golang 关闭 channel 的优雅方式学习总结

作者：xhaoxiong
原文链接：https://ld246.com/article/1537117459863
来源网站：链滴

！无论怎样都不应该在接收端关闭channel,因为在接收端无法判断发送端是否还会向通道中发送元素值

！试图向已经关闭的channel发送数据会导致panic

！关闭已经关闭的channel会导致panic

以下均为亲自测试及其总结，demo仅供参考，总结自云栖社区

关闭原则（正常方式）：

不在接收端关闭(常用于1-N,由发送端来关闭)

不要关闭有多个并发发送者的channel(常用于N-1,由接收端来通知)

3种优雅的关闭场景模型：1-N N-1 M-N （生产者-消费者）以及几种不优雅的关闭式

场景1

1 个发送者 1个接收者（最简单的）

发送端直接关闭channel

```
/**
*@Author: haoxiongxiao
*@Date: 2018/9/16
*@Description: CREATE GO FILE close_channel
*/
package main

import (
    "fmt"
    "time"
)

func main() {
    notify := make(chan int)

    datach := make(chan int, 100)

    go func() {
        <-notify
        fmt.Println("2 秒后接受到信号开始发送")
        for i := 0; i < 100; i++ {
            datach <- i
        }
        fmt.Println("发送端关闭数据通道")
        close(datach)

    }()

    time.Sleep(2 * time.Second)
```

```
    fmt.Println("开始通知发送信息")
    notify <- 1
    time.Sleep(1 * time.Second)
    fmt.Println("通知1秒后接收到数据通道数据 ")
    for {
        if i, ok := <-datach; ok {
            fmt.Println(i)

        } else {
            fmt.Println("接收不到数据中止循环")
            break
        }

    }

    time.Sleep(5 * time.Second)
}
```

1个发送者 N个接收者（进行扩展）

```
func main(){
    notify := make(chan int)

    datach := make(chan int, 100)

    go func() {
        <-notify
        fmt.Println("2 秒后接受到信号开始发送")
        for i := 0; i < 100000; i++ {
            datach <- i
        }
        fmt.Println("发送端关闭数据通道")
        close(datach)

    }()

    time.Sleep(2 * time.Second)
    fmt.Println("开始通知发送信息")
    notify <- 1
    time.Sleep(1 * time.Second)
    fmt.Println("3秒后接受到数据通道数据 此时datach 在接收端已经关闭")
    for i := 0; i < 5; i++ {
        go func() {
            for {
                if i, ok := <-datach; ok {
                    fmt.Println(i)

                } else {
                    break
                }
            }
```

```
        }()

    }
    time.Sleep(5 * time.Second)
    }
```

场景2

N 个发送者 1个接收者

添加一个 停止通知 接收端告诉发送端不要发送了

```
/**
*@Author: haoxiongxiao
*@Date: 2018/9/16
*@Description: CREATE GO FILE close_channel
*/
package main

import (
    "fmt"
    "time"
    "math/rand"
)

type T int

func main() {
    dataCh := make(chan T, 1)
    stopCh := make(chan T)
    //notifyCh := make(chan T)
    for i := 0; i < 10000; i++ {
        go func(i int) {

            for {
                value := T(rand.Intn(10000))

                select {
                case <-stopCh:
                    fmt.Println("接收到停止发送的信号")
                    return
                case dataCh <- value:

                }
            }
        }(i)
    }

    time.Sleep(1 * time.Second)
    fmt.Println("1秒后开始接收数据")
    for {
        if d, ok := <-dataCh; ok {
            fmt.Println(d)
```

```
            if d == 9999 {
                fmt.Println("当在接收端接收到9999时告诉发送端不要发送了")
                close(stopCh)
                return
            }
        } else {
            break
        }

    }
    }
```

场景3

N个发送者 M个接收者

```
/**
*@Author: haoxiongxiao
*@Date: 2018/9/16
*@Description: CREATE GO FILE close_channel
*/
package main

import (
    "fmt"
    "time"
    "math/rand"
    "strconv"
    "log"
)

type T int

func main() {

    dataCh := make(chan T, 100)
    toStop := make(chan string)
    stopCh := make(chan T)

    //简约版调度器
    go func() {
        if t, ok := <-toStop; ok {
            log.Println(t)
            close(stopCh)
        }
    }()
    //生产者
    for i := 0; i < 30; i++ {
        go func(i int) {
            for {
                id := strconv.Itoa(i)
                value := T(rand.Intn(10000))
                if value == 9999 {
                    select {
```

```
                case toStop <- "sender# id:" + id + "to close":
                default:

                }
            }

            select {
            case <-stopCh:
                return
            default:

            }

            select {
            case <-stopCh:
                return
            case dataCh <- value:

            }
        }

    }(i)
}
//消费者
for i := 0; i < 20; i++ {
    go func(i int) {
        id := strconv.Itoa(i)
        for {
            select {
            case <-stopCh:
                return
            default:

            }

            select {
            case <-stopCh:
                return
            case value := <-dataCh:
                if value == 9998 {
                    select {
                    case toStop <- "receiver# id:" + id + "to close":
                    default:

                    }
                }
                log.Println("receiver value :", value)
            }
        }
    }(i)
}
time.Sleep(10 * time.Second)
}
```

在M-N 模型中的第一个

```
select {
    case <-stopCh:
        return
    default:

}
```

是为了尽可能早的去退出

以上为了方便测试可以将

```
if value==rand（范围内的值）{

}
```

来进行调试观察

检查channel是否关闭（有局限性）

```
/**
*@Author: haoxiongxiao
*@Date: 2018/9/16
*@Description: CREATE GO FILE close_channel
*/
package main

import (
    "fmt"
)
type T int

func main() {
    ch := make(chan T)
    close(ch)
    fmt.Println(IsClosed(ch))
}

func IsClosed(ch chan T) bool {
    select {
    case <-ch:

        return true
    default:

    }
    return false
}
```

在对channel状态检查之后，channel已经修改,不能反映当时的状态,因此以上函数具有局限性

非正常方式（以下几种从来没尝试过，今天学到了）

1、使用锁来关闭

```
type T int

type MyChannel struct {
    C      chan T
    closed bool
    mutex  sync.Mutex
}

func NewMyChannel() *MyChannel {
    return &MyChannel{C: make(chan T)}
}

func (this *MyChannel) SafeClose() {
    this.mutex.Lock()
    if !this.closed {
        close(this.C)
        this.closed = true
    }

    this.mutex.Unlock()
}

func (this *MyChannel) IsClosed() bool {
    this.mutex.Lock()
    defer this.mutex.Unlock()
    return this.closed
}
```

2、使用sync.once来关闭

```
type MyChannel struct {
     C chan T
     once sync.Once
    }

func NewMyChannel() *MyChannel{
     return &MyChannel{C: make(chan T)}
    }

func (this *MyChannel) SafeClose() {
     this.once.Do(func(){
         close(this.C)
         })
    }
```

3、使用recover()来关闭

```
type T int
```

```
func SafeSend(ch chan T, value T) (closed bool) {
    defer func() {
        if recover() != nil {
            closed = true
        }
    }()

    ch <- value

    return false
}

func SafeClose(ch chan T) (closed bool) {
    defer func() {
        if recover() != nil {
            closed = false
        }
    }()
    close(ch)
    return true
}
```

以上3种非优雅的关闭方式暂未测试，明天再写demo