Go语言学习(十)bytes包处理字节切片

原创 2016年04月02日 17:04:59

标签：

[go语言](http://so.csdn.net/so/search/s.do?q=go%E8%AF%AD%E8%A8%80&t=blog" \t "http://blog.csdn.net/mchenys/article/details/_blank)

5196

bytes包提供了对字节切片进行读写操作的一系列函数   
字节切片处理的函数比较多,分为基本处理函数,比较函数,后缀检查函数,索引函数,分割函数,   
大小写处理函数和子切片处理函数等.

1.字节切片基本处理函数api

1.1Contains()函数

//Contains()函数的功能是检查字节切片b是否包含子切片subslice,如果包含返回true,否则返回false.func Contains(b,subslice []bytes) bool

1

2

1.2Count()函数

//Count()函数的功能是计算字节切片sep在字节切片s中非重叠显示的个数.func Count(s,sep[]byte) int

1

2

1.3Repeat()函数

//Repeat()函数的功能是把切片b复制count个,然后合成一个新的字节切片返回.func Repeat(b[]byte,count int) []byte

1

2

1.4Replace()函数

/\*Replace()函数的功能是返回字节切片s的一个副本,并把前n个不重叠的

子切片old替换为new;如果n<0,则不限制替换的数量.参数n为替换的次数\*/func Replace(s,old,new []byte,n int) []byte

1

2

3

1.5Runes()函数

//Runes()函数的功能是把s转换为UTF-8编码的字节序列,并返回对应的Unicode切片.func Runes(s []byte) []rune

1

2

1.6Join()函数

Join函数的功能是用字节切片sep把s中的每个字节切片连成一个字节切片并返回.func Join(s [][]byte,sep[]byte) []byte

1

2

[实例]

package main

import(

"fmt"

"bytes"

)func main(){

//Contains

b := []byte("mChenys") //字符串强转为byte切片

sublice1 := []byte("m")

sublice2 := []byte("M")

fmt.Println(bytes.Contains(b,sublice1))//true

fmt.Println(bytes.Contains(b,sublice2))//false

//Count

s := []byte("hahaahaaa")

sep1 := []byte("hah")

sep2 := []byte("aa")

sep3 := []byte("a")

fmt.Println(bytes.Count(s,sep1))//1

fmt.Println(bytes.Count(s,sep2))//2

fmt.Println(bytes.Count(s,sep3))//6

//Repeat

b = []byte("ha")

fmt.Println(string(bytes.Repeat(b,1)))//ha

fmt.Println(string(bytes.Repeat(b,2)))//haha

//Replace

s = []byte("hello,world")

old := []byte("o")

news := []byte("ee")

fmt.Println(string(bytes.Replace(s,old,news,0)))//hello,world

fmt.Println(string(bytes.Replace(s,old,news,1)))//hellee,world

fmt.Println(string(bytes.Replace(s,old,news,2)))//hellee,weerld

fmt.Println(string(bytes.Replace(s,old,news,-1)))//hellee,weerld

//Runes

s = []byte("你好世界")

r := bytes.Runes(s)

fmt.Println("转换前字符串的长度: ",len(s))//12

fmt.Println("转换后字符串的长度: ",len(r))//4

//Join

s := [][]byte{[]byte("你好"),[]byte("世界")}

sep1 := []byte(",")

fmt.Println(string(bytes.Join(s,sep1)))//你好,世界

sep2 := []byte("#")

fmt.Println(string(bytes.Join(s,sep2)))//你好#世界

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

2.字节切片比较函数

2.1Compare()函数

/\*Compare()函数的功能是根据字节的值比较字节切片a和b的大小,如果a=b,返回0,

如果a>b返回1,如果a<b返回-1.\*/func Compare(a,b[]byte) int

1

2

3

2.2Equal()函数

/\*Equal()函数的功能是用来比较2个字节切片是否相等,如果参数为nil,则等同于

空的字节切片,如果a=b,则返回true,否则返回false.区分大小写\*/func Equal(a,b[]byte) bool

1

2

3

2.3EqualFold()函数

/\*EqualFold()函数的功能是把s和t转换成UTF-8字符串进行比较,

并且忽略大小写,如果s=t,返回true,否则,返回false.\*/func EqualFold(s,t[]byte) bool

1

2

3

[实例]

package main

import(

"fmt"

"bytes"

)func main(){

//Compare

a := []byte("abc")

b := []byte("a")

fmt.Println(bytes.Compare(a,b))//1

b =[]byte("abcd")

fmt.Println(bytes.Compare(a,b))//-1

b =[]byte("abC")

fmt.Println(bytes.Compare(a,b))//1 小写字母大于大写字母

b =[]byte("b")

fmt.Println(bytes.Compare(a,b))//-1 从第一个字节开始比较,如果相同再比较长度

//Equal

a = []byte("abc")

b = []byte("ABC")

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

fmt.Println(bytes.Equal(a,nil))//false

b = []byte("abc")

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

//EqualFold

a = []byte("abc")

b = []byte("ABC")

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

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

3.字节切片前后缀检查

3.1HasPrefix()函数

//HasPrefix()函数的功能是检查字节切片s的前缀是否为prefix,如果是返回true,如果不是返回false.func HasPrefix(s,prefix[]byte) bool

1

2

3.2HashSuffix()函数

//HashSuffix()函数的功能是检查字节切片s的后缀是否为suffix,如果是返回true,否则返回false.func HashSuffix(s,suffix[]byte) bool

1

2

[实例]

package main

import(

"fmt"

"bytes"

)func main(){

//HasPrefix

s := []byte("mChenys")

prefix := []byte("m")

fmt.Println(bytes.HasPrefix(s,prefix))//true

prefix = []byte("men")

fmt.Println(bytes.HasPrefix(s,prefix))//false

//HashSuffix

suffix := []byte("ys")

fmt.Println(bytes.HasSuffix(s,suffix))//true

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

4.字节切片位置索引函数

字节切片位置索引函数共有8个,Index(),IndexAny(),IndexByte(),IndexFunc(),IndexRune(),   
LastIndex(),LastIndexAny()和LastIndexFunc().

4.1index()函数

//返回sep在s中第一次出现的位置索引(从0开始),如果sep中不在s中则返回-1.func Index(s,sep []byte) int

1

2

4.2IndexAny()函数

/\*把s解析为UTF-8编码的字节序列,返回chars中任何一个字符在s中第一次

出现的索引位置;如果s中不包含chars中任何一个字符,则返回-1.\*/func IndexAny(s []byte,chars string) int

1

2

3

4.3IndexByte()函数

//函数的功能是检查字节c在s中第一次出现的位置索引;如果s中不包含c则返回-1.func IndexByte(s[]byte,c byte) int

1

2

4.4IndexFunc()函数

/\*的功能是把s解析为UTF-8字节序列,并返回一个满足f(c)=true的字符c的位置

索引,如果没有满足则返回-1.\*/func IndexFunc(s[]byte,f func(r rune)bool) int

1

2

3

4.5IndexRune()函数

/\*功能是把s解析为UTF-8字节序列,并返回rune类型的字符r在s中的位置索引,

如果s中不包含r则返回-1.\*/func IndexRune(s[]byte,r rune) int

1

2

3

4.6LastIndex()函数

//的功能是返回sep在s中最后一次出现的位置索引,如果s中不包含sep,则返回-1func LastIndex(s,sep[]byte) int

1

2

4.7LastIndexAny()函数

/\*的功能是把s解析为UTF-8字节序列,返回chars中任何一个字符在s中最后

出现的位置索引,如果chars为空或者s中不包含chars中的任意字符,则返回-1\*/func LastIndexAny(s[]byte,chars string) int

1

2

3

4.8LastIndexFunc()函数

/\*功能是把s解析成UTF-8字节序列,返回满足f(s)=true的字符c在s中最后

一次出现的位置索引,如果没有找到则返回-1.\*/func LastIndexFunc(s[]byte,f func(r rune)bool) int

1

2

3

[实例]

package main

import(

"fmt"

"bytes"

)func main(){

//Index

a := []byte("aaaaa")

fmt.Println(bytes.Index(a,[]byte("a")))//0

fmt.Println(bytes.Index(a,[]byte("aa")))//0

fmt.Println(bytes.Index(a,[]byte("b")))//-1

//IndexAny

fmt.Println(bytes.IndexAny(a,"a"))//0

fmt.Println(bytes.IndexAny(a,"aa"))//0

//IndexByte

s := []byte("google")

var ch byte = 'g'

fmt.Println(bytes.IndexByte(s,ch))//0

//IndexFunc,可以接收匿名函数

fmt.Println(bytes.IndexFunc(s,func (a rune)bool{

if a == 'o'{

return true

}else{

return false

}

}))//1

//IndexRune

fmt.Println(bytes.IndexRune(s,'e'))//5

fmt.Println(bytes.IndexRune(s,'a'))//-1

//LastIndex

fmt.Println(bytes.LastIndex(s,[]byte("g")))//3

fmt.Println(bytes.LastIndex(s,[]byte("e")))//5

fmt.Println(bytes.LastIndex(s,[]byte("o")))//2

//LastIndexAny

fmt.Println(bytes.LastIndexAny(s,"gle"))//5

fmt.Println(bytes.LastIndexAny(s,"l"))//4

fmt.Println(bytes.LastIndexAny(s,"ge"))//5

//LastIndexFunc

fmt.Println(bytes.LastIndexFunc(s,func(r rune)bool{

if r=='g'{

return true

}else {

return false

}

}))//3

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

5.字节切片分割函数

字节切片分割函数共有6个,Fields(),FieldsFunc(),Split(),SplitN(),   
SplitAfter()和SplitAfterN()

5.1Fields()函数

/\*功能是把字节切片s按照一个或者连续多个空白字符分割成多个字节切片,

如果s只包含空白字符则返回空字节切片,其中参数s准备分割的字节切片.\*/func Fields(s[]byte) [][]byte wq

1

2

3

5.2FieldsFunc()函数

/\*功能是把s解析为UTF-8字节序列,对于每个Unicode字符c,如果f(c)

返回true就把c作为分割字符对s进行拆分.如果所有字符都满足f(c)为true,则返回空切片.\*/func FieldsFunc(s []byte,f func(r rune)bool) [][]byte

1

2

3

5.3Split()函数

/\*功能是把s用sep分割成多个字节切片并返回,如果sep为空,Split则把s切分成

每个字节切片对应一个UTF-8字符,Split()等效于参数为n的splitN()函数.\*/func Split(s,sep[]byte)[][]byte

1

2

3

5.4SplitAfter()函数

/\*功能使用sep作为后缀把s切分成多个字节切片并返回。如果sep为空，

则把s切分成每个字节切片对应一个UTF-8字符\*/func SplitAfter(s,sep[]byte)[][]byte

1

2

3

5.5SplitAfterN()函数

/\*功能是用sep作为后缀把s切分成多个字节切片并返回。如果sep为空，则把s切分成每个字节切片

对应一个UTF-8字符。参数n决定返回切片的长度：如果n>0,最多返回n个子字节切片，

子切片可能包含未切分的字节序列；如果n=0，返回空切片如果n< 0返回所有子切片。\*/func SplitAfterN(s,sep[]byte,n int)[][]byte

1

2

3

4

5.6SplitN()函数

/\*功能是把s用sep分割成多个字节切片并返回,如果sep为空,Split则把s

切分成每个字节切片对应一个UTF-8字符,参数n决定返回长度,n>0最多返回那个子切片;

n==0什么都不返回,n<0返回空子切片.\*/func SplitN(s,sep []byte,n int)[][]byte

1

2

3

4

[实例]

package main

import(

"fmt"

"bytes"

)func main(){

//Fields ,返回的是2维切片

s := []byte("a b c")

for \_,v := range bytes.Fields(s){

//遍历获取1维切片,再强转为字符串

fmt.Print(string(v)+",") //a,b,c,

}

//FieldsFunc,返回是2维切片,接收匿名函数

for \_,v := range bytes.FieldsFunc(s,func(r rune)bool{

if r == ' '{

return true //按照空白字符分割

}else{

return false

}

}){

fmt.Print(string(v)+",")//a,b,c,

}

//Split

s = []byte("吃饭和睡觉")

for \_,v := range bytes.Split(s,[]byte("和")){

fmt.Print(string(v)+",")//吃饭,睡觉,

}

for \_,v := range bytes.Split(s,nil){

fmt.Print(string(v)+",")//吃,饭,和,睡,觉,

}

//SplitAfter

s = []byte("abbcbbd")

for \_,v := range bytes.SplitAfter(s,[]byte("bb")){

fmt.Print(string(v)+",")//abb,cbb,d,

}

for \_,v := range bytes.SplitAfter(s,nil){

fmt.Print(string(v)+",")//a,b,b,c,b,b,d,

}

//SplitAfterN

s = []byte("hehehe")

for \_,v := range bytes.SplitAfterN(s,[]byte("he"),0){

fmt.Print(string(v)+",") //什么都不输出

}

for \_,v := range bytes.SplitAfterN(s,[]byte("he"),1){

fmt.Print(string(v)+",")//hehehe,

}

for \_,v := range bytes.SplitAfterN(s,[]byte("he"),-1){

fmt.Print(string(v)+",")//he,he,he,,

}

//SplitN

s = []byte("hahaha")

for \_,v := range bytes.SplitN(s,[]byte("ha"),0){

fmt.Print(string(v)+",") //什么都不输出

}

for \_,v := range bytes.SplitN(s,[]byte("ha"),1){

fmt.Print(string(v)+",")//hahaha,

}

for \_,v := range bytes.SplitN(s,[]byte("ha"),-1){

fmt.Print(string(v)+",")//,,,,

}

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

6.字节切片大小写处理

共有7个函数,Title(),ToTitle(),ToTitleSpecial(),ToLower(),ToLowerSpecial(),ToUpper()   
和ToUpperSpecial().

6.1Title()函数

//的功能是返回一个s的副本,把s中每个单词的首字母改成Unicode字符大写.func Title(s[]byte) []byte

1

2

6.2ToTitle()函数

//的功能是返回s的一个副本,并把其中所有Unicode字符转为大写.func ToTitle(s []byte) []byte

1

2

6.3ToTitleSpecial()函数

/\*功能是返回s的一个副本,并把其中所有Unicode字符根据\_case指定

的规则转成大写.\*/func ToTitleSpecial(\_case unicode.SpecialCase,s []byte) []byte

1

2

3

6.4ToLower()函数

//的功能是返回s的一个副本,并把其中的所有Unicode字符转为小写.func ToLower(s []byte)[]byte

1

2

6.5ToLowerSpecial()函数

/\*功能是返回s的一个副本，并把其中所有Unicode字符都根据\_case

指定的规则转换成小写.\*/func ToLowerSpecial(\_case unicode.SpecialCase, s []byte) []byte

1

2

3

6.6ToUpper()函数

//是返回s的一个副本,并把其中所有Unicode字符都转为大写.func ToUpper(s []byte) []byte

1

2

6.7ToUpperSpecial()函数

/\*的功能是返回s的一个副本,并把其中所有Unicode字符都根据\_case

指定的规则转成大写.\*/func ToUpperSpecial(\_case unicode.SpecialCase, s []byte) []byte

1

2

3

package main

import(

"fmt"

"bytes"

"unicode"

)func main(){

s := []byte("abc")

fmt.Println(string(bytes.Title(s)))//Abc

fmt.Println(string(bytes.ToTitle(s)))//ABC

fmt.Println(string(bytes.ToTitleSpecial(unicode.AzeriCase,s)))//ABC

s = []byte("ABC")

fmt.Println(string(bytes.ToLower(s)))//abc

fmt.Println(string(bytes.ToLowerSpecial(unicode.AzeriCase,s)))//abc

s = []byte("abc")

fmt.Println(string(bytes.ToUpper(s)))//ABC

fmt.Println(string(bytes.ToUpperSpecial(unicode.AzeriCase,s)))//ABC

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

7.子字节切片处理函数

共有9个,Trim(),TrimFunc(),TrimLeft(),TrimLeftFunc(),TrimRight(),TrimRightFunc(),   
TrimSpace(),TrimPrefix()和TrimSuffix().

7.1Trim()函数

/\*的功能是返回s的子字节切片，cutset中任意出现在s的首部和尾部的

连续字符将被删除。\*/func Trim(s []byte, cutset string) []byte

1

2

3

7.2TrimFunc()函数

//功能是返回s的子字节切片，删除s首部和尾部连接的满足f(c)=true的字符c。func TrimFunc(s []byte, f func(r rune) bool) []byte

1

2

7.3TrimLeft()函数

//的功能是返回s的子字节切片，cutset中任意出现在s首部的连续字符被删除。func TrimLeft(s []byte, cutset string) []byte

1

2

7.4TrimLeftFunc()函数

//功能是返回s的一个子字节切片、删除s首部连续满足f(c)=true的字符c。func TrimLeftFunc(s []byte, f func(r rune) bool) []byte

1

2

7.5TrimRight()函数

//的功能是返回s的子字节切片，cutset中任意出现在s尾部的连续字符被删除。func TrimRight(s []byte, cutset string) []byte

1

2

7.6TrimRightFunc()函数

//功能是返回s的一个子字节切片、删除s尾部连续满足f(c)=true的字符cfunc TrimRightFunc(s []byte, f func(r rune) bool) []byte

1

2

7.7TrimSpace()函数

/\*功能是返回s的一个子字节切片，并删除s中开始和结尾处的连续的

Unicode空白字符。\*/func TrimSpace(s []byte) []byte

1

2

3

7.8TrimPrefix()函数

//功能是返回s的一个子字节切片,并删除前缀为prefix的部分func TrimPrefix(s, prefix []byte) []byte

1

2

7.9TrimSuffix()函数

//功能是返回s的一个子字节切片,并删除后缀为suffix的部分func TrimSuffix(s, suffix []byte) []byte

1

2

[实例]

package main

import(

"fmt"

"bytes"

)func main(){

//Trim

s := []byte(" abc ")

fmt.Println(string(bytes.Trim(s," "))+"d")//abcd

//TrimFunc

s = []byte("hello world")

fmt.Println(string(bytes.TrimFunc(s,func(r rune)bool{

if r=='h' || r=='d'{

return true

}else{

return false

}

}))) //ello worl

s = []byte("helloh")

fmt.Println(string(bytes.TrimFunc(s,func(r rune)bool{

if r=='h' || r=='o'{

return true

}else{

return false

}

}))) //ell

s = []byte("helloh")

fmt.Println(string(bytes.TrimFunc(s,func(r rune)bool{

if r=='h' && r=='o'{

return true

}else{

return false

}

}))) //helloh

//TrimLeft

fmt.Println(string(bytes.TrimLeft(s,"h")))//elloh

fmt.Println(string(bytes.TrimLeft(s,"l")))//helloh

//TrimLeftFunc

fmt.Println(string(bytes.TrimLeftFunc(s,func(r rune)bool{

if r == 'h' || r=='l'{

return true

}else{

return false

}

}))) //elloh

//TrimRight

fmt.Println(string(bytes.TrimRight(s,"oh")))//hell

//TrimRightFunc

fmt.Println(string(bytes.TrimRightFunc(s,func(r rune)bool{

if r == 'h'{

return true

}else{

return false

}

})))//hello

//TrimSpace

s = []byte(" abc ")

fmt.Println("d"+string(bytes.TrimSpace(s))+"d")//dabcd

//TrimPrefix

s = []byte("mChenys")

fmt.Println(string(bytes.TrimPrefix(s,[]byte("mC"))))//henys

fmt.Println(string(bytes.TrimPrefix(s,[]byte("en"))))//mChenys

//TrimSuffix

s = []byte("mChenys")

fmt.Println(string(bytes.TrimSuffix(s,[]byte("ys"))))//mChen

fmt.Println(string(bytes.TrimSuffix(s,[]byte("en"))))//mChenys

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

文章来源：http://blog.csdn.net/mchenys/article/details/51044589

版权声明：本文为博主原创文章，未经博主允许不得转载。