Go 边看边练 -《Go 学习笔记》系列（十四）（已完结）

作者：88250

原文链接：https://ld246.com/article/1439194647152

来源网站：链滴

上一篇： https://hacpai.com/article/1438938175118

ToC

7.2.1 单向

可以将 channel 隐式转换为单向队列，只收或只发。

```
c := make(chan int, 3)
var send chan<- int = c // send-only
var recv <-chan int = c // receive-only
send <- 1
// <-send // Error: receive from send-only type chan<- int
<-recv
// recv <- 2 // Error: send to receive-only type <-chan int
```

不能将单向 channel 转换为普通 channel。

```
d := (chan int)(send) // Error: cannot convert type chan<- int to type chan int
d := (chan int)(recv) // Error: cannot convert type <-chan int to type chan int
```

7.2.2 选择

如果需要同时处理多个 channel，可使用 select 语句。它随机选择一个可用 channel 做收发操作，执行 default case。

<iframe style="border:1px solid" src="https://wide.b3log.org/playground/73b28a1e3a400e666893c7a99e66e0d.go?embed=true" width="99%" height="930"></iframe>

在循环中使用 select default case 需要小心，避免形成洪水。

7.2.3 模式

用简单工厂模式打包并发任务和 channel。

```
func NewTest() chan int {
    c := make(chan int)
    rand.Seed(time.Now().UnixNano())

    go func() {
        time.Sleep(time.Second)
        c <- rand.Int()
    }()

    return c
}

func main() {
    t := NewTest()
    println(<-t) // 等待 goroutine 结束返回。
}
```

用 channel 实现信号量 (semaphore)。

<iframe style="border:1px solid" src="https://wide.b3log.org/playground/42099b128a70f0d61acb9eee89ce982.go?embed=true" width="99%" height="730"></iframe>

用 closed channel 发出退出通知。

<iframe style="border:1px solid" src="https://wide.b3log.org/playground/fba53ba1f4116b4c998f67a716160df.go?embed=true" width="99%" height="930"></iframe>

用 select 实现超时 (timeout)。

```
func main() {
    w := make(chan bool)
    c := make(chan int, 2)

    go func() {
        select {
        case v := <-c: fmt.Println(v)
        case <-time.After(time.Second * 3): fmt.Println("timeout.")
        }

        w <- true
    }()
```

```
    // c <- 1 // 注释掉，引发 timeout。
    <-w
}
```

channel 是第一类对象，可传参 (内部实现为指针) 或者作为结构成员。

```
type Request struct {
    data []int
    ret chan int
}

func NewRequest(data ...int) *Request {
    return &Request{ data, make(chan int, 1) }
}

func Process(req *Request) {
    x := 0
    for _, i := range req.data {
        x += i
    }

    req.ret <- x
}

func main() {
    req := NewRequest(10, 20, 30)
    Process(req)
    fmt.Println(<-req.ret)
}
```

全系列完

- **本系列是基于雨痕的《Go 学习笔记》（第四版）整理汇编而成，非常感谢雨痕的辛勤付出与分享！**
- 转载请注明：文章转载自： **黑客与画家的社区** [https://hacpai.com]
- 如果你觉得本章节做得不错，请在下面打赏一下吧~

社区小贴士

- 关注标签 [golang] 可以方便查看 Go 相关帖子
- 关注作者后如有新帖将会收到通知