

36 | unicode与字符编码

2018-11-02 郝林

Go语言核心36讲

[进入课程 >](#)



讲述：黄洲君

时长 16:45 大小 5.76M



到目前为止，我们已经一起陆陆续续地学完了 Go 语言中那些最重要也最有特色的概念、语法和编程方式。我对于它们非常喜爱，简直可以用如数家珍来形容了。

在开始今天的内容之前，我先来做一个简单的总结。

Go 语言经典知识总结

基于混合线程的并发编程模型自然不必多说。

在数据类型方面有：

基于底层数组的切片；

用来传递数据的通道；
作为一等类型的函数；
可实现面向对象的结构体；
能无侵入实现的接口等。

在语法方面有：

异步编程神器go语句；
函数的最后关卡defer语句；
可做类型判断的switch语句；
多通道操作利器select语句；
非常有特色的异常处理函数panic和recover。

除了这些，我们还一起讨论了**测试 Go 程序**的主要方式。这涉及了 Go 语言自带的程序测试套件，相关的概念和工具包括：

独立的测试源码文件；
三种功用不同的测试函数；
专用的testing代码包；
功能强大的go test命令。

另外，就在前不久，我还为你深入讲解了 Go 语言提供的那些**同步工具**。它们也是 Go 语言并发编程工具箱中不可或缺的一部分。这包括了：

经典的互斥锁；
读写锁；
条件变量；
原子操作。

以及**Go 语言特有的一些数据类型**，即：

单次执行小助手`sync.Once`；

临时对象池`sync.Pool`；

帮助我们实现多 goroutine 协作流程的`sync.WaitGroup`、`context.Context`；

一种高效的并发安全字典`sync.Map`。

毫不夸张地说，如果你真正地掌握了上述这些知识，那么就已经获得了 Go 语言编程的精髓。

在这之后，你再去研读 Go 语言标准库和那些优秀第三方库中的代码的时候，就一定会事半功倍。同时，在使用 Go 语言编写软件的时候，你肯定也会如鱼得水、游刃有余的。

我用了大量的篇幅讲解了 Go 语言中最核心的知识点，真心希望你已经搞懂了这些内容。

在后面的日子里，我会与你一起去探究 Go 语言标准库中最常用的那些代码包，弄清它们的用法、了解它们的机理。当然了，我还会顺便讲一讲那些必备的周边知识。

前导内容 1：Go 语言字符编码基础

首先，让我们来关注字符编码方面的问题。这应该是在计算机软件领域中非常基础的一个问题了。

我在前面说过，Go 语言中的标识符可以包含“任何 Unicode 编码可以表示的字母字符”。我还说过，虽然我们可以直接把一个整数值转换为一个`string`类型的值。

但是，被转换的整数值应该可以代表一个有效的 Unicode 代码点，否则转换的结果就将是“?”，即：一个仅由高亮的问号组成的字符串值。

另外，当一个`string`类型的值被转换为`[]rune`类型值的时候，其中的字符串会被拆分成一个一个的 Unicode 字符。

显然，Go 语言采用的字符编码方案从属于 Unicode 编码规范。更确切地说，Go 语言的代码正是由 Unicode 字符组成的。Go 语言的所有源代码，都必须按照 Unicode 编码规范中的 UTF-8 编码格式进行编码。

换句话说，Go 语言的源码文件必须使用 UTF-8 编码格式进行存储。如果源码文件中出现了非 UTF-8 编码的字符，那么在构建、安装以及运行的时候，go 命令就会报告错误 “illegal UTF-8 encoding”。

在这里，我们首先要对 Unicode 编码规范有所了解。不过，在讲述它之前，我先来简要地介绍一下 ASCII 编码。

前导内容 2：ASCII 编码

ASCII 是英文 “American Standard Code for Information Interchange” 的缩写，中文译为美国信息交换标准代码。它是由美国国家标准学会（ANSI）制定的单字节字符编码方案，可用于基于文本的数据交换。

它最初是美国的国家标准，后又被国际标准化组织（ISO）定为国际标准，称为 ISO 646 标准，并适用于所有的拉丁文字字母。

ASCII 编码方案使用单个字节（byte）的二进制数来编码一个字符。标准的 ASCII 编码用一个字节的最高比特（bit）位作为奇偶校验位，而扩展的 ASCII 编码则将此位也用于表示字符。ASCII 编码支持的可打印字符和控制字符的集合也被叫做 ASCII 编码集。

我们所说的 Unicode 编码规范，实际上是另一个更加通用的、针对书面字符和文本的字符编码标准。它为世界上现存的所有自然语言中的每一个字符，都设定了一个唯一的二进制编码。

它定义了不同自然语言的文本数据在国际间交换的统一方式，并为全球化软件创建了一个重要的基础。

Unicode 编码规范以 ASCII 编码集为出发点，并突破了 ASCII 只能对拉丁字母进行编码的限制。它不但提供了可以对世界上超过百万的字符进行编码的能力，还支持所有已知的转义序列和控制代码。

我们都知道，在计算机系统的内部，抽象的字符会被编码为整数。这些整数的范围被称为代码空间。在代码空间之内，每一个特定的整数都被称为一个代码点。

一个受支持的抽象字符会被映射并分配给某个特定的代码点，反过来讲，一个代码点总是可以被看成一个被编码的字符。

Unicode 编码规范通常使用十六进制表示法来表示 Unicode 代码点的整数值，并使用 “U+” 作为前缀。比如，英文字母字符 “a” 的 Unicode 代码点是 U+0061。在 Unicode 编码规范中，一个字符能且只能由与它对应的那个代码点表示。

Unicode 编码规范现在的最新版本是 11.0，并会于 2019 年 3 月发布 12.0 版本。而 Go 语言从 1.10 版本开始，已经对 Unicode 的 10.0 版本提供了全面的支持。对于绝大多数的应用场景来说，这已经完全够用了。

Unicode 编码规范提供了三种不同的编码格式，即：UTF-8、UTF-16 和 UTF-32。其中的 UTF 是 UCS Transformation Format 的缩写。而 UCS 又是 Universal Character Set 的缩写，但也可以代表 Unicode Character Set。所以，UTF 也可以被翻译为 Unicode 转换格式。它代表的是字符与字节序列之间的转换方式。

在这几种编码格式的名称中，“-” 右边的整数的含义是，以多少个比特位作为一个编码单元。以 UTF-8 为例，它会以 8 个比特，也就是一个字节，作为一个编码单元。并且，它与标准的 ASCII 编码是完全兼容的。也就是说，在 [0x00, 0x7F] 的范围内，这两种编码表示的字符都是相同的。这也是 UTF-8 编码格式的一个巨大优势。

UTF-8 是一种可变宽的编码方案。换句话说，它会用一个或多个字节的二进制数来表示某个字符，最多使用四个字节。比如，对于一个英文字母，它仅用一个字节的二进制数就可以表示，而对于一个中文字符，它需要使用三个字节才能够表示。不论怎样，一个受支持的字符总是可以由 UTF-8 编码为一个字节序列。以下会简称后者为 UTF-8 编码值。

现在，在你初步地了解了这些知识之后，请认真地思考并回答下面的问题。别担心，我会在后面进一步阐述 Unicode、UTF-8 以及 Go 语言对它们的运用。

问题：一个 `string` 类型的值在底层是怎样被表达的？

典型回答 是在底层，一个 `string` 类型的值是由一系列相对应的 Unicode 代码点的 UTF-8 编码值来表达的。

问题解析

在 Go 语言中，一个 `string` 类型的值既可以被拆分为一个包含多个字符的序列，也可以被拆分为一个包含多个字节的序列。

前者可以由一个以 `rune` 为元素类型的切片来表示，而后者则可以由一个以 `byte` 为元素类型的切片代表。

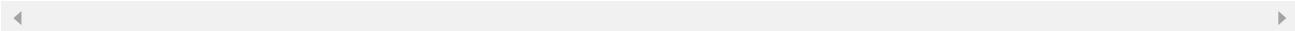
`rune` 是 Go 语言特有的一个基本数据类型，它的一个值就代表一个字符，即：一个 Unicode 字符。

比如，'G'、'o'、'爱'、'好'、'者' 代表的就都是一个 Unicode 字符。

我们已经知道，UTF-8 编码方案会把一个 Unicode 字符编码为一个长度在 [1, 4] 范围内的字节序列。所以，一个 `rune` 类型的值也可以由一个或多个字节来代表。

 复制代码

```
1 type rune = int32
```



根据 `rune` 类型的声明可知，它实际上就是 `int32` 类型的一个别名类型。也就是说，一个 `rune` 类型的值会由四个字节宽度的空间来存储。它的存储空间总是能够存下一个 UTF-8 编码值。

一个 `rune` 类型的值在底层其实就是一个 UTF-8 编码值。前者是（便于我们人类理解的）外部展现，后者是（便于计算机系统理解的）内在表达。

请看下面的代码：

 复制代码

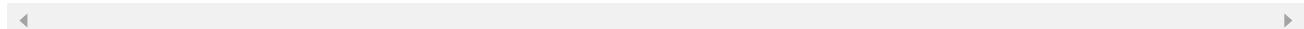
```
1 str := "Go 爱好者"
2 fmt.Printf("The string: %q\n", str)
3 fmt.Printf("  => runes(char): %q\n", []rune(str))
4 fmt.Printf("  => runes(hex): %x\n", []rune(str))
5 fmt.Printf("  => bytes(hex): [% x]\n", []byte(str))
```



字符串值 "Go 爱好者" 如果被转换为 `[]rune` 类型的值的话，其中的每一个字符（不论是英文字符还是中文字符）就都会独立成为一个 `rune` 类型的元素值。因此，这段代码打印出的第二行内容就会如下所示：

 复制代码

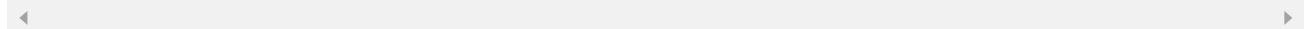
```
1 => runes(char): ['G' 'o' '爱' '好' '者']
```



又由于，每个rune类型的值在底层都是由一个UTF-8编码值来表达的，所以我们可以换一种方式来展现这个字符序列：

 复制代码

```
1 => runes(hex): [47 6f 7231 597d 8005]
```



可以看到，五个十六进制数与五个字符相对应。很明显，前两个十六进制数`47`和`6f`代表的整数都比较小，它们分别表示字符'G'和'o'。

因为它们都是英文字符，所以对应的UTF-8编码值用一个字节表达就足够了。一个字节的编码值被转换为整数之后，不会大到哪里去。

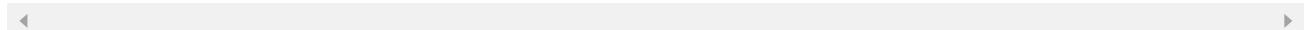
而后三个十六进制数`7231`、`597d`和`8005`都相对较大，它们分别表示中文字符'爱'、'好'和'者'。

这些中文字符对应的UTF-8编码值，都需要使用三个字节来表达。所以，这三个数就是把对应的三个字节的编码值，转换为整数后得到的结果。

我们还可以进一步地拆分，把每个字符的UTF-8编码值都拆成相应的字节序列。上述代码中的第五行就是这么做的。它会得到如下的输出：

 复制代码

```
1 => bytes(hex): [47 6f e7 88 b1 e5 a5 bd e8 80 85]
```



这里得到的字节切片比前面的字符切片明显长了很多。这正是因为一个中文字符的UTF-8编码值需要用三个字节来表达。

这个字节切片的前两个元素值与字符切片的前两个元素值是一致的，而在这之后，前者的每三个元素值才对应字符切片中的一个元素值。

注意，对于一个多字节的 UTF-8 编码值来说，我们可以把它当做一个整体转换为单一的整数，也可以先把它拆成字节序列，再把每个字节分别转换为一个整数，从而得到多个整数。

这两种表示法展现出来的内容往往很不一样。比如，对于中文字符'爱'来说，它的 UTF-8 编码值可以展现为单一的整数7231，也可以展现为三个整数，即：e7、88和b1。

string 值

Go爱好者

[]rune 值

G

o

爱

好

者

0x47

0x6f

0x7231

0x597d

0x8005

[]byte 值

[47]

[6f]

[e7 88 b1]

[e5 a5 bd]

[e8 80 85]

(字符串值的底层表示)

总之，一个string类型的值会由若干个 Unicode 字符组成，每个 Unicode 字符都可以由一个rune类型的值来承载。

这些字符在底层都会被转换为 UTF-8 编码值，而这些 UTF-8 编码值又会以字节序列的形式表达和存储。因此，一个string类型的值在底层就是一个能够表达若干个 UTF-8 编码值的字节序列。

知识扩展

问题 1：使用带有range子句的for语句遍历字符串值的时候应该注意什么？

带有 range 子句的 for 语句会先把被遍历的字符串值拆成一个字节序列，然后再试图找出这个字节序列中包含的每一个 UTF-8 编码值，或者说每一个 Unicode 字符。

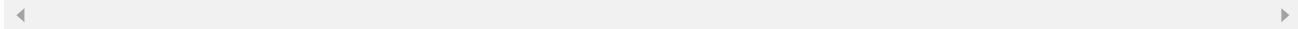
这样的 for 语句可以为两个迭代变量赋值。如果存在两个迭代变量，那么赋给第一个变量的值，就将会是当前字节序列中的某个 UTF-8 编码值的第一个字节所对应的那个索引值。

而赋给第二个变量的值，则是这个 UTF-8 编码值代表的那个 Unicode 字符，其类型会是 rune。

例如，有这么几行代码：

 复制代码

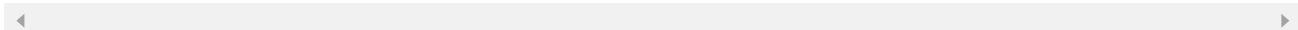
```
1 str := "Go 爱好者"
2 for i, c := range str {
3     fmt.Printf("%d: %q [%x]\n", i, c, []byte(string(c)))
4 }
```



这里被遍历的字符串值是"Go爱好者"。在每次迭代的时候，这段代码都会打印出两个迭代变量的值，以及第二个值的字节序列形式。完整的打印内容如下：

 复制代码

```
1 0: 'G' [47]
2 1: 'o' [6f]
3 2: '爱' [e7 88 b1]
4 5: '好' [e5 a5 bd]
5 8: '者' [e8 80 85]
```



第一行内容中的关键信息有 0、'G' 和 [47]。这是由于这个字符串值中的第一个 Unicode 字符是 'G'。该字符是一个单字节字符，并且由相应的字节序列中的第一个字节表达。这个字节的十六进制表示为 47。

第二行展示的内容与之类似，即：第二个 Unicode 字符是 'o'，由字节序列中的第二个字节表达，其十六进制表示为 6f。

再往下看，第三行展示的是'爱'，也是第三个 Unicode 字符。因为它是一个中文字符，所以由字节序列中的第三、四、五个字节共同表达，其十六进制表示也不再是单一的整数，而是`e7 88 b1`组成的序列。

下面要注意了，正是因为'爱'是由三个字节共同表达的，所以第四个 Unicode 字符'好'对应的索引值并不是3，而是2加3后得到的5。

这里的2代表的是'爱'对应的索引值，而3代表的则是'爱'对应的 UTF-8 编码值的宽度。对于这个字符串值中的最后一个字符'者'来说也是类似的，因此，它对应的索引值是8。

由此可以看出，这样的`for`语句可以逐一地迭代出字符串值里的每个 Unicode 字符。但是，相邻的 Unicode 字符的索引值并不一定是连续的。这取决于前一个 Unicode 字符是否为单字节字符。

正因为如此，如果我们想得到其中某个 Unicode 字符对应的 UTF-8 编码值的宽度，就可以用下一个字符的索引值减去当前字符的索引值。

初学者可能会对`for`语句的这种行为感到困惑，因为它给予两个迭代变量的值看起来并不总是对应的。不过，一旦我们了解了它的内在机制就会拨云见日、豁然开朗。

总结

我们今天把目光聚焦在了 Unicode 编码规范、UTF-8 编码格式，以及 Go 语言对字符串和字符的相关处理方式上。

Go 语言的代码是由 Unicode 字符组成的，它们都必须由 Unicode 编码规范中的 UTF-8 编码格式进行编码并存储，否则就会导致 go 命令的报错。

Unicode 编码规范中的编码格式定义的是：字符与字节序列之间的转换方式。其中的 UTF-8 是一种可变宽的编码方案。

它会用一个或多个字节的二进制数来表示某个字符，最多使用四个字节。一个受支持的字符，总是可以由 UTF-8 编码为一个字节序列，后者也可以被称为 UTF-8 编码值。

Go 语言中的一个 `string` 类型值会由若干个 Unicode 字符组成，每个 Unicode 字符都可以由一个 `rune` 类型的值来承载。

这些字符在底层都会被转换为 UTF-8 编码值，而这些 UTF-8 编码值又会以字节序列的形式表达和存储。因此，一个 `string` 类型的值在底层就是一个能够表达若干个 UTF-8 编码值的字节序列。

初学者可能会对带有 `range` 子句的 `for` 语句遍历字符串值的行为感到困惑，因为它给予两个迭代变量的值看起来并不总是对应的。但事实并非如此。

这样的 `for` 语句会先把被遍历的字符串值拆成一个字节序列，然后再试图找出这个字节序列中包含的每一个 UTF-8 编码值，或者说每一个 Unicode 字符。

相邻的 Unicode 字符的索引值并不一定是连续的。这取决于前一个 Unicode 字符是否为单字节字符。一旦我们清楚了这些内在机制就不会再困惑了。

对于 Go 语言来说，Unicode 编码规范和 UTF-8 编码格式算是基础之一了。我们应该了解到它们对 Go 语言的重要性。这对于正确理解 Go 语言中的相关数据类型以及日后的相关编程编写都会很有好处。

思考题

今天的思考题是：判断一个 Unicode 字符是否为单字节字符通常有几种方式？

[戳此查看 Go 语言专栏文章配套详细代码。](#)



GO语言核心36讲

3个月带你通关 GO 语言

郝林

《Go 并发编程实战》作者
GoHackers 技术社群发起人
前轻松筹大数据负责人



新版升级：点击「 请朋友读」，10位好友免费读，邀请订阅更有**现金奖励**。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 35 | 并发安全字典sync.Map (下)

下一篇 37 | strings包与字符串操作

精选留言 (12)

写留言



wade

2018-11-07

7

而后三个十六进制数7231、597d和8005都相对较大，它们分别表示中文字符'爱'、'好'和'者'。这些中文字符对应的 UTF-8 编码值，都需要使用三个字节来表达。所以，这三个数就是把对应的三个字节来表达。所以，这三个数就是把对应的三个字节的编码值，转换为整数后得到的结果。

...

展开 ▼



冰激凌的眼...

2018-11-05

7

src文件编码是utf8

string是utf8编码的mb , len(string)是字节的长度
string可以转化为[]rune , unicode码 , 32bit的数字 , 当字符看 , len([]rune)为字符长度
string可以转化为[]byte , utf8编码字节串 , len([]byte)和len(string)是一样的
for range的时候 , 迭代出首字节下标和rune , 首字符下标可能跳跃(视上一个字符编码长...
展开▼



冰激凌的眼...

2018-11-02

3

看rune大小
转成byte看长度
加个小尾巴,range看间隔
展开▼



薛WEI

2018-11-15

2

rune怎么翻译 ? 有道查的 : 神秘的记号。为什么这么命名这个类型 ? 有没有什么故事 ?



Gryllus

2018-11-02

2

终于追上了进度
展开▼

作者回复: 🧑



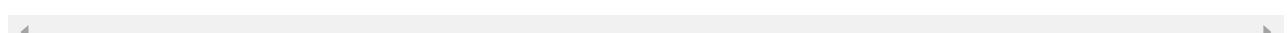
Andylee

2018-12-16

1

这篇文章把unicode和utf8区分的不是很清楚 , 正如上面有个网友说的rune切变16进制输出是字符的unicode的码点 , 而byte切片输出的才是utf8的编码
展开▼

作者回复: 这么说没错 , 不过rune在底层也是字节串。





jacke

2019-05-12



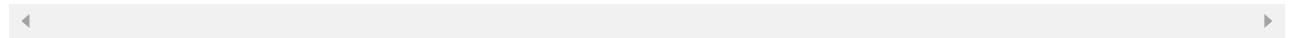
string 底层是[]byte数组，我的疑问是：例子里面看出来，string转化为tune的时候，tune里面保存的是utf-8的代码点数据，string转化为[]byte的时候保存的是utf-8代码点对应的字节序。

上面这些转化逻辑在哪里实现的？fmt.print里面？看了fmt.print找不到，string转为[]byte的实现函数stringtoslicebyte也没看到这部分逻辑

展开 ▼

作者回复：你要知道，string 类型的值本身就是由 UTF-8 编码的一个个字节组成的，同时也可以看做是由一个个 Unicode 字符组成的。这不是在转换的时候才去做的。

你既然已经找到了 stringtoslicebyte 函数，那就应该再去看看那个源码文件中的其他代码。可以从 rawstring 函数看起。等都看完了你就应该明白了。



Geek_1ed70...

2019-04-22



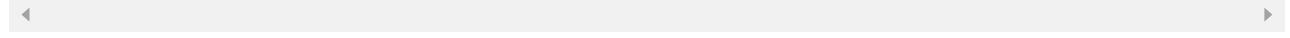
+ isrunesingle.go

```
```go
package show_rune_length
```

...

展开 ▼

作者回复：可以用 unicode/utf8 代码包中的 RuneCount 函数。



Geek\_1ed70...

2019-03-14



您是说一个汉字的rune值 在计算机底层会被转成utf-8编码来 给计算机读取是吗？

比如一个“严”字 unicode为20005(十进制), utf-8编码是11100100 10111000  
10100101(二进制),十进制就是14989477 , 我们平时打印只能看到 20005 它是什么时候转成14989477的啊

展开 ▼

作者回复: 存储的时候会以二进制的形式。另外如果你要看Unicode代码点。你这么转换比较混乱。你可以参看fmt包的文档，看看怎样才能把对应的进制值打印出来。



melody\_fu...

2019-03-06



有点小晕，想请问下 rune 类型在内存的表现形式是 unicde 编码值，还是utf-8 编码值，你所说的底层指的是？

展开 ▼

作者回复: rune的底层表达使用的是Unicode代码点。

但是，底层的存储用UTF-8编码。

表达和存储这两者的关系你能分清楚吧？



王小勃

2019-01-27



打卡：

1、len函数对于字符串，得到的是字节长度

2、utf-8 我以前看到的资料是 1-6个字节的可变长编码，go如果用rune，对于超过4个字节的utf-8字符怎么处理？

展开 ▼

作者回复: 目前utf-8是1~4个字节的。



ryon

2018-11-05



这章讲得还可以

展开 ▼