
prealloc

prealloc is a Go static analysis tool to find slice declarations that could potentially be preallocated.

Installation

```
1 go install github.com/alexkohler/prealloc@latest
```

Usage

Similar to other Go static analysis tools (such as golint, go vet), prealloc can be invoked with one or more filenames, directories, or packages named by its import path. Prealloc also supports the . . . wildcard.

```
1 prealloc [flags] files/directories/packages
```

Flags

- **-simple** (default true) - Report preallocation suggestions only on simple loops that have no returns/breaks/continues/gotos in them. Setting this to false may increase false positives.
- **-rangeloops** (default true) - Report preallocation suggestions on range loops.
- **-forloops** (default false) - Report preallocation suggestions on for loops. This is false by default due to there generally being weirder things happening inside for loops (at least from what I've observed in the Standard Library).
- **-set_exit_status** (default false) - Set exit status to 1 if any issues are found.

Purpose

While the Go *does* attempt to avoid reallocation by growing the capacity in advance, this sometimes isn't enough for longer slices. If the size of a slice is known at the time of its creation, it should be specified.

Consider the following benchmark: (this can be found in prealloc_test.go in this repo)

```
1 import "testing"
2
3 func BenchmarkNoPreallocate(b *testing.B) {
4     existing := make([]int64, 10, 10)
5     b.ResetTimer()
```

```

6     for i := 0; i < b.N; i++ {
7         // Don't preallocate our initial slice
8         var init []int64
9         for _, element := range existing {
10             init = append(init, element)
11         }
12     }
13 }
14
15 func BenchmarkPreallocate(b *testing.B) {
16     existing := make([]int64, 10, 10)
17     b.ResetTimer()
18     for i := 0; i < b.N; i++ {
19         // Preallocate our initial slice
20         init := make([]int64, 0, len(existing))
21         for _, element := range existing {
22             init = append(init, element)
23         }
24     }
25 }

```

```

1 $ go test -bench=. -benchmem
2 goos: linux
3 goarch: amd64
4 BenchmarkNoPreallocate-4      30000000          510 ns/op          248 B/
   op                          5 allocs/op
5 BenchmarkPreallocate-4      200000000          111 ns/op           80 B/
   op                          1 allocs/op

```

As you can see, not preallocating can cause a performance hit, primarily due to Go having to reallocate the underlying array. The pattern benchmarked above is common in Go: declare a slice, then write some sort of range or for loop that appends or indexes into it. The purpose of this tool is to flag slice/loop declarations like the one in [BenchmarkNoPreallocate](#).

Example

Some examples from the Go 1.9.2 source:

```

1 $ prealloc go/src/....
2 archive/tar/reader_test.go:854 Consider preallocating ss
3 archive/zip/zip_test.go:201 Consider preallocating all
4 cmd/api/goapi.go:301 Consider preallocating missing
5 cmd/api/goapi.go:476 Consider preallocating files
6 cmd/asm/internal/asm/endtoend_test.go:345 Consider preallocating extra
7 cmd/cgo/main.go:60 Consider preallocating ks
8 cmd/cgo/ast.go:149 Consider preallocating pieces
9 cmd/compile/internal/ssa/flagalloc.go:64 Consider preallocating
   oldSched

```

```
10 cmd/compile/internal/ssa/regalloc.go:719 Consider preallocatingphis
11 cmd/compile/internal/ssa/regalloc.go:718 Consider preallocating
    oldSched
12 cmd/compile/internal/ssa/regalloc.go:1674 Consider preallocating
    oldSched
13 cmd/compile/internal/ssa/gen/rulegen.go:145 Consider preallocating ops
14 cmd/compile/internal/ssa/gen/rulegen.go:145 Consider preallocating ops
15 cmd/dist/build.go:893 Consider preallocating all
16 cmd/dist/build.go:1246 Consider preallocating plats
17 cmd/dist/build.go:1264 Consider preallocating results
18 cmd/dist/buildgo.go:59 Consider preallocating list
19 cmd/doc/pkg.go:363 Consider preallocating names
20 cmd/fix/typecheck.go:219 Consider preallocating b
21 cmd/go/internal/base/path.go:34 Consider preallocating out
22 cmd/go/internal/get/get.go:175 Consider preallocating out
23 cmd/go/internal/load/pkg.go:1894 Consider preallocating dirent
24 cmd/go/internal/work/build.go:2402 Consider preallocating abs0files
25 cmd/go/internal/work/build.go:2731 Consider preallocating abs0files
26 cmd/internal/objfile/pe.go:48 Consider preallocating syms
27 cmd/internal/objfile/pe.go:38 Consider preallocating addrs
28 cmd/internal/objfile/goobj.go:43 Consider preallocating syms
29 cmd/internal/objfile/elf.go:35 Consider preallocating syms
30 cmd/link/internal/ld/lib.go:1070 Consider preallocating argv
31 cmd/vet/all/main.go:91 Consider preallocating pp
32 database/sql/sql.go:66 Consider preallocating list
33 debug/macho/file.go:506 Consider preallocating all
34 internal/trace/order.go:55 Consider preallocating batches
35 mime/quotedprintable/reader_test.go:191 Consider preallocating outcomes
36 net/dnsclient_unix_test.go:954 Consider preallocating conflines
37 net/interface_solaris.go:85 Consider preallocating ifat
38 net/interface_linux_test.go:91 Consider preallocating ifmat4
39 net/interface_linux_test.go:100 Consider preallocating ifmat6
40 net/internal/socktest/switch.go:34 Consider preallocating st
41 os/os_windows_test.go:766 Consider preallocating args
42 runtime/pprof/internal/profile/filter.go:77 Consider preallocating
    lines
43 runtime/pprof/internal/profile/profile.go:554 Consider preallocating
    names
44 text/template/parse/node.go:189 Consider preallocating decl
```

```
1 // cmd/api/goapi.go:301
2 var missing []string
3 for feature := range optionalSet {
4     missing = append(missing, feature)
5 }
6
7 // cmd/fix/typecheck.go:219
8 var b []ast.Expr
9 for _, x := range a {
10     b = append(b, x)
11 }
```

```

12
13 // net/internal/socktest/switch.go:34
14 var st []Stat
15 sw.smu.RLock()
16 for _, s := range sw.stats {
17     ns := *s
18     st = append(st, ns)
19 }
20 sw.smu.RUnlock()
21
22 // cmd/api/goapi.go:301
23 var missing []string
24 for feature := range optionalSet {
25     missing = append(missing, feature)
26 }

```

Even if the size the slice is being preallocated to is small, there's still a performance gain to be had in explicitly specifying the capacity rather than leaving it up to `append` to discover that it needs to preallocate. Of course, preallocation doesn't need to be done *everywhere*. This tool's job is just to help suggest places where one should consider preallocating.

How do I fix prealloc's suggestions?

During the declaration of your slice, rather than using the zero value of the slice with `var`, initialize it with Go's built-in `make` function, passing the appropriate type and length. This length will generally be whatever you are ranging over. Fixing the examples from above would look like so:

```

1 // cmd/api/goapi.go:301
2 missing := make([]string, 0, len(optionalSet))
3 for feature := range optionalSet {
4     missing = append(missing, feature)
5 }
6
7 // cmd/fix/typecheck.go:219
8 b := make([]ast.Expr, 0, len(a))
9 for _, x := range a {
10     b = append(b, x)
11 }
12
13 // net/internal/socktest/switch.go:34
14 st := make([]Stat, 0, len(sw.stats))
15 sw.smu.RLock()
16 for _, s := range sw.stats {
17     ns := *s
18     st = append(st, ns)
19 }
20 sw.smu.RUnlock()
21

```

```

22 // cmd/api/goapi.go:301
23 missing := make ([]string, 0, len(optionalSet))
24 for feature := range optionalSet {
25     missing = append(missing, feature)
26 }

```

Note: If performance is absolutely critical, it may be more efficient to use `copy` instead of `append` for larger slices. For reference, see the following benchmark:

```

1 func BenchmarkSize200PreallocateCopy(b *testing.B) {
2     existing := make([]int64, 200, 200)
3     b.ResetTimer()
4     for i := 0; i < b.N; i++ {
5         // Preallocate our initial slice
6         init := make([]int64, len(existing))
7         copy(init, existing)
8     }
9 }

```

```

1 $ go test -bench=. -benchmem
2 goos: linux
3 goarch: amd64
4 BenchmarkSize200NoPreallocate-4          500000          3080 ns/op
   4088 B/op          9 allocs/op
5 BenchmarkSize200Preallocate-4           1000000          1163 ns/op
   1792 B/op          1 allocs/op
6 BenchmarkSize200PreallocateCopy-4        2000000           807 ns/op
   1792 B/op          1 allocs/op

```

TODO

- Configuration on whether or not to run on test files
- Support for embedded ifs (currently, prealloc will only find breaks/returns/continues/gotos if they are in a single if block, I'd like to expand this to supporting multiple if blocks in the future).
- Globbing support (e.g. prealloc *.go)

Contributing

Pull requests welcome!

Other static analysis tools

If you've enjoyed prealloc, take a look at my other static analysis tools! - nakedret - Finds naked returns. - unimport - Finds unnecessary import aliases.