# strings包和time及math包

本章讲解。

本章重点为大家介绍如下的内容：

* strings包常用函数
* strconv包常用函数比特币系统整体架构
* time包
* math包
* math/rand包——随机数包
* 键盘输入比特币系统前端模块

## 字符串处理包概述

### strings与strconv包简介：

strings包实现了用于操作字符的简单函数。

strconv包实现了字符串与其它基本数据类型之间的类型转换。

大多数语言的字符串处理库提供的函数都大同小异，且越高级的语言提供的函数越多，比如c提供的字符串处理函数go基本都有，但是go提供的c未必有。

掌握基本的字符串处理函数后，更丰富的字符串处理函数都是通过封装基本的处理函数实现。所以熟悉go strings库后基本就能借此封装符合自己需求的，应用于特定场景的字符串处理函数了。

### string的基本用法

例：

package main

import (

"fmt"

"unicode/utf8"

)

func main() {

s := "我爱Go语言"

fmt.Println(s)

fmt.Println("字节长度：", len(s))

fmt.Println("-------------------")

//获取字符串引用

for i, ch := range s {

fmt.Printf("%d:%c \n", i, ch)

}

fmt.Println("-------------------")

//遍历所有字节

for i, ch := range []byte(s) {

fmt.Printf("%d:%X \n", i, ch)

}

fmt.Println("字节长度:", len(s))

fmt.Println("-------------------")

//获取所有字符

//rune相当于go的char（字符型）

count := 0

for i, ch := range []rune(s) {

fmt.Printf("%d:%c \n", i, ch)

count++

}

fmt.Println("字符长度:", count)

fmt.Println("字符长度：", utf8.RuneCountInString(s))

fmt.Println("-------------------")

}

## strings包中的字符串处理函数

### 检索字符串

1.func Contains(s, substr string) bool

判断字符串s是否包含substr字符串

fmt.Println(strings.Contains("seafood", "foo"))

fmt.Println(strings.Contains("seafood", "bar"))

fmt.Println(strings.Contains("seafood", ""))

fmt.Println(strings.Contains("", ""))

Output:

true

false

true

true

2.func ContainsAny(s, chars string) bool

判断字符串s是否包含chars字符串中的任一字符

fmt.Println(strings.ContainsAny("team", "i"))

fmt.Println(strings.ContainsAny("failure", "u & i"))

fmt.Println(strings.ContainsAny("foo", ""))

fmt.Println(strings.ContainsAny("", ""))

Output:

false

true

false

false

3.func Count(s, sep string) int

返回字符串s包含字符串sep的个数

fmt.Println(strings.Count("cheese", "e"))

fmt.Println(strings.Count("five", "")) // before & after each rune

Output:

3

5

4.func Index(s, sep string) int

返回字符串s中字符串sep首次出现的位置

fmt.Println(strings.Index("chicken", "ken"))

fmt.Println(strings.Index("chicken", "dmr"))

Output:

4

-1

5.func IndexAny(s, chars string) int

返回字符串chars中的任一unicode码值r在s中首次出现的位置

fmt.Println(strings.IndexAny("chicken", "aeiouy"))

fmt.Println(strings.IndexAny("crwth", "aeiouy"))

Output:

2

-1

6.func IndexFunc(s string, f func(rune) bool) int

返回字符串s中满足函数f(r)==true字符首次出现的位置

f := func(c rune) bool {

return unicode.Is(unicode.Han, c)

}

fmt.Println(strings.IndexFunc("Hello, 世界", f))

fmt.Println(strings.IndexFunc("Hello, world", f))

Output:

7

-1

7.func IndexRune(s string, r rune) int

返回unicode码值r在字符串中首次出现的位置

fmt.Println(strings.IndexRune("chicken", 'k'))

fmt.Println(strings.IndexRune("chicken", 'd'))

Output:

4

-1

8.func LastIndex(s, sep string) int

返回字符串s中字符串sep最后一次出现的位置

fmt.Println(strings.Index("go gopher", "go"))

fmt.Println(strings.LastIndex("go gopher", "go"))

fmt.Println(strings.LastIndex("go gopher", "rodent"))

Output:

0

3

-1

### 分割字符串：

1.func Fields(s string) []string

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）分割的多个字符串。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。

fmt.Printf("Fields are: %q", strings.Fields(" foo bar baz "))

Output:

Fields are: ["foo" "bar" "baz"]

2.func FieldsFunc(s string, f func(rune) bool) []string

类似Fields，但使用函数f来确定分割符（满足f的unicode码值）。如果字符串全部是分隔符或者是空字符串的话，会返回空切片。

f := func(c rune) bool {

return !unicode.IsLetter(c) && !unicode.IsNumber(c)

}

fmt.Printf("Fields are: %q", strings.FieldsFunc(" foo1;bar2,baz3...", f))

Output:

Fields are: ["foo1" "bar2" "baz3"]

3.func Split(s, sep string) []string

用去掉s中出现的sep的方式进行分割，会分割到结尾，并返回生成的所有片段组成的切片（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码值一个字符串。

fmt.Printf("%q\n", strings.Split("a,b,c", ","))

fmt.Printf("%q\n", strings.Split("a man a plan a canal panama", "a "))

fmt.Printf("%q\n", strings.Split(" xyz ", ""))

fmt.Printf("%q\n", strings.Split("", "Bernardo O'Higgins"))

Output:

["a" "b" "c"]

["" "man " "plan " "canal panama"]

[" " "x" "y" "z" " "]

[""]

4.func SplitAfter(s, sep string) []string

用从s中出现的sep后面切断的方式进行分割，会分割到结尾，并返回生成的所有片段组成的切片（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码值一个字符串。

fmt.Printf("%q\n", strings.SplitAfter("a,b,c", ","))

Output:

["a," "b," "c"]

5.func SplitAfterN(s, sep string, n int) []string

用从s中出现的sep后面切断的方式进行分割，会分割到结尾，并返回生成的所有片段组成的切片（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码值一个字符串。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

fmt.Printf("%q\n", strings.SplitAfterN("a,b,c", ",", 2))

Output:

["a," "b,c"]

6.func SplitN(s, sep string, n int) []string

用去掉s中出现的sep的方式进行分割，会分割到结尾，并返回生成的所有片段组成的切片（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码值一个字符串。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

fmt.Printf("%q\n", strings.SplitN("a,b,c", ",", 2))

z := strings.SplitN("a,b,c", ",", 0)

fmt.Printf("%q (nil = %v)\n", z, z == nil)

Output:

["a" "b,c"]

[] (nil = true)

### 大小写转换：

1.func Title(s string) string

将字符串s每个单词首字母大写返回

fmt.Println(strings.Title("her royal highness"))

Output:

Her Royal Highness

2.func ToLower(s string) string

将字符串s转换成小写返回

fmt.Println(strings.ToLower("Gopher"))

Output:

gopher

3.func ToTitle(s string) string

将字符串s转换成大写返回

fmt.Println(strings.ToTitle("loud noises"))

fmt.Println(strings.ToTitle("хлеб"))

Output:

LOUD NOISES

ХЛЕБ

4.func ToTitleSpecial(\_case unicode.SpecialCase, s string) string

将字符串s中所有字符按\_case指定的映射转换成大写返回

5.func ToUpper(s string) string

将字符串s转换成大写返回

fmt.Println(strings.ToUpper("Gopher"))

Output:

GOPHER

### trim修剪函数：

1.func Trim(s string, cutset string) string

将字符串s中首尾包含cutset中的任一字符去掉返回

fmt.Printf("[%q]", strings.Trim(" !!! Achtung! Achtung! !!! ", "! "))

Output:

["Achtung! Achtung"]

2.func TrimFunc(s string, f func(rune) bool) string

将字符串s首尾满足函数f(r)==true的字符去掉返回

3.func TrimLeft(s string, cutset string) string

将字符串s左边包含cutset中的任一字符去掉返回

4.func TrimLeftFunc(s string, f func(rune) bool) string

将字符串s左边满足函数f(r)==true的字符去掉返回

5.func TrimPrefix(s, prefix string) string

将字符串s中前缀字符串prefix去掉返回

var s = "Goodbye,, world!"

s = strings.TrimPrefix(s, "Goodbye,")

s = strings.TrimPrefix(s, "Howdy,")

fmt.Print("Hello" + s)

Output:

Hello, world!

6.func TrimRight(s string, cutset string) string

将字符串s右边包含cutset中的任一字符去掉返回

7.func TrimRightFunc(s string, f func(rune) bool) string

将字符串s右边满足函数f(r)==true的字符去掉返回

8.func TrimSpace(s string) string

将字符串s首尾空白去掉返回

fmt.Println(strings.TrimSpace(" \t\n a lone gopher \n\t\r\n"))

Output:

a lone gopher

9.func TrimSuffix(s, suffix string) string

将字符串s中后缀字符串prefix去掉返回

var s = "Hello, goodbye, etc!"

s = strings.TrimSuffix(s, "goodbye, etc!")

s = strings.TrimSuffix(s, "planet")

fmt.Print(s, "world!")

Output:

Hello, world!

### 比较字符串

1.func Compare(a, b string) int

按字典顺序比较a和b字符串大小

fmt.Println(strings.Compare("a", "b"))

fmt.Println(strings.Compare("a", "a"))

fmt.Println(strings.Compare("b", "a"))

Output:

-1

0

1

2.func EqualFold(s, t string) bool

判断s和t两个utf8字符串是否相等，忽略大小写

fmt.Println(strings.EqualFold("Go", "go"))

Output:

true

3.func Repeat(s string, count int) string

将字符串s重复count次返回

fmt.Println("ba" + strings.Repeat("na", 2))

Output:

banana

4.func Replace(s, old, new string, n int) string

替换字符串s中old字符为new字符并返回，n<0是替换所有old字符串

fmt.Println(strings.Replace("oink oink oink", "k", "ky", 2))

fmt.Println(strings.Replace("oink oink oink", "oink", "moo", -1))

Output:

oinky oinky oink

moo moo moo

5.func Join(a []string, sep string) string

将a中的所有字符串连接成一个字符串，使用字符串sep作为分隔符

s := []string{"foo", "bar", "baz"}

fmt.Println(strings.Join(s, ", "))

Output:

foo, bar, baz

## strconv包中常用函数

### Parse类函数（将字符串转其它类型）

1.func Atoi(s string) (int, error)

Atoi 返回 ParseInt(s, 10, 0) 转换为 int 类型的结果。【alphabet：字母】

v := "10"

if s, err := strconv.Atoi(v); err == nil {

fmt.Printf("%T, %v", s, s)

}

Output:

int, 10

2.func ParseInt(s string, base int, bitSize int) (i int64, err error)

ParseInt 解释给定基础（2到36）中的字符串 s 并返回相应的值 i。如果 base == 0，则基数由字符串的前缀隐含：base 16代表“0x”，base 8代表“0”，否则以10为底数。

v32 := "-354634382"

if s, err := strconv.ParseInt(v32, 10, 32); err == nil {

fmt.Printf("%T, %v\n", s, s)

}

if s, err := strconv.ParseInt(v32, 16, 32); err == nil {

fmt.Printf("%T, %v\n", s, s)

}

v64 := "-3546343826724305832"

if s, err := strconv.ParseInt(v64, 10, 64); err == nil {

fmt.Printf("%T, %v\n", s, s)

}

if s, err := strconv.ParseInt(v64, 16, 64); err == nil {

fmt.Printf("%T, %v\n", s, s)

}

Output:

int64, -354634382

int64, -3546343826724305832

3.func ParseUint(s string, base int, bitSize int) (uint64, error)

ParseUint 就像 ParseInt，但是对于无符号数字。

v := "42"

if s, err := strconv.ParseUint(v, 10, 32); err == nil {

fmt.Printf("%T, %v\n", s, s)

}

if s, err := strconv.ParseUint(v, 10, 64); err == nil {

fmt.Printf("%T, %v\n", s, s)

}

Output:

uint64, 42

uint64, 42

4.func ParseFloat(s string, bitSize int) (float64, error)

ParseFloat 将字符串 s 转换为浮点数，精度由 bitSize：32指定，float32为64; float64为64。当 bitSize = 32时，结果仍然具有 float64 类型，但可以在不更改其值的情况下将其转换为 float32。

v := "3.1415926535"

if s, err := strconv.ParseFloat(v, 32); err == nil {

fmt.Printf("%T, %v\n", s, s)

}

if s, err := strconv.ParseFloat(v, 64); err == nil {

fmt.Printf("%T, %v\n", s, s)

}

Output:

float64, 3.1415927410125732

float64, 3.1415926535

5.func ParseBool(str string) (bool, error)

ParseBool 返回字符串表示的布尔值。

它接受1，t，T，TRUE，true，True，

0，f，F，FALSE，false，False。

任何其他值都会返回错误。

v := "true"

if s, err := strconv.ParseBool(v); err == nil {

fmt.Printf("%T, %v\n", s, s)

}

Output:

bool, true

### Format类函数

1.func Itoa(i int) string

Itoa 是 FormatInt(int64(i), 10) 的缩写。

i := 10

s := strconv.Itoa(i)

fmt.Printf("%T, %v\n", s, s)

Output:

string, 10

2.func FormatInt(i int64, base int) string

FormatInt 返回给定基数中的i的字符串表示，对于2 <= base <= 36.结果对于数字值> = 10使用小写字母 'a' 到 'z' 。

v := int64(-42)

s10 := strconv.FormatInt(v, 10)

fmt.Printf("%T, %v\n", s10, s10)

s16 := strconv.FormatInt(v, 16)

fmt.Printf("%T, %v\n", s16, s16)

Output:

string, -42

string, -2a

3.func FormatUint(i uint64, base int) string

FormatUint 返回给定基数中的 i 的字符串表示，对于2 <= base <= 36.结果对于数字值> = 10使用小写字母 'a' 到 'z' 。

v := uint64(42)

s10 := strconv.FormatUint(v, 10)

fmt.Printf("%T, %v\n", s10, s10)

s16 := strconv.FormatUint(v, 16)

fmt.Printf("%T, %v\n", s16, s16)

Output:

string, 42

string, 2a

4.func FormatFloat(f float64, fmt byte, prec, bitSize int) string

FormatFloat 根据格式 fmt 和 precision prec 将浮点数f转换为字符串。它将结果进行四舍五入，假设原始数据是从 bitSize 位的浮点值获得的（float32为32，float64为64）。

格式 fmt 是 'b'，'e'，'E'，'f'，'g'或 'G'。

v := 3.1415926535

s32 := strconv.FormatFloat(v, 'E', -1, 32)

fmt.Printf("%T, %v\n", s32, s32)

s64 := strconv.FormatFloat(v, 'E', -1, 64)

fmt.Printf("%T, %v\n", s64, s64)

Output:

string, 3.1415927E+00

string, 3.1415926535E+00

5.func FormatBool(b bool) string

FormatBool 根据 b 的值返回“true”或“false”

v := true

s := strconv.FormatBool(v)

fmt.Printf("%T, %v\n", s, s)

Output:

string, true

## time包

需要先import "time"。time包提供了时间的显示和测量用的函数。日历的计算采用的是公历。

### time包中类型及方法

1.type Weekday

func (d Weekday) String() string

2.type Month

func (m Month) String() string

3.type Location

func LoadLocation(name string) (\*Location, error)

func FixedZone(name string, offset int) \*Location

func (l \*Location) String() string

4.type Time

func Date(year int, month Month, day, hour, min, sec, nsec int, loc \*Location) Time

func Parse(layout, value string) (Time, error)

func ParseInLocation(layout, value string, loc \*Location) (Time, error)

func Now() Time

func Unix(sec int64, nsec int64) Time

func (t Time) Location() \*Location

func (t Time) Zone() (name string, offset int)

func (t Time) IsZero() bool

func (t Time) Local() Time

func (t Time) UTC() Time

func (t Time) In(loc \*Location) Time

func (t Time) Unix() int64

func (t Time) UnixNano() int64

func (t Time) Equal(u Time) bool

func (t Time) Before(u Time) bool

func (t Time) After(u Time) bool

func (t Time) Date() (year int, month Month, day int)

func (t Time) Clock() (hour, min, sec int)

func (t Time) Year() int

func (t Time) Month() Month

func (t Time) ISOWeek() (year, week int)

func (t Time) YearDay() int

func (t Time) Day() int

func (t Time) Weekday() Weekday

func (t Time) Hour() int

func (t Time) Minute() int

func (t Time) Second() int

func (t Time) Nanosecond() int

func (t Time) Add(d Duration) Time

func (t Time) AddDate(years int, months int, days int) Time

func (t Time) Sub(u Time) Duration

func (t Time) Round(d Duration) Time

func (t Time) Truncate(d Duration) Time

func (t Time) Format(layout string) string

func (t Time) String() string

func (t Time) GobEncode() ([]byte, error)

func (t \*Time) GobDecode(data []byte) error

func (t Time) MarshalBinary() ([]byte, error)

func (t \*Time) UnmarshalBinary(data []byte) error

func (t Time) MarshalJSON() ([]byte, error)

func (t \*Time) UnmarshalJSON(data []byte) error

func (t Time) MarshalText() ([]byte, error)

func (t \*Time) UnmarshalText(data []byte) error

5.type Duration

func ParseDuration(s string) (Duration, error)

func Since(t Time) Duration

func (d Duration) Hours() float64

func (d Duration) Minutes() float64

func (d Duration) Seconds() float64

func (d Duration) Nanoseconds() int64

func (d Duration) String() string

6.type Timer

func NewTimer(d Duration) \*Timer

func AfterFunc(d Duration, f func()) \*Timer

func (t \*Timer) Reset(d Duration) bool

func (t \*Timer) Stop() bool

7.type Ticker

func NewTicker(d Duration) \*Ticker

func (t \*Ticker) Stop()

8.const (

Nanosecond Duration = 1 //纳秒ns

Microsecond = 1000 \* Nanosecond //微妙us

Millisecond = 1000 \* Microsecond //毫秒ms

Second = 1000 \* Millisecond //秒s

Minute = 60 \* Second

Hour = 60 \* Minute

)

### time包中核心方法介绍

1.func Now() Time

Now返回当前本地时间。

2.func (t Time) Local() Time

Local将时间转成本地时区，但指向同一时间点的Time。

3.func (t Time) UTC() Time

UTC将时间转成UTC和零时区，但指向同一时间点的Time。

国际上通过英国伦敦格林尼治天文台原址的那条经线称为0°经线，也叫本初子午线。

4.func Date(year int, month Month, day, hour, min, sec, nsec int, loc \*Location) Time

Date可以根据指定数值，返回一个时间。时区为loc，时间格式为：year-month-day hour:min:sec + nsec nanoseconds的时间点。loc可以是time.Local.time.UTC。string转time

5.func Parse(layout, value string) (Time, error)

Parse能将一个格式化的时间字符串解析成它所代表的时间。就是string转time

layout定义了参考时间：Mon Jan 2 15:04:05 -0700 MST 2006

如果缺少表示时区的信息，Parse会将时区设置为UTC。layout简写格式：Mon Jan 2 15:04:05 2006

预定义的ANSIC.UnixDate.RFC3339和其他版式描述了参考时间的标准或便捷表示。

6.func (t Time) Format(layout string) string

Format根据layout指定的格式返回t代表的时间点的格式化文本表示。就是time转string

layout定义了参考时间：Mon Jan 2 15:04:05 -0700 MST 2006

7.func (t Time) String() string

String将时间格式化成字符串(time转string，相当于是固定格式的Format方法)，格式为："2006-01-02 15:04:05.999999999 -0700 MST"

8.func (t Time) Unix() int64

Unix将t表示为Unix时间（时间戳，int64），即从时间点January 1, 1970 UTC到时间点t所经过的时间（单位秒）。

9.func (t Time) UnixNano() int64

UnixNano将t表示为Unix时间，即从时间点January 1, 1970 UTC到时间点t所经过的时间（单位纳秒）。

10.func (t Time) Equal(u Time) bool

判断两个时间是否相同，会考虑时区的影响，因此不同时区标准的时间也可以正确比较。本方法和用t==u不同，这种方法还会比较地点和时区信息。

11.func (t Time) Before(u Time) bool

如果t代表的时间点在u之前，返回真；否则返回假。

12.func (t Time) After(u Time) bool

如果t代表的时间点在u之后，返回真；否则返回假。

13.func (t Time) Date() (year int, month Month, day int)

返回时间点t对应的年.月.日。

14.func (t Time) Year() int

返回时间点t对应的年份。

15.func (t Time) Month() Month

返回时间点t对应那一年的第几月。

16.func (t Time) Day() int

返回时间点t对应那一月的第几日。

17.func (t Time) Weekday() Weekday

返回时间点t对应的那一周的周几。

18.func (t Time) Clock() (hour, min, sec int)

返回t对应的那一天的时.分.秒。

19.func (t Time) Hour() int

返回t对应的那一天的第几小时，范围[0, 23]。

20.func (t Time) Minute() int

返回t对应的那一小时的第几分种，范围[0, 59]。

21.func (t Time) Second() int

返回t对应的那一分钟的第几秒，范围[0, 59]。

22.func (t Time) Nanosecond() int

返回t对应的那一秒内的纳秒偏移量，范围[0, 999999999]。

23.func (t Time) Sub(u Time) Duration

返回一个时间段t-u。如果结果超出了Duration可以表示的最大值/最小值，将返回最大值/最小值。要获取时间点t-d（d为Duration），可以使用t.Add(-d)。

24.func (d Duration) Hours() float64

Hours将时间段表示为float64类型的小时数。

25.func (d Duration) Minutes() float64

Minutes将时间段表示为float64类型的分钟数。

26.func (d Duration) Seconds() float64

Seconds将时间段表示为float64类型的秒数。

27.func (d Duration) Nanoseconds() int64

Nanoseconds将时间段表示为int64类型的纳秒数，等价于int64(d)。

28.func (d Duration) String() string

返回时间段采用"72h3m0.5s"格式的字符串表示。最前面可以有符号，数字+单位为一个单元，开始部分的0值单元会被省略；如果时间段<1s，会使用"ms"."us"."ns"来保证第一个单元的数字不是0；如果时间段为0，会返回"0"。

29.func ParseDuration(s string) (Duration, error)

ParseDuration解析一个时间段字符串。一个时间段字符串是一个序列，每个片段包含可选的正负号.十进制数.可选的小数部分和单位后缀，如"300ms"."-1.5h"."2h45m"。合法的单位有"ns"."us" /"µs"."ms"."s"."m"."h"。

30.func (t Time) Add(d Duration) Time

Add返回时间点t+d。

31.func (t Time) AddDate(years int, months int, days int) Time

AddDate返回增加了给出的年份.月份和天数的时间点Time。例如，时间点January 1, 2011调用AddDate(-1, 2, 3)会返回March 4, 2010。

AddDate会将结果规范化，类似Date函数的做法。因此，举个例子，给时间点October 31添加一个月，会生成时间点December 1。（从时间点November 31规范化而来）

## math包

使用时需要import "math"，math包提供了基本的数学常数和数学函数。

### math包中函数

func NaN() float64

func IsNaN(f float64) (is bool)

func Inf(sign int) float64

func IsInf(f float64, sign int) bool

func Float32bits(f float32) uint32

func Float32frombits(b uint32) float32

func Float64bits(f float64) uint64

func Float64frombits(b uint64) float64

func Signbit(x float64) bool

func Copysign(x, y float64) float64

func Ceil(x float64) float64

func Floor(x float64) float64

func Trunc(x float64) float64

func Modf(f float64) (int float64, frac float64)

func Nextafter(x, y float64) (r float64)

func Abs(x float64) float64

func Max(x, y float64) float64

func Min(x, y float64) float64

func Dim(x, y float64) float64

func Mod(x, y float64) float64

func Remainder(x, y float64) float64

func Sqrt(x float64) float64

func Cbrt(x float64) float64

func Hypot(p, q float64) float64

func Sin(x float64) float64

func Cos(x float64) float64

func Tan(x float64) float64

func Sincos(x float64) (sin, cos float64)

func Asin(x float64) float64

func Acos(x float64) float64

func Atan(x float64) float64

func Atan2(y, x float64) float64

func Sinh(x float64) float64

func Cosh(x float64) float64

func Tanh(x float64) float64

func Asinh(x float64) float64

func Acosh(x float64) float64

func Atanh(x float64) float64

func Log(x float64) float64

func Log1p(x float64) float64

func Log2(x float64) float64

func Log10(x float64) float64

func Logb(x float64) float64

func Ilogb(x float64) int

func Frexp(f float64) (frac float64, exp int)

func Ldexp(frac float64, exp int) float64

func Exp(x float64) float64

func Expm1(x float64) float64

func Exp2(x float64) float64

func Pow(x, y float64) float64

func Pow10(e int) float64

func Gamma(x float64) float64

func Lgamma(x float64) (lgamma float64, sign int)

func Erf(x float64) float64

func Erfc(x float64) float64

func J0(x float64) float64

func J1(x float64) float64

func Jn(n int, x float64) float64

func Y0(x float64) float64

func Y1(x float64) float64

func Yn(n int, x float64) float64

### math包中核心函数介绍

1.func IsNaN(f float64) (is bool)

报告f是否表示一个NaN（Not A Number）值。

2.func Ceil(x float64) float64

返回不小于x的最小整数（的浮点值）

3.func Floor(x float64) float64

返回不大于x的最小整数（的浮点值）

4.func Trunc(x float64) float64

返回x的整数部分（的浮点值）。

5.func Abs(x float64) float64

返回x的绝对值

6.func Max(x, y float64) float64

返回x和y中最大值

7.func Min(x, y float64) float64

返回x和y中最小值

8.func Dim(x, y float64) float64

函数返回x-y和0中的最大值

9.func Mod(x, y float64) float64

取余运算，可以理解为 x-Trunc(x/y)\*y，结果的正负号和x相同

10.func Sqrt(x float64) float64

返回x的二次方根

11.func Cbrt(x float64) float64

返回x的三次方根，特例如下：

12.func Hypot(p, q float64) float64

返回Sqrt(p\*p + q\*q)

13.func Pow(x, y float64) float64

返回x^y

14.func Sin(x float64) float64

求正弦。

15.func Cos(x float64) float64

求余弦。

16.func Tan(x float64) float64

求正切。

17.func Log(x float64) float64

求自然对数

18.func Log2(x float64) float64

求2为底的对数。

19.func Log10(x float64) float64

求10为底的对数。

## 随机数(math/rand包)

1.使用时需要import "math/rand"，rand包实现了伪随机数生成器。

2.随机数从资源生成。包水平的函数都使用的默认的公共资源。该资源会在程序每次运行时都产生确定的序列。如果需要每次运行产生不同的序列，应使用Seed函数进行初始化。默认资源可以安全的用于多协程并发。

### rand包中类型及方法

1.type Source

func NewSource(seed int64) Source

2.type Rand

func New(src Source) \*Rand

func (r \*Rand) Seed(seed int64)

func (r \*Rand) Int() int

func (r \*Rand) Int31() int32

func (r \*Rand) Int63() int64

func (r \*Rand) Uint32() uint32

func (r \*Rand) Intn(n int) int

func (r \*Rand) Int31n(n int32) int32

func (r \*Rand) Int63n(n int64) int64

func (r \*Rand) Float32() float32

func (r \*Rand) Float64() float64

func (r \*Rand) NormFloat64() float64

func (r \*Rand) ExpFloat64() float64

func (r \*Rand) Perm(n int) []int

3.type Zipf

func NewZipf(r \*Rand, s float64, v float64, imax uint64) \*Zipf

func (z \*Zipf) Uint64() uint64

func Seed(seed int64)

func Int() int

func Int31() int32

func Int63() int64

func Uint32() uint32

func Intn(n int) int

func Int31n(n int32) int32

func Int63n(n int64) int64

func Float32() float32

func Float64() float64

func NormFloat64() float64

func ExpFloat64() float64

func Perm(n int) []int

### rand包中核心方法介绍

1.func NewSource(seed int64) Source

使用给定的种子创建一个伪随机资源。

2.func New(src Source) \*Rand

返回一个使用src生产的随机数来生成其他各种分布的随机数值的\*Rand。

3.func (r \*Rand) Seed(seed int64)

使用给定的seed来初始化生成器到一个确定的状态。

4.func (r \*Rand) Int() int

返回一个非负的伪随机int值。

5.func (r \*Rand) Intn(n int) int

返回一个取值范围在[0,n)的伪随机int值，如果n<=0会panic。

6.func (r \*Rand) Float64() float64

返回一个取值范围在[0.0, 1.0]的伪随机float64值。

### 获取随机数的几种方式：

1.通过默认随机数种子获取随机数

rand.Int()

rand.Float64()

rand.Intn(n) // 例如获取0-n之间随机数

总是生成固定的随机数。默认情况下，随机数种子都是1。seed是一个64位整数。

2.动态随机数种子生成随机资源，实例随机对象来获取随机数

s1 := rand.NewSource(time.Now().UnixNano())

r1 := rand.New(s1)

randnum := r1.Intn(n) // 例如获取0-n之间随机数

3.简写形式：动态变化随机数种子来获取随机数

（1）.获取整型随机数[0,10]

rand.Seed(time.Now().UnixNano()）

rand.Intn(10)

（2）. 获取浮点型0.0至1.0之间的随机数

rand.Seed(time.Now().UnixNano())

rand.Float64()

（3）. 获取两数之间随机数[m , n]

rand.Seed(time.Now().UnixNano())

随机数 = rand.Intn(n - m + 1) + m

例如：获取[5,11]之间随机数： rand.Intn(7) + 5

## 键盘输入

### scanln

1.fmt.scanln()

2.示例代码：

package main

import "fmt"

func main() {

username := ""

age := 0

fmt.Scanln(&username, &age)

fmt.Println("账号信息为：", username, age)

fmt.Printf("用户名是：%q ， 年龄是：%d \n", username, age)

fmt.Printf("用户名是：%s ， 年龄是：%d \n", username, age)

fmt.Println(&username)

}

### 随机数+键盘输入案例——猜数字游戏

package main

import (

"fmt"

"math/rand"

"strings"

"time"

)

func main() {

play()

}

func play() {

target := generateRandNum(10, 100)

//fmt.Println("产生随机数：", target)

fmt.Println("请输入随机数：")

fmt.Println(strings.Repeat("-" , target))

//记录猜测的次数

count := 0

for {

count++

yourNum := 0

fmt.Scanln(&yourNum)

//fmt.Scanf("%d", &yourNum)

if yourNum < target {

fmt.Println("小了❌")

} else if yourNum > target {

fmt.Println("大了❌")

} else {

fmt.Println("正确✅")

fmt.Printf("您一共猜测了 %d 次！\n", count)

fmt.Println("-------------------------")

play()

}

alertInfo(count, target)

}

}

//生成随机数

func generateRandNum(min int, max int) int {

rand.Seed(time.Now().UnixNano())

return rand.Intn(max-min+1) + min

}

//提示信息

func alertInfo(count int, target int) {

if count >= 6 {

fmt.Printf("您一共猜测了 %d 次都没有猜中，太笨了！😓\n", count)

fmt.Println("正确结果是：", target)

fmt.Println("-----------------------------")

fmt.Println("")

play()

}

}

[第6章 strings包和time及math包 1](#_Toc241629271)

[6.1 字符串处理包概述 1](#_Toc171581220)

[6.1.1 strings与strconv包简介： 1](#_Toc1842510266)

[6.1.2 string的基本用法 1](#_Toc355850922)

[6.2 strings包中的字符串处理函数 3](#_Toc44489159)

[6.2.1 检索字符串 3](#_Toc404986157)

[6.2.2 分割字符串： 4](#_Toc1226663356)

[6.2.3 大小写转换： 5](#_Toc688013092)

[6.2.4 trim修剪函数： 6](#_Toc1384081796)

[6.2.5 比较字符串 7](#_Toc719881068)

[6.3 strconv包中常用函数 8](#_Toc118242678)

[6.3.1 Parse类函数（将字符串转其它类型） 8](#_Toc882315671)

[6.3.2 Format类函数（将其它类型格式化成字符串） 8](#_Toc704899962)

[6.4 time包 9](#_Toc1733864482)

[6.4.1 time包中类型及方法 9](#_Toc1854742831)

[6.4.2 time包中核心方法介绍 12](#_Toc1937624412)

[6.5 math包 16](#_Toc1211469376)

[6.5.1 math包中函数 16](#_Toc873345225)

[6.5.2 math包中核心函数介绍 18](#_Toc262469330)

[6.6 随机数(math/rand包) 20](#_Toc390618372)

[6.6.1 rand包中类型及方法 20](#_Toc265469325)

[6.6.2 rand包中核心方法介绍 22](#_Toc1419410456)

[6.6.3 获取随机数的几种方式： 22](#_Toc1783183116)

[6.7 键盘输入 23](#_Toc1824336727)

[6.7.1 scanln 23](#_Toc2003342470)

[6.7.2 随机数+键盘输入案例——猜数字游戏 24](#_Toc1928275624)