数据结构

ASCII

ASCII（American Standard Code for Information Interchange，美国信息交换标准代码）是基于拉丁字母的一套单字节编码系统

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

熟记下面表格中特殊字符

序号	转义字符	十进制数	说明
1	\x00	0	null字符，表中第一项，C语言中的字符串结束符
2	\x09 \t	9	tab字符
3	\x0d\x0a \r\n	13 10	回车和换行
4	\x30~\x39	48~57	字符0~9
5	\x31	49	字符1
6	\x41	65	字符A
7	\x61	97	字符a

注意：这里的1指的是字符1，不是数字1

UTF-8、GBK都兼容了ASCII

```
"a\x09b\x0ac \x31\x41\x61" 表示什么？
'A' > 'a'  谁大？字符比较
"a" > "A"  谁大？字符串比较
"AA" > "Aa" # 谁大？
```

字符

本质上来说，计算机中一切都是字节的，字符串也是多个字节组合而成，就是一个字节形成的有序序列。但是对于多字节编码的中文来说，用一个字节描述不了，需要多个字节表示一个字符，Go提供了rune类型。

- byte：兼容ASCII码的字符，是byte类型，即uint8别名，占用1个字节
- rune：汉字等字符，unicode，是rune类型，即int32别名，占用4个字节
- 一个字符字面量使用单引号引起来

字符串与字节序列转换

```
s1 := "abc"
s2 := "测试"
fmt.Println(len(s1), len(s2)) // 3, 6 字节

// 强制类型转换 string => []byte; string => []rune
// 注意[]byte表示字节序列；[]rune表示rune序列
fmt.Println([]byte(s1))
fmt.Println([]rune(s1))
fmt.Println([]byte(s2)) // utf-8 bytes，长度为6即6个字节
fmt.Println([]rune(s2)) // unicode切片，长度为2，每个元素4字节
fmt.Printf("%x, %x", 27979, 35797)
fmt.Println("~~~~~~~~~~~~~~~~~~~~~~~~~~")

// []byte => string
fmt.Println(string([]byte{49, 65, 97}))
// []rune => string
fmt.Println(string([]rune{27979, 35797}))
// rune使用unicode，但是字符串内部使用utf-8

fmt.Println(s2[0], s2[1], s2[2]) // 索引取到了什么？
fmt.Println(string([]byte{230, 181, 139})) // 打印什么？
```

- string(整数)，强制类型转换一个整数，相当于把整数当unicode码，去查一个字符，最后返回字符串
- string(整数序列)，强制类型转换一个整数序列，也是转成字符串

字符串

- 字面常量，只读，不可变
- 线性数据结构，可以索引
- 值类型
- utf-8编码

长度

使用内建函数len，返回字符串占用的字节数。时间复杂度为O(1)，字符串是字面常量，定义时已经知道长度，记录下来即可。

索引

不支持负索引，索引范围[0, len(s)-1]。

即使是有中文，索引指的是按照**字节的偏移量**。

时间复杂度O(1)，使用索引计算该字符相对开头的偏移量即可。

对于顺序表来说，使用索引效率查找效率是最高的。

`s[i]` 获取**索引**i处的UTF-8编码的一个字节。

遍历

C风格使用**索引**遍历，相当于字节遍历。

```
s := "magedu马哥教育"
for i := 0; i < len(s); i++ {
    fmt.Printf("%d, %T, %[2]d %[2]c\n", i, s[i])
}
```

for-range遍历，安全地遍历字符。

```
s := "magedu马哥教育"
for i, v := range s {
    fmt.Printf("%d, %T: %[2]d %[2]c; %T: %[3]d %[3]c\n", i, s[i], v) // 注意i不连续。v是汉字的unicode
}
```

strings库

strings提供了大多数字符串操作函数，使用方便。

注意：字符串是字面常量，不可修改，很多操作都是返回新的字符串。

拼接

- Join：使用间隔符拼接字符串切片
- Builder：推荐的字符串拼接方式

```
s0 := "www.magedu.edu"
s1 := "马哥教育"
fmt.Printf("%s %p; %s %p\n", s0, &s0, s1, &s1)

s2 := s0 + "-" + s1
fmt.Printf("%s %p\n", s2, &s2)

s3 := strings.Join([]string{s0, s1}, "-")
fmt.Printf("%s %p\n", s3, &s3)
```

```
10
11  s4 := fmt.Sprintf("%s-%s", s0, s1)
12  fmt.Printf("%s %p\n", s4, &s4)
13
14  var b strings.Builder
15  b.WriteString(s0)
16  b.WriteByte('-')
17  b.WriteString(s1)
18  s5 := b.String()
19  fmt.Printf("%s %p\n", s5, &s5)
```

简单拼接字符串常用+、fmt.Sprintf。如果手里正好有字符串的序列，可以考虑Join。如果反复多次拼接，strings.Builder是推荐的方式。

查询

- Index：从左至右搜索，返回子串第一次出现的字节索引位置。未找到，返回-1。子串为空，也返回0。
- LastIndex：从右至左搜索，返回子串第一次出现的字节索引位置。未找到，返回-1。
- IndexByte、IndexRune与Index类似；LastIndexByte与LastIndex类似。
- IndexAny：从左至右搜索，找到给定的字符集字符串中任意一个字符就返回索引位置。未找到返回-1。
- Contains*方法本质上就是Index*方法，只不过返回bool值，方便使用bool值时使用。
- LastIndexAny与IndexAny搜索方向相反。
- Count：从左至右搜索子串，返回子串出现的次数。

时间复杂度是O(n)，效率不高，该用则用，但要少用。

```
1  s := "www.magedu.com马哥教育"
2  fmt.Println('马', "马"[0])
3  fmt.Println(
4      strings.Index(s, "马"),
5      strings.IndexAny(s, "马"),
6      strings.IndexByte(s, 233),
7      strings.IndexRune(s, 39532),
8      strings.Contains(s, "马"),
9      strings.Count(s, "m"),
10 )
```

大小写

- ToLower：转换为小写
- ToUpper：转换为大写

前后缀

- HasPrefix：是否以子串开头
- HasSuffix：是否以子串结尾

效率高吗?

移除

- TrimSpace：去除字符串两端的空白字符。
- TrimPrefix、TrimSuffix：如果开头或结尾匹配，则去除。否则，返回原字符串的副本。
- TrimLeft：字符串开头的字符如果在字符集中，则全部移除，直到碰到第一个不在字符集中的字符为止。
- TrimRight：字符串结尾的字符如果在字符集中，则全部移除，直到碰到第一个不在字符集中的字符为止。
- Trim：字符串两头的字符如果在字符集中，则全部移除，直到左或右都碰到第一个不在字符集中的字符为止。

```
fmt.Println(strings.TrimSpace("\v\n\r \tabc\txyz\t \v\r\n"))
fmt.Println(strings.TrimPrefix("www.magedu.edu-马哥教育", "www."))
fmt.Println(strings.TrimSuffix("www.magedu.edu-马哥教育", ".edu"))
fmt.Println(strings.TrimLeft("abcdddeabeccc", "abcd"))
fmt.Println(strings.TrimRight("abcdddeabeccc", "abcd"))
fmt.Println(strings.Trim("abcdddeabeccc", "abcd"))
```

分割

- Split：按照给定的分割子串去分割，返回切割后的字符串切片。
  - 切割字符串是被切掉的，不会出现在结果中
  - 没有切到，也会返回一个元素的切片，元素就是被切的字符串
  - 分割字符串为空串，那么返回将被切割字符串按照每个rune字符分解后转成string存入切片返回
- `SplitN(s, sep string, n int) []string`
  - n == 0，返回空切片，切成0个子串
  - n > 0，返回切片元素的个数
    - n == 1，返回一个元素切片，元素为s，相当于Split的没有切到
    - n > 1，按照sep切割。返回多个元素的切片。按照sep切成的段数最多有x段，当n < x时，会有部分剩余字符串未切；n == x时，字符串s正好从头到尾切完，返回所有段的切片；n > x时，和n == x一样。n表示切割出来的子串的上限，即至多切片里面有n个元素
  - n < 0，等价Split，能切多少切出多少
- SplitAfter和Split相似，就是不把sep切掉
- SplitAfterN和SplitN相似，也不把sep切掉
- `Cut(s, sep string) (before, after string, found bool)`
  - 内部使用Index找sep，所以是从左至右搜索切割点。可以认为就是切一刀，一刀两段
  - 没有切到，返回 `s, "", false`
  - 切到了，匹配切割符的部分要切掉，返回 `切割符前部分，切割符后部分，true`

替换

- `Replace(s, old, new string, n int) string`
  - n < 0，等价ReplaceAll，全部替换
  - n == 0，或old == new，就返回s
  - n > 0，至多替换n次，如果n超过找到old子串的次数x，也就只能替换x次了
  - 未找到替换处，就返回s

其他

- Repeat：使用给定的字符串重复n次拼接成一个新字符串。
- Map：按照给定处理每个rune字符的函数依次处理每个字符后，拼接成字符串返回。注意Map是一对一的映射，不能减少元素个数。

```
s := "www.magedu.edu-马哥教育"
for i, v := range s {
    fmt.Printf("%d %c; ", i, v)
}

fmt.Println(strings.Map(func(r rune) rune {
    if 'a' <= r && r <= 'z' {
        return r - 0x20 // 请问这是干什么？
    }
    return r
}, s))
```

类型转换

数值类型转换

- 低精度向高精度转换可以，高精度向低精度转换会损失精度
- 无符号向有符号转换，最高位是符号位
- byte和int可以互相转换
- float和int可以相互转换，float到int会丢失精度
- **bool和int不能相互转换**
- 不同长度的int和float之间可以互相转换

```
var i int8 = -1
var j uint8 = uint8(i)
fmt.Println(i, j) // 请问j是多少
```

```
fmt.Println(int(3.14)) // 错误，不允许无类型float常量转到int

var a = 3.14 // 定义有类型变量转换就没有问题
fmt.Printf("%T: %[1]v => %d\n", a, int(a)) // float64: 3.14 => 3
```

```
// byte rune本质上就是整数和无类型常量可以直接计算，自动转换
b := 'a'
c := b + 1
fmt.Println(c) // 请问c显示什么，什么类型
```

但是，如果不使用无类型常量，有类型的计算如果类型不一致要报错，因为Go是对类型要求非常严苛的语言，要强制类型转换。

```
b := 'a'
e := 1
c := b + e // rune和int类型不能加，必须转换。比如c := int(b) + e或c := b + rune(e)
fmt.Println(c)
```

类型别名和类型定义

```
var a byte = 'C'
var b uint8 = 49
fmt.Println(a, b, a+b) // 为什么类型不同，可以相加？
```

原因是在源码中定义了`type byte = uint8`，byte是uint8的别名。

Go v1.9开始使用了类型别名，将byte等的定义修改成了类型别名的方式。

别名说明就是uint8的另外一个名字，和uint8是一回事。再看一段代码，正确吗？

```
type myByte uint8

var a byte = 'C'
var b uint8 = 49
fmt.Println(a, b, a+b) // 为什么类型不同，可以相加？

var c myByte = 50
fmt.Println(a, c, a + c) // 可以吗？为什么？
```

答案是不可以。原因就是Go原因不允许不同类型随便运算。就算我们眼睛看到可以，也不行，必须强制类型转换，转换是否成功，程序员自己判断。

```
type myByte uint8 // 类型定义
type byte = uint8 // 类型别名
```

byte只是存在在代码中，为了方便阅读或使用，编译完成后，不会有byte类型。

字符串转换

在字符那一节，我们介绍过了字符串与[]byte、字符串与[]rune之间互转的例子，这里不再赘述。

```
s1 := "127"
fmt.Println(strconv.Atoi(s1))              // 十进制理解
fmt.Println(strconv.ParseInt(s1, 16, 32)) // 十六进制理解0x127

s2 := "115.6"
fmt.Println(strconv.ParseFloat(s2, 64))

fmt.Println(strconv.ParseBool("true"))
```