8a 0x43b155 0x4efaf5 0x4f2334 0x4f2316 0x5b7222 0x5d47f2 0x5d3584 0x6ef1dd 0x6eef27 0xa691b7 0x4427e2 0x472ef1
# 0x43b154 internal/poll.runtime_pollWait+0x54 /usr/local/go/src/runtime/netpoll.go:203
# 0x4efaf4 internal/poll.(*pollDesc).wait+0x44 /usr/local/go/src/internal/poll/fd_poll_runtime.go:87
# 0x4f2333 internal/poll.(*pollDesc).waitRead+0x1d3 /usr/local/go/src/internal/poll/fd_poll_runtime.go:92
# 0x4f2315 internal/poll.(*FD).Accept+0x1b5 /usr/local/go/src/internal/poll/fd_unix.go:384
# 0x5b7221 net.(*netFD).accept+0x41 /usr/local/go/src/net/fd_unix.go:238
# 0x5d47f1 net.(*TCPListener).accept+0x31 /usr/local/go/src/net/tcpsock_posix.go:139
# 0x447c02 net.(*TCPListener).Accept+0x63 /usr/local/src/net/tcpsock_posix.go:763
```

可以看到，刚开始协程total: 12，然后做一些引起开协程并结束动作，看看协程的状态：



```
goroutine profile: total 21
1 @ 0x416f84 0x457e0c 0x71d532 0x472ef1
# 0x457e0b os/signal.signal_recv+0x9b /usr/local/go/src/runtime/sigqueue.go:147
# 0x71d531 os/signal.loop+0x21 /usr/local/go/src/os/signal/signal_unix.go:23

1 @ 0x442bc0 0x411b47 0x41180b 0x9f1752 0xa03053 0xa54eba 0xa036b7 0x472ef1
# 0x9f1751 golang.org/x/crypto/ssh.(*Session).Wait+0x51 /usr/local/go/src/golang.org/x/crypto/ssh/session.go:403
# 0xa03052 gowork/model.(*LogicSSHWSession).Wait+0x32 /home/zhongyue/gowork_space/gowork/model/logic_ssh_ws_session.go:100
# 0xa54eb9 gowork/controller.NewSSHShell.func3+0x29 /home/zhongyue/gowork_space/gowork/controller/controller.go:100
# 0xa036b6 gowork/model.(*waitGroupWrapper).Wrap.func1+0x26 /home/zhongyue/gowork_space/gowork/model/wait_group_wrapper.go:100
```

这时协程已经失控了。

我们使用命令行go自带的工具命令

go tool pprof <http://localhost:8023/debug/pprof/goroutine>会返回一个交互对话框，

这里使用 top, list xxx, web 命令查看具体信息

```

zhongyue@zhongyue ~/gowork_space/gowork (master)
└─> go tool pprof http://localhost:8023/debug/pprof/goroutine
Fetching profile over HTTP from http://localhost:8023/debug/pprof/goroutine
Saved profile in /home/zhongyue/pprof/pprof.main.goroutine.001.pb.gz
File: main
Build ID: 0a652f140892617b5effa0e14577f0d840a700f8
Type: goroutine
Time: Apr 3, 2020 at 2:34pm (CST)
Entering interactive mode (type "help" for commands, "o" for options)
(pprof) top
Showing nodes accounting for 27, 100% of 27 total
Showing top 10 nodes out of 82
      flat flat% sum%      cum cum%
    24 88.89% 88.89%    24 88.89% runtime.gopark
     1  3.70% 92.59%     1  3.70% net/http.(*connReader).backgroundRead
     1  3.70% 96.30%     1  3.70% runtime.notetsleepg
     1  3.70% 100%     1  3.70% runtime/pprof.writeRuntimeProfile
     0  0% 100%     2  7.41% bufio.(*Reader).Read
     0  0% 100%     1  3.70% database/sql.(*DB).connectionOpener
     0  0% 100%     1  3.70% database/sql.(*DB).connectionResetter
     0  0% 100%     1  3.70% github.com/DeanThompson/ginpprof.GoroutineHandler.func1
     0  0% 100%     3 11.11% github.com/gin-contrib/sessions.Sessions.func1
     0  0% 100%     3 11.11% github.com/gin-gonic/gin.(*Context).Next

```

list 具体某一条

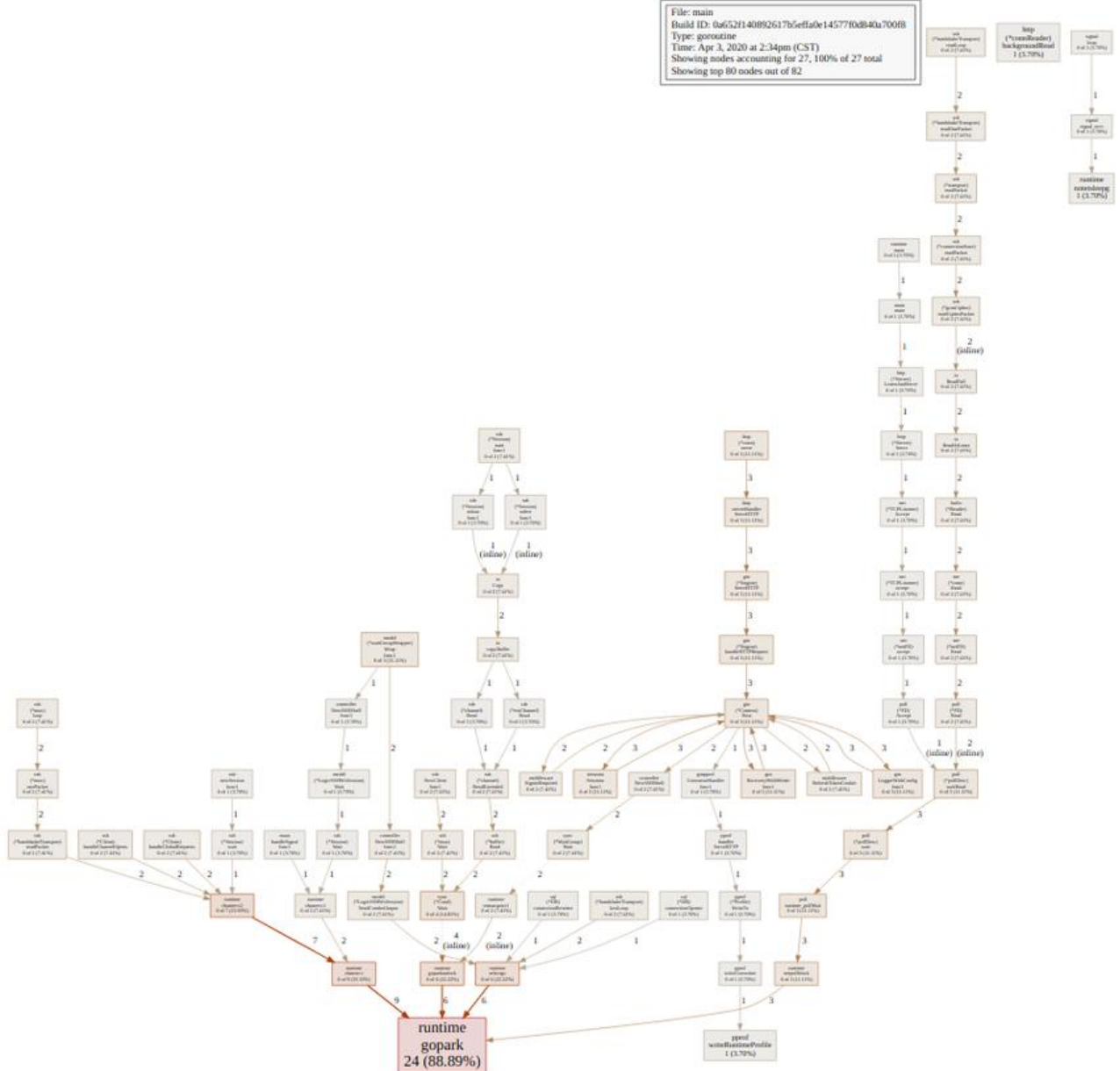
```

(pprof) list Read
Total: 27
ROUTINE ===== bufio.(*Reader).Read in /usr/local/go/src/bu
 0          2 (flat, cum) 7.41% of Total
.          .      221:          }
.          .      222:          // One read.
.          .      223:          // Do not use b.fill, which will loop
.          .      224:          b.r = 0
.          .      225:          b.w = 0
.          2      226:          n, b.err = b.rd.Read(b.buf)
.          .      227:          if n < 0 {
.          .      228:              panic(errNegativeRead)
.          .      229:          }
.          .      230:          if n == 0 {
.          .      231:              return 0, b.readErr()
ROUTINE ===== golang.org/x/crypto/ssh.(*buffer).Read in /h
 0          2 (flat, cum) 7.41% of Total
.          .      89:          if b.closed {
.          .      90:              err = io.EOF
.          .      91:              break
.          .      92:          }
.          .      93:          // out of buffers, wait for producer
.          2      94:          b.Cond.Wait()

```

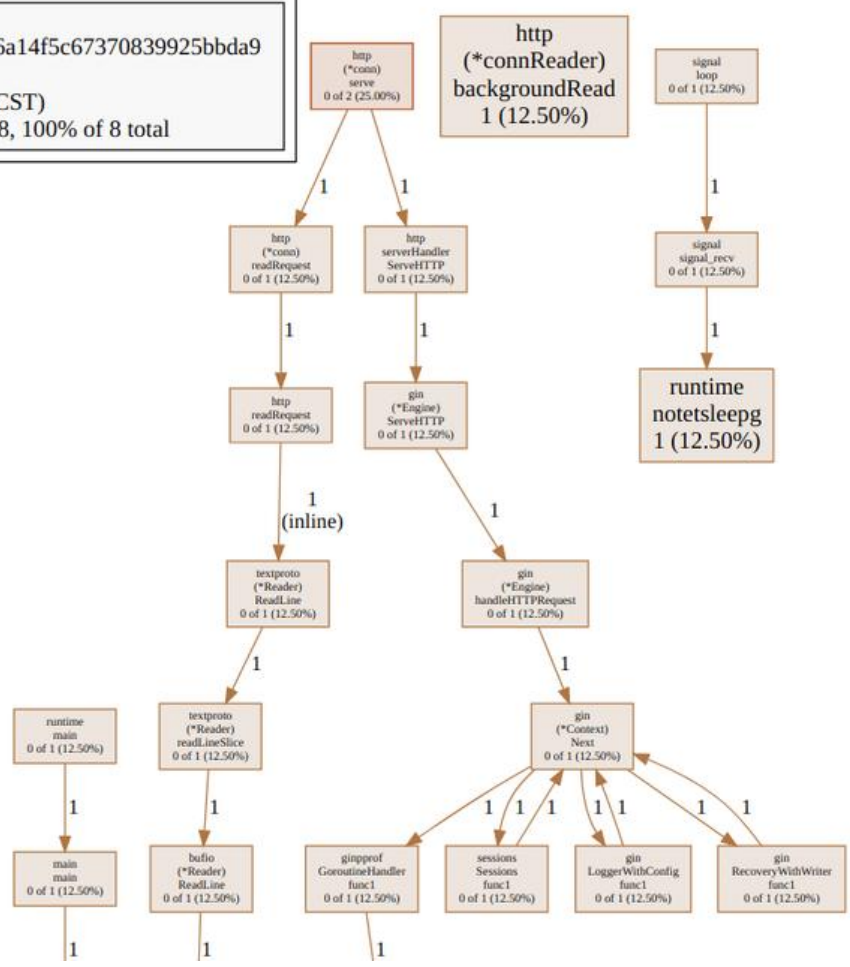
web命令，虽然这个命令的名字叫“web”，但它的实际行为是产生一个.svg文件，并调用你的系统里设置的默认打开.svg的程序打开它。如果遇到failed to execute dot. Is Graphviz installed? Error: exec: "dot": executable file not found in \$PATH 错误，需要安装Graphviz，ubuntu下安装直接su o apt-get install graphviz。好了，我在软件中做了一些ssh的操作，这些操作会引起后台开启协程在浏览器中会看到下图：

File: main  
Build ID: 0a652f140892617b5effa0e14577f0db40a700ff  
Type: goroutine  
Time: Apr 3, 2020 at 2:34pm (CST)  
Showing nodes accounting for 27, 100% of 27 total  
Showing top 80 nodes out of 82



这里图片不是很清晰，主要想说明pprof调试确定协程泄露的方法。解决掉协程泄露失控的问题后，下：

File: main  
 Build ID: 17319513ab9afb9666a14f5c67370839925bbda9  
 Type: goroutine  
 Time: Apr 3, 2020 at 4:03pm (CST)  
 Showing nodes accounting for 8, 100% of 8 total



感谢 <https://blog.wolfogre.com/posts/go-pprof-practice>