Бенчмаркинг/профайлинг в Go

Нина Пакшина

Инструменты в Go

- Бенчмарки (go test)
- Профайлинг в файл (runtime/pprof)
- Веб-профайлинг (net/http/pprof)

Бенчмарки

- Простота использования
- Не влияет на продакшн
- Можно сравнивать бенчмарки
- Нужно писать бенчмарки
- Отображает только время исполнения операции целиком

Benchmark

```go
const (
    factNumber = 3628800
)

func BenchmarkRecursive(b *testing.B) {
    for i := 0; i < b.N; i++ {
        Recursive(factNumber)
    }
}
```

Запуск

go test -v ./... -bench=. -benchtime 100x

```
ninako@MacBook-Pro-Nina factorial % go test -v ./... -bench=. -benchtime 100x
goos: darwin
goarch: arm64
pkg: profiling/factorial
BenchmarkRecursive
BenchmarkRecursive-8          100          30646691 ns/op
BenchmarkDynamic
BenchmarkDynamic-8            100          14020201 ns/op
PASS
ok      profiling/factorial     5.017s
```

Сравнение

```go
func Calculate() []int {
    var slice []int
    for i := 0; i < maxSliceSize; i++ {
        slice = append(slice, Recursive(i))
    }
    return slice
}
```

```go
func BenchmarkCalculate(b *testing.B) {
    for i := 0; i < b.N; i++ {
        Calculate()
    }
}
```

```
go test ./... -bench="BenchmarkCalculate"
-run=^# -count=5 | tee old.txt
```

```
ninako@MacBook-Pro-Nina factorial % go test ./... -bench="Be
goos: darwin
goarch: arm64
pkg: profiling/factorial
BenchmarkCalculate-8           4         298302354 ns/op
BenchmarkCalculate-8           4         297754146 ns/op
BenchmarkCalculate-8           4         298117698 ns/op
BenchmarkCalculate-8           4         297987448 ns/op
BenchmarkCalculate-8           4         298118031 ns/op
PASS
ok      profiling/factorial     12.787s
```

```go
func Calculate() []int {
    var slice []int
    for i := 0; i < maxSliceSize; i++ {
        slice = append(slice, Dynamic(i))
    }
    return slice
}
```

```
go test ./... -bench="BenchmarkCalculate"
-run=^# -count=5 | tee new.txt
```

Benchstat

```
go get golang.org/x/perf/cmd/benchstat
```

```
benchstat old.txt new.txt
```

```
goos: darwin
goarch: arm64
pkg: profiling/factorial
            |   old.txt    |                new.txt                 |
            |    sec/op    |    sec/op     vs base                  |
Calculate-8   298.1m ± ∞ ¹   158.4m ± ∞ ¹  -46.87% (p=0.008 n=5)
¹ need >= 6 samples for confidence interval at level 0.95
```

Профилирование pprof

- Большое количество инструментов для анализа
- Позволяет анализировать код “по кирпичикам”
- Запускается из рабочего кода
- Влияет на производительность
- Возможность удаленного анализа

Типы профилирования

- **CPU** (использование процессора приложением)
- **Heap** / **Memory** (использование приложением памяти)
- **Goroutine** (функции, создающих наибольшее количество горутин)
- **Block** (функции, вызывающих наибольшее количество блокировок)
- **Thread** (функции, создающих наибольшее количество потоков)
- **Mutex** (функции с наибольшей конкуренцией за мьютексы)

Можно получить с помощью бенчмарков:

```
go test -bench='BenchmarkCalculate' -cpuprofile='cpu.prof'
-memprofile='mem.prof'
```

Профилирование ЦПУ

go tool pprof cpu.prof

```
ninako@MacBook-Pro-Nina factorial % go tool pprof cpu.prof
Type: cpu
Time: Sep 23, 2023 at 3:01pm (MSK)
Duration: 2.53s, Total samples = 1.82s (72.07%)
Entering interactive mode (type "help" for commands, "o" for options)
(pprof)
```

Топ потребления ЦПУ

```
(pprof) top5
Showing nodes accounting for 1.82s, 100% of 1.82s total
Showing top 5 nodes out of 23
      flat  flat%   sum%        cum   cum%
     1.67s 91.76% 91.76%      1.75s 96.15%  profiling/factorial.Recursive
     0.08s  4.40% 96.15%      0.08s  4.40%  runtime.asyncPreempt
     0.03s  1.65% 97.80%      0.03s  1.65%  runtime.madvise
     0.03s  1.65% 99.45%      0.03s  1.65%  runtime.memclrNoHeapPointers
     0.01s  0.55%   100%      0.01s  0.55%  runtime.writeHeapBits.flush
```

Топ 5

granularity=lines

```
(pprof) top5
Showing nodes accounting for 1790ms, 98.35% of 1820ms total
Showing top 5 nodes out of 27
      flat  flat%   sum%        cum   cum%
    1430ms 78.57% 78.57%     1750ms 96.15%  profiling/factorial.Recursive /Users/ninako/Code/Golang/profiling/factorial/fact.go:11
     230ms 12.64% 91.21%      230ms 12.64%  profiling/factorial.Recursive /Users/ninako/Code/Golang/profiling/factorial/fact.go:7
      80ms  4.40% 95.60%       80ms  4.40%  runtime.asyncPreempt /usr/local/go/src/runtime/preempt_arm64.s:7
      30ms  1.65% 97.25%       30ms  1.65%  runtime.madvise /usr/local/go/src/runtime/sys_darwin.go:253
      20ms  1.10% 98.35%       20ms  1.10%  runtime.memclrNoHeapPointers /usr/local/go/src/runtime/memclr_arm64.s:174
(pprof)
```

Топ 5

hide=runtime

```
   hide=runtime
Showing nodes accounting for 1.79s, 98.35% of 1.82s total
Showing top 5 nodes out of 7
      flat  flat%   sum%        cum   cum%
     1.51s 82.97% 82.97%      1.75s 96.15%  profiling/factorial.Recursive /Use
     0.23s 12.64% 95.60%      0.23s 12.64%  profiling/factorial.Recursive /Use
     0.04s  2.20% 97.80%      0.04s  2.20%  profiling/factorial.Calculate /Use
     0.01s  0.55% 98.35%      0.01s  0.55%  profiling/factorial.Recursive /Use
         0     0% 98.35%      0.04s  2.20%  profiling/factorial.BenchmarkCalcu
```

`focus= - все сбросить`

Отобразить потребление построчно

list profiling/factorial.Recursive

```
(pprof) list profiling/factorial.Recursive
Total: 1.82s
ROUTINE ======================== profiling/factorial.Recursive in /
     1.75s      1.99s (flat, cum) 109.34% of Total
     230ms      230ms      7:func Recursive(n int) int {
         .          .      8:   if n == 0 {
      10ms       10ms      9:           return 1
         .          .     10:   } else {
     1.51s      1.75s     11:           return n * Recursive(n-1)
         .          .     12:   }
         .          .     13:}
         .          .     14:
         .          .     15:func Dynamic(n int) int {
         .          .     16:   memo := make([]int, n+1)
(pprof)
```

go tool pprof mem.prof

go tool pprof mem.prof

```
(pprof) list factorial.Calculate
Total: 3.28GB
ROUTINE ======================== profiling/factorial.Calculate in /Users/nin
    6.84MB     3.28GB (flat, cum) 99.92% of Total
         .          .     24:func Calculate() []int {
  553.04kB   553.04kB     25:   slice := make([]int, maxSliceSize)
         .          .     26:   for i := 0; i < maxSliceSize; i++ {
    6.30MB     3.27GB     27:           slice = append(slice, Dynamic(i))
         .          .     28:   }
         .          .     29:   return slice
         .          .     30:}
```

Tree

```
Showing nodes accounting for 3519.85MB, 99.91% of 3523.06MB total
Dropped 8 nodes (cum <= 17.62MB)
----------------------------------------------------------+-------------
      flat  flat%   sum%        cum   cum%   calls calls% + context
----------------------------------------------------------+-------------
                                     3519.09MB   100% |   profiling/factorial.Calculate (inline)
 3519.09MB 99.89% 99.89%  3519.09MB 99.89%                | profiling/factorial.Dynamic
----------------------------------------------------------+-------------
                                     3519.85MB   100% |   profiling/factorial.BenchmarkCalculate (inline)
    0.77MB 0.022% 99.91%  3519.85MB 99.91%                | profiling/factorial.Calculate
                                     3519.09MB   100% |   profiling/factorial.Dynamic (inline)
----------------------------------------------------------+-------------
                                     3519.85MB   100% |   testing.(*B).runN
         0     0% 99.91%  3519.85MB 99.91%                | profiling/factorial.BenchmarkCalculate
                                     3519.85MB   100% |   profiling/factorial.Calculate (inline)
----------------------------------------------------------+-------------
         0     0% 99.91%  3519.85MB 99.91%                | testing.(*B).run1.func1
                                     3519.85MB   100% |   testing.(*B).runN
----------------------------------------------------------+-------------
                                     3519.85MB   100% |   testing.(*B).run1.func1
```

Peek

```
(pprof) peek  profiling/factorial.Calculate
Active filters:
   hide=runtime
Showing nodes accounting for 3523.06MB, 100% of 3523.06MB total
----------------------------------------------------------+-------------
      flat  flat%   sum%        cum   cum%   calls calls% + context
----------------------------------------------------------+-------------
                                          3519.85MB   100% |   profiling/factorial.BenchmarkCalculate (inline)
    0.77MB 0.022% 0.022%  3519.85MB 99.91%                | profiling/factorial.Calculate
                                          3519.09MB   100% |   profiling/factorial.Dynamic (inline)
----------------------------------------------------------+-------------
```

Профилирование

Запуск из fileprofile/main.go

```go
func main() {
    cpuFile, err := os.Create( name: "cpu.prof")
    if err != nil {
        log.Fatal(err)
    }
    err = pprof.StartCPUProfile(cpuFile)
    if err != nil {
        log.Fatal(err)
    }
    defer func() {
        pprof.StopCPUProfile()
    }()
```

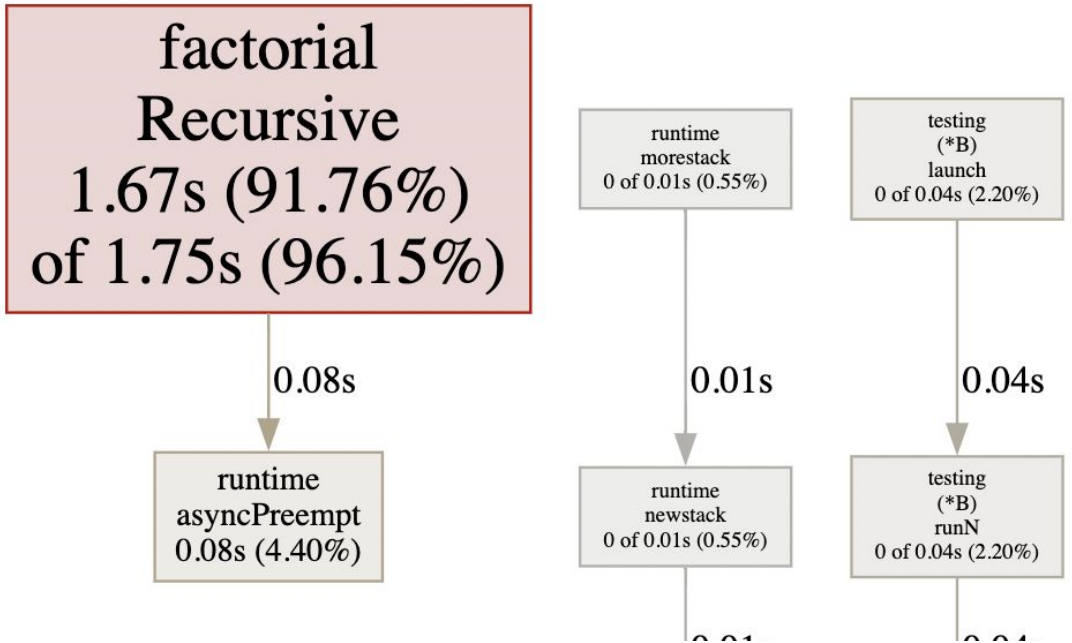
Тэги

```
pprof.Do(ctx, pprof.Labels( args...: "calculate", "calculate"), func(ctx context.Context) {
    factorial.Calculate()
})
```

```
(pprof) tags
 calculate: Total 1.3s
            1.3s (  100%): calculate
```

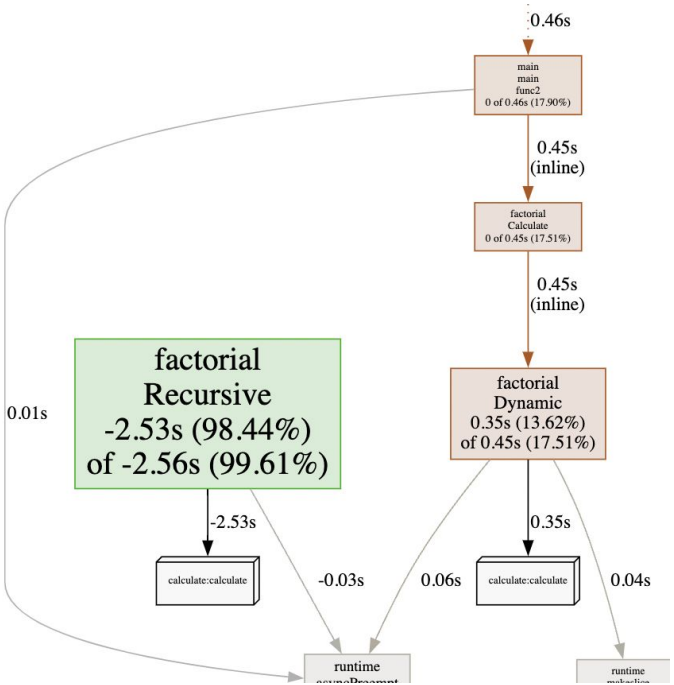
Отображение диаграмм

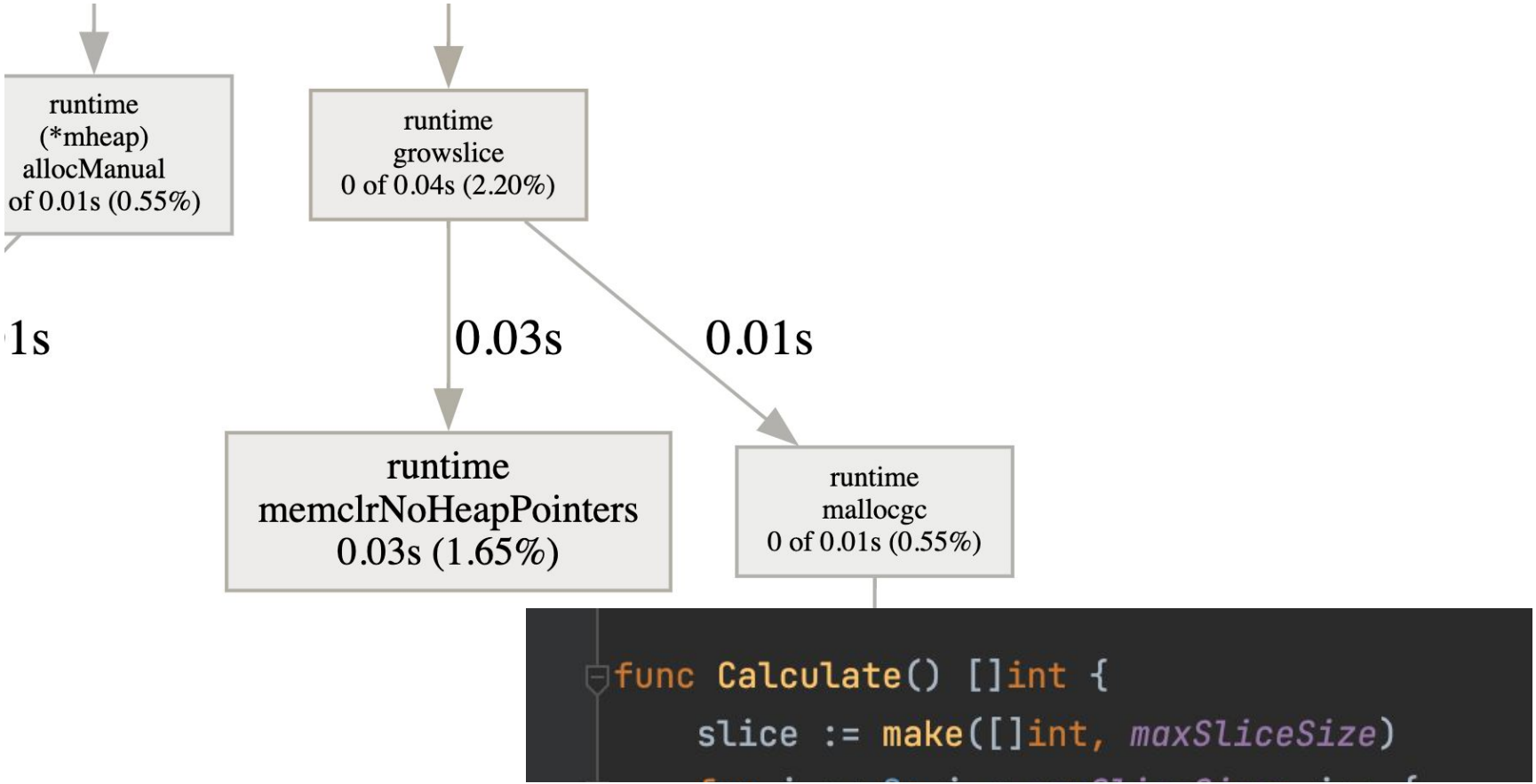
go tool pprof -http=:6060 cpu.prof



Сравнить два варианта

go tool pprof -http=:6060 -diff_base cpu_1.prof cpu.prof




runtime
(*mheap)
allocManual
of 0.01s (0.55%)
runtime
growslice
0 of 0.04s (2.20%)
1s
0.03s
0.01s
runtime
memclrNoHeapPointers
0.03s (1.65%)
runtime
mallocgc
0 of 0.01s (0.55%)
func Calculate() []int {
slice := make([]int, maxSliceSize)

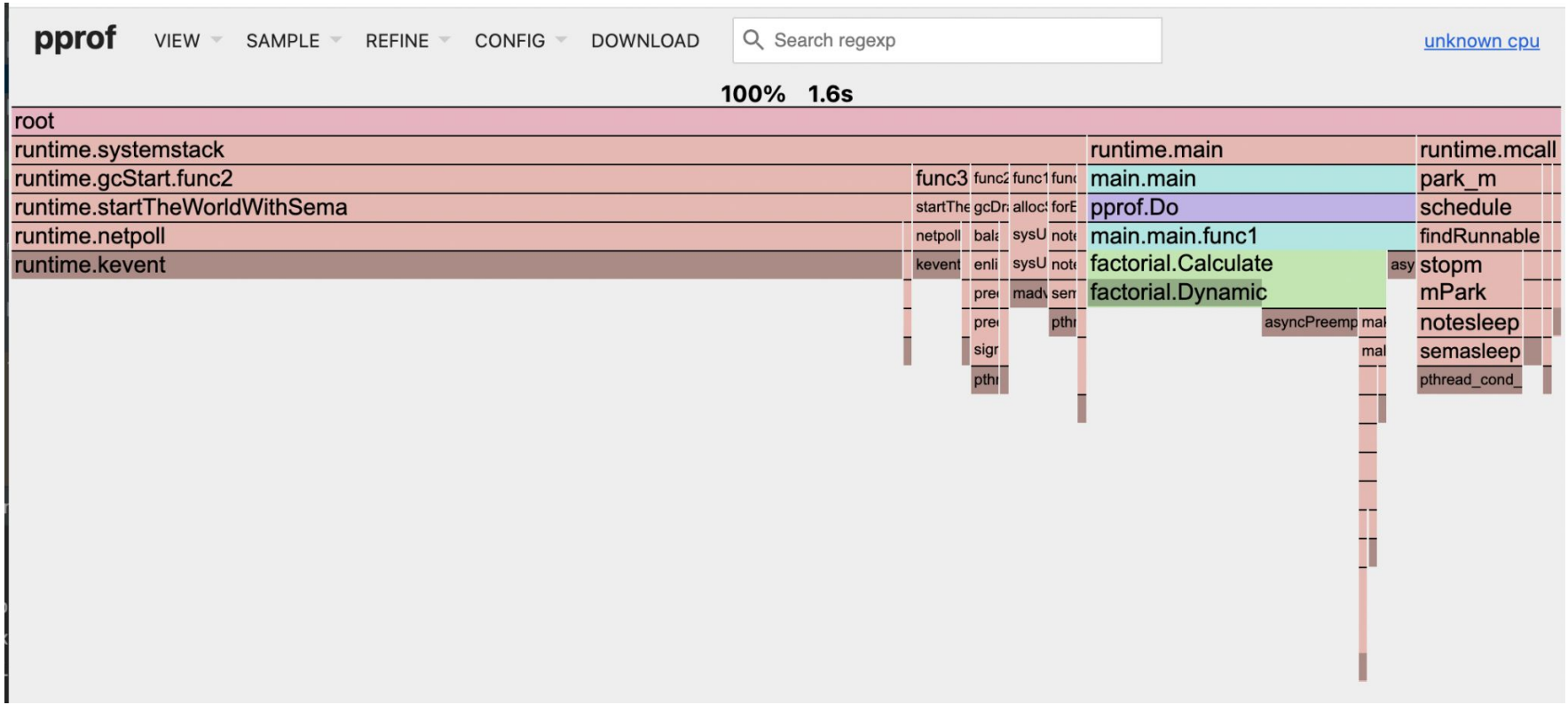
На что обращать внимание

runtime.mallogc - можно посмотреть какие данные могут избежать аллокации

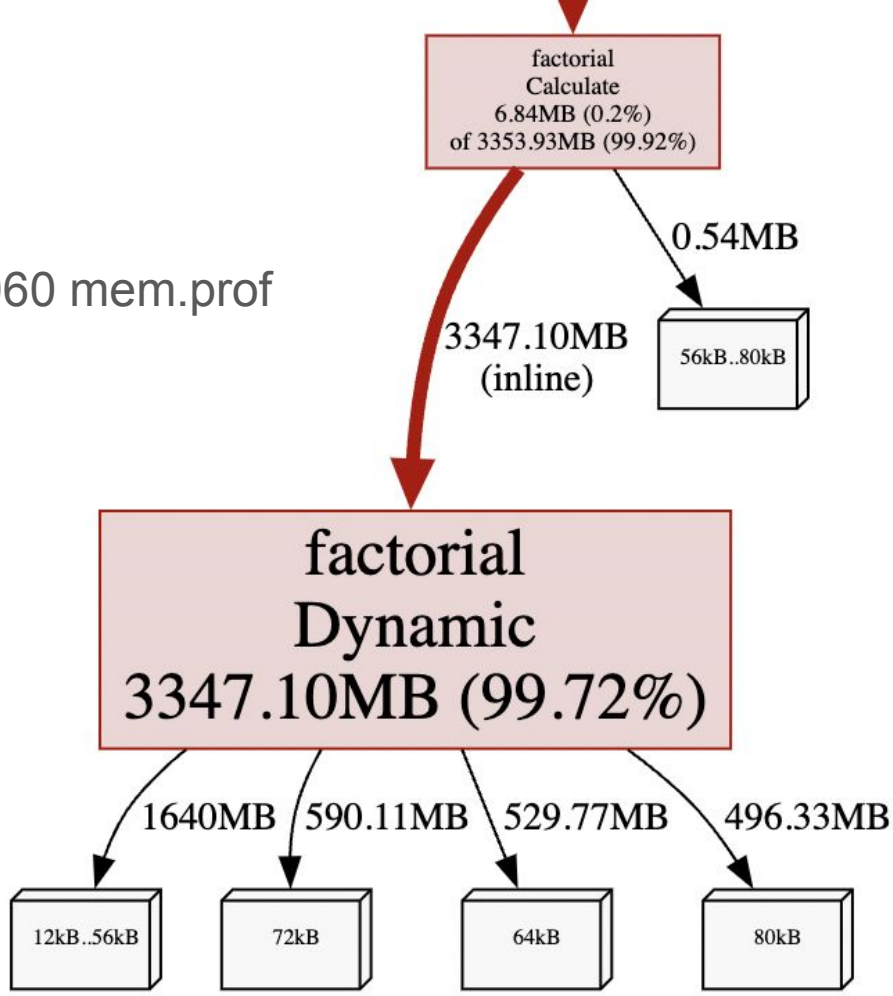
runtime.growslice - инициализировать слайсы

syscall.Read или **syscall.Write -** чтение или запись в режиме kernel, использовать буферизацию

Flame Graph


pprof
VIEW
SAMPLE
REFINE
CONFIG
DOWNLOAD
Search regexp
unknown cpu
100% 1.6s
root
runtime.systemstack
runtime.gcStart.func2
runtime.startTheWorldWithSema
runtime.netpoll
runtime.kevent
func3
startThe
netpoll
kevent
runtime.main
main.main
pprof.Do
main.main.func1
factorial.Calculate
factorial.Dynamic
asyncPreemp
runtime.mcall
park_m
schedule
findRunnable
stopm
mPark
notesleep
semasleep
pthread_cond_

go tool pprof -http=:6060 mem.prof



Запуск в реальном времени

`localhost:5555/debug/pprof`

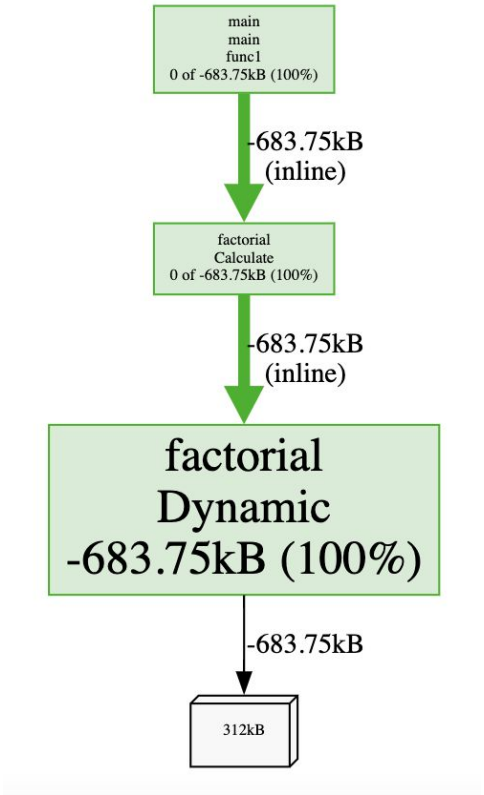
```go
import (
    "log"
    "net/http"
    _ "net/http/pprof"
    "profiling/factorial"
)

func main() {
    go log.Fatal(http.ListenAndServe( addr: "localhost:5555",  handler: nil))
    for {
        factorial.Calculate()
    }
}
```

Диагностика утечки памяти

- Триггерим сборщик мусора с разницей в несколько секунд http://localhost:5555/debug/pprof/heap?gc=1
- Сравниваем разницу:

  go tool pprof -http=:6060 -diff_base heap1.prof heap2.prof

Диагностика



Профайлинг мьютексов и блоков

```
runtime.SetBlockProfileRate( rate: 1)
runtime.SetMutexProfileFraction( rate: 1)
```

go tool pprof -http=:5555 http://localhost:6060/debug/pprof/mutex

go tool pprof -http=:6061 http://localhost:6060/debug/pprof/block