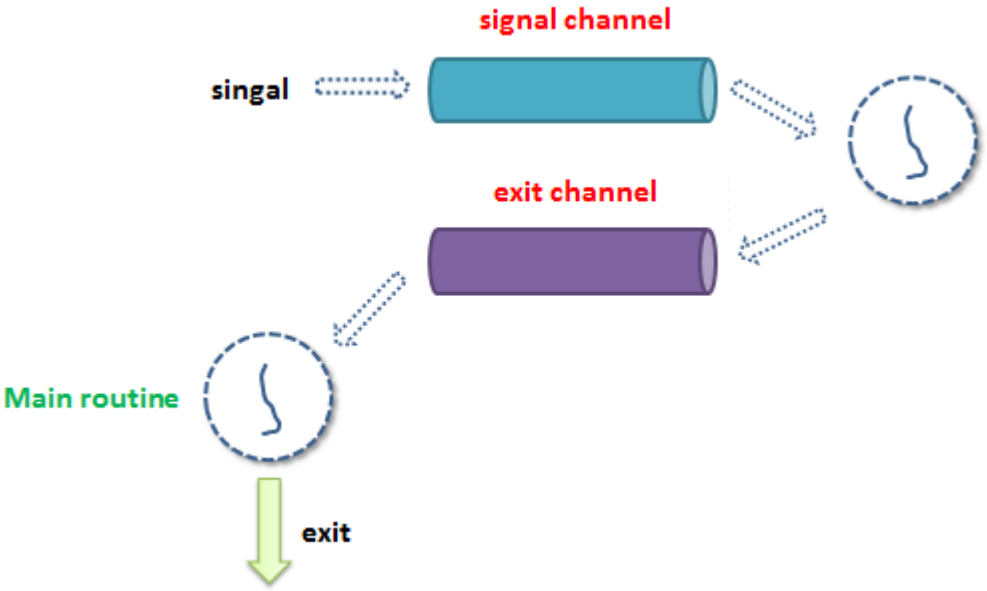
NSQ 编程实践

1. 利用 channel 实现协程优雅退出

- **Example1:**

以 nsqd 为例，主程序在启动时会创建一个 signal channel，一旦产生我们注册的信号，会向 signal channel 写入内容。一个单独的协程阻塞在 signal channel 的读上。在一个单独的协程中，一旦可以读出，就向另外一个退出管道(exit Channel)写入”1”。主协程初始化完成后，一直阻塞在该 exit Channel 的读，读返回说明需要退出了，调用退出函数。




```
exitChan := make(chan int)

signalChan := make(chan os.Signal, 1)

    go func() {

     <-signalChan

     exitChan <- 1

}()

signal.Notify(signalChan, syscall.SIGINT, syscall.SIGTERM)

……

nsqd.Main()
```

<-exitChan

nsqd.Exit()

- **Example 2**

Nsqd 实现的时候同样会创建一个 exitChan 用于它和内部一些协程之间的通信。如 lookup 协程和 statusd 协程

n.waitGroup.Wrap(func() { n.lookupLoop() })

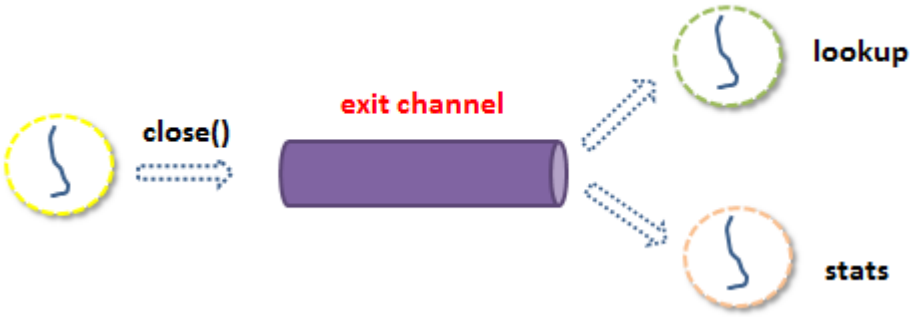
if n.options.StatsdAddress != "" {

n.waitGroup.Wrap(func() { n.statsdLoop() })

}

在这两个协程的处理函数中，都会侦听 nsqd 创建的 exitChan，一旦能读出，就说明需要退出了。而在 nsqd 的 Exit 函数中会调用关闭 chan 的方法，这样那些子协程就能够优雅地退出了。

close(n.exitChan)



2. 利用 channel 实现协程间同步

- **Example**

还是以 nsqd 为例，它有一个 notifyChan 专门用来与 lookup 协程进行同步。nsqd 协程中的 Notify 方法

```
func (n *NSQD) Notify(v interface{}) {
    select {
        case <-n.exitChan:
        case n.notifyChan <- v:
```

```
            n.Lock()
            err := n.PersistMetadata()
            if err != nil {
                log.Printf("ERROR: failed to persist metadata - %s", err.Error())
            }
            n.Unlock()
        }
    }
```

如果有事件发生，那么就向 notifyChan 中写入 v，lookup 协程会阻塞在该 notifyChan 的读事件上：

case val := <-n.notifyChan:

```
var cmd *nsq.Command
    var branch string
        switch val.(type) {
```

这里就实现了主协程中向子协程发送一个通知，并将通知的信息通过 interface 来传递的功能，子协程收到通知并解析出参数，进行接下来的处理（向 nsqlookupd 发送更新 channel 或者 topic 的命令）。

3. Godep 解决第三方包依赖

具体使用方法可参考

- https://github.com/tools/godep
- http://www.goinggo.net/2013_10_01_archive.html