字符串类型

字符串就是一串固定长度的字符连接起来的字符序列。Go 的字符串是由单个字节连接起来的。Go 语言的字符串的字节使用 UTF-8 编码标识 Unicode 文本。

字符串是 UTF-8 字符的一个序列（当字符为 ASCII 码时则占用 1 个字节，其它字符根据需要占用 2-4 个字节）。UTF-8 是被广泛使用的编码格式，是文本文件的标准编码，其它包括 XML 和 JSON 在内，也都使用该编码。由于该编码对占用字节长度的不定性，Go 中的字符串也可能根据需要占用 1 至 4 个字节，这与其它语言如 C++、Java 或者 Python 不同（Java 始终使用 2 个字节）。Go 这样做的好处是不仅减少了内存和硬盘空间占用，同时也不用像其它语言那样需要对使用 UTF-8 字符集的文本进行编码和解码。

字符串是一种值类型，且值不可变，即创建某个文本后你无法再次修改这个文本的内容；更深入地讲，字符串是字节的定长数组。

Go语言字符串是一个任意**字节的常量序列**。[] byte类型字节数组

go语言字符串字面量

在Go语言中，字符串字面量使用**双引号** `""` 或者**反引号** `'` 来创建。双引号用来创建可解析的字符串，支持转义，但不能用来引用多行；反引号用来创建原生的字符串字面量，可能由多行组成，但不支持转义，并且可以包含除了反引号外其他所有字符。双引号创建可解析的字符串应用最广泛，反引号用来创建原生的字符串则多用于书写多行消息，HTML以及正则表达式。

Go 支持以下 2 种形式的字面值：

- 解释字符串：

  该类字符串使用双引号 `""` 括起来，其中的相关的转义字符将被替换，这些转义字符包括：

  - `\n`： 换行符
  - `\r`： 回车符
  - `\t`： tab 键
  - `\u` 或 `\U`：Unicode 字符
  - `\\`： 反斜杠自身
  - `\f` ： 换页
  - `\v` ： 垂直制表符
  - `\'` ： 单引号 (只用在 ''' 形式的rune符号面值中)
  - `\"` ： 双引号 (只用在 "..." 形式的字符串面值中)
  - `\a` ： 响铃
  - `\b` ： 退格
- 非解释字符串：

  该类字符串使用反引号 `'` 括起来，支持换行，例如：

  ```
  `This is a raw string \n` 中的 `\n\` 会被原样输出。
  ```

`string` 类型的零值为长度为零的字符串，即空字符串 `""`。

字符串切片截取：

一般的比较运算符（`==`、`!=`、`<`、`<=`、`>=`、`>`）通过在内存中按字节比较来实现字符串的对比。你可以通过函数 `len()` 来获取字符串所占的字节长度，例如：`len(str)`。

字符串的内容（纯字节）可以通过标准索引法来获取，在中括号 `[]` 内写入索引，索引从 0 开始计数：

- 字符串 str 的第 1 个字节：`str[0]`
- 第 i 个字节：`str[i - 1]`
- 最后 1 个字节：`str[len(str)-1]`

```
package main

import (
    "fmt"
)

func main() {
    var str string = "hello world"
    n := 3
    m := 5
    fmt.Println(str[n])   //获取字符串索引位置为n的原始字节  108
    fmt.Println(str[n:m]) //截取字符串索引位置为n到m-1的字符串  lo
    fmt.Println(str[n:])  //截取字符串索引位置为n到len(s)-1的字符串  lo world
    fmt.Println(str[:m])  //截取字符串索引位置为0到m-1的字符串  hello
}
```

需要注意的是，这种转换方案只对纯 ASCII 码的字符串有效。

注意事项 获取字符串中某个字节的地址的行为是非法的，例如：`&str[i]`。如果试图访问超出字符串索引范围的字节将会导致panic异常，`str[len(str)]`。

字符串可以用==和<进行比较；比较通过逐个字节比较完成的，因此比较的结果是字符串自然编码的顺序。

字符串的值是不可变的：一个字符串包含的字节序列永远不会被改变，当然我们也可以给一个字符串变量分配一个新字符串值。可以像下面这样将一个字符串追加到另一个字符串：

```
s := "left foot"
t := s
s += ", right foot"
```

这并不会导致原始的字符串值被改变，但是变量s将因为+=语句持有一个新的字符串值，但是t依然是包含原先的字符串值。

```
fmt.Println(s) // "left foot, right foot"
fmt.Println(t) // "left foot"
```

因为字符串是不可修改的，因此尝试修改字符串内部数据的操作也是被禁止的：

```
s[0] = 'L' // compile error: cannot assign to s[0]
```

不变性意味着如果两个字符串共享相同的底层数据的话也是安全的，这使得复制任何长度的字符串代价是低廉的。同样，一个字符串s和对应的子字符串切片s[7:]的操作也可以安全地共享相同的内存，因此字符串切片操作代价也是低廉的。在这两种情况下都没有必要分配新的内存。

字符串拼接符 `+`

两个字符串 `s1` 和 `s2` 可以通过 `s := s1 + s2` 拼接在一起。

`s2` 追加在 `s1` 尾部并生成一个新的字符串 `s`。

你可以通过以下方式来对代码中多行的字符串进行拼接：

```
package main

import "fmt"

func main() {
    s1 := "hello"
    s2 := "world"
    s := s1 + s2
    fmt.Printf("s: %v\n", s)

    str := "hello " +
        "baozi"
    fmt.Printf("str: %v\n", str)
}
```

由于编译器行尾自动补全分号的缘故，加号 `+` 必须放在第一行。

拼接的简写形式 `+=` 也可以用于字符串：

```
package main

import (
    "fmt"
)

func main() {
    s := "hel" + "lo,"
    s += "world!"
    fmt.Println(s) //输出 “hello, world!”
}
```

也可以使用fmt.Sprintf() 函数：

```
package main

import "fmt"

func main() {
    name := "包子"
    age := "18"

    msg := fmt.Sprintf("%s,%s", name, age)
    fmt.Printf("msg: %v\n", msg)
}
#结果
msg: 包子,18
```

在循环中使用加号 `+` 拼接字符串并不是最高效的做法，更好的办法是使用函数 `strings.Join()`，有没有更好地办法了？有！使用字节缓冲（`bytes.Buffer`）拼接更加给力。

strings.Join()示例：

```
package main

import (
    "fmt"
    "strings"
)

func main() {
    name := "包子"
    age := "18"

    msg := strings.Join([]string{name, age}, ",")
    fmt.Printf("msg: %v\n", msg)
}
#结果
msg: 包子,18
```

bytes.Buffer示例：

```
package main

import (
    "bytes"
    "fmt"
)

func main() {
    var bf bytes.Buffer
    bf.WriteString("包子")
    bf.WriteString(", ")
    bf.WriteString("18")
    fmt.Printf("bf.String(): %v\n", bf.String())
}
#结果
bf.String(): 包子, 18
```

Go语言字符串常用的方法

方法	说明
len()	获取字符串长度
+	拼接字符串
fmt.Sprintf()	拼接字符串
strings.Split()	分割字符串
strings.Contains()	判断是否包含
strings.HasPrefix(), strings.HasSuffix()	前缀后缀判断，是否已字符开头或结尾
strings.Index(),strings.LastIndex()	判断子字符串在字符串中首次出现的位置和在末尾出现的位置
strings.Replace()	替换字符串：Replace(原字符串,原字段,新字段,替换次数) -1表示全部替换，0表示不替换
strings.Count()	统计字符串出现次数
strings.Repeat()	是字符串重复输出多次
strings.ToUpper(), strings.ToLower()	字符串大小写转换
strings.TrimSpace()	函数去除空白字符
strings.TrimLeft(), strings.TrimRight()	从左往右删除和从右往左删除
strings.Join()	切片拼接成字符串
strconv.Itoa()	函数将数值转换成字符串
strconv.Atoi()	函数进行字符串转换整数