Go语言 标准库 builtin包

builtin包提供了一些类型声明、变量和常量声明，还有一些便利函数，这个包不需要导入，这些变量和函数就可以直接使用。

常用函数

append

```
func append(slice []Type, elems ...Type) []Type
```

应用	说明
`slice = append(slice, elem1, elem2)`	直接在slice后面添加单个元素，添加元素类型可以和slice相同，也可以不同
`slice = append(slice, anotherSlice)`	直接将另外一个slice添加到slice后面，但其本质还是讲anotherSlice中的元素一个一个添加到slice中，和上一种方式类似

示例：

```
package main

import "fmt"

func main() {
    s1 := []int{1, 2, 3}
    i := append(s1, 4)
    fmt.Printf("i: %v\n", i)

    s2 := []int{7, 8, 9}
    i2 := append(s1, s2...)
    fmt.Printf("i2: %v\n", i2)
}
#结果
i: [1 2 3 4]
i2: [1 2 3 7 8 9]
```

len

返回，数组、切片、字符串、通道的长度

示例：

```
package main

import "fmt"
```

```
func main() {
    s1 := "hello world"
    i := len(s1)
    fmt.Printf("i: %v\n", i)

    s2 := []int{1, 2, 3}
    fmt.Printf("len(s2): %v\n", len(s2))
}
#结果
i: 11
len(s2): 3
```

print、println

打印输出到控制台。

示例:

```
package main

import "fmt"

func main() {
    name := "tom"
    age := 20
    print(name, " ", age, "\n")
    fmt.Println("---------")
    println(name, " ", age)
}
#结果
tom 20
---------
tom   20
```

panic

抛出一个panic异常

```
package main

import "fmt"

func main() {
    defer fmt.Println("panic 异常后执行...")
    panic("panic 错误...")
    fmt.Println("end...")
}
#结果
panic 异常后执行...
panic: panic 错误...

goroutine 1 [running]:
main.main()
```

```
    d:/GoPro/hello.go:7 +0x73
exit status 2
```

> 可以看出，在抛出panic异常之后的代码块将不会执行，而在抛出panic异常之前的defer函数仍然被调用执行。

new和make

new和make区别：

1. make只能用来分配及初始化类型为slice，map，chan的数据；new可以分配任意类型的数据
2. new分配返回的是指针，即类型*T；make返回引用，即T；
3. new分配的空间被清零，make分配后，会进行初始化。

new

示例：

```
package main

import (
    "fmt"
)

func testNew() {
    b := new(bool)
    fmt.Println(*b)
    i := new(int)
    fmt.Println(*i)
    s := new(string)
    fmt.Println(*s)
}

func main() {
    testNew()
}
#结果
false
0
```

make

内建函数make(T, args)与new(T)的用途不一样。它只用来创建slice、map和channel，并且返回一个初始化的（而不是置零），类型为T的值（而不是*T）。之所以有所不同，是因为这三个类型的背后引用了使用前必须初始化的数据结构。例如，slice是一个三元描述符，包含一个指向数据（在数组中）的指针，长度，以及容量，在这些项被初始化之前，slice都是nil的。对于slice，map和channel，make初始化这些内部数据结构，并准备好可用的值。

例如：

```
make([]int, 10 , 100)
```

说明：分配一个有100个int的数组，然后创建一个长度为10，容量为100的slice结构，该slice引用包含前10个元素的数组。对应的，new([]int)返回一个指向新分配的，被置零的slice结构体的指针，即指向值为nil的slice的指针。

示例：

```
package main

import "fmt"

func main() {
    var p *[]int = new([]int)     // allocates slice structure; *p == ni; rarely
useful
    var v []int = make([]int, 10) // the slice v now refers to a new array of 100
ints

    fmt.Printf("p: %v\n", p)
    fmt.Printf("v: %v\n", v)

    var p1 *[]int = new([]int)
    *p1 = make([]int, 5, 10)

    // Idiomatic: 习惯的做法
    v1 := make([]int, 10)

    fmt.Printf("p1: %v\n", p1)
    fmt.Printf("v1: %v\n", v1)
}
#结果
p: &[]
v: [0 0 0 0 0 0 0 0 0 0]
p1: &[0 0 0 0 0]
v1: [0 0 0 0 0 0 0 0 0 0]
```