1、基本介绍

bytes包实现了`操作[]byte`的常用函数。本包的函数`和strings包的函数相当类似`。

2、比较函数

`Compare、Equal和EqualFold`三个

func Compare

```
func Compare(a, b []byte) int
```

Compare函数返回一个整数表示两个[]byte切片按字典序比较的结果（类同C的strcmp）。如果a==b返回0；如果a<b返回-1；否则返回+1。nil参数视为空切片。

```go
func main() {

    // Compare
    // Interpret Compare's result by comparing it to zero.
    var array [10]byte = [...]byte{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
    var a, b []byte = array[0:6], array[1:6]

    if bytes.Compare(a, b) < 0 {
        // a less b
        fmt.Println("a less b")
    }
    if bytes.Compare(a, b) <= 0 {
        // a less or equal b
        fmt.Println("a less or equal b")
    }
    if bytes.Compare(a, b) > 0 {
        // a greater b
        fmt.Println("a greater b")
    }
    if bytes.Compare(a, b) >= 0 {
        // a greater or equal b
        fmt.Println("a greater or equal b")
    }
    // Prefer Equal to Compare for equality comparisons.
    if bytes.Equal(a, b) {
        // a equal b
        fmt.Println("a equal b")
    }
    if !bytes.Equal(a, b) {
        // a not equal b
        fmt.Println("a not equal b")
    }

}
```


func Equal

```
func Equal(a, b []byte) bool
```

判断两个切片的内容是否完全相同。

func EqualFold

```
func EqualFold(s, t []byte) bool
```

判断两个utf-8编码切片（将unicode大写、小写、标题三种格式字符视为相同）是否相同。

```go
func main() {

    // Equal和EqualFold
    var array [7]byte = [...]byte{'a', 'b', '3', '4', '5', '6', '7'}
    var array2 [7]byte = [...]byte{'a', 'B', '3', '4', '5', '6', '7'}
    var a, b []byte = array[0:6], array2[0:6]

    fmt.Println(bytes.Equal(a, b))     // false
    fmt.Println(bytes.EqualFold(a, b)) // true

}
```

3、Runes函数

func Runes

```
func Runes(s []byte) []rune
```

Runes函数返回和s等价的[]rune切片。（将utf-8编码的unicode码值分别写入单个rune）

转换之后，这个切片可以处理中文

```go
func main() {

    // Equal和EqualFold
    var array [7]byte = [...]byte{'a', 'b', '3', '4', '5', '6', '7'}
    var array2 [7]byte = [...]byte{'a', 'B', '3', '4', '5', '6', '7'}
    var a, b []byte = array[0:6], array2[0:6]

    fmt.Println(bytes.Equal(a, b))     // false
    fmt.Println(bytes.EqualFold(a, b)) // true

    var data []rune = bytes.Runes(a)
    data = append(data, '你')
    fmt.Println(data)
    fmt.Println(a)

}
```

```
问题 1  输出  调试控制台  终端  COMMENTS
PS F:\tools\golang\bytes> go run .\main.go
false
true
[97 98 51 52 53 54 20320]
[97 98 51 52 53 54]
PS F:\tools\golang\bytes>
```

4、判断前后缀

func HasPrefix

```
func HasPrefix(s, prefix []byte) bool
```

判断s是否有前缀切片prefix。

func HasSuffix

```
func HasSuffix(s, suffix []byte) bool
```

判断s是否有后缀切片suffix。

```
func main() {

    // Equal和EqualFold
    var array [7]byte = [...]byte{'a', 'b', '3', '4', '5', '6', '7'}
    var array2 [7]byte = [...]byte{'a', 'B', '3', '4', '5', '6', '7'}
    var a, prefix []byte = array[0:6], array2[0:3]

    is := bytes.HasPrefix(a, prefix)
    fmt.Println(is) // false

    prefix[1] = 'b'
    is = bytes.HasPrefix(a, prefix)
    fmt.Println(is) // true
}
```


5、包含函数

func Contains

```
func Contains(b, subslice []byte) bool
```

判断切片b是否包含子切片subslice。


```
func main() {

    // Contains
    var array [7]byte = [...]byte{'a', 'b', '3', '4', '5', '6', '7'}
    var array2 [7]byte = [...]byte{'a', 'B', '3', '4', '5', '6', '7'}
    var a, subslice []byte = array[0:6], array2[0:3]

    is := bytes.Contains(a, subslice)
    fmt.Println(is) // false

    subslice[1] = 'b'

    is = bytes.Contains(a, subslice)
    fmt.Println(is) // true

    is = bytes.ContainsAny(a, "abcdef")
    fmt.Println(is) // true

    is = bytes.ContainsRune(a, '6')
    fmt.Println(is) // true
}
```

判断是否存在切片subslice，要求内容完全一致

判断切片a中是否存在字符串"abcdef"中的一个字符，只要有一个就为true

判断切片a中是否有rune类型的元素

5、Count函数

func Count

```
func Count(s, sep []byte) int
```

Count计算s中有多少个不重叠的sep子切片。

```go
func main() {

    // Contains
    var array [7]byte = [...]byte{'a', 'b', '3', '4', '5', '6', '7'}
    var array2 [7]byte = [...]byte{'a', 'B', '3', '4', '5', '6', '7'}
    var a, sep []byte = array[0:6], array2[0:3]

    count := bytes.Count(a, sep)
    fmt.Println(count)  // 0

    sep[1] = 'b'
    count = bytes.Count(a, sep)
    fmt.Println(count)  // 1
}
```


6、Index系列

索引子切片、byte、rune类型在原切片s中的位置。

func Index

```
func Index(s, sep []byte) int
```

子切片sep在s中第一次出现的位置，不存在则返回-1。

func IndexByte

```
func IndexByte(s []byte, c byte) int
```

字符c在s中第一次出现的位置，不存在则返回-1。

func IndexRune

```
func IndexRune(s []byte, r rune) int
```

unicode字符r的utf-8编码在s中第一次出现的位置，不存在则返回-1。

func IndexAny

```
func IndexAny(s []byte, chars string) int
```

字符串chars中的任一utf-8编码在s中第一次出现的位置，如不存在或者chars为空字符串则返回-1。

func LastIndex

```
func LastIndex(s, sep []byte) int
```

切片sep在字符串s中最后一次出现的位置，不存在则返回-1。

func LastIndexAny

```
func LastIndexAny(s []byte, chars string) int
```

字符串chars中的任一utf-8字符在s中最后一次出现的位置，如不存在或者chars为空字符串则返回-1。

7、Title系列

func Title

```
func Title(s []byte) []byte
```

返回s中每个单词的首字母都改为标题格式的拷贝。

BUG: Title用于划分单词的规则不能很好的处理Unicode标点符号。

func ToTitle

```
func ToTitle(s []byte) []byte
```

返回将所有字母都转为对应的标题版本的拷贝。

```go
func main() {

    // Title ToTitle
    var array [7]byte = [...]byte{'a', 'b', '3', '4', '5', '6', '7'}
    var array2 [7]byte = [...]byte{'a', 'B', '3', '4', '5', '6', '7'}
    var a, _ []byte = array[0:6], array2[0:3]

    dst := bytes.Title(a)
    fmt.Printf("%s\n", dst) // Ab3456

    dst = bytes.ToTitle(a)
    fmt.Printf("%s\n", dst) //AB3456
}
```


8、转大写和小写

func ToLower

```
func ToLower(s []byte) []byte
```

返回将所有字母都转为对应的小写版本的拷贝。

func ToUpper

```
func ToUpper(s []byte) []byte
```

返回将所有字母都转为对应的大写版本的拷贝。

9、重复和替换

func Repeat

```
func Repeat(b []byte, count int) []byte
```

返回count个b串联形成的新的切片。

func Replace

```
func Replace(s, old, new []byte, n int) []byte
```

返回将s中前n个不重叠old切片序列都替换为new的新的切片拷贝，如果n<0会替换所有old子切片。

```go
func main() {

    var array [7]byte = [...]byte{'a', 'b', '3', '4', '5', '6', '7'}
    var array2 [7]byte = [...]byte{'a', 'B', '3', '4', '5', '6', '7'}
    var a, _ []byte = array[0:6], array2[0:3]

    fmt.Println(a)                  // [97 98 51 52 53 54]
    fmt.Println(bytes.Repeat(a, 2)) // [97 98 51 52 53 54 97 98 51 52 53 54]

    result := bytes.ReplaceAll(a, []byte{'a', 'b'}, []byte{'c', 'd'})
    fmt.Println(result) // [99 100 51 52 53 54]
}
```

10、Trim系列

func Trim

```
func Trim(s []byte, cutset string) []byte
```

返回将s前后端所有cutset包含的unicode码值都去掉的子切片。（共用底层数组）

func TrimSpace

```
func TrimSpace(s []byte) []byte
```

返回将s前后端所有空白（unicode.IsSpace指定）都去掉的子切片。（共用底层数组）

func TrimLeft

```
func TrimLeft(s []byte, cutset string) []byte
```

返回将s前端所有cutset包含的unicode码值都去掉的子切片。（共用底层数组）

func TrimRight

```
func TrimRight(s []byte, cutset string) []byte
```

返回将s后端所有cutset包含的unicode码值都去掉的子切片。（共用底层数组）

func TrimSuffix

```
func TrimSuffix(s, suffix []byte) []byte
```

返回去除s可能的后缀suffix的子切片。（共用底层数组）

Example

```
var b = []byte("Hello, goodbye, etc!")
b = bytes.TrimSuffix(b, []byte("goodbye, etc!"))
b = bytes.TrimSuffix(b, []byte("gopher"))
b = append(b, bytes.TrimSuffix([]byte("world!"), []byte("x!"))...)
os.Stdout.Write(b)
```

Output:

```
Hello, world!
```

func TrimPrefix

```
func TrimPrefix(s, prefix []byte) []byte
```

返回去除s可能的前缀prefix的子切片。（共用底层数组）

Example

```
var b = []byte("Goodbye,, world!")
b = bytes.TrimPrefix(b, []byte("Goodbye,"))
b = bytes.TrimPrefix(b, []byte("See ya,"))
fmt.Printf("Hello%s", b)
```

Output:

```
Hello, world!
```

11、Fields分割函数

func Fields

```
func Fields(s []byte) [][]byte
```

返回将字符串按照空白（unicode.IsSpace确定，可以是一到多个连续的空白字符）分割的多个子切片。如果字符串全部是空白或者是空字符串的话，会返回空切片。

```
 9  func main() {
10
11      var s = []byte("Hello, goodbye, etc!")
12
13      dst := bytes.Fields(s)
14      for i := 0; i < len(dst); i++ {
15          os.Stdout.Write(dst[i])
16          os.Stdout.Write([]byte{'\n'})
17      }
18  }
```

```
问题 1   输出   调试控制台   终端   COMMENTS

PS F:\tools\golang\bytes> go run .\main.go
Hello,
goodbye,
etc!
```


12、Split系列

func Split

```
func Split(s, sep []byte) [][]byte
```

用去掉s中出现的sep的方式进行分割，会分割到结尾，并返回生成的所有[]byte切片组成的切片（每一个sep都会进行一次切割，即使两个sep相邻，也会进行两次切割）。如果sep为空字符，Split会将s切分成每一个unicode码值一个[]byte切片。

func SplitN

```
func SplitN(s, sep []byte, n int) [][]byte
```

用去掉s中出现的sep的方式进行分割，会分割到最多n个子切片，并返回生成的所有[]byte切片组成的切片（每一个sep都会进行一次切割，即使两个sep相邻，也会进行两次切割）。如果sep为空字符，Split会将s切分成每一个unicode码值一个[]byte切片。参数n决定返回的切片的数目：

```
n > 0 : 返回的切片最多n个子字符串；最后一个子字符串包含未进行切割的部分。
n == 0: 返回nil
n < 0 : 返回所有的子字符串组成的切片
```

13、Join连接函数

func Join

```
func Join(s [][]byte, sep []byte) []byte
```

将一系列[]byte切片连接为一个[]byte切片，之间用sep来分隔，返回生成的新切片。

```go
func main() {

    var src [][]byte = [][]byte{
        []byte{'a', 'b', 'c', 'd', 'e'},
        []byte{'f', 'g', 'h', 'i'},
        []byte{'j', 'k', 'l', 'm'},
    }

    var dst []byte = bytes.Join(src, []byte{'x', 'x'})

    os.Stdout.Write(dst) // abcdexxfghixxjklm

}
```