**Тест cpu SlowSearch**

$ go tool pprof cpu.out

Type: cpu

Time: Sep 28, 2021 at 11:07pm (MSK)

Duration: 2.54s, Total samples = 2.26s (88.99%)

Entering interactive mode (type "help" for commands, "o" for options)

(pprof) list SlowSearch

Total: 2.26s

ROUTINE ======================== hw3.SlowSearch in C:\Go\golang-stepik-2021q3\3\99\_hw\common.go

0 710ms (flat, cum) 31.42% of Total

. . 17: file, err := os.Open(filePath)

. . 18: if err != nil {

. . 19: panic(err)

. . 20: }

. . 21:

. 20ms 22: fileContents, err := ioutil.ReadAll(file)

. . 23: if err != nil {

. . 24: panic(err)

. . 25: }

. . 26:

. . 27: r := regexp.MustCompile("@")

. . 28: seenBrowsers := []string{}

. . 29: uniqueBrowsers := 0

. . 30: foundUsers := ""

. . 31:

. . 32: lines := strings.Split(string(fileContents), "\n")

. . 33:

. . 34: users := make([]map[string]interface{}, 0)

. . 35: for \_, line := range lines {

. 10ms 36: user := make(map[string]interface{})

. . 37: // fmt.Printf("%v %v\n", err, line)

. 220ms 38: err := json.Unmarshal([]byte(line), &user)

. . 39: if err != nil {

. . 40: panic(err)

. . 41: }

. . 42: users = append(users, user)

. . 43: }

. . 44:

. . 45: for i, user := range users {

. . 46:

. . 47: isAndroid := false

. . 48: isMSIE := false

. . 49:

. . 50: browsers, ok := user["browsers"].([]interface{})

. . 51: if !ok {

. . 52: // log.Println("cant cast browsers")

. . 53: continue

. . 54: }

. . 55:

. . 56: for \_, browserRaw := range browsers {

. . 57: browser, ok := browserRaw.(string)

. . 58: if !ok {

. . 59: // log.Println("cant cast browser to string")

. . 60: continue

. . 61: }

. 160ms 62: if ok, err := regexp.MatchString("Android", browser); ok && err == nil {

. . 63: isAndroid = true

. . 64: notSeenBefore := true

. . 65: for \_, item := range seenBrowsers {

. . 66: if item == browser {

. . 67: notSeenBefore = false

. . 68: }

. . 69: }

. . 70: if notSeenBefore {

. . 71: // log.Printf("SLOW New browser: %s, first seen: %s", browser, user["name"])

. . 72: seenBrowsers = append(seenBrowsers, browser)

. . 73: uniqueBrowsers++

. . 74: }

. . 75: }

. . 76: }

. . 77:

. . 78: for \_, browserRaw := range browsers {

. . 79: browser, ok := browserRaw.(string)

. . 80: if !ok {

. . 81: // log.Println("cant cast browser to string")

. . 82: continue

. . 83: }

. 290ms 84: if ok, err := regexp.MatchString("MSIE", browser); ok && err == nil {

. . 85: isMSIE = true

. . 86: notSeenBefore := true

. . 87: for \_, item := range seenBrowsers {

. . 88: if item == browser {

. . 89: notSeenBefore = false

. . 90: }

. . 91: }

. . 92: if notSeenBefore {

. . 93: // log.Printf("SLOW New browser: %s, first seen: %s", browser, user["name"])

. . 94: seenBrowsers = append(seenBrowsers, browser)

. . 95: uniqueBrowsers++

. . 96: }

. . 97: }

. . 98: }

. . 99:

. . 100: if !(isAndroid && isMSIE) {

. . 101: continue

. . 102: }

. . 103:

. . 104: // log.Println("Android and MSIE user:", user["name"], user["email"])

. . 105: email := r.ReplaceAllString(user["email"].(string), " [at] ")

. 10ms 106: foundUsers += fmt.Sprintf("[%d] %s <%s>\n", i, user["name"], email)

. . 107: }

. . 108:

. . 109: fmt.Fprintln(out, "found users:\n"+foundUsers)

. . 110: fmt.Fprintln(out, "Total unique browsers", len(seenBrowsers))

. . 111:}

(pprof)

**Тест mem SlowSearch**

$ go tool pprof mem.out

Type: alloc\_space

Time: Sep 28, 2021 at 11:07pm (MSK)

Entering interactive mode (type "help" for commands, "o" for options)

(pprof) list SlowSearch

Total: 1.94GB

ROUTINE ======================== hw3.SlowSearch in C:\Go\golang-stepik-2021q3\3\99\_hw\common.go

65.89MB 995.28MB (flat, cum) 50.20% of Total

. . 17: file, err := os.Open(filePath)

. . 18: if err != nil {

. . 19: panic(err)

. . 20: }

. . 21:

. 153.01MB 22: fileContents, err := ioutil.ReadAll(file)

. . 23: if err != nil {

. . 24: panic(err)

. . 25: }

. . 26:

. . 27: r := regexp.MustCompile("@")

. . 28: seenBrowsers := []string{}

. . 29: uniqueBrowsers := 0

. . 30: foundUsers := ""

. . 31:

23.85MB 23.85MB 32: lines := strings.Split(string(fileContents), "\n")

. . 33:

. . 34: users := make([]map[string]interface{}, 0)

. . 35: for \_, line := range lines {

5MB 5MB 36: user := make(map[string]interface{})

. . 37: // fmt.Printf("%v %v\n", err, line)

24.51MB 107.52MB 38: err := json.Unmarshal([]byte(line), &user)

. . 39: if err != nil {

. . 40: panic(err)

. . 41: }

1MB 1MB 42: users = append(users, user)

. . 43: }

. . 44:

. . 45: for i, user := range users {

. . 46:

. . 47: isAndroid := false

. . 48: isMSIE := false

. . 49:

. . 50: browsers, ok := user["browsers"].([]interface{})

. . 51: if !ok {

. . 52: // log.Println("cant cast browsers")

. . 53: continue

. . 54: }

. . 55:

. . 56: for \_, browserRaw := range browsers {

. . 57: browser, ok := browserRaw.(string)

. . 58: if !ok {

. . 59: // log.Println("cant cast browser to string")

. . 60: continue

. . 61: }

. 410.74MB 62: if ok, err := regexp.MatchString("Android", browser); ok && err == nil {

. . 63: isAndroid = true

. . 64: notSeenBefore := true

. . 65: for \_, item := range seenBrowsers {

. . 66: if item == browser {

. . 67: notSeenBefore = false

. . 68: }

. . 69: }

. . 70: if notSeenBefore {

. . 71: // log.Printf("SLOW New browser: %s, first seen: %s", browser, user["name"])

. . 72: seenBrowsers = append(seenBrowsers, browser)

. . 73: uniqueBrowsers++

. . 74: }

. . 75: }

. . 76: }

. . 77:

. . 78: for \_, browserRaw := range browsers {

. . 79: browser, ok := browserRaw.(string)

. . 80: if !ok {

. . 81: // log.Println("cant cast browser to string")

. . 82: continue

. . 83: }

. 281.12MB 84: if ok, err := regexp.MatchString("MSIE", browser); ok && err == nil {

. . 85: isMSIE = true

. . 86: notSeenBefore := true

. . 87: for \_, item := range seenBrowsers {

. . 88: if item == browser {

. . 89: notSeenBefore = false

. . 90: }

. . 91: }

. . 92: if notSeenBefore {

. . 93: // log.Printf("SLOW New browser: %s, first seen: %s", browser, user["name"])

512.06kB 512.06kB 94: seenBrowsers = append(seenBrowsers, browser)

. . 95: uniqueBrowsers++

. . 96: }

. . 97: }

. . 98: }

. . 99:

. . 100: if !(isAndroid && isMSIE) {

. . 101: continue

. . 102: }

. . 103:

. . 104: // log.Println("Android and MSIE user:", user["name"], user["email"])

. 512.03kB 105: email := r.ReplaceAllString(user["email"].(string), " [at] ")

11.03MB 11.03MB 106: foundUsers += fmt.Sprintf("[%d] %s <%s>\n", i, user["name"], email)

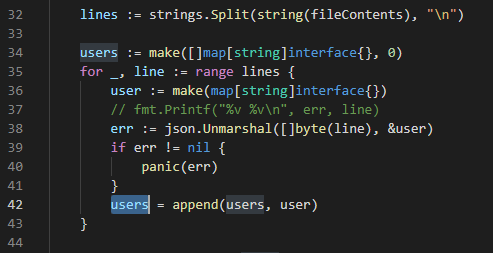
. . 107: }

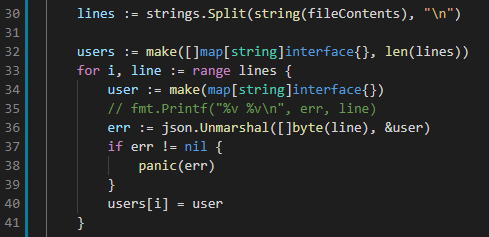
. . 108:

. 1MB 109: fmt.Fprintln(out, "found users:\n"+foundUsers)

. . 110: fmt.Fprintln(out, "Total unique browsers", len(seenBrowsers))

. . 111:}

1. Выделение памяти, которое можно предотвратить  
   



BenchmarkSlow-4 51 23122878 ns/op 19461166 B/op 189794 allocs/op

BenchmarkFast-4 50 23161196 ns/op 19457827 B/op 189784 allocs/op

1. Выделение памяти, которое можно предотвратить

Было:



Стало:



1. 24.51MB 107.52MB 38: err := json.Unmarshal([]byte(line), &user)

Сделать процесс конвертации легче

err := easyjson.Unmarshal([]byte(line), &user)

Генерация кода под описанную структуру

type User struct {

    Username string   `json:"name,string"`

    Email    string   `json:"email,string"`

    Browser  []string `json:"browsers,string"`

}

package main

import (

    json "encoding/json"

    easyjson "github.com/mailru/easyjson"

    jlexer "github.com/mailru/easyjson/jlexer"

    jwriter "github.com/mailru/easyjson/jwriter"

)

// suppress unused package warning

var (

    \_ \*json.RawMessage

    \_ \*jlexer.Lexer

    \_ \*jwriter.Writer

    \_ easyjson.Marshaler

)

func easyjsonB8df9358DecodeHw3(in \*jlexer.Lexer, out \*User) {

    isTopLevel := in.IsStart()

    if in.IsNull() {

        if isTopLevel {

            in.Consumed()

        }

        in.Skip()

        return

    }

    in.Delim('{')

    for !in.IsDelim('}') {

        key := in.UnsafeFieldName(false)

        in.WantColon()

        if in.IsNull() {

            in.Skip()

            in.WantComma()

            continue

        }

        switch key {

        case "name":

            out.Username = string(in.String())

        case "email":

            out.Email = string(in.String())

        case "browsers":

            if in.IsNull() {

                in.Skip()

                out.Browser = nil

            } else {

                in.Delim('[')

                if out.Browser == nil {

                    if !in.IsDelim(']') {

                        out.Browser = make([]string, 0, 4)

                    } else {

                        out.Browser = []string{}

                    }

                } else {

                    out.Browser = (out.Browser)[:0]

                }

                for !in.IsDelim(']') {

                    var v1 string

                    v1 = string(in.String())

                    out.Browser = append(out.Browser, v1)

                    in.WantComma()

                }

                in.Delim(']')

            }

        default:

            in.SkipRecursive()

        }

        in.WantComma()

    }

    in.Delim('}')

    if isTopLevel {

        in.Consumed()

    }

}

func easyjsonB8df9358EncodeHw3(out \*jwriter.Writer, in User) {

    out.RawByte('{')

    first := true

    \_ = first

    {

        const prefix string = ",\"name\":"

        out.RawString(prefix[1:])

        out.String(string(in.Username))

    }

    {

        const prefix string = ",\"email\":"

        out.RawString(prefix)

        out.String(string(in.Email))

    }

    {

        const prefix string = ",\"browsers\":"

        out.RawString(prefix)

        if in.Browser == nil && (out.Flags&jwriter.NilSliceAsEmpty) == 0 {

            out.RawString("null")

        } else {

            out.RawByte('[')

            for v2, v3 := range in.Browser {

                if v2 > 0 {

                    out.RawByte(',')

                }

                out.String(string(v3))

            }

            out.RawByte(']')

        }

    }

    out.RawByte('}')

}

// MarshalJSON supports json.Marshaler interface

func (v User) MarshalJSON() ([]byte, error) {

    w := jwriter.Writer{}

    easyjsonB8df9358EncodeHw3(&w, v)

    return w.Buffer.BuildBytes(), w.Error

}

// MarshalEasyJSON supports easyjson.Marshaler interface

func (v User) MarshalEasyJSON(w \*jwriter.Writer) {

    easyjsonB8df9358EncodeHw3(w, v)

}

// UnmarshalJSON supports json.Unmarshaler interface

func (v \*User) UnmarshalJSON(data []byte) error {

    r := jlexer.Lexer{Data: data}

    easyjsonB8df9358DecodeHw3(&r, v)

    return r.Error()

}

// UnmarshalEasyJSON supports easyjson.Unmarshaler interface

func (v \*User) UnmarshalEasyJSON(l \*jlexer.Lexer) {

    easyjsonB8df9358DecodeHw3(l, v)

}

GORYU@DESKTOP-S0BTTTV MINGW64 /c/Go/golang-stepik-2021q3/3/99\_hw (hw2)

$ go test -bench . -benchmem -cpuprofile=cpu.out -memprofile=mem.out -memprofilerate=1

goos: windows

goarch: amd64

pkg: hw3

cpu: Intel(R) Core(TM) i3-9100F CPU @ 3.60GHz

BenchmarkSlow-4 2 646558750 ns/op 19492256 B/op 189810 allocs/op

BenchmarkFast-4 3 423328467 ns/op 18524578 B/op 144902 allocs/op

PASS

ok hw3 5.770s

1. 392kB 392kB 40: users := make([]User, 0, len(lines))

Убрать накопление юзеров и работать построчно объединив 2 цикла в 1

20.51MB 27: fileContents, err := ioutil.ReadAll(file)

Сделать построчное чтение

fileScanner := bufio.NewScanner(file)

28kB 38: for i := 0; fileScanner.Scan(); i++ {

1. 1s 65: if ok, err := regexp.MatchString("Android", browser); ok && err == nil {

54.27MB 65: if ok, err := regexp.MatchString("Android", browser); ok && err == nil {  
790ms 83: if ok, err := regexp.MatchString("MSIE", browser); ok && err == nil {

Скомпилировать regexp заранее + объединить 2 цикла в 1

rM := regexp.MustCompile("MSIE")

    rA := regexp.MustCompile("Android")

….

okM := false

        okA := false

        isAndroid := false

        isMSIE := false

        for \_, browserRaw := range user.Browser {

            okM = rM.MatchString(browserRaw)

            okA = rA.MatchString(browserRaw)

            if okM || okA {

                if okA {

                    isAndroid = true

                } else {

                    isMSIE = true

                }

                notSeenBefore := true

                for \_, item := range seenBrowsers {

                    if item == browserRaw {

                        notSeenBefore = false

                    }

                }

                if notSeenBefore {

                    // log.Printf("SLOW New browser: %s, first seen: %s", browser, user["name"])

                    seenBrowsers = append(seenBrowsers, browserRaw)

                    uniqueBrowsers++

                }

            }

        }

$ go test -bench . -benchmem -cpuprofile=cpu.out -memprofile=mem.out -memprofilerate=1

goos: windows

goarch: amd64

pkg: hw3

cpu: Intel(R) Core(TM) i3-9100F CPU @ 3.60GHz

BenchmarkSlow-4 2 552252150 ns/op 19508768 B/op 189803 allocs/op

BenchmarkFast-4 52 21649519 ns/op 1979758 B/op 9823 allocs/op

PASS

ok hw3 3.546s