用golang.org/x/text实现国际化和本地化

Marcel van Lohuizen
Google, Go team

I18n and L10n for Go using x/text

Overview

- golang.org/x/text subrepository
- What is it for?
- Current status
- Examples
- Conclusion

概览

- golang.org/x/text 子代码库
- 用途?
- 现状
- 例子
- 结论

I18n and L10n

- Searching and Sorting
- Upper, lower, title case
- Bi-directional text
- Injecting translated text
- Formatting of numbers, currency, date, time
- Unit conversion

国际化与本地化

- 搜索和排序
- 大小写和标题大小写
- 双向文本
- 注入翻译文本
- 数字, 货币, 日期时间格式
- 单位转换

golang.org/x/text 现状

语言标签
- language
  - display

字符串等式
- collate
- search
- secure
  - precis

文本处理
- cases
- encoding
  - ...
- runes
- *segment*
- transform
- unicode
  - *bidi*
  - cldr
  - norm
  - rangetable
- width

格式化
- currency
- *date*
- message
- *number*
- *measure*
  - *area*
  - *length*
  - *...*
- *feature*
  - *gender*
  - *plural*

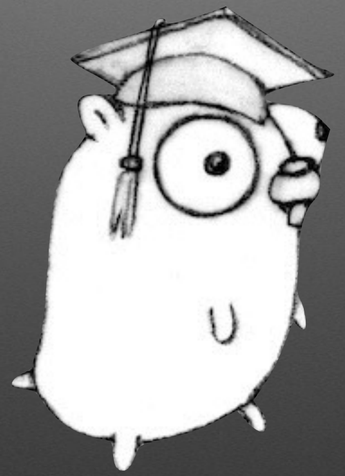
Go's Requirements

- Streaming
- Statically-linked binaries
- Multiple languages served simultaneously
- Performance
- Simple API

Go语言的要求

- 支持文本流 (io.Reader, io.Writer)
- 静态链接库
- 同时服务多种语言
- 性能
- 简单的API

Go中的Unicode



Unicode Go Refresher

Go uses UTF-8

Go natively handles UTF-8:

```
const beijing = "北京市"
for index, runeValue := range beijing {
    fmt.Printf("%#U 从第%d字节开始\n", runeValue, index)
}
```

Output:

```
U+5317 '北' 从第0字节开始
U+4EAC '京' 从第3字节开始
U+5E02 '市' 从第6字节开始
```

Go使用UTF-8

Go语言原生支持UTF-8:

输出:

String Model

- Always UTF-8
- Same model for source code as for text handling!
- No random access
- No meta data (except for byte length) or string “object”
- Strings not in canonical form

字符串模型总结

- 始终使用UTF-8
- 对原代码使用同样的编码处理方式
- 不支持随机访问
- 不提供元数据（除字节长度）或者字符串对象
- 并不要求字符串必须是归一化后的

Sequential nature of text

文本的序列本质

```
const flags = "🇲🇨🇳🇱"        // 国家代码 "mc" + "nl"
fmt.Println(flags[4:])
```

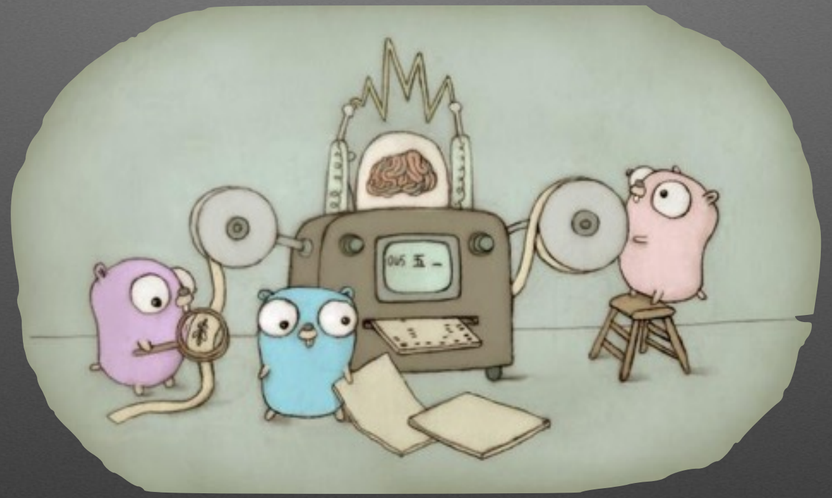
(continued)

- Text processing is inherently sequential, even for UTF-32
- Multi-rune characters: “e + ´ = é”
- Segmentation
- Casing

文本的序列本质

- 文本处理本质上是顺序化的，即使对UTF-32的多字节字符
- 多字节字符 (multiple runes): “e + ´ = é”
- 分词
- 大小写

转换文本



Transforming Text

Transformer 接口

```
type Transformer interface {
    Transform(dst, src []byte, atEOF bool) (nDst, nSrc int, err error)
    Reset()
}
```

Using Transformers

- A transform is typically used with one of the helpers functions.
- Most packages provide convenience wrappers

使用 Transformers

通常使用transform包提供的辅助函数：

```
encoder := simplifiedchinese.GBK.NewEncoder()
s, _, _ := transform.String(encoder, "你好")
```

同时大部分软件包提供了方便的封装

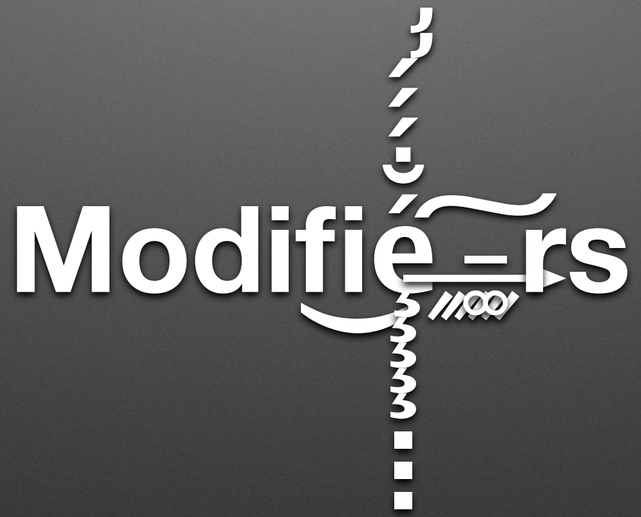
```
s := encoder.String("你好")

w := norm.NFC.Writer(w)
```

Normalization

标准化

```
norm.NFC.Writer(w)    // 以NFC格式向w写入文本流
```



x/text/unicode/norm包提供支持文本流并且安全的O(n) Unicode标准化算法

x/text/unicode/norm implements a stream-safe and secure O(n) normalization algorithm

Package cases

cases包

标题大小写：

```
toTitle := cases.Title(language.Dutch)

fmt.Println(toTitle.String("'n ijsberg"))
```

输出:

```
'n IJsberg
```

Languages may require different casing algorithms!

不同的语言可能需要不同的大小写算法

Transformers

- 实现了Transformer接口的x/text包：
  - cases
  - encoding/...
  - runes
  - transform
  - width
  - secure/precis
  - unicode/norm
  - unicode/bidi

搜索与排序

Searching and Sorting

Multilingual Search and Sort

- Accented characters: e < é < f
- Multi-letter characters: "ch" in Spanish
- Equivalences:
  å ⇔ aa in Danish
  ß ⇔ ss in German
- Reordering: Z < Å in Danish
- Compatibility equivalence:
  K (U+004B) ⇔ K (U+212A)
- Reverse sorting of accents in Canadian French

多语言搜索与排序

- 带音调的字符：e < é < f
- 多字母的字符：”ch"（西班牙语）
- 等价字符：å ⇔ aa（丹麦语）， ß ⇔ ss（德语）
- 重排序：Z < Å（丹麦语）
- 兼容性等价：K (U+004B) ⇔ K (U+212A)
- 反序排列加拿大法语中带音调的字符

Search and Replace

搜索与替换

- 用 bytes.Replace 把 "a cafe" 替换成 "many cafes"
  1. “We went to a cafe.”
  2. “We went to a café.”
  3. “We went to a cafe/u0301.”
- 第三个例句的结果:

“We went to many cafes/u0301.” ＝NFC ⇒

“We went to many cafeś.”

Simple byte-oriented search and replace will not work!

简单的单字节搜索替换并不适用！

search Example

x/text/search 例子

```
m := search.New(language.Danish, search.IgnoreCase, search.IgnoreDiacritics)

start, end := m.IndexString(text, s)

match := s[start:end]
```

SEARCH	TEXT	MATCH
aarhus	Århus a\u0303\u031b	Århus
a		a\u0303\u031b
a\u031b\u0303		a\u0303\u031b

collate Example

x/text/collate 例子

Output:

[上海市 北京市 广州市]

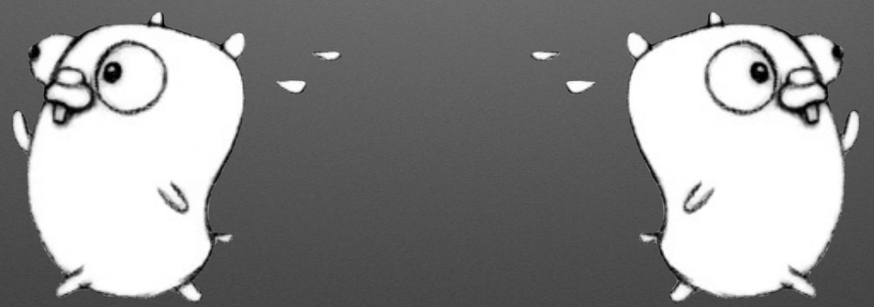
[北京市 广州市 上海市]

[上海市 广州市 北京市]

```
import (
    "fmt"
    "golang.org/x/text/collate"
    "golang.org/x/text/language"
)

func main() {
    a := []string{"北京市", "上海市", "广州市"}
    for _, tag := range []string{"en","zh", "zh-u-co-stroke"} {
        collate.New(language.Make(tag)).SortStrings(a)
        fmt.Println(a)
    }
}
```

文本分割



Segmentation

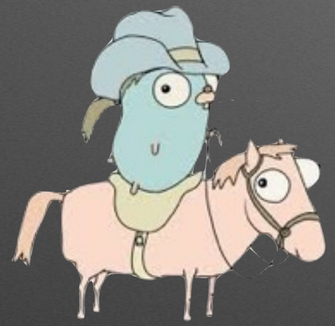
Segmentation Support

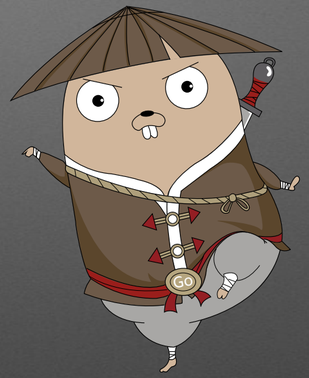
- Planned:
  - API for segmentation
  - Supported by Unicode:
    - word, line, sentence, paragraph
- Not planned:
  - Language-specific segmentation
  - Community support welcome

对文本分割的支持

- 计划中的：
  - 提供分割功能的API
  - Unicode所支持的:
    - 单词（以空格分隔的），行，句子，段落
- 尚未计划的：
  - 针对特定语言的文本分割
  - 欢迎来自社区的帮助

语言标签





Language Tags

Language Tag Examples

语言标签例子

<lang> [-<script>] [-<region>] [-<variant>]* [-<extension>]*

zh	中文 (默认是简体中文)
zh-Hant	繁体中文（台湾）
zh-HK	繁体中文（香港）
zh-Latn-pinyin	中文拼音
zh-HK-u-co-pinyin	中文，拼音顺序

Matching is Non-Trivial

- Swiss German speakers usually understand German `gsw` ⇒ `de`
- The converse is not often true! `de` ⇏ `gsw`
- `cmn` is Mandarin Chinese, `zh` is more commonly used
- `hr` matches `sr-Latn`

The Matcher in x/text/language solves this problem

语言匹配并不简单

- 说瑞士德语的人通常能听懂德语 `gsw` ⇒ `de`
- 但反过来就不是！ `de` ⇏ `gsw`
- `cmn`是普通话，`zh`更常用
- `hr` 匹配 `sr-Latn`

在x/text/language里的matcher能解决这个问题

Language Matching in Go

Go中的语言匹配

```
import (
    “http”,
    ”golang.org/x/text/language”
)

// Languages supported by your application
var matcher = language.NewMatcher([]language.Tag{
    language.SimplifiedChinese,     // zh-Hans
    language.AmericanEnglish,       // en-US
})

func handle(w http.ResponseWriter, r *http.Request) {
    prefs, _, _ := language.ParseAcceptLanguage(r.Header.Get(“Accept-Language”))

    tag, _, _ := matcher.Match(prefs…)
    // use tag; it includes carried over user preference
}
```

Language Matching Recap

- Find best supported language for list of user-preferred languages
- Use matched tag to select language-specific resources
  - translations
  - sort order
  - case operations
- Resulting tag has carried over user settings

语言匹配总结

- 找到用户偏爱的语言中支持最好的一种
- 使用匹配到的标签选择语言相关的资源
  - 翻译
  - 排序
  - 大小写处理
- 结果标签中携带有用户的设置

注入翻译文本

Translation Insertion

Translating Text

- Mark text within your code To Be Translated
- Extract the text from your code
- Send to translators
- Insert translated messages back into your code

翻译文本

- 在代码中把文本标记为“需要翻译”
- 将这些文本从代码中提取出来
- 发送给翻译人员
- 将翻译之后的文本插入原来的代码中

Mark Text “To Be Translated”

将文本标记为“需要翻译”

之前：

```
import ”fmt”

// Report that person visited a city.
fmt.Printf(“%[1]s went to %[2]s.”, person, city)
```

之后：

```
import ”golang.org/x/text/message”

p := message.NewPrinter(userLang)

// Report that person visited a city.
p.Printf(“%[1]s went to %[2]s.”, person, city)
```

Extract and send for translation

提取并发送待翻译的文本

```
{
    Description: "Report that person visited a city.",
    Original:    "{person} went to {city}.",
    Key:          "%s went to %s.",
}
```

Insert Translations in Code

在代码中插入翻译结果

```
import ”golang.org/x/text/message”

message.SetString(language.Dutch,
                  "%s went to %s",
                  "%s is in %s geweest.”)

message.SetString(language.SimplifiedChinese,
                  "%s went to %s",
                  "%s去了%s。")
```

Planned extensions

- Go tooling: automate extraction and insertion
- Planned:
  - number formatting
  - selection based on plurals, gender, etc.
- golang.org/design/12750-localization

规划

- Go工具：自动抽取及插入
- 计划中的：
  - 格式化数字
  - 基于单复数、性别等信息的选择
- golang.org/design/12750-localization

Conclusion

- Human languages are hard to deal with
- Let x/text can simplify it for you

结语

- 人类语言好难对付
- 让x/text帮你化简吧

Community feedback

- East-Asian Width
- gofmt and East-Asian characters
- Vertical support

社区反馈

- 东亚语言（全角）
- 东亚字符的格式

Q & A

谢谢

Marcel van Lohuizen

- 参考
  - godoc.org/golang.org/x/text
  - blog.golang.org/matchlang
  - blog.golang.org/normalization
  - blog.golang.org/strings
  - golang.org/issue/12750