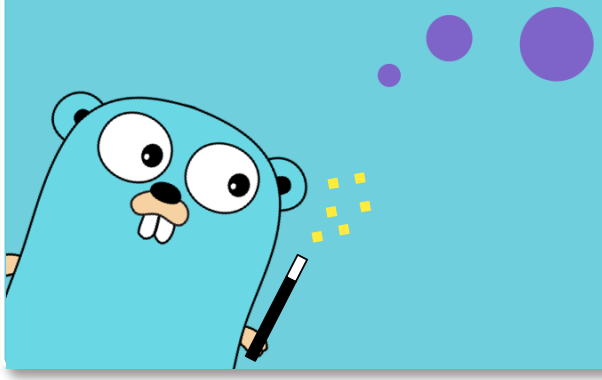
Rune Magic in Go

February 2019 Golang Meetup
Amsterdam

nulab backlog by nulab cacoo by nulab typetalk by nulab



Gabriele Vaccari
Software Engineer @ Nulab
(Go, Java, Elixir and some frontend)

A little terminology

character - abstract and ethereal entity (like God) 

code point - unique non-negative integer value that identifies a character, written U+<hex>. Example: U+0041

encoding - set of rules for representing a code point in memory, i.e. as bytes. Example: UTF-8

UTF-8

— UTF-8 can encode all Unicode code points

How it works:

A chinese character: 汉
it's unicode value: U+6C49
convert 6C49 to binary: 01101100 01001001 (two-bytes code point)

the code point is two bytes long

encoding patterns

1st Byte	2nd Byte	3rd Byte	4th Byte	Number of Free Bits	Max Unicode Val
0xxxxxxx				7	007F hex
110xxxxx	10xxxxxx			(5+6)=11	07FF hex
1110xxxx	10xxxxxx	10xxxxxx		(4+6+6)=16	FFFF hex
11110xxx	10xxxxxx	10xxxxxx	10xxxxxx	(3+6+6+6)=21	10FFFF hex

we need 16 free bits to encode 汉

UTF-8 pattern: 1110xxxx 10xxxxxx 10xxxxxx
Encoded character: 1110**0110** 10**110001** 10**001001**

Strings in Go

— a string is just a slice of bytes (and it's immutable)

— don't confuse string values (bytes) and string literals (UTF-8 bytes)!

— Go source code - and string literals - is always encoded in UTF-8

```
s := "Nulab" // s is assigned a string literal

fmt.Println(reflect.TypeOf(s[0])) // uint8

fmt.Printf("%x\n", s[0]) // 4e
fmt.Printf("%d\n", s[0]) // 78
fmt.Printf("%c\n", s[0]) // N


bytes := []byte{0x4e, 0x75, 0x6c, 0x61, 0x62}

fmt.Printf("%s\n", bytes) // Nulab
```

Runes in Go

— rune is Go's name for "code point", and is a built-in type

— rune is an alias for int32 (4 bytes), because that's the maximum length of an UTF-8-encoded character.

—⚠ when you range over a string, you get runes!

```
var r rune = '乾'                // U+4e7e

fmt.Println(reflect.TypeOf(r))  // int32

fmt.Printf("%x\n", r)           // 4e7e

var wide rune = 0x1f76a         //

fmt.Println(len(string(wide)))  // 4

        var overflow rune = 0x1f76a0000 // constant 8445886464 overflows rune
```

Practical Use Case

— we want to store UTF-8 strings into a Postgres varchar(N) field.

— we want to truncate strings with len(s) > N and append the elision suffix `...` (three dots)

— how to do that without corrupting the content of the string?

```
func validateObject(obj *store.NotificationObject) error {
if obj == nil {
return errors.New("you broke it!")
}
// Cut free text fields which are too long
obj.Name = elide(obj.Name)
obj.Content = elide(obj.Content)
return nil
}
```

Bug-prone Approaches

— naive substring field[:max] ✖

```
func NaiveSubstring() {
const max = 5

a := "截断错误"
// UTF-8
// e6 88 aa (截) e6 96 ad (断) e9 94 99 (错) e8 af af (误)
        //                   ^ max
b := a[:max]
fmt.Printf("%s", b) // 截<EFBFBD> (ouch!)
}
```

— iterate over len(field) ✖

still not ideal, because Postgres length function counts *characters!*

Go: `len("怎么算长度") // 15`

Postgres: `LENGTH('怎么算长度') // 5`

Solution

— Golang unicode/utf8 library 

```
const (
ThreeDotsElision  = "..."
        FreeTextMaxLength = VarcharLimit - 3 // make room for three dots
)

func Elide(field string) string {

if len(field) > FreeTextMaxLength {

                // Convert to byte array since it's required by the utf8 func
                // Also it's always true that len(bytes) >= len(chars)
bytes := []byte(field)

for len(bytes) > FreeTextMaxLength {
_, size := utf8.DecodeLastRune(bytes)
bytes = bytes[:len(bytes)-size]
}
return string(bytes) + ThreeDotsElision
}
return field
}
```

Thank you!

Resources:

https://github.com/vibridi/golang-meetup-02-19

https://blog.golang.org/strings

https://unicodebook.readthedocs.io/

https://www.joelonsoftware.com/

nulab backlog by nulab cacoo by nulab typetalk by nulab