**1、定时器**

Go语言中定时器可以实现在指定时间点执行特定的事件，定时器的实质是单向通道，time.Timer结构体类型中有一个time.Time类型的单向chan，具体声明如下：

type Timer struct {

    C <-chan Time

    r runtimeTimer

}

只能通过两种方式来进行初始化：time.NewTimer()和time.AfterFunc()，看如下代码：

**time.NewTimer():**

// NewTimer方法接受一个时间段d表示自此刻起经历d时间段后，定时器到期；返回值为\*Timer类型，到期后将到期时的时间写入\*Timer的C(chan time.Time)字段中

    t := time.NewTimer(2 \* time.Second)

    now := time.Now()

    fmt.Printf("Now time: %v\n", now)

    // 改行代码会阻塞当前的goroutine，直到t.C中传入一个元素

    expire := <-t.C

    fmt.Printf("Expire time: %v\n", expire)

**time.AfterFunc():**

var t \*time.Timer

    f := func() {

        fmt.Printf("Expriation time: %v\n", time.Now())

        fmt.Printf("C`s length: %v\n", len(t.C))

    }

    // 在定时器过期时执行一个自定义的无接受参数和输出参数的函数

    t = time.AfterFunc(1\*time.Second, f)

    // 为了结果输出完全

    time.Sleep(2 \* time.Second)

如果不使用时器的Stop()和Reset()方法，可以直接用定时器的快捷方法，如下：

c := <-time.After(5 \* time.Second)

如果在定时器到期之前，使用Stop()，那么就不会再有元素写入通道内，那么等待接受该通道元素所在的goroutine将被阻塞，恢复被停止的定时器的唯一途径是使用Reset()方法重置；定时器可以复用，尤其是在for循环中复用可以减少程序的资源占用，这时需要Reset()方法来重置定时器。

**2、断续器**

Go提供了循环多次的执行某一任务的工具，续断器，实质是单向通道，time.结构体类型中有一个time.Time类型的单向chan，具体声明如下：

type Ticker struct {

    C <-chan Time

    r runtimeTimer

}

续断器使用NewTicker()来初始化，代码如下：

fmt.Println("begin")  
//初始化间隔期，间隔2s  
ticks := time.NewTicker(5 \* time.Second)  
//指定执行次数  
//c := make(chan int, 2)  
i := 0  
// for...range...相当于从通道取一个元素  
for \_ = range ticks.C{  
 //c <- 1  
 fmt.Printf("%vExecute the task.\n", time.Now())  
 fmt.Println(i)  
 i++  
}  
//time.Sleep(1 \* time.Minute)  
//ticks.Stop()  
fmt.Println("Done.")

如果不使用续断器的Stop()方法，可以直接使用续断器的快捷方法，如下

tick := <-time.Tick(5 \* time.Second)