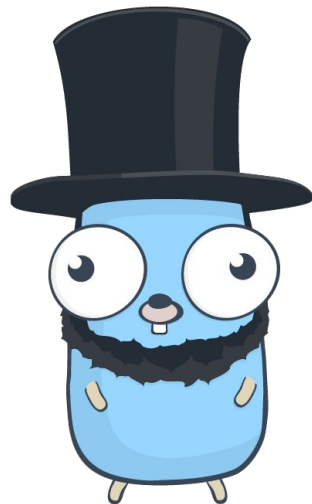


Profile all the things!

Using pprof in Go

John Potocny
Capital Go 2017



About Me

- [@johnpotocny1](#)
- Backend Developer @ VividCortex
- We're hiring!



VividCortex

So, let's talk about pprof

- Been around since before Go1 was released
- Initial support for CPU and heap profiling
- Full list of default profiles as of Go1.8:
 - CPU
 - Goroutine
 - Heap
 - ThreadCreate
 - Block
 - Mutex (New in Go1.8!)
- Godoc here: <https://golang.org/pkg/runtime/pprof/>

How do I use pprof?

- <https://play.golang.org/p/FvEMZw0GhD>
- Alternative: import “net/http/pprof”
 - Demo1

← → ↻ ⓘ localhost:8080/debug/pprof/

/debug/pprof/

profiles:

0 [block](#)

6 [goroutine](#)

0 [heap](#)

0 [mutex](#)

6 [threadcreate](#)

[full goroutine stack dump](#)

```
1 package main
2
3 import (
4     "fmt"
5     "io"
6     "log"
7     "net/http"
8     _ "net/http/pprof"
9 )
10
11 func main() {
12     http.HandleFunc("/", handler)
13     http.ListenAndServe("localhost:8080", nil)
14 }
```

How do I use pprof? (cont.)

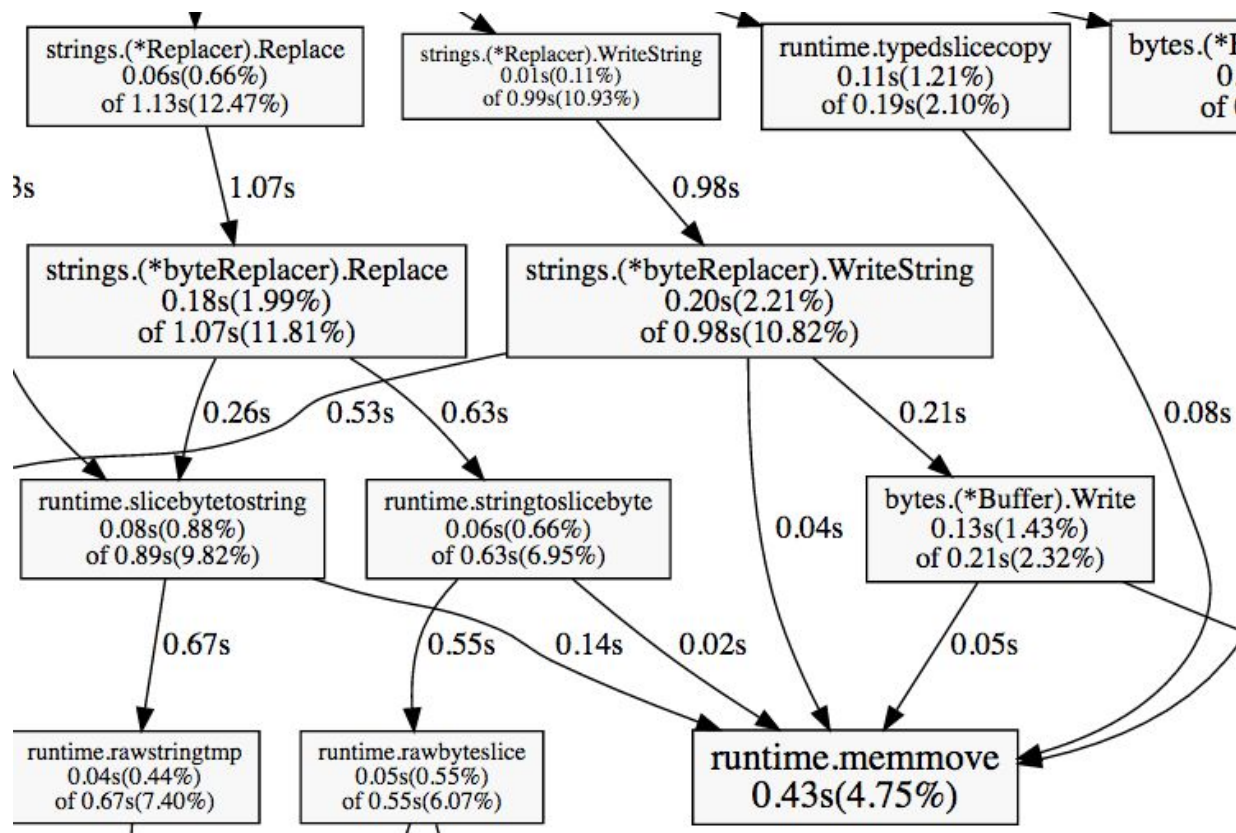
- Use 'go tool pprof' to fetch profiles from a program with a pprof server:
- Can also use write profiles when running 'go test'
 - ex: `go test -cpuprofile=cpu.out; go tool pprof presentation.test cpu.out`

```
~:502 $ go tool pprof stage-api0:8085/debug/pprof/profile
Fetching profile from http://stage-api0:8085/debug/pprof/profile
Please wait... (30s)
Saved profile in /Users/johnpotocny/pprof/pprof.api-metrics.stage-api0:8085.samples.cpu.001.pb.gz
Entering interactive mode (type "help" for commands)
(pprof) top
570ms of 1520ms total (37.50%)
Showing top 10 nodes out of 174 (cum >= 60ms)
      flat  flat%   sum%        cum   cum%   encoding/json.(*decodeState).scanWhile
      80ms   5.26%   5.26%       140ms   9.21%   encoding/json.(*decodeState).object
      70ms   4.61%   9.87%       920ms  60.53%   runtime.scanobject
      70ms   4.61%  14.47%       140ms   9.21%   reflect.(*rtype).Name
      60ms   3.95%  18.42%       120ms   7.89%   runtime.mapaccess1
      60ms   3.95%  22.37%        90ms   5.92%
```

Add the binary for more visibility!

```
presentation:504 $ go tool pprof --seconds=120 presentation localhost:8080/debug/pprof/profile
Fetching profile from http://localhost:8080/debug/pprof/profile?seconds=120
Please wait... (2m0s)
Saved profile in /Users/johnpotocny/pprof/pprof.presentation.localhost:8080.samples.cpu.004.pb.gz
Entering interactive mode (type "help" for commands)
(pprof) list handler
Total: 6.26s
ROUTINE ===== main.handler in /Users/johnpotocny/vividcortex/go/src/github.com/
      0      10ms (flat, cum) 0.16% of Total
      .      .      31:  http.ListenAndServe("localhost:8080", nil)
      .      .      32:}
      .      .      33:
      .      .      34:func handler(w http.ResponseWriter, r *http.Request) {
      .      .      35:  latest := incrementCounter()
      .      10ms  36:  _, err := io.WriteString(w, fmt.Sprint(latest))
      .      .      37:  if err != nil {
      .      .      38:      log.Println("error writing response:", err)
      .      .      39:  }
      .      .      40:}
```

Types of Profiles: CPU



Types of Profiles: Heap

- 4 different modes to view
 - `--inuse_space` (default): shows live heap memory in bytes
 - `--inuse_objects`: shows # of live heap objects
 - `--alloc_objects`: shows # of allocated objects
 - `--alloc_space`: shows allocated bytes over time

```
presentation:507 $ go tool pprof --alloc_objects presentation localhost:8080/debug/pprof/heap
Fetching profile from http://localhost:8080/debug/pprof/heap
Saved profile in /Users/johnpotocny/pprof/pprof.presentation.localhost:8080.alloc_objects.allc
Entering interactive mode (type "help" for commands)
(pprof) top
3623085 of 4861308 total (74.53%)
Dropped 10 nodes (cum <= 24306)
Showing top 10 nodes out of 53 (cum >= 190068)
      flat flat% sum%      cum cum%      context.WithCancel
    742786 15.28% 15.28%    1045005 21.50% runtime.rawstringtmp
    655368 13.48% 28.76%     655368 13.48% runtime.makemap
    349540  7.19% 35.95%     349540  7.19% net.sockaddrToTCP
    327694  6.74% 42.69%     327694  6.74% syscall.anyToSockaddr
    327689  6.74% 49.43%     327689  6.74% net/http.(*conn).readRequest
    290276  5.97% 55.40%     2475559 50.92%
```


Types of Profiles: Goroutine

- List goroutines aggregated by stack
- Use debug=2 to have all routines separated, with current state included.

```
localhost:8080/debug/pprof/goroutine?debug=1

goroutine profile: total 5
1 @ 0x102df2a 0x1028e67 0x10284a9 0x115b5f8 0x115b664 0x115ce27 0x116f120 0x1252988 0x1059af1
# 0x10284a8 net.runtime_pollWait+0x58 /Users/johnpotocny/gol.8/src/runtime/netpoll.go:164
# 0x115b5f7 net.(*pollDesc).wait+0x37 /Users/johnpotocny/gol.8/src/net/fd_poll_runtime.go:75
# 0x115b663 net.(*pollDesc).waitRead+0x33 /Users/johnpotocny/gol.8/src/net/fd_poll_runtime.go:80
# 0x115ce26 net.(*netFD).Read+0x1b6 /Users/johnpotocny/gol.8/src/net/fd_unix.go:250
# 0x116f11f net.(*conn).Read+0x6f /Users/johnpotocny/gol.8/src/net/net.go:181
# 0x1252987 net/http.(*connReader).backgroundRead+0x57 /Users/johnpotocny/gol.8/src/net/http/server.go:656

1 @ 0x102df2a 0x1028e67 0x10284a9 0x115b5f8 0x115b664 0x115ce27 0x116f120 0x1252e80 0x1108657 0x11095db 0x11097c7 0x11e2c1f 0x11e2a1f
# 0x10284a8 net.runtime_pollWait+0x58 /Users/johnpotocny/gol.8/src/runtime/netpoll.go:164
# 0x115b5f7 net.(*pollDesc).wait+0x37 /Users/johnpotocny/gol.8/src/net/fd_poll_runtime.go:75
# 0x115b663 net.(*pollDesc).waitRead+0x33 /Users/johnpotocny/gol.8/src/net/fd_poll_runtime.go:80
# 0x115ce26 net.(*netFD).Read+0x1b6 /Users/johnpotocny/gol.8/src/net/fd_unix.go:250
# 0x116f11f net.(*conn).Read+0x6f /Users/johnpotocny/gol.8/src/net/net.go:181
# 0x1252e7f net/http.(*connReader).Read+0x13f /Users/johnpotocny/gol.8/src/net/http/server.go:754
# 0x1108656 bufio.(*Reader).fill+0x116 /Users/johnpotocny/gol.8/src/bufio/bufio.go:97
# 0x11095da bufio.(*Reader).ReadSlice+0xba /Users/johnpotocny/gol.8/src/bufio/bufio.go:338
# 0x11097c6 bufio.(*Reader).ReadLine+0x36 /Users/johnpotocny/gol.8/src/bufio/bufio.go:367
# 0x11e2c1e net/textproto.(*Reader).readLineSlice+0x5e /Users/johnpotocny/gol.8/src/net/textproto/reader.go:55
# 0x11e2a1e net/textproto.(*Reader).ReadLine+0x2e /Users/johnpotocny/gol.8/src/net/textproto/reader.go:36
# 0x124ce74 net/http.readRequest+0xa4 /Users/johnpotocny/gol.8/src/net/http/request.go:918
# 0x1254252 net/http.(*conn).readRequest+0x212 /Users/johnpotocny/gol.8/src/net/http/server.go:934
# 0x1258659 net/http.(*conn).serve+0x499 /Users/johnpotocny/gol.8/src/net/http/server.go:1763
```

Types of Profiles: ThreadCreate

- Shows which goroutines lead to creation of threads in an app
 - Broken? See <https://github.com/golang/go/issues/6104>

```
← → ↻ ⓘ localhost:8080/debug/pprof/threadcreate?
```

```
threadcreate profile: total 6
```

```
6 @
```

```
# 0x0
```

```
demo4:505 $ go tool pprof demo4 localhost:8080/debug/pprof/threadcreate
Fetching profile from http://localhost:8080/debug/pprof/threadcreate
Saved profile in /Users/johnpotocny/pprof/pprof.demo4.localhost:8080.thr
Entering interactive mode (type "help" for commands)
(pprof) top
0 of 6 total ( 0%)
      flat flat%   sum%        cum   cum%
(pprof) █
```

Types of Profiles: Block

- Shows time spent blocked on synchronization (mutexes, channels, etc.)
- Not enabled by default call `runtime.SetBlockProfileRate()` to enable
 - Argument is rate in ns to sample blocked routines
 - 1 = sample all events
 - 0 = disable
 - 100 = sample every 100ns

Types of Profiles: Block (cont.)

Demo2

```
demo2:622 $ go tool pprof demo2 localhost:8080/debug/pprof/block
Fetching profile from http://localhost:8080/debug/pprof/block
Saved profile in /Users/johnpotocny/pprof/pprof.demo2.localhost:8080.contentions.delay.008.pb.gz
Entering interactive mode (type "help" for commands)
(pprof) list handler
Total: 837.85ms
ROUTINE ===== main.handler in /Users/johnpotocny/vividcortex/go/src/github.com/emo2.go
   0   366.91ms (flat, cum) 43.79% of Total
     .           .      33:   return latest
     .           .      34:}
     .           .      35:
     .           .      36:func handler(w http.ResponseWriter, r *http.Request) {
     .           .      37:   // Introduce a 1us block.
     .           .      38:   <-time.After(time.Microsecond)
     .   366.76ms      39:   _, err := io.WriteString(w, fmt.Sprintf(incrementCounter()))
     .   150.92us      40:   if err != nil {
     .           .      41:       log.Println("error writing response:", err)
     .           .      42:   }
     .           .      43:}
(pprof) █
```

Types of Profiles: Mutex

- Added in Go1.8
- Block showed us blocked routines; Mutex profile shows us blockers
 - Specifically, which routines spend time holding a mutex
- Disabled by default; call `runtime.SetMutexProfileFraction()` to enable

Type of Profiles: Mutex (cont.)

Demo3

```
demo3:629 $ go tool pprof demo3 localhost:8080/debug/pprof/mutex
Fetching profile from http://localhost:8080/debug/pprof/mutex
Saved profile in /Users/johnpotocny/pprof/pprof.demo3.localhost:8080.contentions.delay.003.pb.gz
Entering interactive mode (type "help" for commands)
(pprof) list handle
Total: 8.49ms
ROUTINE ===== main.handleReadCounter in /Users/johnpotocny/vividcortex/go/src/
on/demo3/demo3.go
      0      8.49ms (flat, cum)   100% of Total
      .      .      42:   globalCounter.Lock()
      .      .      43:   _, err := io.WriteString(w, fmt.Sprintf(globalCounter.count))
      .      .      44:   if err != nil {
      .      .      45:       log.Println("error writing response:", err)
      .      .      46:   }
      .      8.49ms      47:   globalCounter.Unlock()
      .      .      48:}
(pprof) █
```

Types of Profiles: Mutex (cont.)

- What if we use a RWMutex?
 - RLock/RUnlock are not tracked; RWMutex.Lock/Unlock are though
 - Milestone for Go1.9; <https://github.com/golang/go/issues/18496>

```
demo3:634 $ go tool pprof demo3 localhost:8080/debug/pprof/mutex
Fetching profile from http://localhost:8080/debug/pprof/mutex
Saved profile in /Users/johnpotocny/pprof/pprof.demo3.localhost:8080.
Entering interactive mode (type "help" for commands)
(pprof) list handle
profile is empty
(pprof) top
profile is empty
(pprof) █
```


Types of Profiles: Custom!

- runtime/pprof exposes an API for creating our own profiles!

```
15
16 const profileName = "activeRequests"
17
18 var myProfile *pprof.Profile
19
20 func init() {
21     // Check that the profile doesn't exist;
22     // if we try to create it twice we get a panic!
23     if myProfile = pprof.Lookup(profileName); myProfile == nil {
24         myProfile = pprof.NewProfile(profileName)
25     }
26 }
```

← → ↻ ⓘ localhost:8080/debug/pprof/

/debug/pprof/

profiles:

0 [activeRequests](#)

0 [block](#)

5 [goroutine](#)

0 [heap](#)

0 [mutex](#)

6 [threadcreate](#)

[full goroutine stack dump](#)

Custom Profiles (cont.)

```
34 // trackRequest wraps an http handler with tracing information
35 // so we can profile in-flight requests.
36 func trackRequest() func() {
37     // Allocate a byte and use its memory address as a key for the profile.
38     key := new(byte)
39
40     // Add the key to track that we have another request in progress.
41     // Pass skip=1 to omit the myProfile.Add call in the stack recorded;
42     // higher values will remove more calls and start higher up the stack.
43     myProfile.Add(key, 1)
44
45     // Introduce artificial delay in the request we're tracking,
46     // since they don't do much work.
47     time.Sleep(time.Millisecond * 500)
48
49     return func() {
50         // Remove the counter for the now completed request.
51         myProfile.Remove(key)
52     }
53 }

```



```
69 func handleIncCounter(w http.ResponseWriter, r *http.Request) {
70     defer trackRequest()()
71     incrementCounter()
72 }

```

Custom Profiles (cont.)

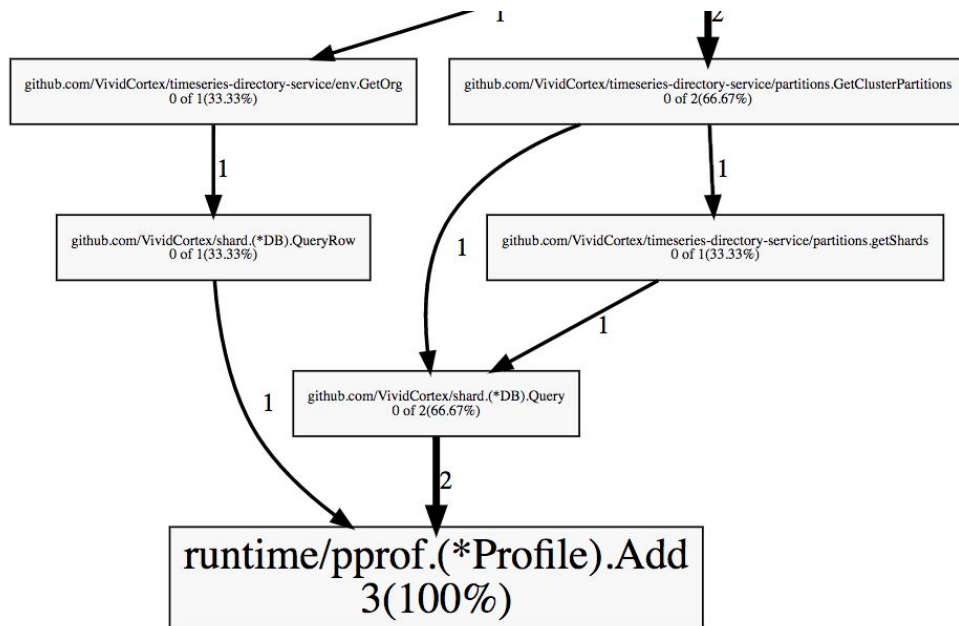
```
(pprof) list handle
Total: 2
ROUTINE ===== main.handleIncCounter in /Users/johnpotocny/vividcortex/go/s
demo4/demo4.go
      0      1 (flat, cum) 50.00% of Total
      .      .    65:
      .      .    66:   return latest
      .      .    67:}
      .      .    68:
      .      .    69:func handleIncCounter(w http.ResponseWriter, r *http.Request) {
      .      1    70:   defer trackRequest()()
      .      .    71:   incrementCounter()
      .      .    72:}
      .      .    73:
```

When to Use Custom Profiles

- Track a critical resource in your application
 - Open files
 - Active DB handles
 - Open Network sockets
 - In-use worker routines
- Make it easier to identify resource leaks in our apps
- We can also identify contention around bounded resources

A real-world case for custom Profiles

- I added support for tracking active-queries in a service at VividCortex!
 - Now it's easier for us to track db-concurrency in our apps



What's next for pprof?

- Support tracking file descriptors? <https://github.com/golang/go/issues/16379>
- Profile non-heap memory too? <https://github.com/golang/go/issues/15848>
- Count-oriented custom profiles? <https://github.com/golang/go/issues/18454>
 - Allow aggregating events forever rather than showing current-state only
 - Accepted proposal, Go1.9 milestone
- Support profiles with labels: <https://github.com/golang/go/issues/17280>
 - Accepted proposal, Go1.9 milestone

Thank You!

