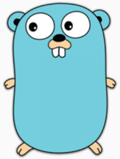
Programming in Go

Matt Holiday
Christmas 2020



Regular Expressions & Search

Searching in strings

Use the `strings` package for simple searches

Carefully use the `regexp` package for complex searches and validation

“Some people, when confronted with a problem, think ‘I know, I’ll use regular expressions.’ Now they have two problems.” — Jamie Zawinski

Searching in strings

Go’s regular expression syntax is a subset of what some other languages have

That avoids the performance impact of *catastrophic backtracking*

See articles by Russ Cox on regular expressions:
`https://swtch.com/~rsc/regexp/`

Simple string searches

Boolean searches:

- `strings.HasPrefix(s, substr)`
- `strings.HasSuffix(s, substr)`
- `strings.Contains(s, substr)`

Location searches:

- `strings.LastIndex(s, substr)`
- `strings.LastIndexByte(s, char)`

Search and replace:

- `strings.Replace(s, substr, replacement, count)`
- `strings.ReplaceAll(s, substr, replacement)`

Location by regex

A regular expression can match a variable number of runes
(a plain string search can't do that)

```go
func main() {
    te := "aba abba abbba"
    re := regexp.MustCompile("b+")
    mm := re.FindAllString(te, -1)
    id := re.FindAllStringIndex(te, -1)

    fmt.Println(mm) // [b bb bbb]
    fmt.Println(id) // [[1 2] [5 7] [10 13]]

    for _, d := range id {
        fmt.Println(te[d[0]:d[1]])    // b bb bbb
    }
}
```

Replacement by regex

A regular expression can match a variable number of runes
(a plain string search can't do that)

```go
func main() {
    te := "aba abba abbba"
    re := regexp.MustCompile("b+")
    up := re.ReplaceAllStringFunc(te, strings.ToUpper)

    fmt.Println(up)   // aBa aBBa aBBBa
}
```

Regular expressions

Syntax for repetition and character classes:

- `.` is any character
- `.*` is zero or more
- `.+` is one or more
- `.?` is zero or one (prefer one)
- `a{n}` is *n* repetitions of the letter “a”
- `a{n,m}` is *n* to *m* repetitions of the letter “a”
- `[a-z]` is a **character class** (here letters a-z)
- `[^a-z]` is an *negated* class (here anything *except* a-z)

Regular expressions

Syntax for location:

- `xy` is “x” followed by “y” (a [sub]string!)
- `x|y` is either “x” *or* “y”
- `^x` is “x” at the beginning
- `x$` is “x” at the end
- `^x$` is “x” by itself (it’s the whole string)
- `\b` is a word boundary
- `\bx\b` is the word “x” by itself (inside the string)
- `(x)` is a **capture group**

Regular expressions

Some built-in character classes:

- `\d` is a decimal digit
- `\w` is a *word* character (`[0-9A-Za-z_]`)
- `\s` is *whitespace*
- `[[:alpha:]]` is any alphabetic character
- `[[:alnum:]]` is any alphanumeric character
- `[[:punct:]]` is any punctuation character
- `[[:print:]]` is any printable character
- `[[:xdigit:]]` is any hexadecimal character

See https://golang.org/pkg/regexp/syntax/

UUID validation

A UUID has the form `072665ee-a034-4cc3-a2e8-9f1822c4ebbb`

So we can try to validate one by matching a regular expression

A very simple (and not quite correct) example:

```
uure := `[a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}`
ufmt := regexp.MustCompile(uure)

if !ufmt.MatchString(ustr) {
    return fmt.Errorf("%s is not a UUID", ustr)
}
```

UUID validation

What was wrong:

- hex characters could be in upper case
- RFC 4122 has specific requirements for certain characters

The format is really `xxxxxxxx-xxxx-Vxxx-Wxxx-xxxxxxxxxxxx`

where `V` is a version (1–5)

and `W` is a format marker (`10bb` — one of `8`, `9`, `a`, `b`)

UUID validation

```go
var uure = `[[:xdigit:]]{8}-[[:xdigit:]]{4}-`+
           `[1-5][[:xdigit:]]{3}-[89abAB][[:xdigit:]]{3}-[[:xdigit:]]{12}`
var ufmt = regexp.MustCompile(uure)

var test = []string{
    "072665ee-a034-4cc3-a2e8-9f1822c4ebbb",
    "072665ee-a034-6cc3-a2e8-9f1822c4ebbb", // wrong version
    "072665ee-a034-4cc3-72e8-9f1822c4ebbb", // wrong format
}

func main() {
    for _, t := range test {
        if !uu.MatchString(t) {
            fmt.Println(t, "fails")
        }
    }
}
```

Search and replace with capture

```go
var phre = `\(([[:digit:]]{3})\) ([[:digit:]]{3})-([[:digit:]]{4})`
var pfmt = regexp.MustCompile(phre)

func main() {
    orig := "call me at (214) 514-3232 today"
    match := pfmt.FindStringSubmatch(orig)

    fmt.Printf("%q\n", match)

    if len(match) > 3 {
        fmt.Printf("+1 %s-%s-%s\n", match[1], match[2], match[3])
    }
}

// ["(214) 514-3232" "214" "514" "3232"]   // match [submatch ...]
// +1 214-514-3232
```

Search and replace with capture

```go
var phre = `\(([[:digit:]]{3})\) ([[:digit:]]{3})-([[:digit:]]{4})`
var pfmt = regexp.MustCompile(phre)

func main() {
    orig := "call me at (214) 514-3232 today"
    match := pfmt.FindStringSubmatch(orig)

    fmt.Printf("%q\n", match)

    intl := pfmt.ReplaceAllString(orig, "+1 ${1}-${2}-${3}")

    fmt.Println(intl)
}

// ["(214) 514-3232" "214" "514" "3232"]
// call me at +1 214-514-3232 today
```

URL validation with capture

```go
var uure = `^(http|https)://([a-zA-Z0-9\-\.]+\.[a-zA-Z]{2,4})`+
           `(?::([0-9]+))?\/?([a-zA-Z0-9\-\._\?\,\'\/\\\+&amp;%\$#\=~]*)$`
var ufmt = regexp.MustCompile(uure)

var test = []string{
    "http://matt.com/hello",
    "http://matt.com:8080/hello/",
    "http://matt.com:8080/hello?a=1&b=2",
    "http://matt.com:8080/hello?a=1&b=2&c=3",
}

func main() {
    for i, t := range test {
        match := u.FindStringSubmatch(t)
        fmt.Printf("%d: %q\n", i, match)
    }
}
```

URL validation with capture

```go
var uure = `^(http|https)://([a-zA-Z0-9\-\.]+\.[a-zA-Z]{2,4})`+
           `(?::([0-9]+))?\/?([a-zA-Z0-9\-\._\?\,\'\/\\\+&amp;%\$#\=~]*)$`
var ufmt = regexp.MustCompile(uure)
var vars = regexp.MustCompile(`(\w+)=(\w+)`)
var test = []string{
    "http://matt.com:8080/hello?a=1&b=2",
    "http://matt.com:8080/hello?a=1&b=2&c=3",
}

func main() {
    for i, t := range test {
        match := u.FindStringSubmatch(t)
        fmt.Printf("%d: %q\n", i, match)
        if len(match) > 4 && strings.Contains(match[4], "?") {
            fmt.Printf("   %q\n", vars.FindAllStringSubmatch(match[4], -1))
        }
    }
}
```