Go语言 select 多路复用

在某些场景下我们需要同时从多个通道接收数据。通道在接收数据时，如果没有数据可以接收将会发生阻塞。你也许会写出如下代码使用遍历的方式来实现：

```
for{
    // 尝试从ch1接收值
    data, ok := <-ch1
    // 尝试从ch2接收值
    data, ok := <-ch2
    …
}
```

这种方式虽然可以实现从多个通道接收值的需求，但是运行性能会差很多。为了应对这种场景，Go内置了select关键字，可以同时响应多个通道的操作。

select的使用类似于switch语句，它有一系列case分支和一个默认的分支。每个case会对应一个通道的通信（接收或发送）过程。select会一直等待，直到某个case的通信操作完成时，就会执行case分支对应的语句。具体格式如下：

```
    select {
    case <-chan1:
       // 如果chan1成功读到数据，则进行该case处理语句
    case chan2 <- 1:
       // 如果成功向chan2写入数据，则进行该case处理语句
    default:
       // 如果上面都没有成功，则进入default处理流程
    }
```

1. select是Go中的一个控制结构，类似于`switch`语句，用于处理异步IO操作。`select`会监听case语句中channel的读写操作，当case中channel读写操作为非阻塞状态（即能读写）时，将会触发相应的动作。

   > select中的case语句必须是一个channel操作
   >
   > select中的default子句总是可运行的。

2. 如果有多个`case`都可以运行，`select`会随机公平地选出一个执行，其他不会执行。
3. 如果没有可运行的`case`语句，且有`default`，那么就会执行default的动作。
4. 如果没有可运行的case语句，且没有default语句，select阻塞，直到某个case通信可以运行

- select可以**同时监听一个或多个channel**，直到其中一个channel ready

```
package main

import (
    "fmt"
    "time"
)

func test1(ch chan string) {
```

```
    time.Sleep(time.Second * 5)
    ch <- "test1"
}
func test2(ch chan string) {
    time.Sleep(time.Second * 2)
    ch <- "test2"
}

func main() {
    // 2个管道
    output1 := make(chan string)
    output2 := make(chan string)
    // 跑2个子协程，写数据
    go test1(output1)
    go test2(output2)
    // 用select监控
    select {
    case s1 := <-output1:
        fmt.Println("s1=", s1)
    case s2 := <-output2:
        fmt.Println("s2=", s2)
    }
}
```

- 如果多个channel同时ready，则随机选择一个执行

```
package main

import (
    "fmt"
)

func main() {
    // 创建2个管道
    int_chan := make(chan int, 1)
    string_chan := make(chan string, 1)
    go func() {
        //time.Sleep(2 * time.Second)
        int_chan <- 1
    }()
    go func() {
        string_chan <- "hello"
    }()
    select {
    case value := <-int_chan:
        fmt.Println("int:", value)
    case value := <-string_chan:
        fmt.Println("string:", value)
    }
    fmt.Println("main结束")
}
```

- 可以用于判断管道是否存满

```

```

```go
package main

import (
    "fmt"
    "time"
)

// 判断管道有没有存满
func main() {
    // 创建管道
    output1 := make(chan string, 10)
    // 子协程写数据
    go write(output1)
    // 取数据
    for s := range output1 {
        fmt.Println("res:", s)
        time.Sleep(time.Second)
    }
}

func write(ch chan string) {
    for {
        select {
        // 写数据
        case ch <- "hello":
            fmt.Println("write hello")
        default:
            fmt.Println("channel full")
        }
        time.Sleep(time.Millisecond * 500)
    }
}
```