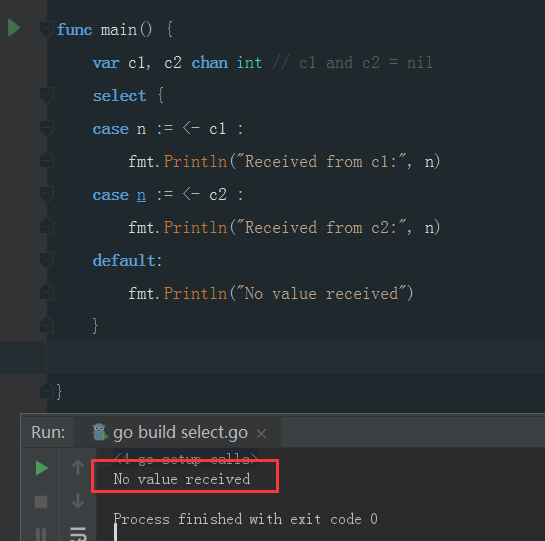
# 什么是select

select是Go中的一个控制结构，类似于用于通信的switch语句。每个case必须是一个通信操作，要么是发送要么是接收。

select随机执行一个可运行的case。如果没有case可运行，它将阻塞，直到有case可运行。

# 非阻塞式通信

Channel收、发数据都是阻塞式的，如果使用select + default，其它case阻塞时就会执行default

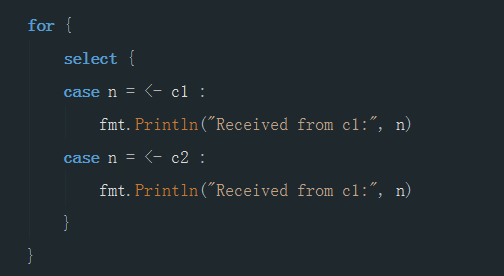


# for + select

**交替处理多个channel通信**

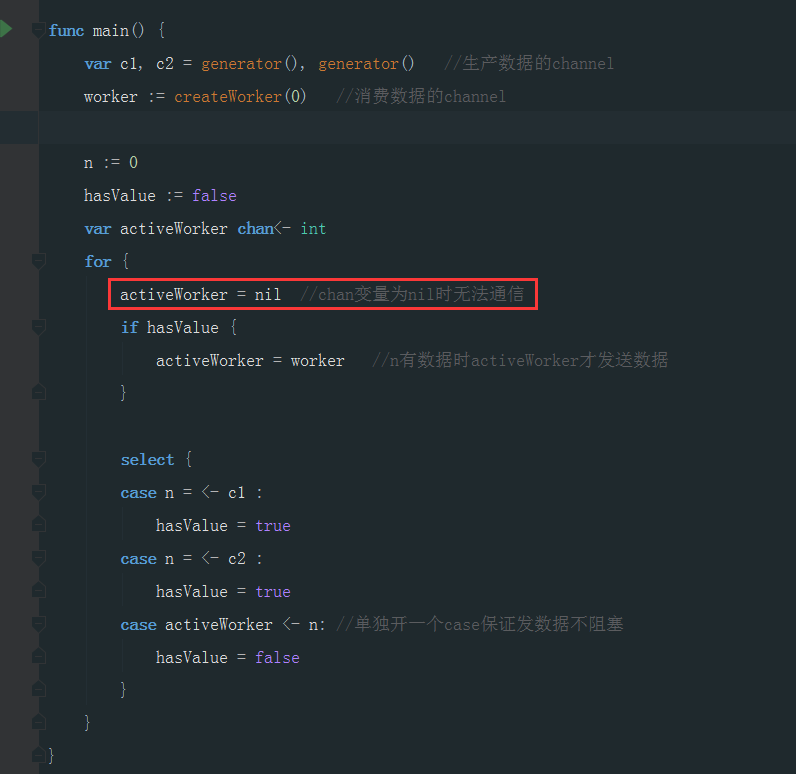
For循环的时候不要加default，因为channel不是时时刻刻都有数据，大多数时候是阻塞的，所以有default就会大部分时候都在执行default的语句

如果没有default字句，select将阻塞，直到某个通信可以运行



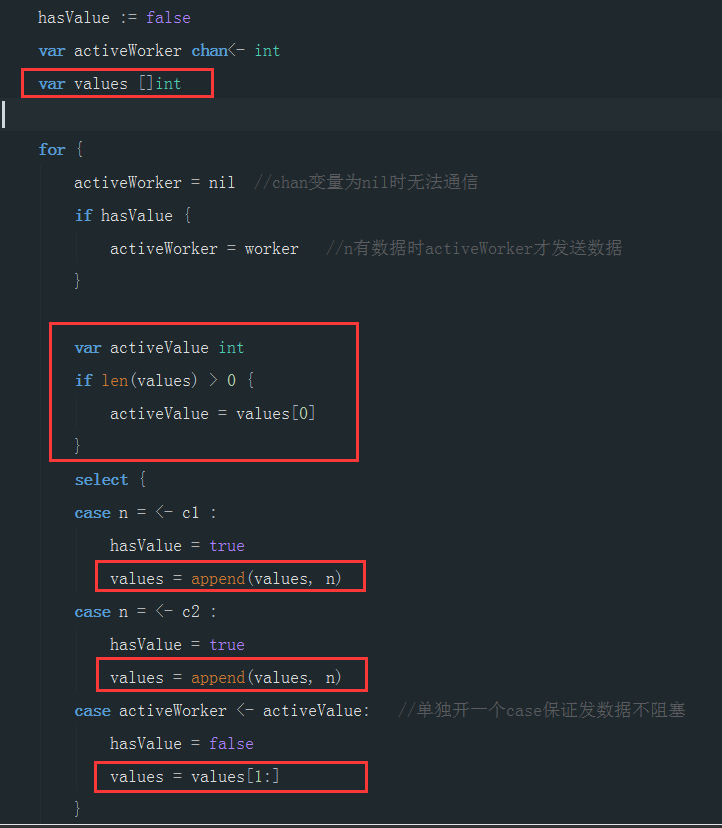
**把channel变量置为nil，使得case子句不成立**

在case子句中channel发送数据，数据源n的数据是由另外一个channel提供的，那么n没有新数据时，必须把channel变量置为nil，以免不断发送重复数据



**将channel收到的数据用slice存起来以免被覆盖**

发送数据和接收数据的速度很可能不是一致的，所以需要用slice把接收到的数据存起来

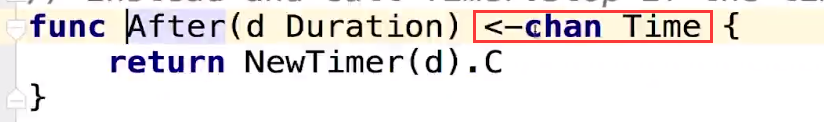


# 计时器Time.After()的使用

**time.After()的返回值是<-chan Time变量。**

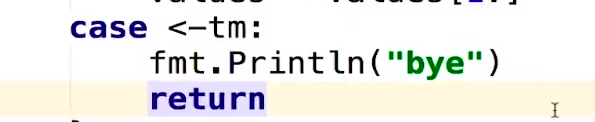
**倒计时后向<-chan Time发送数据。**

可以配合select使用，用来倒计时后执行某操作。



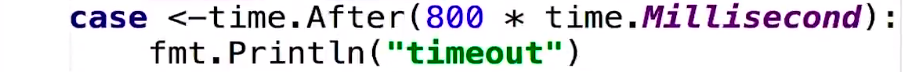
这里10秒后结束select + for





每次执行select都会计算case的值

这里计时器直接写在case里面，每次都会返回新的<-chan Time变量，所以每次都会重新计时。如果800毫秒内没有执行其它case，就会打印timeout。



# 定时器time.Tick()的使用

**time.Tick()的返回值是<-chan Time变量。**

**定时向<-chan Time发送数据。**

每秒钟计算一次values的长度



