Профилирование и оптимизация программ на Go

Олег Федосеев,
руководитель отдела backend-разработки
@olegfedoseev
o.fedoseev@office.ngs.ru

НГС
ТЕХНОЛОГИИ

Кто я?

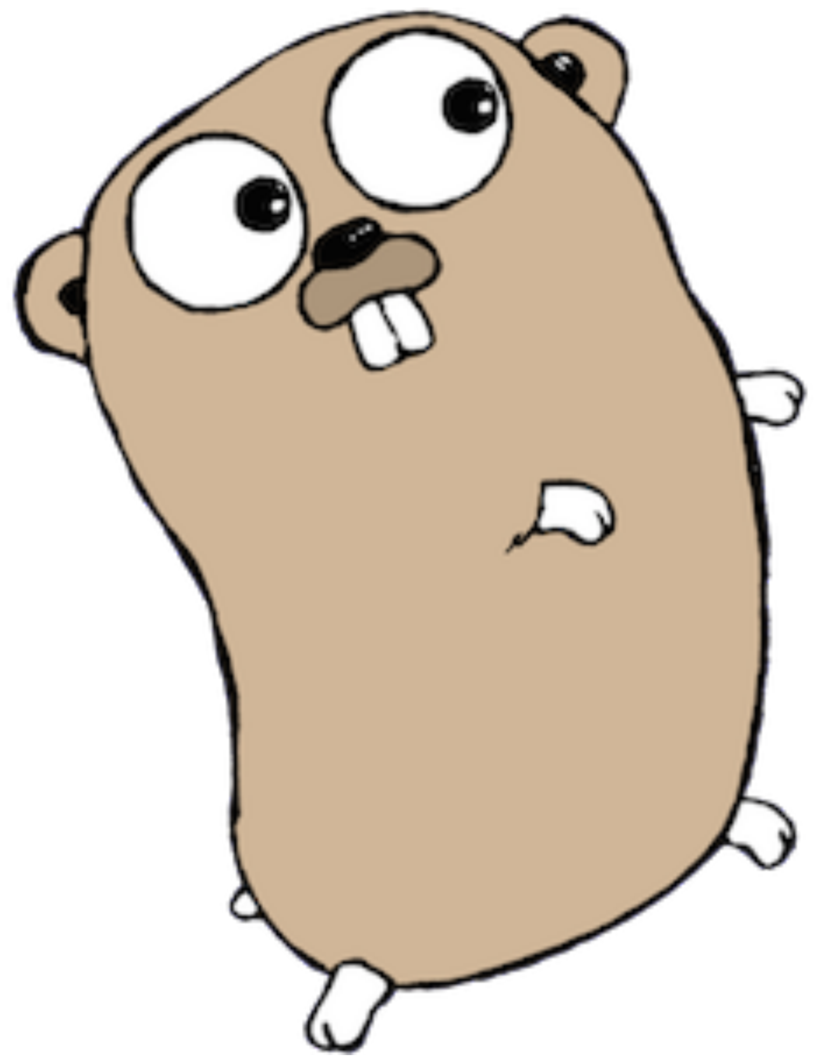
- Руковожу всей backend разработкой в **НГС**
- Пишу код более 8 лет
- Последние пару лет предпочитаю **Go**

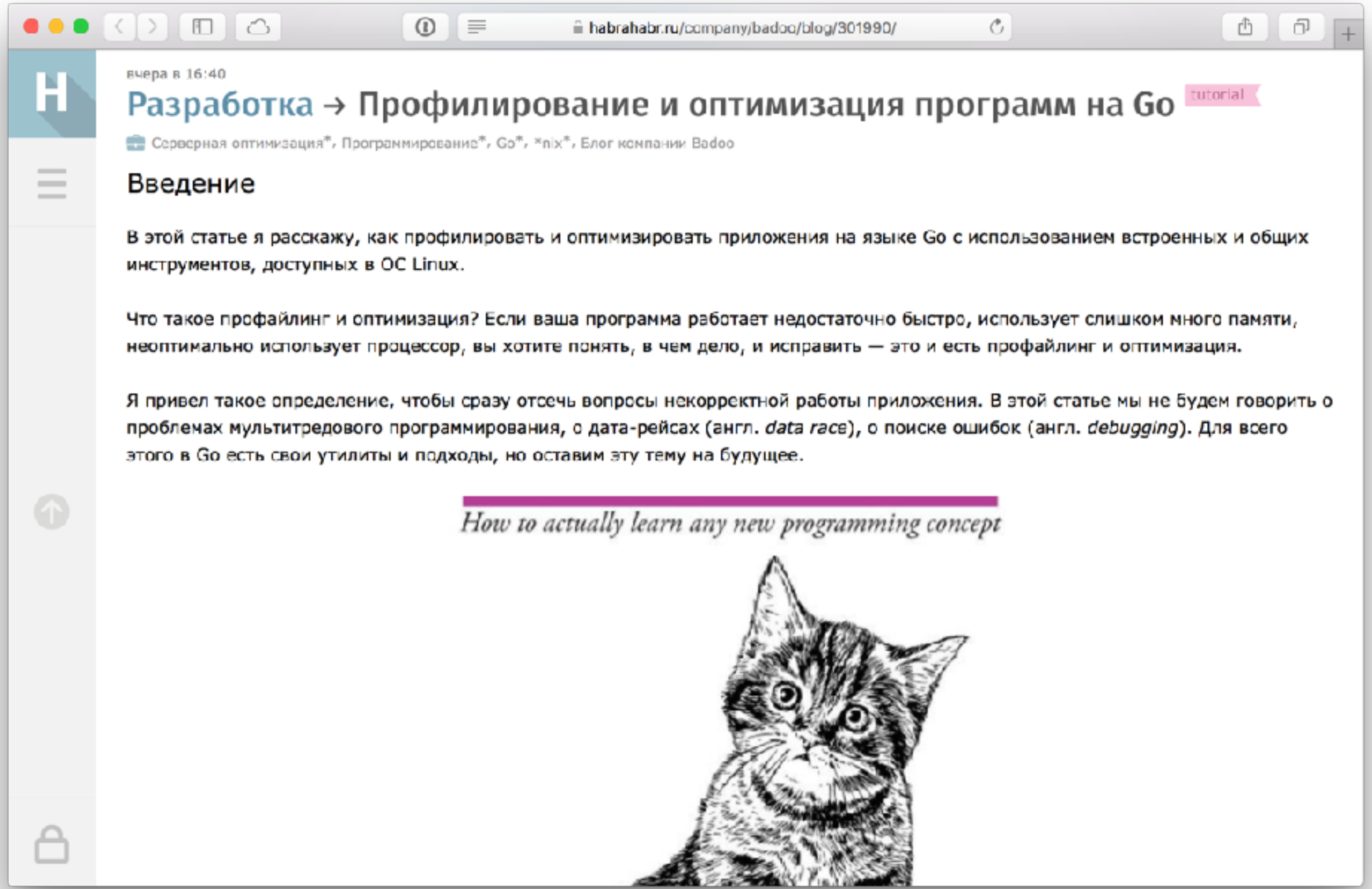
НГС ТЕХНОЛОГИИ

А так же мы делаем

N1.RU зарплата.ру

Лирическое отступление




habrahabr.ru/company/badoo/blog/301990/

вчера в 16:40

Разработка → Профилирование и оптимизация программ на Go

Серверная оптимизация*, Программирование*, Go*, *nix*, Блог компании Badoo

Введение

В этой статье я расскажу, как профилировать и оптимизировать приложения на языке Go с использованием встроенных и общих инструментов, доступных в ОС Linux.

Что такое профайлинг и оптимизация? Если ваша программа работает недостаточно быстро, использует слишком много памяти, неоптимально использует процессор, вы хотите понять, в чем дело, и исправить — это и есть профайлинг и оптимизация.

Я привел такое определение, чтобы сразу отсечь вопросы некорректной работы приложения. В этой статье мы не будем говорить о проблемах мультитредового программирования, о дата-рейсах (англ. *data race*), о поиске ошибок (англ. *debugging*). Для всего этого в Go есть свои утилиты и подходы, но оставим эту тему на будущее.

https://habrahabr.ru/company/badoo/blog/301990/

Краткий план

- Собираем данные
- Анализируем данные
- Примеры оптимизаций
- **Демо**!

Package pprof

```
import "runtime/pprof"
```

Overview
Index

Overview

Package pprof writes runtime profiling data in the format expected by the pprof visualization tool. For more information about pprof, see http://code.google.com/p/google-perftools/.

Index

func Profiles() []*Profile
func StartCPUProfile(w io.Writer) error
func StopCPUProfile()
func WriteHeapProfile(w io.Writer) error
type Profile
    func Lookup(name string) *Profile
    func NewProfile(name string) *Profile
    func (p *Profile) Add(value interface{}, skip int)
    func (p *Profile) Count() int
    func (p *Profile) Name() string
    func (p *Profile) Remove(value interface{})
    func (p *Profile) WriteTo(w io.Writer, debug int) error
Bugs

import “runtime/pprof”

```
var cpuprofile = flag.String("cpuprofile", "", "write cpu profile to file")

func main() {
    flag.Parse()
    if *cpuprofile != "" {
        f, err := os.Create(*cpuprofile)
        if err != nil {
            log.Fatal(err)
        }
        pprof.StartCPUProfile(f)
        defer pprof.StopCPUProfile()
    }
    ...
```

Package pprof

```
import "net/http/pprof"
```

Overview

Package pprof serves via its HTTP server runtime profiling data in the format expected by the pprof visualization tool. For more information about pprof, see http://code.google.com/p/google-perftools/.

The package is typically only imported for the side effect of registering its HTTP handlers. The handled paths all begin with /debug/pprof/.

To use pprof, link this package into your program:

```
import _ "net/http/pprof"
```

If your application is not already running an http server, you need to start one. Add "net/http" and "log" to your imports and the following code to your main function:

```
go func() {
	log.Println(http.ListenAndServe("localhost:6060", nil))
}()
```

Then use the pprof tool to look at the heap profile:

```
go tool pprof http://localhost:6060/debug/pprof/heap
```

Benchmarks

Functions of the form

```
func BenchmarkXxx(*testing.B)
```

are considered benchmarks, and are executed by the "go test" command when its -bench flag is provided.
Benchmarks are run sequentially.

For a description of the testing flags, see https://golang.org/cmd/go/#hdr-Description_of_testing_flags.

A sample benchmark function looks like this:

```
func BenchmarkHello(b *testing.B) {
    for i := 0; i < b.N; i++ {
        fmt.Sprintf("hello")
    }
}
```

The benchmark function must run the target code b.N times. During benchmark execution, b.N is adjusted until
the benchmark function lasts long enough to be timed reliably. The output

```
BenchmarkHello    10000000    282 ns/op
```

means that the loop ran 10000000 times at a speed of 282 ns per loop.

Собираем данные

- Явный start-stop https://golang.org/pkg/runtime/pprof/
- Http-интерфейс https://golang.org/pkg/net/http/pprof/
- Бенчмарки https://golang.org/pkg/testing/

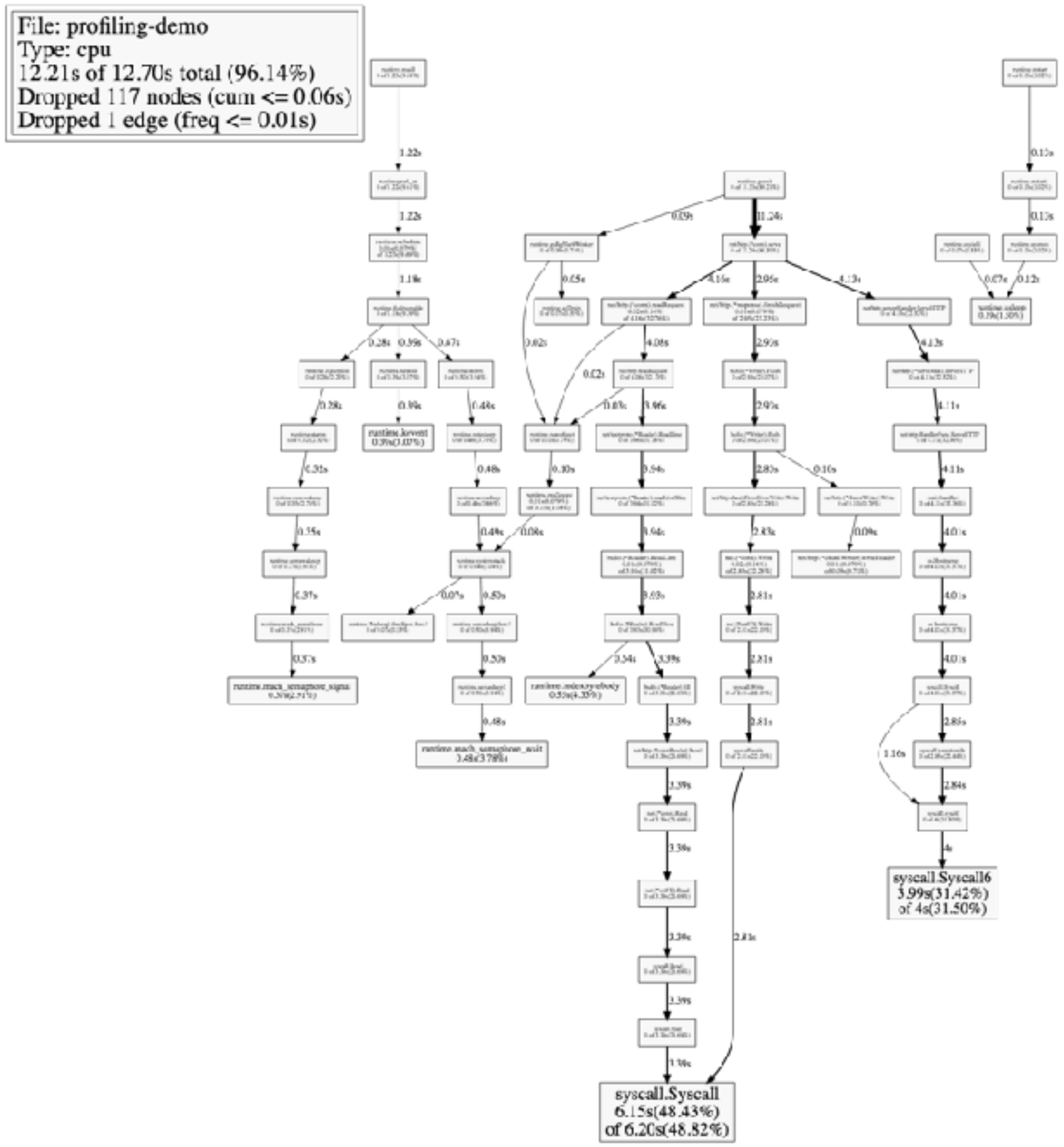
go tool pprof

```
(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
```

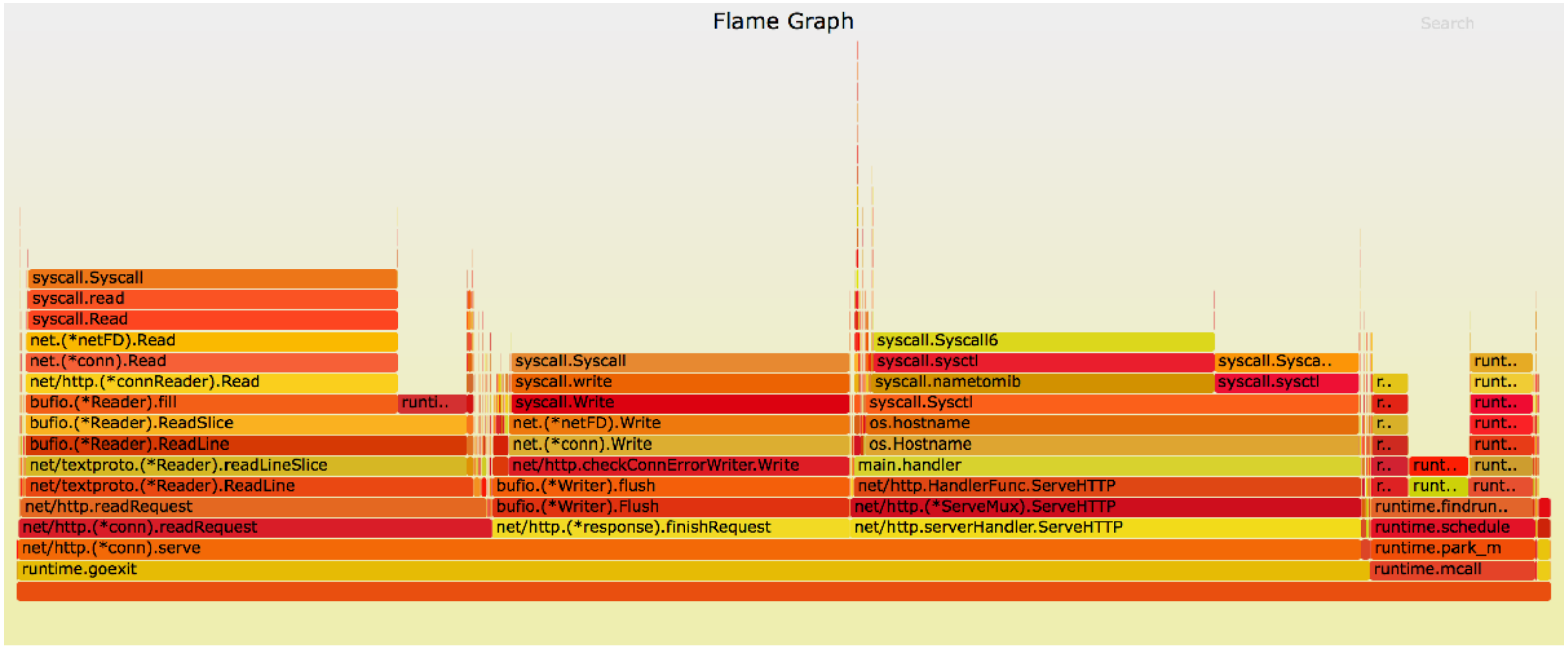
go tool pprof

```
(pprof) list main.handler
Total: 12.70s
ROUTINE ======================== main.handler in profiling-demo/main.go
         0      4.11s (flat, cum) 32.36% of Total
         .          .     28:      })
         .          .     29:      log.Fatal(http.ListenAndServe(":8080", nil))
         .          .     30:}
         .          .     31:
         .          .     32:func handler(w http.ResponseWriter, r *http.Request) {
         .      4.01s     33:   host, err := os.Hostname()
         .          .     34:      if err != nil {
         .          .     35:              log.Printf("Failed to get hostname: %v", err)
         .          .     36:              http.Error(w, err.Error(), http.StatusInternalServerError)
         .          .     37:      }
         .          .     38:
         .          .     39:      now := time.Now()
         .          .     40:
```

go tool pprof


File: profiling-demo
Type: cpu
12.21s of 12.70s total (96.14%)
Dropped 117 nodes (cum <= 0.06s)
Dropped 1 edge (freq <= 0.01s)
runtime.kevent
0.39s(3.07%)
runtime.mach_semaphore_signal
0.36s(2.91%)
runtime.mach_semaphore_wait
0.48s(3.78%)
runtime.memclrNoHeapPointers
0.55s(4.33%)
syscall.Syscall6
3.99s(31.42%)
of 4s(31.50%)
syscall.Syscall
6.15s(48.43%)
of 6.20s(48.82%)

go tool pprof



Спасибо!

Олег Федосеев
oleg.fedoseev@me.com
@olegfedoseev