Tony Bai

一个程序员的心路历程

- 关于我
- 文章列表

Go程序调试、分析与优化

- 八月 25, 2015
- 17 条评论

Brad Fitzpatrick在YAPC Asia 2015（Yet Another Perl Conference）上做了一次技术分享，题为："Go Debugging, Profiling, and Optimization"。个人感觉这篇分享中价值最大的是BradFitz现场演示的一个有关如何对Go程序进行调试、分析和优化的 Demo，Brad将demo上传到了他个人在github.com的repo中，但不知为何，repo中的代码似乎与repo里talk.md中的说明不甚一致(btw，我并没有看video)。于 是打算在这里按照Brad的思路重新走一遍demo的演示流程(所有演示代码在这里可以下载到)。

一、实验环境

```
$uname -a
Linux pc-tony 3.13.0-61-generic #100~precise1-Ubuntu SMP Wed Jul 29
12:06:40 UTC 2015 x86_64 x86_64 x86_64 GNU/Linux
```

注意：在`Darwin`或`Windows`下，`profile`的结果可能与这里有很大不同（甚至完全不一样的输出和瓶颈热点）。

```
$go version
go version go1.5 linux/amd64

$ go env
GOARCH="amd64"
GOBIN="/home1/tonybai/.bin/go15/bin"
GOEXE=""
GOHOSTARCH="amd64"
GOHOSTOS="linux"
GOOS="linux"
GOPATH="/home1/tonybai/proj/GoProjects"
GORACE=""
GOROOT="/home1/tonybai/.bin/go15"
GOTOOLDIR="/home1/tonybai/.bin/go15/pkg/tool/linux_amd64"
GO15VENDOREXPERIMENT="1"
CC="gcc"
GOGCCFLAGS="-fPIC -m64 -pthread -fmessage-length=0"
CXX="g++"
```

```
CGO_ENABLED="1"
```

代码基于Brad的github.com/bradfitz/talk-yapc-asia-2015。

二、待优化程序(step0)

待优化程序，也就是原始程序，我们放在step0中：

```
//go-debug-profile-optimization/step0/demo.go

package main

import (
    "fmt"
    "log"
    "net/http"
    "regexp"
)

var visitors int

func handleHi(w http.ResponseWriter, r *http.Request) {
    if match, _ := regexp.MatchString(`^\w*$`, r.FormValue("color"));
!match {
        http.Error(w, "Optional color is invalid",
http.StatusBadRequest)
        return
    }
    visitors++
    w.Header().Set("Content-Type", "text/html; charset=utf-8")
    w.Write([]byte("<h1 style='color: " + r.FormValue("color") +
        "'>Welcome!</h1>You are visitor number " + fmt.Sprint(visitors)
+ "!"))
}

func main() {
    log.Printf("Starting on port 8080")
    http.HandleFunc("/hi", handleHi)
    log.Fatal(http.ListenAndServe("127.0.0.1:8080", nil))
}

$go run demo.go
2015/08/25 09:42:35 Starting on port 8080
```

在浏览器输入：http://localhost:8080/hi

一切顺利的话，页面会显示：

```
Welcome!
```

```
You are visitor number 1!
```

三、添加测试代码

按照talk.md中的说明，brad repo中demo中根本没有测试代码(commit 2427d0faa12ed1fb05f1e6a1e69307c11259c2b2)。

于是我根据作者的意图，新增了demo_test.go，采用TestHandleHi_Recorder和TestHandleHi_TestServer对HandleHi进行测试：

```
//go-debug-profile-optimization/step0/demo_test.go
package main

import (
    "bufio"
    "net/http"
    "net/http/httptest"
    "strings"
    "testing"
)

func TestHandleHi_Recorder(t *testing.T) {
    rw := httptest.NewRecorder()
    handleHi(rw, req(t, "GET / HTTP/1.0\r\n\r\n"))
    if !strings.Contains(rw.Body.String(), "visitor number") {
        t.Errorf("Unexpected output: %s", rw.Body)
    }
}

func req(t *testing.T, v string) *http.Request {
    req, err := http.ReadRequest(bufio.NewReader(strings.NewReader(v)))
    if err != nil {
        t.Fatal(err)
    }
    return req
}

func TestHandleHi_TestServer(t *testing.T) {
    ts := httptest.NewServer(http.HandlerFunc(handleHi))
    defer ts.Close()
    res, err := http.Get(ts.URL)
    if err != nil {
        t.Error(err)
        return
    }
    if g, w := res.Header.Get("Content-Type"), "text/html; charset=utf-
8"; g != w {
        t.Errorf("Content-Type = %q; want %q", g, w)
```

```
    }
    slurp, err := ioutil.ReadAll(res.Body)
    defer res.Body.Close()
    if err != nil {
        t.Error(err)
        return
    }
    t.Logf("Got: %s", slurp)
}

$ go test -v
=== RUN   TestHandleHi_Recorder
— PASS: TestHandleHi_Recorder (0.00s)
=== RUN   TestHandleHi_TestServer
— PASS: TestHandleHi_TestServer (0.00s)
    demo_test.go:45: Got: <h1 style='color: '>Welcome!</h1>You are
visitor number 2!
PASS
ok      _/home1/tonybai/proj/opensource/github/experiments/go-debug-
profile-optimization/step0    0.007s
```

测试通过！

至此，step0使命结束。

四、Race Detector(竞态分析)

并发设计使得程序可以更好更有效的利用现代处理器的多核心。但并发设计很容易引入竞态，导致严重bug。Go程序中竞态就是当多个goroutine并发 访问某共享数据且未使用同步机制时，且至少一个goroutine进行了写操作。不过go工具自带race分析功能。在分析优化step0中demo代码 前，我们先要保证demo代码中不存在竞态。

工具的使用方法就是在go test后加上-race标志，在step0目录下：

```
$ go test -v -race
=== RUN   TestHandleHi_Recorder
— PASS: TestHandleHi_Recorder (0.00s)
=== RUN   TestHandleHi_TestServer
— PASS: TestHandleHi_TestServer (0.00s)
    demo_test.go:45: Got: <h1 style='color: '>Welcome!</h1>You are
visitor number 2!
PASS
ok      _/home1/tonybai/proj/opensource/github/experiments/go-debug-
profile-optimization/step0    1.012s
```

-race通过做运行时分析做竞态分析，虽然不存在误报，但却存在实际有竞态，但工具没发现的情况。接下来我们改造一下测试代码，让test并发起来：

向step1(copy自step0)中demo_test.go中添加一个test method:

```
//go-debug-profile-optimization/step1/demo_test.go
… …
func TestHandleHi_TestServer_Parallel(t *testing.T) {
    ts := httptest.NewServer(http.HandlerFunc(handleHi))
    defer ts.Close()
    var wg sync.WaitGroup
    for i := 0; i < 2; i++ {
        wg.Add(1)
        go func() {
            defer wg.Done()
            res, err := http.Get(ts.URL)
            if err != nil {
                t.Error(err)
                return
            }
            if g, w := res.Header.Get("Content-Type"), "text/html;
charset=utf-8"; g != w {
                t.Errorf("Content-Type = %q; want %q", g, w)
            }
            slurp, err := ioutil.ReadAll(res.Body)
            defer res.Body.Close()
            if err != nil {
                t.Error(err)
                return
            }
            t.Logf("Got: %s", slurp)
        }()
    }
    wg.Wait()
}
… …
```

执行竞态test:

```
$ go test -v -race
=== RUN   TestHandleHi_Recorder
— PASS: TestHandleHi_Recorder (0.00s)
=== RUN   TestHandleHi_TestServer
— PASS: TestHandleHi_TestServer (0.00s)
    demo_test.go:46: Got: <h1 style='color: '>Welcome!</h1>You are
visitor number 2!
=== RUN   TestHandleHi_TestServer_Parallel
==================
WARNING: DATA RACE
Read by goroutine 22:
  _/home1/tonybai/proj/opensource/github/experiments/go-debug-profile-
```

```
optimization/step1.handleHi()
      /home1/tonybai/proj/opensource/github/experiments/go-debug-
profile-optimization/step1/demo.go:17 +0xf5
  net/http.HandlerFunc.ServeHTTP()
      /tmp/workdir/go/src/net/http/server.go:1422 +0×47
  net/http/httptest.(*waitGroupHandler).ServeHTTP()
      /tmp/workdir/go/src/net/http/httptest/server.go:200 +0xfe
  net/http.serverHandler.ServeHTTP()
      /tmp/workdir/go/src/net/http/server.go:1862 +0×206
  net/http.(*conn).serve()
      /tmp/workdir/go/src/net/http/server.go:1361 +0x117c

Previous write by goroutine 25:
  _/home1/tonybai/proj/opensource/github/experiments/go-debug-profile-
optimization/step1.handleHi()
      /home1/tonybai/proj/opensource/github/experiments/go-debug-
profile-optimization/step1/demo.go:17 +0×111
  net/http.HandlerFunc.ServeHTTP()
      /tmp/workdir/go/src/net/http/server.go:1422 +0×47
  net/http/httptest.(*waitGroupHandler).ServeHTTP()
      /tmp/workdir/go/src/net/http/httptest/server.go:200 +0xfe
  net/http.serverHandler.ServeHTTP()
      /tmp/workdir/go/src/net/http/server.go:1862 +0×206
  net/http.(*conn).serve()
      /tmp/workdir/go/src/net/http/server.go:1361 +0x117c

Goroutine 22 (running) created at:
  net/http.(*Server).Serve()
      /tmp/workdir/go/src/net/http/server.go:1910 +0×464

Goroutine 25 (running) created at:
  net/http.(*Server).Serve()
      /tmp/workdir/go/src/net/http/server.go:1910 +0×464
==================
— PASS: TestHandleHi_TestServer_Parallel (0.00s)
    demo_test.go:71: Got: <h1 style='color: '>Welcome!</h1>You are
visitor number 3!
    demo_test.go:71: Got: <h1 style='color: '>Welcome!</h1>You are
visitor number 4!
PASS
Found 1 data race(s)
exit status 66
FAIL    _/home1/tonybai/proj/opensource/github/experiments/go-debug-
profile-optimization/step1    1.023s
```

工具发现demo.go第17行：

```
   visitors++
```

是一处潜在的竞态条件。

visitors被多个goroutine访问但未采用同步机制。

既然发现了竞态条件，我们就需要fix it。有多种fix方法可选：

1、使用channel
2、使用Mutex
3、使用atomic

Brad使用了atomic：

```
//go-debug-profile-optimization/step1/demo.go
… …
var visitors int64 // must be accessed atomically

func handleHi(w http.ResponseWriter, r *http.Request) {
    if match, _ := regexp.MatchString(`^\w*$`, r.FormValue("color"));
!match {
        http.Error(w, "Optional color is invalid",
http.StatusBadRequest)
        return
    }
    visitNum := atomic.AddInt64(&visitors, 1)
    w.Header().Set("Content-Type", "text/html; charset=utf-8")
    w.Write([]byte("<h1 style='color: " + r.FormValue("color") +
        "'>Welcome!</h1>You are visitor number " + fmt.Sprint(visitNum)
+ "!"))
}
… …
```

再做一次测试：

```
$ go test -v -race
=== RUN   TestHandleHi_Recorder
— PASS: TestHandleHi_Recorder (0.00s)
=== RUN   TestHandleHi_TestServer
— PASS: TestHandleHi_TestServer (0.00s)
    demo_test.go:46: Got: <h1 style='color: '>Welcome!</h1>You are
visitor number 2!
=== RUN   TestHandleHi_TestServer_Parallel
— PASS: TestHandleHi_TestServer_Parallel (0.00s)
    demo_test.go:71: Got: <h1 style='color: '>Welcome!</h1>You are
visitor number 3!
    demo_test.go:71: Got: <h1 style='color: '>Welcome!</h1>You are
visitor number 4!
PASS
ok      _/home1/tonybai/proj/opensource/github/experiments/go-debug-
profile-optimization/step1    1.020s
```

竞态条件被消除了！

至此，step1结束了使命！

五、CPU Profiling

要做CPU Profilling，我们需要benchmark数据，Go test提供benchmark test功能，我们只要写对应的Benchmark测试方法即可：

```
//go-debug-profile-optimization/step2/demo_test.go
… …
func BenchmarkHi(b *testing.B) {
    b.ReportAllocs()

    req, err := http.ReadRequest(bufio.NewReader(strings.NewReader("GET
/ HTTP/1.0\r\n\r\n")))
    if err != nil {
        b.Fatal(err)
    }

    for i := 0; i < b.N; i++ {
        rw := httptest.NewRecorder()
        handleHi(rw, req)
    }
}
… …

$ go test -v -run=^$ -bench=.
PASS
BenchmarkHi-4      100000         14808 ns/op        4961 B/op
81 allocs/op
ok      _/home1/tonybai/proj/opensource/github/experiments/go-debug-
profile-optimization/step2    1.648s
```

开始CPU Profiling：

```
$ go test -v -run=^$ -bench=^BenchmarkHi$ -benchtime=2s -
cpuprofile=prof.cpu
PASS
BenchmarkHi-4      200000         14679 ns/op        4961 B/op
81 allocs/op
ok      _/home1/tonybai/proj/opensource/github/experiments/go-debug-
profile-optimization/step2    3.096s
```

执行完benchmark test后，step2目录下出现两个新文件prof.cpu和step2.test，这两个文件将作为后续go tool pprof的输入：

```
$ls
demo.go        demo_test.go    prof.cpu    step2.test*
```

使用go profile viewer工具：

```
$ go tool pprof step2.test prof.cpu
Entering interactive mode (type "help" for commands)
(pprof) top
1830ms of 3560ms total (51.40%)
Dropped 53 nodes (cum <= 17.80ms)
Showing top 10 nodes out of 133 (cum >= 1290ms)
      flat  flat%   sum%        cum   cum%
     480ms 13.48% 13.48%      980ms 27.53%  runtime.growslice
     360ms 10.11% 23.60%      700ms 19.66%  runtime.mallocgc
     170ms  4.78% 28.37%      170ms  4.78%  runtime.heapBitsSetType
     170ms  4.78% 33.15%      200ms  5.62%  runtime.scanblock
     120ms  3.37% 36.52%     1100ms 30.90%  regexp.makeOnePass.func2
     120ms  3.37% 39.89%      550ms 15.45%  runtime.newarray
     110ms  3.09% 42.98%      300ms  8.43%  runtime.makeslice
     110ms  3.09% 46.07%      220ms  6.18%  runtime.mapassign1
     100ms  2.81% 48.88%      100ms  2.81%  runtime.futex
      90ms  2.53% 51.40%     1290ms 36.24%  regexp.makeOnePass

(pprof) top -cum
0.18s of 3.56s total ( 5.06%)
Dropped 53 nodes (cum <= 0.02s)
Showing top 10 nodes out of 133 (cum >= 1.29s)
      flat  flat%   sum%        cum   cum%
         0     0%     0%      3.26s 91.57%  runtime.goexit
     0.02s  0.56%  0.56%      2.87s 80.62%  BenchmarkHi
         0     0%  0.56%      2.87s 80.62%  testing.(*B).launch
         0     0%  0.56%      2.87s 80.62%  testing.(*B).runN
     0.03s  0.84%  1.40%      2.80s 78.65%  step2.handleHi
     0.01s  0.28%  1.69%      2.46s 69.10%  regexp.MatchString
         0     0%  1.69%      2.24s 62.92%  regexp.Compile
         0     0%  1.69%      2.24s 62.92%  regexp.compile
     0.03s  0.84%  2.53%      1.56s 43.82%  regexp.compileOnePass
     0.09s  2.53%  5.06%      1.29s 36.24%  regexp.makeOnePass

(pprof) list handleHi
Total: 3.56s
ROUTINE ======================== handleHi in go-debug-profile-
optimization/step2/demo.go
      30ms      2.80s (flat, cum) 78.65% of Total
         .          .      9:)
         .          .     10:
         .          .     11:var visitors int64 // must be accessed
atomically
         .          .     12:
         .          .     13:func handleHi(w http.ResponseWriter, r
*http.Request) {
```

```
         .      2.47s     14:    if match, _ :=
regexp.MatchString(`^\w*$`, r.FormValue("color")); !match {
         .          .     15:        http.Error(w, "Optional color is
invalid", http.StatusBadRequest)
         .          .     16:        return
         .          .     17:    }
      10ms       20ms     18:    visitNum := atomic.AddInt64(&visitors,
1)
      10ms       90ms     19:    w.Header().Set("Content-Type",
"text/html; charset=utf-8")
      10ms       20ms     20:    w.Write([]byte("<h1 style='color: " +
r.FormValue("color") +
         .      200ms     21:        "'>Welcome!</h1>You are visitor
number " + fmt.Sprint(visitNum) + "!"))
         .          .     22:}
         .          .     23:
         .          .     24:func main() {
         .          .     25:    log.Printf("Starting on port 8080")
         .          .     26:    http.HandleFunc("/hi", handleHi)
(pprof)
```

从top -cum来看，handleHi消耗cpu较大，而handleHi中，又是MatchString耗时最长。

六、第一次优化

前面已经发现MatchString较为耗时，优化手段：让正则式仅编译一次(step3)：

```
// go-debug-profile-optimization/step3/demo.go

… …
var visitors int64 // must be accessed atomically

var rxOptionalID = regexp.MustCompile(`^\d*$`)

func handleHi(w http.ResponseWriter, r *http.Request) {
    if !rxOptionalID.MatchString(r.FormValue("color")) {
        http.Error(w, "Optional color is invalid",
http.StatusBadRequest)
        return
    }

    visitNum := atomic.AddInt64(&visitors, 1)
    w.Header().Set("Content-Type", "text/html; charset=utf-8")
    w.Write([]byte("<h1 style='color: " + r.FormValue("color") +
        "'>Welcome!</h1>You are visitor number " + fmt.Sprint(visitNum)
+ "!"))
}
… …
```

运行一下bench：

```
$ go test -bench=.
PASS
BenchmarkHi-4     1000000          1678 ns/op         720 B/op
       9 allocs/op
ok      _/home1/tonybai/proj/opensource/github/experiments/go-debug-
profile-optimization/step3    1.710s
```

对比之前在step2中运行的bench结果：

```
$ go test -v -run=^$ -bench=.
PASS
BenchmarkHi-4      100000         14808 ns/op        4961 B/op
81 allocs/op
ok      _/home1/tonybai/proj/opensource/github/experiments/go-debug-
profile-optimization/step2    1.648s
```

耗时相同，但优化后的bench运行了100w次，而之前的Bench运行10w次，相当于性能提高10倍。

再看看cpu prof结果：

```
$ go test -v -run=^$ -bench=^BenchmarkHi$ -benchtime=3s -
cpuprofile=prof.cpu
PASS
BenchmarkHi-4     3000000          1640 ns/op         720 B/op
       9 allocs/op
ok      _/home1/tonybai/proj/opensource/github/experiments/go-debug-
profile-optimization/step3    6.540s

$ go tool pprof step3.test prof.cpu
Entering interactive mode (type "help" for commands)
(pprof) top -cum 30
2.74s of 8.07s total (33.95%)
Dropped 72 nodes (cum <= 0.04s)
Showing top 30 nodes out of 103 (cum >= 0.56s)
      flat  flat%   sum%        cum   cum%
         0     0%     0%      7.17s 88.85%  runtime.goexit
     0.05s  0.62%  0.62%      6.21s 76.95%  step3.BenchmarkHi
         0     0%  0.62%      6.21s 76.95%  testing.(*B).launch
         0     0%  0.62%      6.21s 76.95%  testing.(*B).runN
     0.06s  0.74%  1.36%      4.96s 61.46%  step3.handleHi
     1.15s 14.25% 15.61%      2.35s 29.12%  runtime.mallocgc
     0.02s  0.25% 15.86%      1.63s 20.20%  runtime.systemstack
         0     0% 15.86%      1.53s 18.96%  net/http.Header.Set
     0.06s  0.74% 16.60%      1.53s 18.96%
net/textproto.MIMEHeader.Set
     0.09s  1.12% 17.72%      1.22s 15.12%  runtime.newobject
```

```
     0.05s  0.62% 18.34%      1.09s 13.51%  fmt.Sprint
     0.20s  2.48% 20.82%         1s 12.39%  runtime.mapassign1
         0     0% 20.82%      0.81s 10.04%  runtime.mcall
     0.01s  0.12% 20.94%      0.79s  9.79%  runtime.schedule
     0.05s  0.62% 21.56%      0.76s  9.42%  regexp.
(*Regexp).MatchString
     0.09s  1.12% 22.68%      0.71s  8.80%  regexp.(*Regexp).doExecute
     0.01s  0.12% 22.80%      0.71s  8.80%  runtime.concatstring5
     0.20s  2.48% 25.28%      0.70s  8.67%  runtime.concatstrings
         0     0% 25.28%      0.69s  8.55%  runtime.gosweepone
     0.05s  0.62% 25.90%      0.69s  8.55%  runtime.mSpan_Sweep
         0     0% 25.90%      0.68s  8.43%  runtime.bgsweep
     0.04s   0.5% 26.39%      0.68s  8.43%  runtime.newarray
     0.01s  0.12% 26.52%      0.67s  8.30%  runtime.goschedImpl
     0.01s  0.12% 26.64%      0.65s  8.05%  runtime.gosched_m
         0     0% 26.64%      0.65s  8.05%  runtime.gosweepone.func1
     0.01s  0.12% 26.77%      0.65s  8.05%  runtime.sweepone
     0.28s  3.47% 30.24%      0.62s  7.68%  runtime.makemap
     0.17s  2.11% 32.34%      0.59s  7.31%  runtime.heapBitsSweepSpan
     0.02s  0.25% 32.59%      0.58s  7.19%  fmt.(*pp).doPrint
     0.11s  1.36% 33.95%      0.56s  6.94%  fmt.(*pp).printArg
```

handleHi耗时有一定下降。

七、Mem Profiling

在step3目录下执行bench，获取mem分配数据：

```
$ go test -v -run=^$ -bench=^BenchmarkHi$ -benchtime=2s -
memprofile=prof.mem
PASS
BenchmarkHi-4     2000000          1657 ns/op         720 B/op
       9 allocs/op
ok      _/home1/tonybai/proj/opensource/github/experiments/go-debug-
profile-optimization/step3    5.002s
```

使用pprof工具分析mem：

```
$ go tool pprof -alloc_space step3.test prof.mem
Entering interactive mode (type "help" for commands)
(pprof) top
2065.91MB of 2067.41MB total (99.93%)
Dropped 14 nodes (cum <= 10.34MB)
      flat  flat%   sum%        cum   cum%
 1076.35MB 52.06% 52.06%  1076.35MB 52.06%
net/textproto.MIMEHeader.Set
  535.54MB 25.90% 77.97%  2066.91MB   100%  step3.BenchmarkHi
  406.52MB 19.66% 97.63%  1531.37MB 74.07%  step3.handleHi
```

```
   47.50MB  2.30% 99.93%    48.50MB  2.35%  fmt.Sprint
         0     0% 99.93%  1076.35MB 52.06%  net/http.Header.Set
         0     0% 99.93%  2066.91MB   100%  runtime.goexit
         0     0% 99.93%  2066.91MB   100%  testing.(*B).launch
         0     0% 99.93%  2066.91MB   100%  testing.(*B).runN

(pprof) top -cum
2065.91MB of 2067.41MB total (99.93%)
Dropped 14 nodes (cum <= 10.34MB)
      flat  flat%   sum%        cum   cum%
  535.54MB 25.90% 25.90%  2066.91MB   100%  step3.BenchmarkHi
         0     0% 25.90%  2066.91MB   100%  runtime.goexit
         0     0% 25.90%  2066.91MB   100%  testing.(*B).launch
         0     0% 25.90%  2066.91MB   100%  testing.(*B).runN
  406.52MB 19.66% 45.57%  1531.37MB 74.07%  step3.handleHi
         0     0% 45.57%  1076.35MB 52.06%  net/http.Header.Set
 1076.35MB 52.06% 97.63%  1076.35MB 52.06%
net/textproto.MIMEHeader.Set
   47.50MB  2.30% 99.93%    48.50MB  2.35%  fmt.Sprint

(pprof) list handleHi
Total: 2.02GB
     ROUTINE =========step3.handleHi in step3/demo.go
  406.52MB     1.50GB (flat, cum) 74.07% of Total
         .          .     17:        http.Error(w, "Optional color is
invalid", http.StatusBadRequest)
         .          .     18:        return
         .          .     19:    }
         .          .     20:
         .          .     21:    visitNum := atomic.AddInt64(&visitors,
1)
         .     1.05GB     22:    w.Header().Set("Content-Type",
"text/html; charset=utf-8")
         .          .     23:    w.Write([]byte("<h1 style='color: " +
r.FormValue("color") +
  406.52MB   455.02MB     24:        "'>Welcome!</h1>You are visitor
number " + fmt.Sprint(visitNum) + "!"))
         .          .     25:}
         .          .     26:
         .          .     27:func main() {
         .          .     28:    log.Printf("Starting on port 8080")
         .          .     29:    http.HandleFunc("/hi", handleHi)
(pprof)
```

可以看到handleHi22、23两行占用了较多内存。

八、第二次优化

第二次优化的方法:
1、删除w.Header().Set这行
2、用fmt.Fprintf替代w.Write

第二次优化的代码在step4目录中:

```
// go-debug-profile-optimization/step4/demo.go
… …
func handleHi(w http.ResponseWriter, r *http.Request) {
    if !rxOptionalID.MatchString(r.FormValue("color")) {
        http.Error(w, "Optional color is invalid",
http.StatusBadRequest)
        return
    }

    visitNum := atomic.AddInt64(&visitors, 1)
    fmt.Fprintf(w, "<html><h1 stype='color: \"%s\"'>Welcome!</h1>You
are visitor number %d!", r.FormValue("color"), visitNum)
}
… …
```

执行一遍pprof:

```
$ go test -v -run=^$ -bench=^BenchmarkHi$ -benchtime=2s -
memprofile=prof.mem
PASS
BenchmarkHi-4     2000000          1428 ns/op         304 B/op
       6 allocs/op
ok      _/home1/tonybai/proj/opensource/github/experiments/go-debug-
profile-optimization/step4    4.343s

$ go tool pprof -alloc_space step4.test prof.mem
Entering interactive mode (type "help" for commands)
(pprof) top
868.06MB of 868.56MB total (99.94%)
Dropped 5 nodes (cum <= 4.34MB)
      flat  flat%   sum%        cum   cum%
  559.54MB 64.42% 64.42%   868.06MB 99.94%  step4.BenchmarkHi
  219.52MB 25.27% 89.70%   219.52MB 25.27%  bytes.makeSlice
      89MB 10.25% 99.94%   308.52MB 35.52%  step4.handleHi
         0     0% 99.94%   219.52MB 25.27%  bytes.(*Buffer).Write
         0     0% 99.94%   219.52MB 25.27%  bytes.(*Buffer).grow
         0     0% 99.94%   219.52MB 25.27%  fmt.Fprintf
         0     0% 99.94%   219.52MB 25.27%  net/http/httptest.
(*ResponseRecorder).Write
         0     0% 99.94%   868.06MB 99.94%  runtime.goexit
         0     0% 99.94%   868.06MB 99.94%  testing.(*B).launch
         0     0% 99.94%   868.06MB 99.94%  testing.(*B).runN
```

```
(pprof) top -cum
868.06MB of 868.56MB total (99.94%)
Dropped 5 nodes (cum <= 4.34MB)
      flat  flat%   sum%        cum   cum%
  559.54MB 64.42% 64.42%   868.06MB 99.94%  step4.BenchmarkHi
         0     0% 64.42%   868.06MB 99.94%  runtime.goexit
         0     0% 64.42%   868.06MB 99.94%  testing.(*B).launch
         0     0% 64.42%   868.06MB 99.94%  testing.(*B).runN
      89MB 10.25% 74.67%   308.52MB 35.52%  step4.handleHi
         0     0% 74.67%   219.52MB 25.27%  bytes.(*Buffer).Write
         0     0% 74.67%   219.52MB 25.27%  bytes.(*Buffer).grow
  219.52MB 25.27% 99.94%   219.52MB 25.27%  bytes.makeSlice
         0     0% 99.94%   219.52MB 25.27%  fmt.Fprintf
         0     0% 99.94%   219.52MB 25.27%  net/http/httptest.
(*ResponseRecorder).Write
(pprof) list handleHi
Total: 868.56MB
ROUTINE ============ step4.handleHi in step4/demo.go
      89MB   308.52MB (flat, cum) 35.52% of Total
         .          .     17:        http.Error(w, "Optional color is
invalid", http.StatusBadRequest)
         .          .     18:        return
         .          .     19:    }
         .          .     20:
         .          .     21:    visitNum := atomic.AddInt64(&visitors,
1)
      89MB   308.52MB     22:    fmt.Fprintf(w, "<html><h1
stype='color: \"%s\"'>Welcome!</h1>You are visitor number %d!",
r.FormValue("color"), visitNum)
         .          .     23:}
         .          .     24:
         .          .     25:func main() {
         .          .     26:    log.Printf("Starting on port 8080")
         .          .     27:    http.HandleFunc("/hi", handleHi)
(pprof)
```

可以看出内存占用大幅减少。

九、Benchcmp

golang.org/x/tools中有一个工具：benchcmp，可以给出两次bench的结果对比。

github.com/golang/tools是golang.org/x/tools的一个镜像。安装benchcmp步骤：

```
1、go get -u github.com/golang/tools
2、mkdir -p $GOPATH/src/golang.org/x
3、mv $GOPATH/src/github.com/golang/tools $GOPATH/src/golang.org/x
```

```
4、go install golang.org/x/tools/cmd/benchcmp
```

我们分别在step2、step3和step4下执行如下命令：

```
$ go-debug-profile-optimization/step2$ go test -bench=. -
memprofile=prof.mem | tee mem.2
PASS
BenchmarkHi-4      100000         14786 ns/op        4961 B/op
81 allocs/op
ok      _/home1/tonybai/proj/opensource/github/experiments/go-debug-
profile-optimization/step2    1.644s

go-debug-profile-optimization/step3$ go test -bench=. -
memprofile=prof.mem | tee mem.3
PASS
BenchmarkHi-4     1000000          1662 ns/op         720 B/op
       9 allocs/op
ok      _/home1/tonybai/proj/opensource/github/experiments/go-debug-
profile-optimization/step3    1.694s

go-debug-profile-optimization/step4$ go test -bench=. -
memprofile=prof.mem | tee mem.4
PASS
BenchmarkHi-4     1000000          1428 ns/op         304 B/op
       6 allocs/op
ok      _/home1/tonybai/proj/opensource/github/experiments/go-debug-
profile-optimization/step4    1.456s
```

利用benchcmp工具对比结果（benchcmp old new）：

```
$ benchcmp step3/mem.3 step4/mem.4
benchmark         old ns/op     new ns/op     delta
BenchmarkHi-4     1662          1428          -14.08%

benchmark         old allocs     new allocs     delta
BenchmarkHi-4     9              6              -33.33%

benchmark         old bytes     new bytes     delta
BenchmarkHi-4     720           304           -57.78%

$ benchcmp step2/mem.2 step4/mem.4
benchmark         old ns/op     new ns/op     delta
BenchmarkHi-4     14786         1428          -90.34%

benchmark         old allocs     new allocs     delta
BenchmarkHi-4     81             6              -92.59%

benchmark         old bytes     new bytes     delta
BenchmarkHi-4     4961          304           -93.87%
```

可以看出优化后，内存分配大幅减少，gc的时间也随之减少。

十、内存来自哪

我们在BenchmarkHi中清理每次handleHi执行后的内存：

```
//step5/demo_test.go
… …
func BenchmarkHi(b *testing.B) {
    b.ReportAllocs()

    req, err := http.ReadRequest(bufio.NewReader(strings.NewReader("GET
/ HTTP/1.0\r\n\r\n")))
    if err != nil {
        b.Fatal(err)
    }

    for i := 0; i < b.N; i++ {
        rw := httptest.NewRecorder()
        handleHi(rw, req)
        reset(rw)
    }
}

func reset(rw *httptest.ResponseRecorder) {
    m := rw.HeaderMap
    for k := range m {
        delete(m, k)
    }
    body := rw.Body
    body.Reset()
    *rw = httptest.ResponseRecorder{
        Body:      body,
        HeaderMap: m,
    }
}

… …
$ go test -v -run=^$ -bench=^BenchmarkHi$ -benchtime=2s -
memprofile=prof.mem
PASS
BenchmarkHi-4     2000000          1518 ns/op         304 B/op
       6 allocs/op
ok      _/home1/tonybai/proj/opensource/github/experiments/go-debug-
profile-optimization/step5    4.577s

$ go tool pprof -alloc_space step5.test prof.mem
Entering interactive mode (type "help" for commands)
(pprof) top -cum 10
```

```
290.52MB of 291.52MB total (99.66%)
Dropped 14 nodes (cum <= 1.46MB)
      flat  flat%   sum%        cum   cum%
         0     0%     0%   291.02MB 99.83%  runtime.goexit
  179.01MB 61.41% 61.41%   290.52MB 99.66%  step5.BenchmarkHi
         0     0% 61.41%   290.52MB 99.66%  testing.(*B).launch
         0     0% 61.41%   290.52MB 99.66%  testing.(*B).runN
   26.50MB  9.09% 70.50%   111.51MB 38.25%  step5.handleHi
         0     0% 70.50%    85.01MB 29.16%  bytes.(*Buffer).Write
         0     0% 70.50%    85.01MB 29.16%  bytes.(*Buffer).grow
   85.01MB 29.16% 99.66%    85.01MB 29.16%  bytes.makeSlice
         0     0% 99.66%    85.01MB 29.16%  fmt.Fprintf
         0     0% 99.66%    85.01MB 29.16%  net/http/httptest.
(*ResponseRecorder).Write
(pprof) list handleHi
Total: 291.52MB
ROUTINE ========================
_/home1/tonybai/proj/opensource/github/experiments/go-debug-profile-
optimization/step5.handleHi in
/home1/tonybai/proj/opensource/github/experiments/go-debug-profile-
optimization/step5/demo.go
   26.50MB   111.51MB (flat, cum) 38.25% of Total
         .          .     17:        http.Error(w, "Optional color is
invalid", http.StatusBadRequest)
         .          .     18:        return
         .          .     19:    }
         .          .     20:
         .          .     21:    visitNum := atomic.AddInt64(&visitors,
1)
   26.50MB   111.51MB     22:    fmt.Fprintf(w, "<html><h1
stype='color: \"%s\"'>Welcome!</h1>You are visitor number %d!",
r.FormValue("color"), visitNum)
         .          .     23:}
         .          .     24:
         .          .     25:func main() {
         .          .     26:    log.Printf("Starting on port 8080")
         .          .     27:    http.HandleFunc("/hi", handleHi)
(pprof)
```

内存从300MB降到111MB。内存来自哪？看到list handleHi，fmt.Fprintf分配了111.51MB。

我们来看这一行代码：

```
fmt.Fprintf(w, "<h1 style='color: %s'>Welcome!</h1>You are visitor
number %d!",
               r.FormValue("color"), num)
```

fmt.Fprintf的manual：

```
$ go doc fmt.Fprintf
func Fprintf(w io.Writer, format string, a …interface{}) (n int, err
error)

    Fprintf formats according to a format specifier and writes to w. It
returns
    the number of bytes written and any write error encountered.
```

这里回顾一下Go type在runtime中的内存占用：

```
A Go interface is 2 words of memory: (type, pointer).
A Go string is 2 words of memory: (base pointer, length)
A Go slice is 3 words of memory: (base pointer, length, capacity)
```

每次调用fmt.Fprintf，参数以value值形式传入函数时，程序就要为每个变参分配一个占用16bytes的empty interface，然后用传入的类型初始化该interface value。这就是这块累计分配内存较多的原因。

十一、消除所有内存分配

下面的优化代码可能在实际中并不需要，但一旦真的成为瓶颈，可以这么做：

```
//go-debug-profile-optimization/step6/demo.go
……
var bufPool = sync.Pool{
    New: func() interface{} {
        return new(bytes.Buffer)
    },
}

func handleHi(w http.ResponseWriter, r *http.Request) {
    if !rxOptionalID.MatchString(r.FormValue("color")) {
        http.Error(w, "Optional color is invalid",
http.StatusBadRequest)
        return
    }

    visitNum := atomic.AddInt64(&visitors, 1)
    buf := bufPool.Get().(*bytes.Buffer)
    defer bufPool.Put(buf)
    buf.Reset()
    buf.WriteString("<h1 style='color: ")
    buf.WriteString(r.FormValue("color"))
    buf.WriteString("'>Welcome!</h1>You are visitor number ")
    b := strconv.AppendInt(buf.Bytes(), int64(visitNum), 10)
    b = append(b, '!')
    w.Write(b)
}
```

```
……

$  go test -v -run=^$ -bench=^BenchmarkHi$ -benchtime=2s -
memprofile=prof.mem
PASS
BenchmarkHi-4     5000000           780 ns/op         192 B/op
       3 allocs/op
ok      _/home1/tonybai/proj/opensource/github/experiments/go-debug-
profile-optimization/step6    4.709s

 go tool pprof -alloc_space step6.test prof.mem
Entering interactive mode (type "help" for commands)
(pprof) top -cum 10
1.07GB of 1.07GB total (  100%)
Dropped 5 nodes (cum <= 0.01GB)
      flat  flat%   sum%        cum   cum%
    1.07GB   100%   100%     1.07GB   100%  step6.BenchmarkHi
         0     0%   100%     1.07GB   100%  runtime.goexit
         0     0%   100%     1.07GB   100%  testing.(*B).launch
         0     0%   100%     1.07GB   100%  testing.(*B).runN

$ go test -bench=. -memprofile=prof.mem | tee mem.6
PASS
BenchmarkHi-4     2000000           790 ns/op         192 B/op
       3 allocs/op
ok      _/home1/tonybai/proj/opensource/github/experiments/go-debug-
profile-optimization/step6    2.401s

$ benchcmp step5/mem.5 step6/mem.6
benchmark         old ns/op     new ns/op     delta
BenchmarkHi-4     1513          790           -47.79%

benchmark         old allocs     new allocs     delta
BenchmarkHi-4     6               3              -50.00%

benchmark         old bytes     new bytes     delta
BenchmarkHi-4     304           192           -36.84%
```

可以看到handleHi已经不在top列表中了。benchcmp结果也显示内存分配又有大幅下降！

十二、竞争(Contention)优化

为handleHi编写一个Parallel benchmark test:

```
//go-debug-profile-optimization/step7/demo_test.go
… …
func BenchmarkHiParallel(b *testing.B) {
    r, err := http.ReadRequest(bufio.NewReader(strings.NewReader("GET /
HTTP/1.0\r\n\r\n")))
```

```
    if err != nil {
        b.Fatal(err)
    }

    b.RunParallel(func(pb *testing.PB) {
        rw := httptest.NewRecorder()
        for pb.Next() {
            handleHi(rw, r)
            reset(rw)
        }
    })
}
… …
```

执行测试，并分析结果:

```
$ go test -bench=Parallel -blockprofile=prof.block
PASS
BenchmarkHiParallel-4     5000000           305 ns/op
ok      _/home1/tonybai/proj/opensource/github/experiments/go-debug-
profile-optimization/step7    1.947s

$ go tool pprof step7.test  prof.block
Entering interactive mode (type "help" for commands)
(pprof) top -cum 10
3.68s of 3.72s total (98.82%)
Dropped 29 nodes (cum <= 0.02s)
Showing top 10 nodes out of 20 (cum >= 1.84s)
      flat  flat%   sum%        cum   cum%
         0     0%     0%      3.72s   100%  runtime.goexit
     1.84s 49.46% 49.46%      1.84s 49.46%  runtime.chanrecv1
         0     0% 49.46%      1.84s 49.46%  main.main
         0     0% 49.46%      1.84s 49.46%  runtime.main
         0     0% 49.46%      1.84s 49.46%  testing.(*M).Run
         0     0% 49.46%      1.84s 49.43%  testing.(*B).run
         0     0% 49.46%      1.84s 49.43%  testing.RunBenchmarks
         0     0% 49.46%      1.84s 49.36%  step7.BenchmarkHiParallel
     1.84s 49.36% 98.82%      1.84s 49.36%  sync.(*WaitGroup).Wait
         0     0% 98.82%      1.84s 49.36%  testing.(*B).RunParallel
(pprof) list BenchmarkHiParallel
Total: 3.72s
ROUTINE ====== step7.BenchmarkHiParallel in step7/demo_test.go
         0      1.84s (flat, cum) 49.36% of Total
         .          .    113:        rw := httptest.NewRecorder()
         .          .    114:        for pb.Next() {
         .          .    115:            handleHi(rw, r)
         .          .    116:            reset(rw)
         .          .    117:        }
```

```
         .      1.84s    118:    })
         .          .    119:}
ROUTINE ==== step7.BenchmarkHiParallel.func1 in step7/demo_test.go
         0    43.02ms (flat, cum)  1.16% of Total
         .          .    110:    }
         .          .    111:
         .          .    112:    b.RunParallel(func(pb *testing.PB) {
         .          .    113:        rw := httptest.NewRecorder()
         .          .    114:        for pb.Next() {
         .    43.02ms    115:            handleHi(rw, r)
         .          .    116:            reset(rw)
         .          .    117:        }
         .          .    118:    })
         .          .    119:}
(pprof) list handleHi
Total: 3.72s
ROUTINE =====step7.handleHi in step7/demo.go
         0    43.02ms (flat, cum)  1.16% of Total
         .          .     18:        return new(bytes.Buffer)
         .          .     19:    },
         .          .     20:}
         .          .     21:
         .          .     22:func handleHi(w http.ResponseWriter, r
*http.Request) {
         .    43.01ms     23:    if
!rxOptionalID.MatchString(r.FormValue("color")) {
         .          .     24:        http.Error(w, "Optional color is
invalid", http.StatusBadRequest)
         .          .     25:        return
         .          .     26:    }
         .          .     27:
         .          .     28:    visitNum := atomic.AddInt64(&visitors,
1)
         .     2.50us     29:    buf := bufPool.Get().(*bytes.Buffer)
         .          .     30:    defer bufPool.Put(buf)
         .          .     31:    buf.Reset()
         .          .     32:    buf.WriteString("<h1 style='color: ")
         .          .     33:    buf.WriteString(r.FormValue("color"))
         .          .     34:    buf.WriteString("'>Welcome!</h1>You
are visitor number ")
(pprof)
```

handleHi中MatchString这块是一个焦点，这里耗时较多。

优化方法（**step8**）：

```
//go-debug-profile-optimization/step8/demo.go
… …
```

```
var colorRxPool = sync.Pool{
    New: func() interface{} { return regexp.MustCompile(`\w*$`) },
}

func handleHi(w http.ResponseWriter, r *http.Request) {
    if !colorRxPool.Get().
(*regexp.Regexp).MatchString(r.FormValue("color")) {
        http.Error(w, "Optional color is invalid",
http.StatusBadRequest)
        return
    }

    visitNum := atomic.AddInt64(&visitors, 1)
    buf := bufPool.Get().(*bytes.Buffer)
    defer bufPool.Put(buf)
    buf.Reset()
    buf.WriteString("<h1 style='color: ")
    buf.WriteString(r.FormValue("color"))
    buf.WriteString("'>Welcome!</h1>You are visitor number ")
    b := strconv.AppendInt(buf.Bytes(), int64(visitNum), 10)
    b = append(b, '!')
    w.Write(b)
}
… …
```

测试执行与分析：

```
$ go test -bench=Parallel -blockprofile=prof.block
PASS
BenchmarkHiParallel-4      100000         19190 ns/op
ok      _/home1/tonybai/proj/opensource/github/experiments/go-debug-
profile-optimization/step8    2.219s

$ go tool pprof step8.test  prof.block
Entering interactive mode (type "help" for commands)
(pprof) top -cum 10
4.22s of 4.23s total (99.69%)
Dropped 28 nodes (cum <= 0.02s)
Showing top 10 nodes out of 12 (cum >= 2.11s)
      flat  flat%   sum%        cum   cum%
         0     0%     0%      4.23s   100%  runtime.goexit
     2.11s 49.90% 49.90%      2.11s 49.90%  runtime.chanrecv1
         0     0% 49.90%      2.11s 49.89%  main.main
         0     0% 49.90%      2.11s 49.89%  runtime.main
         0     0% 49.90%      2.11s 49.89%  testing.(*M).Run
         0     0% 49.90%      2.11s 49.86%  testing.(*B).run
         0     0% 49.90%      2.11s 49.86%  testing.RunBenchmarks
         0     0% 49.90%      2.11s 49.79%  step8.BenchmarkHiParallel
```

```
     2.11s 49.79% 99.69%      2.11s 49.79%  sync.(*WaitGroup).Wait
         0     0% 99.69%      2.11s 49.79%  testing.(*B).RunParallel
(pprof) list BenchmarkHiParallel
Total: 4.23s
ROUTINE ======step8.BenchmarkHiParallel in step8/demo_test.go
         0      2.11s (flat, cum) 49.79% of Total
         .          .    113:        rw := httptest.NewRecorder()
         .          .    114:        for pb.Next() {
         .          .    115:            handleHi(rw, r)
         .          .    116:            reset(rw)
         .          .    117:        }
         .      2.11s    118:    })
         .          .    119:}
ROUTINE ======step8.BenchmarkHiParallel.func1 in step8/demo_test.go
         0    11.68ms (flat, cum)  0.28% of Total
         .          .    110:    }
         .          .    111:
         .          .    112:    b.RunParallel(func(pb *testing.PB) {
         .          .    113:        rw := httptest.NewRecorder()
         .          .    114:        for pb.Next() {
         .    11.68ms    115:            handleHi(rw, r)
         .          .    116:            reset(rw)
         .          .    117:        }
         .          .    118:    })
         .          .    119:}
(pprof) list handleHi
Total: 4.23s
ROUTINE ======step8.handleHi in step8/demo.go
         0    11.68ms (flat, cum)  0.28% of Total
         .          .     21:var colorRxPool = sync.Pool{
         .          .     22:    New: func() interface{} { return
regexp.MustCompile(`\w*$`) },
         .          .     23:}
         .          .     24:
         .          .     25:func handleHi(w http.ResponseWriter, r
*http.Request) {
         .     5.66ms     26:    if !colorRxPool.Get().
(*regexp.Regexp).MatchString(r.FormValue("color")) {
         .          .     27:        http.Error(w, "Optional color is
invalid", http.StatusBadRequest)
         .          .     28:        return
         .          .     29:    }
         .          .     30:
         .          .     31:    visitNum := atomic.AddInt64(&visitors,
1)
         .     6.02ms     32:    buf := bufPool.Get().(*bytes.Buffer)
         .          .     33:    defer bufPool.Put(buf)
         .          .     34:    buf.Reset()
```

```
         .          .      35:     buf.WriteString("<h1 style='color: ")
         .          .      36:     buf.WriteString(r.FormValue("color"))
         .          .      37:     buf.WriteString("'>Welcome!</h1>You
are visitor number ")
(pprof)
```

优化后，MatchString从43ms降到5.66ms。

Related posts:

1. Goroutine是如何工作的
2. 也谈并发与并行
3. Golang测试技术
4. 近期遇到的3个Golang代码问题
5. 一个有关Golang变量作用域的坑

已有 17 条评论

1. *那抹湛蓝*
   2015/08/27

   在 CPU profiling 中，下面这个命令出错

   go test -v -run=^$ -bench=.

   提示 no matches found: -run=^$

   回复

2. *bigwhite*
   2015/08/27

   -run regexp Run only those tests and examples matching the regular expression.
   ^$是正则式，我在go 1.5 ubuntu amd64下执行没有问题啊。你的go版本和环境是?

   回复

   - *那抹湛蓝*
     2015/08/27

     $ uname -a
     Linux shhl2 3.16.0-30-generic #40~14.04.1-Ubuntu SMP Thu Jan 15 17:43:14 UTC 2015 x86_64 x86_64 x86_64 GNU/Linux
     $ go version
     go version go1.5 linux/amd64

我用的是 zsh

回复

3. bigwhite
2015/08/31

可以将-run去掉试试。

回复

4. 原来微博的昵称真的可以好长好长
2015/09/02

博客的模板比较差啊。。。 代码段看起来非常不舒服。。。 影响文章质量了都~ 不过，内容很精彩实用~ 赞~

回复

- bigwhite
2015/09/05

  有好模板，不妨推荐一个？

  回复

  - 原来微博的昵称真的可以好长好长
2015/09/23

    我也没啥好模板，只是你这个代码较多，格式化展示效果会好很多~

    回复

    - bigwhite
2015/09/23

      嗯，我也在逐渐切换到markdown，希望后续格式会好些。

      回复

5. Hessian海生
2015/10/10

找个代码高亮插件吧

回复

6. 8094731

2016/01/09

在windows 7中，为什么我执行后的没有*.test文件 其他的都好。

回复

- bigwhite
  2016/01/09

  我手头没有windows go开发环境，工作中也不用windows，您自己不妨摸索一下，有结论后，别忘了在这里的评论中补上一句。呵呵。

  回复

7. *Fred*
   2016/12/06

   写的很不错，其中有个问题指出，以免后看者在测试验证时浪费不必要的时间使用pprof工具分析mem：$ go tool pprof –alloc_space step3.test prof.memEntering interactive mode (type “help” for commands)—中划线大写了–alloc_space（错误）-alloc_space

   回复

8. small-pig
   2018/05/06

   //go-debug-profile-optimization/step0/demo.go
   package main

   import (
   “fmt”
   “log”
   “net/http”
   “regexp”
)

   // must be accessed atomically

   var add = make(chan struct{})
   var total = make(chan int)

   func getVistors() int {
   return <-total
   }

   func addVistors() {
   add <- struct{}{}
   }

   func teller() {

```
var visitors int
for {
select {
case <-add:
visitors += 1
case total <- visitors:
}
}
}

func handleHi(w http.ResponseWriter, r *http.Request) {
if match, _ := regexp.MatchString(`^\w*$`, r.FormValue("color")); !match {
http.Error(w, "Optional color is invalid", http.StatusBadRequest)
return
}
addVistors()
visitNum := getVistors()
w.Header().Set("Content-Type", "text/html; charset=utf-8")
w.Write([]byte("Welcome!You are visitor number ” + fmt.Sprint(visitNum) + “!”))
}
func main() {
log.Printf(“Starting on port 8080″)
go teller()
http.HandleFunc(“/hi”, handleHi)
log.Fatal(http.ListenAndServe(“127.0.0.1:8080″, nil))
}
```

请问big white大大,我使用了channel,为什么go test的时候会不成功?
现象是go test timeout。

回复

9. *small-pig*
2018/05/06

不好意思
var c = make(chan struct{}, 1)
func handleVistors() {
c <- struct{}{}
vistors++
<-c
}

我这样写,测试依旧不通过，依旧是timeout。不知道为什么。

回复

-  *bigwhite*
2018/05/07

无论是否是buffered channel，go test执行到TestHandleHi_Recorder时，只是直接调用了handleHi，而你在demo.go中写的teller() goroutine并没有启动啊。因此测试始终阻塞在那里了啊。应该是visitors这个channel上。

回复

10. *small-pig*
2018/05/06

你好，
不太理解step 8 为什么会优化之前的做法[让正则仅编译一次]。
我浅显的理解,sync.Pool不是会不定时被gc掉么,如果被gc了的话,又要重新编译正则了。

回复

- *bigwhite*
2018/05/07

regexp.Regexp内部是有一个mutex的，因此是goroutine-safe的，但mutex在并发时存在一定的性能损耗。step8目的就是建立一个regexp.Regexp pool，这样多个goroutine按需获取，这样不用每收到一个连接就创建一个Regexp，sync.Pool中的对象是暂时性的，在两个GC周期之间未被get出去的object会被gc掉，但pool的确增加对象重用的几率，减少gc的负担，因此一定程度上减少内存分配，提升整体性能。

回复

添加新评论

发表评论前，请滑动滚动条解锁

称呼

邮箱

网站 http://example.com

提交评论

输入关键字搜索 搜索

欢迎使用邮件订阅我的博客

输入邮箱订阅本站，只要有新文章发布，就会第一时间发送邮件通知你哦！

名字： 请输入您的名字

邮箱: 请输入您的电子邮箱

马上订阅

我的业余项目

- smspush短信发送平台

这里是Tony Bai的个人Blog，欢迎访问、订阅和留言！订阅Feed请点击上面图片。

如果您觉得这里的文章对您有帮助，请扫描上方二维码进行捐赠，加油后的Tony Bai将会为您呈现更多精彩的文章，谢谢！

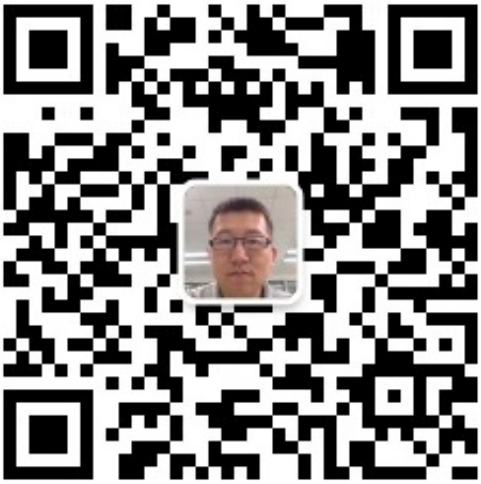
如果您希望通过微信捐赠，请用微信客户端扫描下方赞赏码:

如果您希望通过比特币或以太币捐赠，可以扫描下方二维码：

比特币：

以太币：

如果您喜欢通过微信App浏览本站内容，可以扫描下方二维码，订阅本站官方微信订阅号“iamtonybai”；点击二维码，可直达本人官方微博主页^_^：



本站Powered by Digital Ocean VPS。

选择Digital Ocean VPS主机，即可获得10美元现金充值，可免费使用两个月哟！

著名主机提供商Linode 10$优惠码：linode10，在这里注册即可免费获得。

阿里云推荐码：1WFZ0V，立享9折！

bigwhite.cn@Gmail.com

my LinkedIn profile

文章

- 慕课网免费课“Kubernetes：开启云原生之门”上线
- 写Go代码时遇到的那些问题[第3期]
- defer函数参数求值简要分析
- 对一段Go语言代码输出结果的简要分析
- TB一周萃选[第10期]
- Go 1.10中值得关注的几个变化
- TB一周萃选[第9期]
- TB一周萃选[第8期]
- TB一周萃选[第7期]
- 写Go代码时遇到的那些问题[第2期]

评论

- bigwhite 在 Hello，Termux
  那个防止垃圾评论的plugin的确体验较差，不过我的wordpress版本较低，还懒得升级，好的防垃...
- Hugh 在 Hello，Termux
  如果是想获得管理员权限的话可以用tsu命令替换su命令,原来的命令都还能执行. pkg instal...
- bob 在 ngrok原理浅析
  受益匪浅，已订阅博文免费ngrok服务器铂金ngrok https://ngrok.bob.kim
- bigwhite 在 在Kubernetes集群上部署高可用Harbor镜像仓库
  我的邮箱，bigwhite.cn@aliyun.com，欢迎沟通。您要做的这个平台也不算小，兄台背后...
- bigwhite 在 在Kubernetes集群上部署高可用Harbor镜像仓库
  大大的赞。codefresh.io这个很不错。国内这方面的服务似乎多是绑定某个容器云平台了。没有独立...
- 今何安 在 在Kubernetes集群上部署高可用Harbor镜像仓库
  架构上目前还没有做HA，这个问题不大，目前就只有数据库mysql会存在单点问题，这个后续会切换到直接...
- 今何安 在 在Kubernetes集群上部署高可用Harbor镜像仓库
  经过年后这段时间的准备，我开发了一个精简版的docker镜像仓库产品：https://douwa.t...
- bigwhite 在 部署devstack
  以前没遇到过，现在也没有devstack环境了。不过 google了一下，找到了两个和你遇到相似问题...
- 洪城浪子 在 部署devstack
  请问如果出现g-api该如何解决+functions:wait_for_service:432 ...
- bigwhite 在 Go程序调试、分析与优化
  regexp.Regexp内部是有一个mutex的，因此是goroutine-safe的，但mute...
- 下一页 »

分类

- 光影汇 (7)
- 影音坊 (36)
- 思考控 (66)
- 技术志 (555)
- 教育记 (1)
- 杂货铺 (75)
- 生活簿 (154)
- 职场录 (14)
- 读书吧 (14)
- 运动迷 (107)
- 驴友秀 (40)

标签

Blog Blogger C Cpp docker English GCC github GNU Go Golang Google Java k8s Kernel Kubernetes Linux M10 Opensource Programmer Python Solaris Subversion Ubuntu Unix Windows 世界杯 博客 学习 容器 工作 巴萨 开源 思考 感悟 摄影 旅游 梅西 球王 生活 程序员 编译器 西甲 足球 驴友

归档

- 2018 年五月 (1)
- 2018 年四月 (1)
- 2018 年三月 (3)
- 2018 年二月 (3)
- 2018 年一月 (7)
- 2017 年十二月 (5)
- 2017 年十一月 (4)
- 2017 年十月 (3)
- 2017 年九月 (2)
- 2017 年八月 (3)
- 2017 年七月 (4)
- 2017 年六月 (8)
- 2017 年五月 (5)
- 2017 年四月 (3)
- 2017 年三月 (2)
- 2017 年二月 (5)
- 2017 年一月 (7)

- 2016 年十二月 (7)
- 2016 年十一月 (7)
- 2016 年十月 (3)
- 2016 年九月 (2)
- 2016 年八月 (1)
- 2016 年六月 (2)
- 2016 年五月 (2)
- 2016 年四月 (2)
- 2016 年三月 (2)
- 2016 年二月 (3)
- 2016 年一月 (2)
- 2015 年十二月 (1)
- 2015 年十一月 (1)
- 2015 年十月 (1)
- 2015 年九月 (3)
- 2015 年八月 (5)
- 2015 年七月 (6)
- 2015 年六月 (4)
- 2015 年五月 (1)
- 2015 年四月 (2)
- 2015 年三月 (2)
- 2015 年一月 (2)
- 2014 年十二月 (5)
- 2014 年十一月 (8)
- 2014 年十月 (9)
- 2014 年九月 (2)
- 2014 年八月 (1)
- 2014 年七月 (1)
- 2014 年五月 (2)
- 2014 年四月 (5)
- 2014 年三月 (4)
- 2014 年二月 (1)
- 2014 年一月 (1)
- 2013 年十二月 (3)
- 2013 年十一月 (5)
- 2013 年十月 (6)
- 2013 年九月 (4)
- 2013 年八月 (5)
- 2013 年七月 (6)
- 2013 年六月 (2)
- 2013 年五月 (6)

- 2013 年四月 (3)
- 2013 年三月 (7)
- 2013 年二月 (4)
- 2013 年一月 (6)
- 2012 年十二月 (8)
- 2012 年十一月 (10)
- 2012 年十月 (5)
- 2012 年九月 (3)
- 2012 年八月 (10)
- 2012 年七月 (4)
- 2012 年六月 (2)
- 2012 年五月 (4)
- 2012 年四月 (10)
- 2012 年三月 (8)
- 2012 年二月 (6)
- 2012 年一月 (6)
- 2011 年十二月 (4)
- 2011 年十一月 (4)
- 2011 年十月 (5)
- 2011 年九月 (8)
- 2011 年八月 (7)
- 2011 年七月 (6)
- 2011 年六月 (7)
- 2011 年五月 (8)
- 2011 年四月 (6)
- 2011 年三月 (10)
- 2011 年二月 (7)
- 2011 年一月 (10)
- 2010 年十二月 (7)
- 2010 年十一月 (6)
- 2010 年十月 (7)
- 2010 年九月 (12)
- 2010 年八月 (8)
- 2010 年七月 (3)
- 2010 年六月 (5)
- 2010 年五月 (4)
- 2010 年四月 (2)
- 2010 年三月 (6)
- 2010 年二月 (4)
- 2010 年一月 (6)
- 2009 年十二月 (6)

- 2009 年十一月 (6)
- 2009 年十月 (5)
- 2009 年九月 (8)
- 2009 年八月 (8)
- 2009 年七月 (8)
- 2009 年六月 (2)
- 2009 年五月 (5)
- 2009 年四月 (7)
- 2009 年三月 (12)
- 2009 年二月 (9)
- 2009 年一月 (15)
- 2008 年十二月 (9)
- 2008 年十一月 (5)
- 2008 年十月 (10)
- 2008 年九月 (13)
- 2008 年八月 (13)
- 2008 年七月 (3)
- 2008 年六月 (1)
- 2008 年五月 (7)
- 2008 年四月 (4)
- 2008 年三月 (9)
- 2008 年二月 (11)
- 2008 年一月 (15)
- 2007 年十二月 (11)
- 2007 年十一月 (14)
- 2007 年十月 (4)
- 2007 年九月 (5)
- 2007 年八月 (1)
- 2007 年七月 (10)
- 2007 年六月 (10)
- 2007 年五月 (10)
- 2007 年四月 (8)
- 2007 年三月 (15)
- 2007 年二月 (4)
- 2007 年一月 (17)
- 2006 年十二月 (18)
- 2006 年十一月 (9)
- 2006 年十月 (11)
- 2006 年九月 (6)
- 2006 年八月 (5)
- 2006 年七月 (22)

- 2006 年六月 (35)
- 2006 年五月 (24)
- 2006 年四月 (26)
- 2006 年三月 (25)
- 2006 年二月 (18)
- 2006 年一月 (15)
- 2005 年十二月 (10)
- 2005 年十一月 (10)
- 2005 年九月 (13)
- 2005 年八月 (11)
- 2005 年七月 (6)
- 2005 年六月 (2)
- 2005 年五月 (3)
- 2005 年四月 (6)
- 2005 年三月 (1)
- 2005 年一月 (15)
- 2004 年十二月 (9)
- 2004 年十一月 (14)
- 2004 年十月 (2)
- 2004 年九月 (2)

私人

- 我的女儿

链接

- @douban
- @flickr
- @github
- @googlecode
- @picasa
- @slideshare
- @twitter
- @weibo
- Hoterran
- Lionel Messi
- Puras He
- 梦想风暴
- 过眼云烟

开源项目

- buildc

- cbehave
- lcut

翻译项目

- C语言编码风格和标准
- 《Programming in Haskell》中文翻译项目