第五章：协程和管道

协程

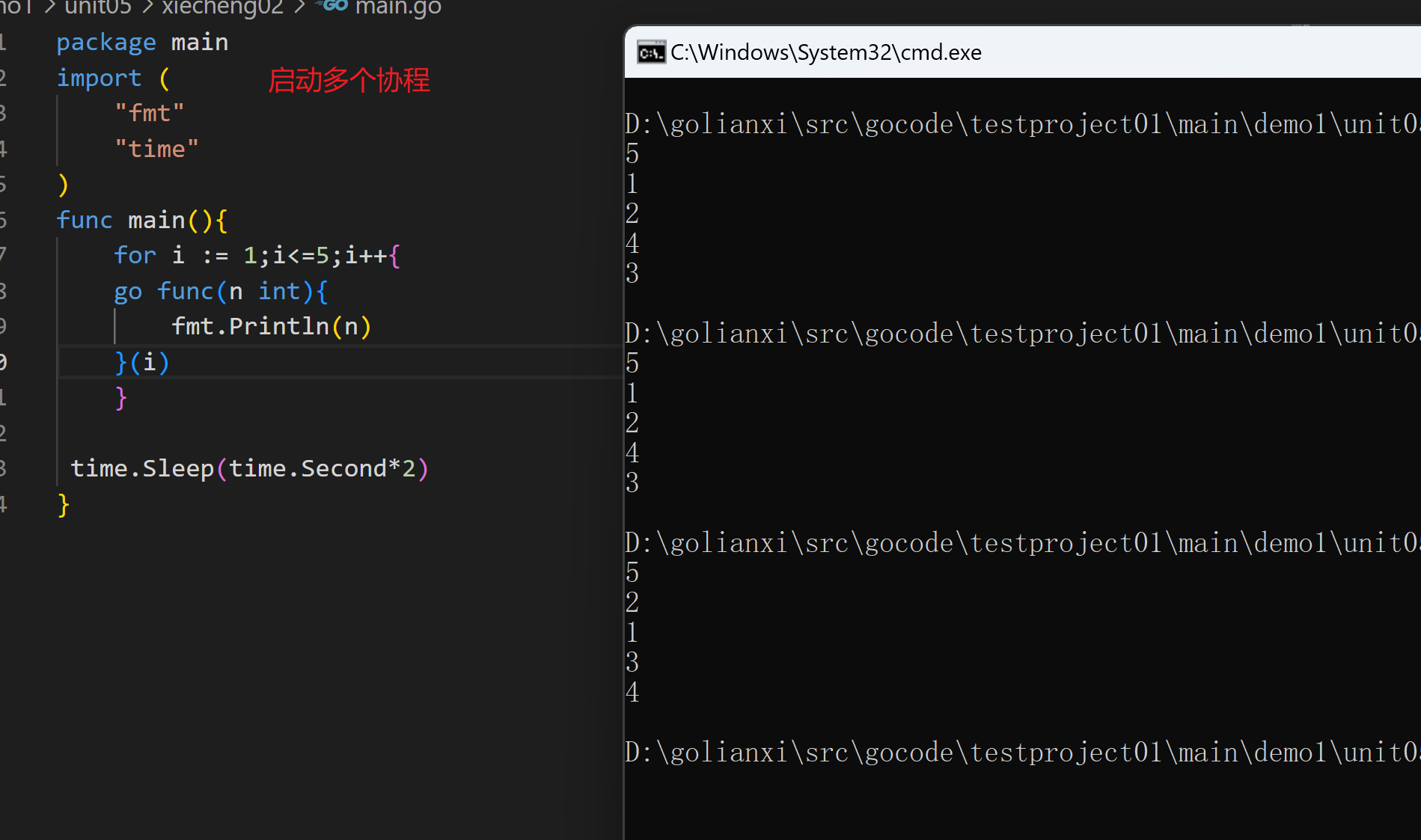
【1】案例:

请编写一个程序，完成如下功能：

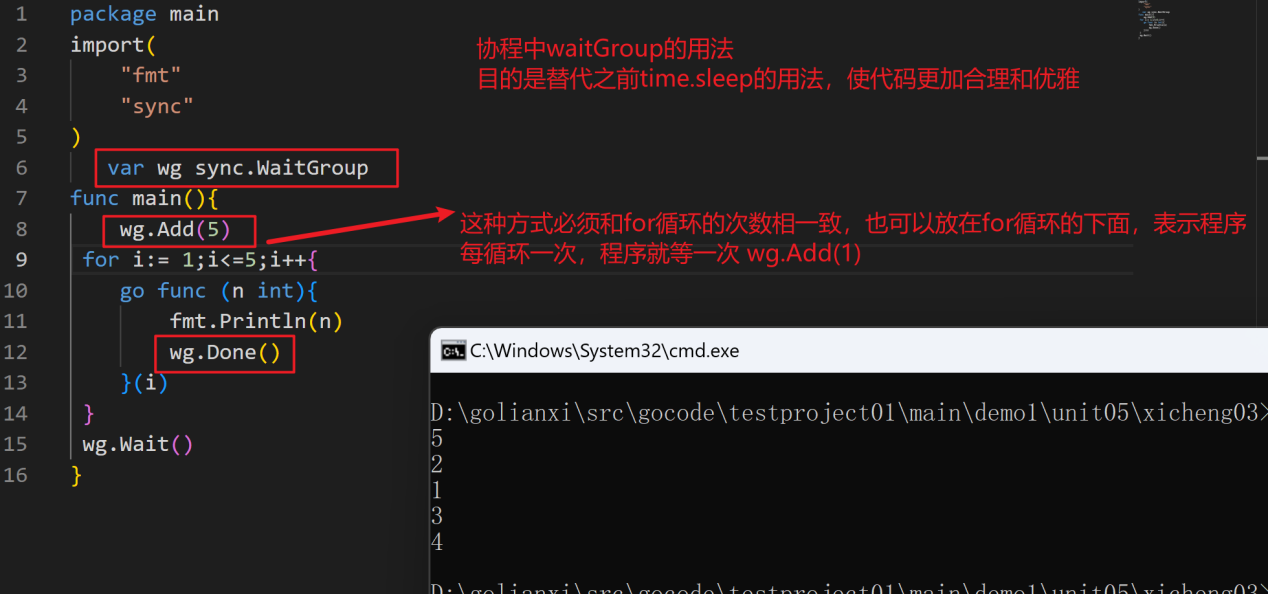
1. 在主线程中，开启一个goroutine，该goroutini每隔1秒输出“hello golang”
2. 在主线程中也每隔一秒输出“hello，msb”，输出10次后，退出程序
3. 要求主线程和goroutine同时执行

代码如下：

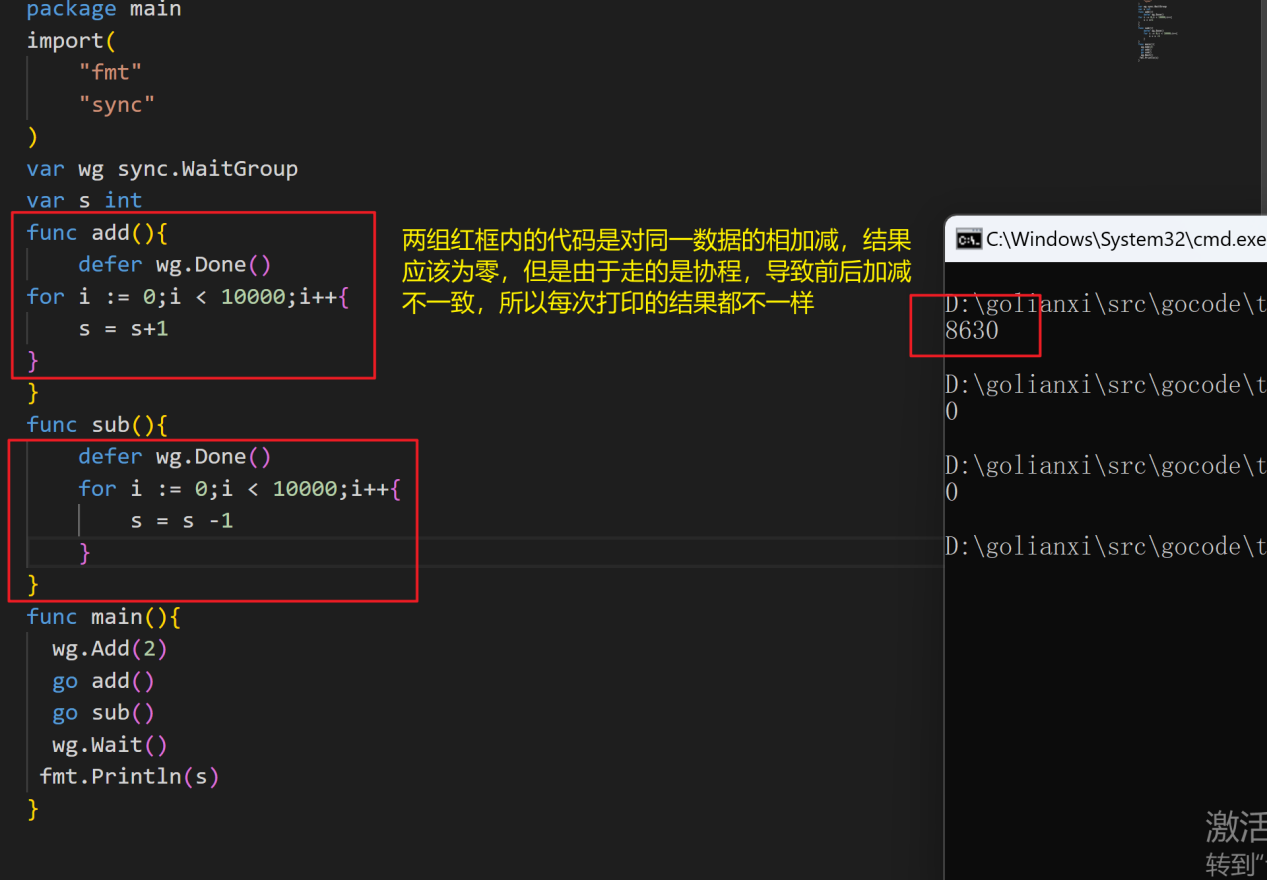
启动多个协程：

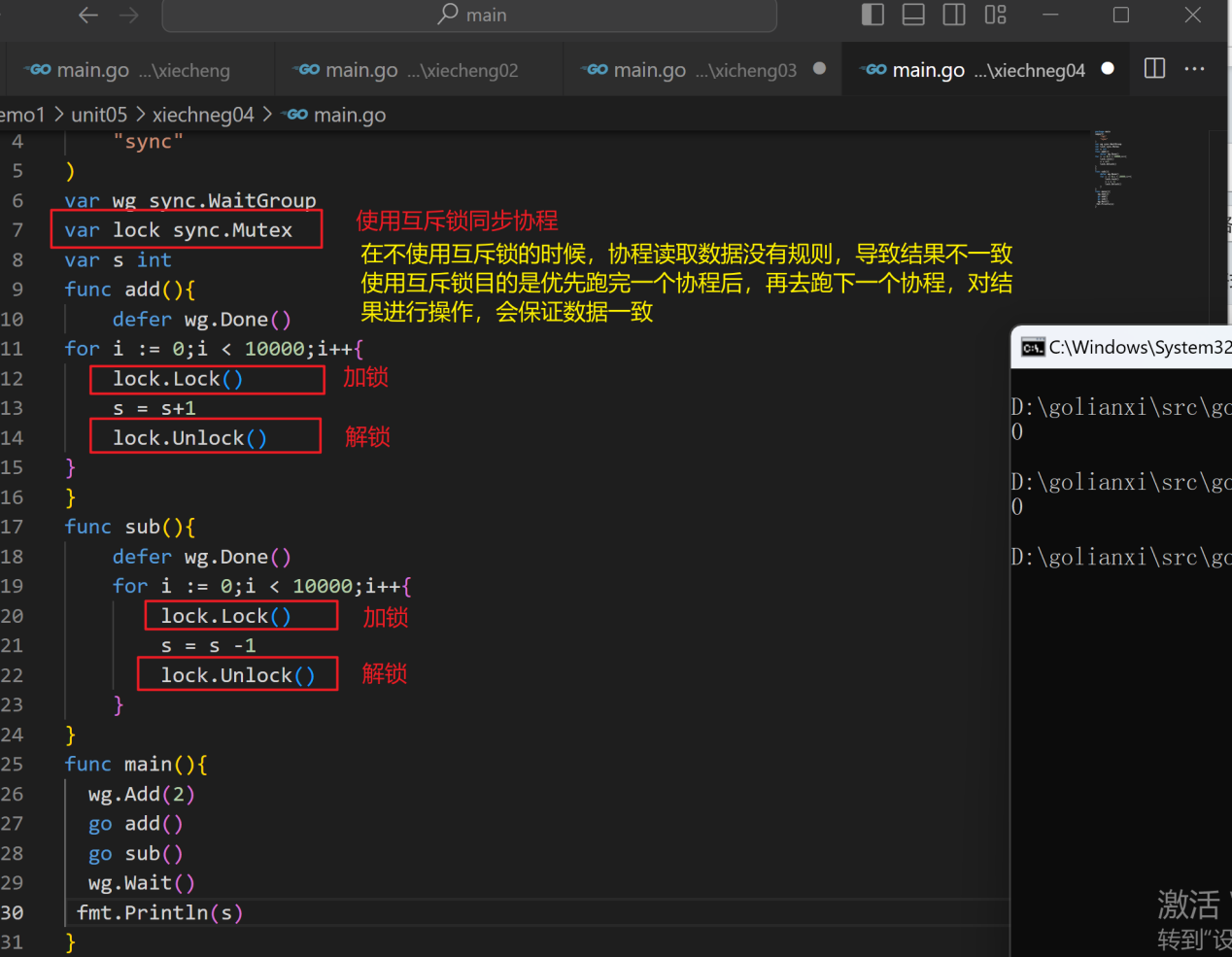


wait.Group的使用场景



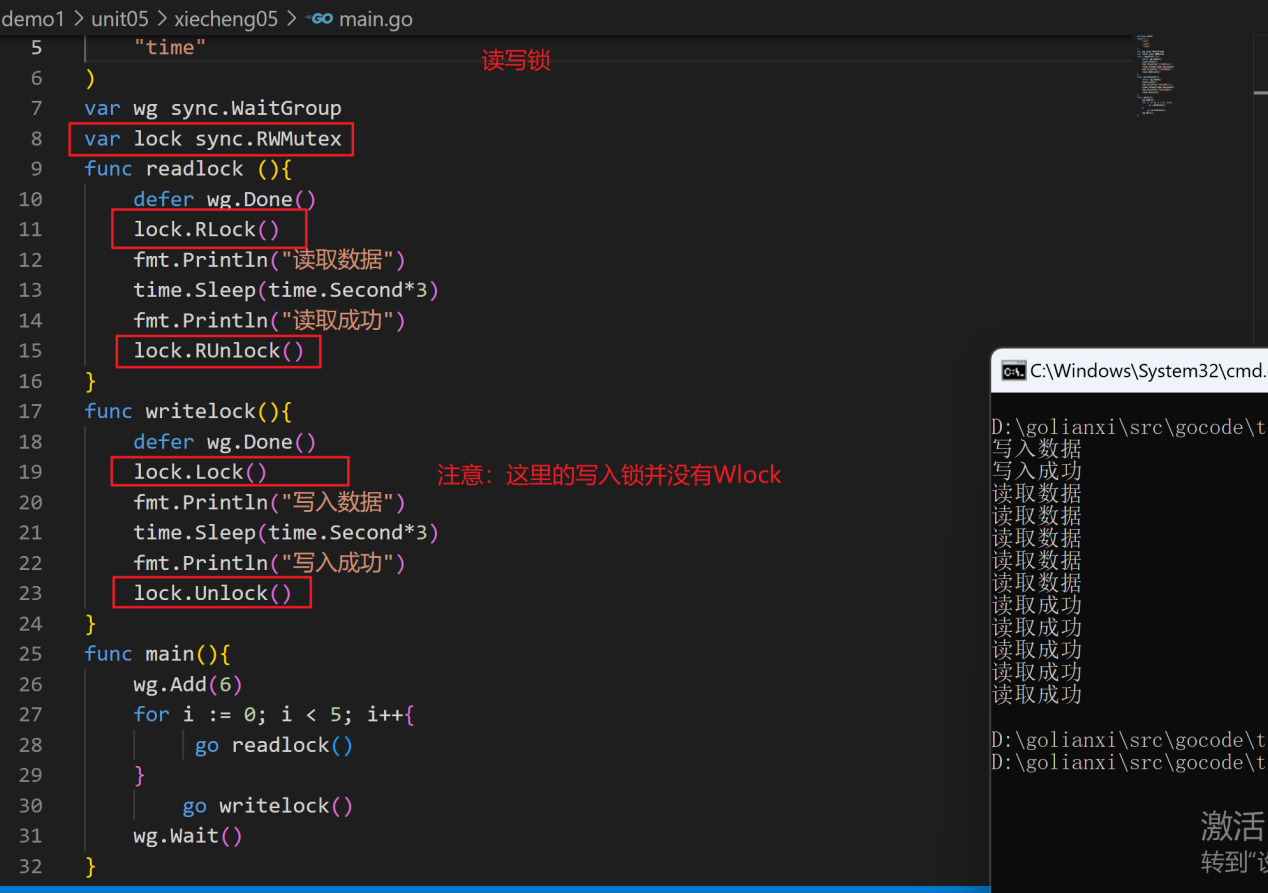
多个协程操作同一数据案例：

互斥锁的使用：



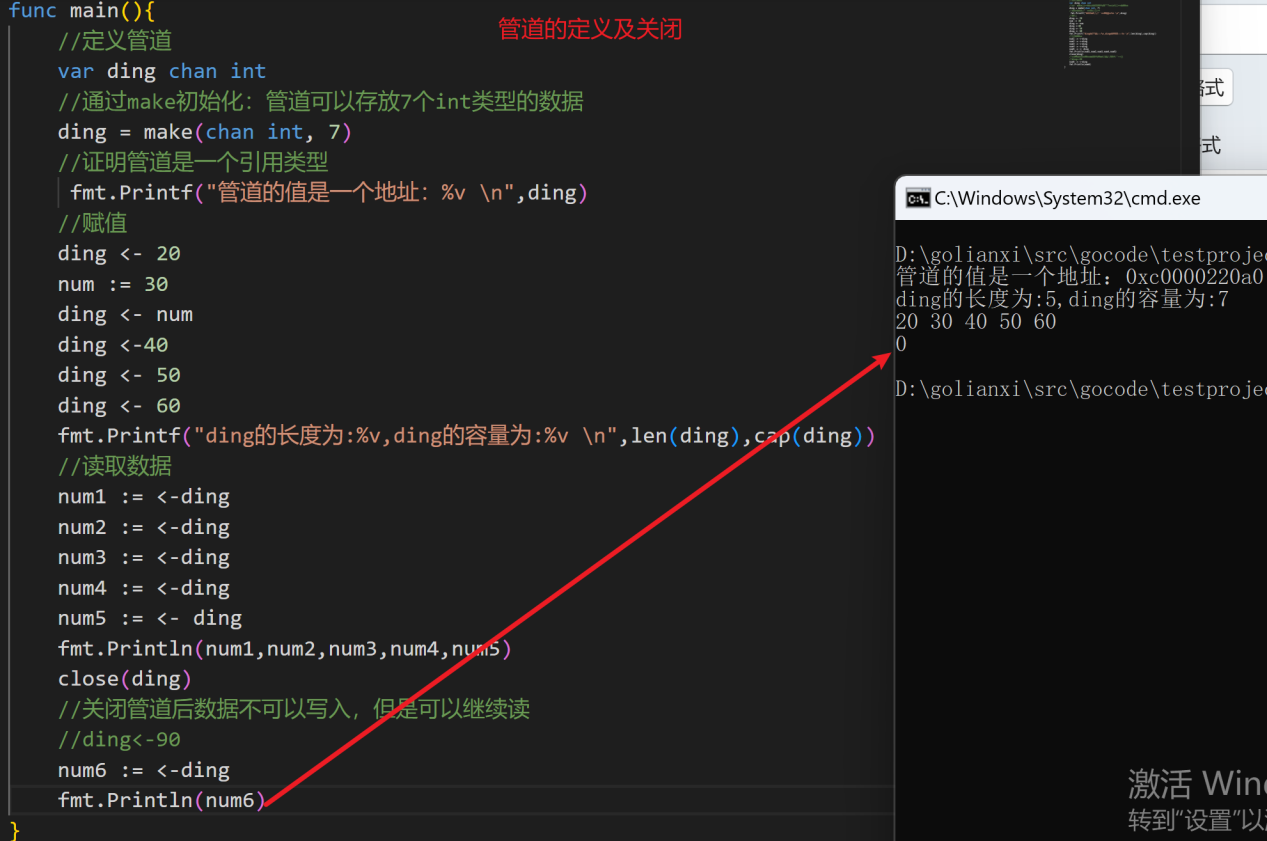
注意：互斥锁的性能和效率偏低

引入读写锁：



管道

管道的定义及如何关闭的注意事项：

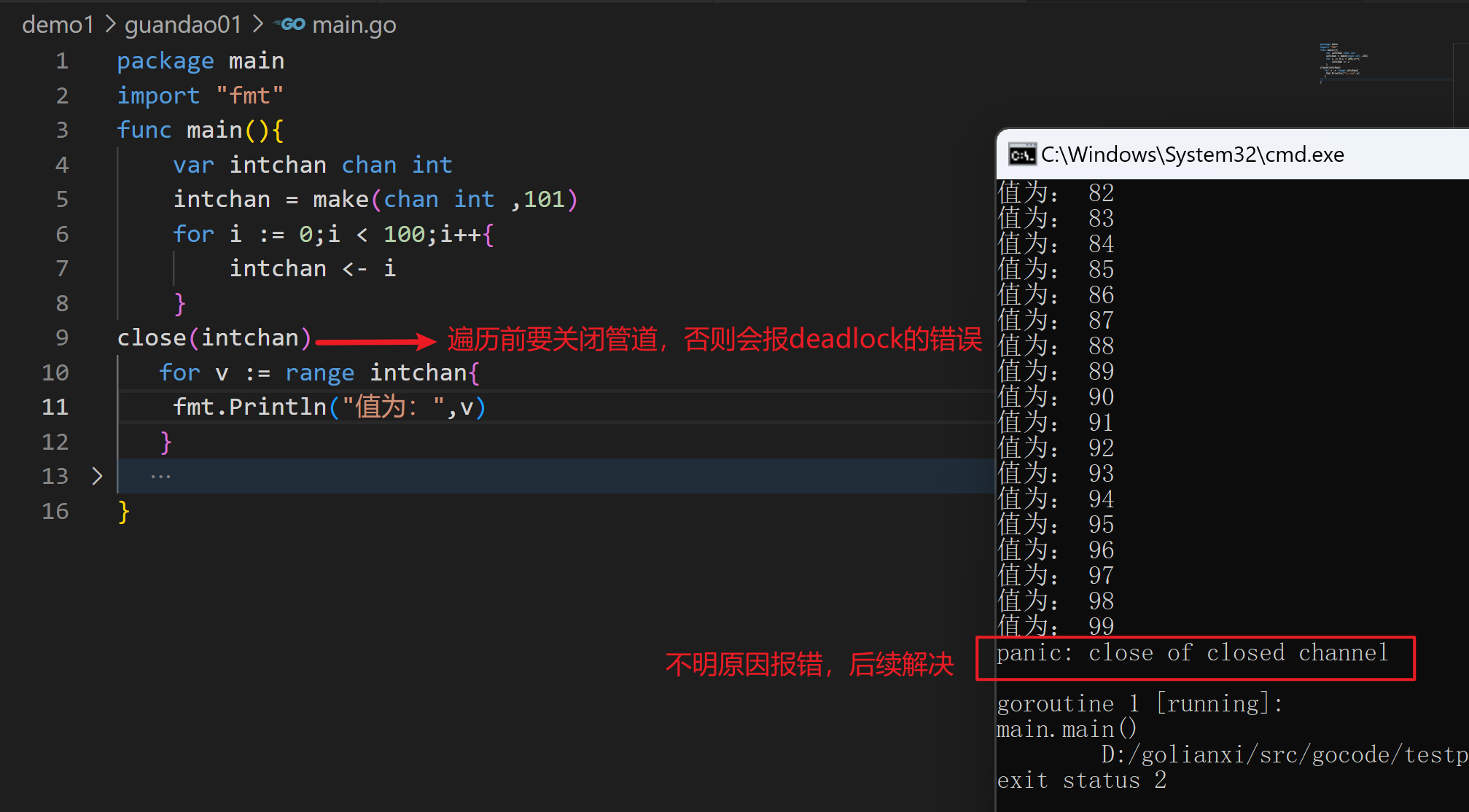


注意：如果管道里面的数据已经取完，如果再取就会报错

管道的遍历：

管道支持for-range的方式进行遍历，但有两点需要注意

1. 在遍历时，如果管道没有关闭，会出现deadlock的错误
2. 在遍历时，如果管道已经关闭，则会正常遍历数据，遍历完后，就会退出遍历

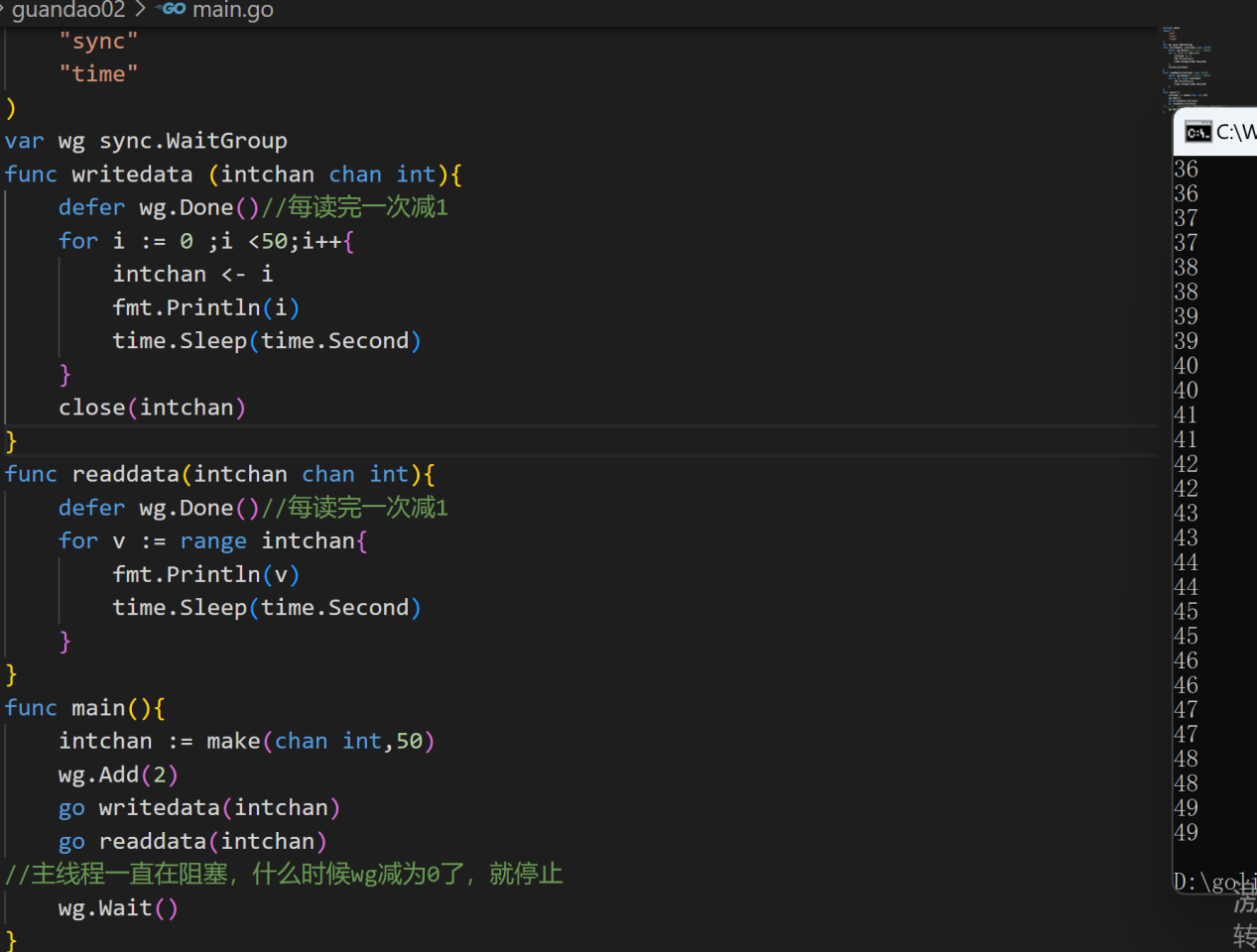


协程和管道共同工作案例

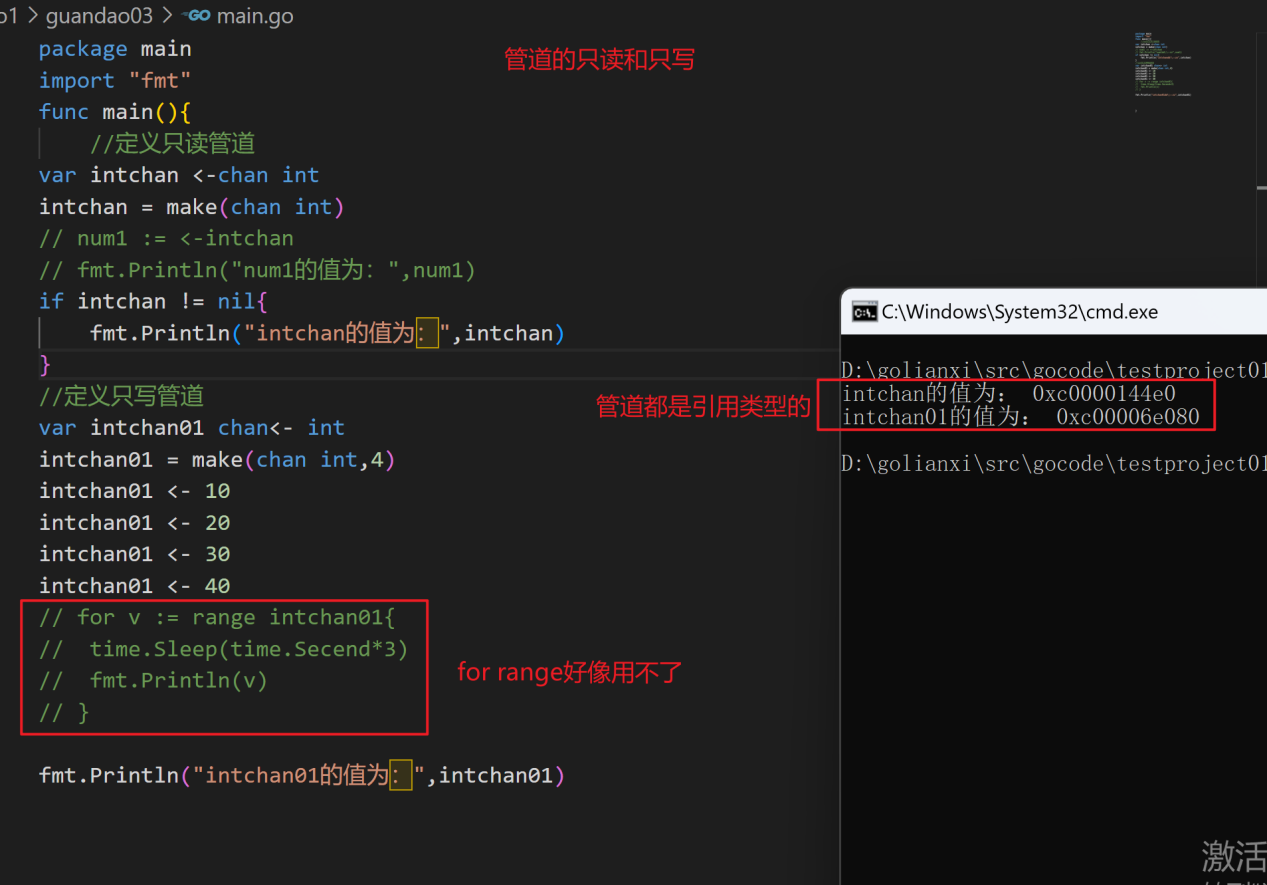
【1】案例需求

请完成协程和管道协同工作的案例，具体要求：

1. 开启一个writeData协程，向管道中写入50个整数
2. 开启一个readData协程，从管道中读取writedata写入的数据
3. 注意：writedata和readdata操作的是同一个管道
4. 主线程需要等待writedata和readdata协程都完成工作才能退出

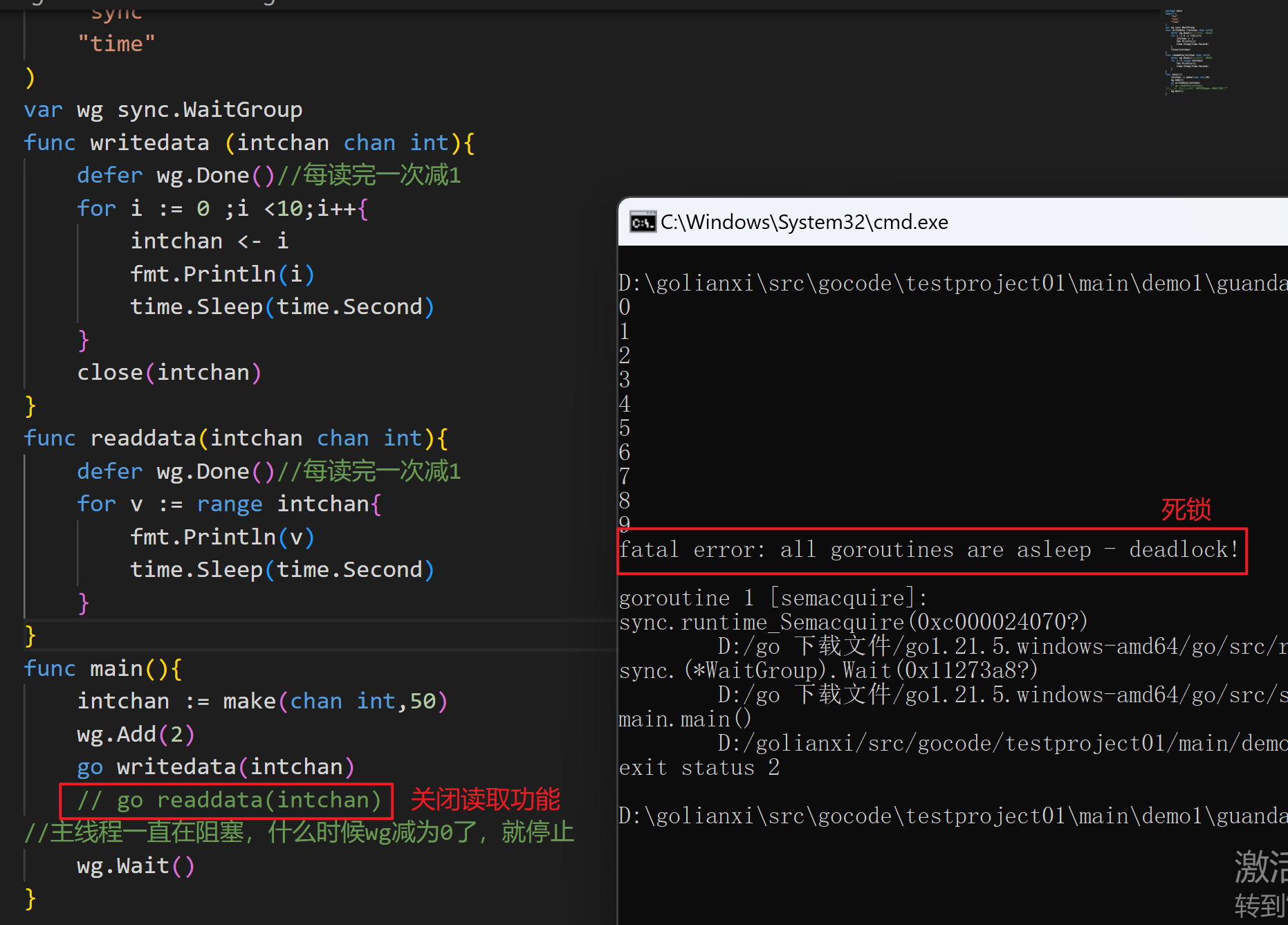


管道的只读和只写



管道的阻塞：

1. 当管道只写入数据，没有读取，就会出现阻塞



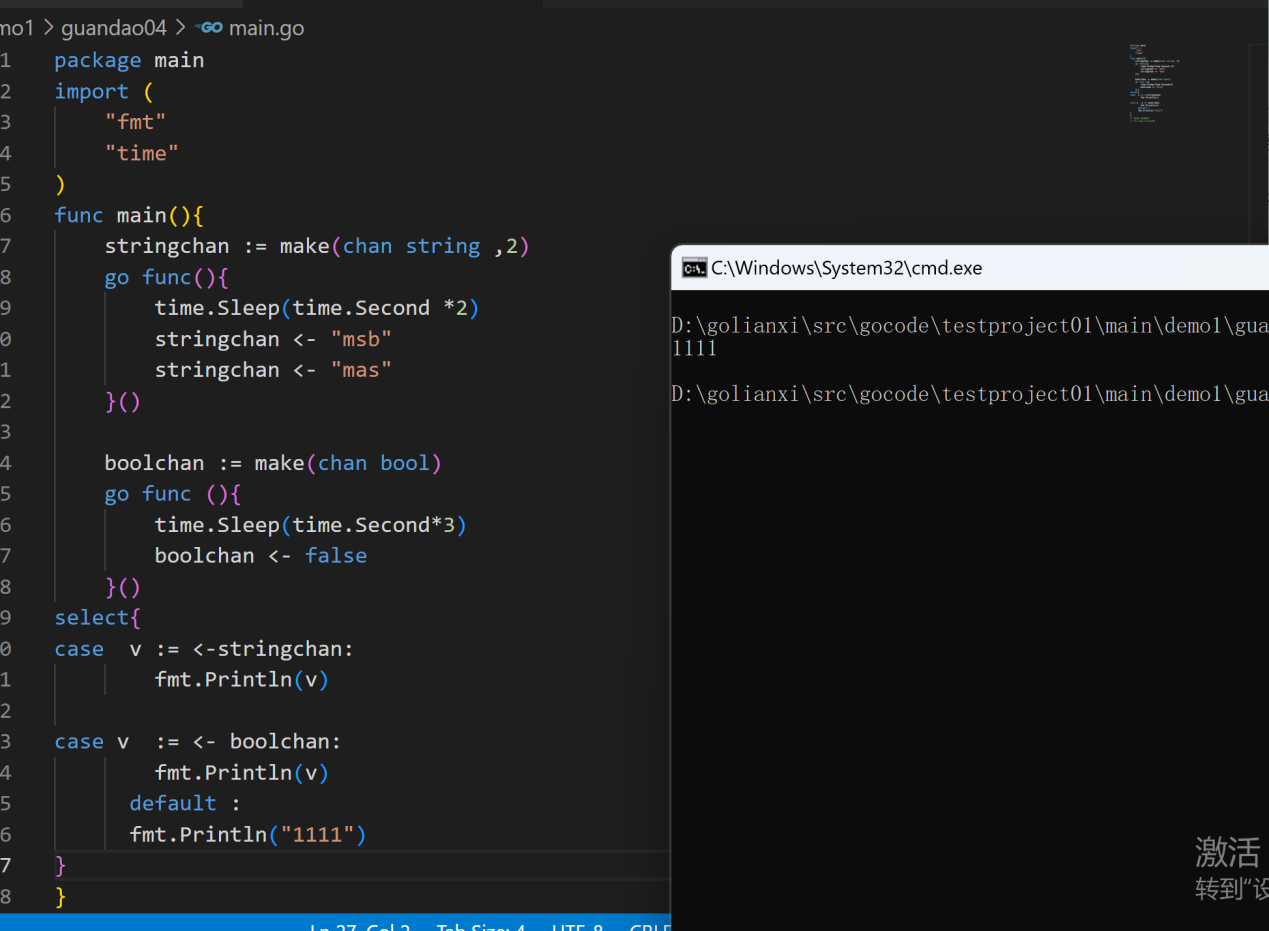
1. 即使写的快，读的慢（管道读写频率不一致），也不会出现阻塞问题，只有在数据不读取的时候会阻塞

Select功能：

【1】select：解决多个管道的选择问题，也可以叫做多路复用，可以从多个管道中随机公平的选择一个来执行

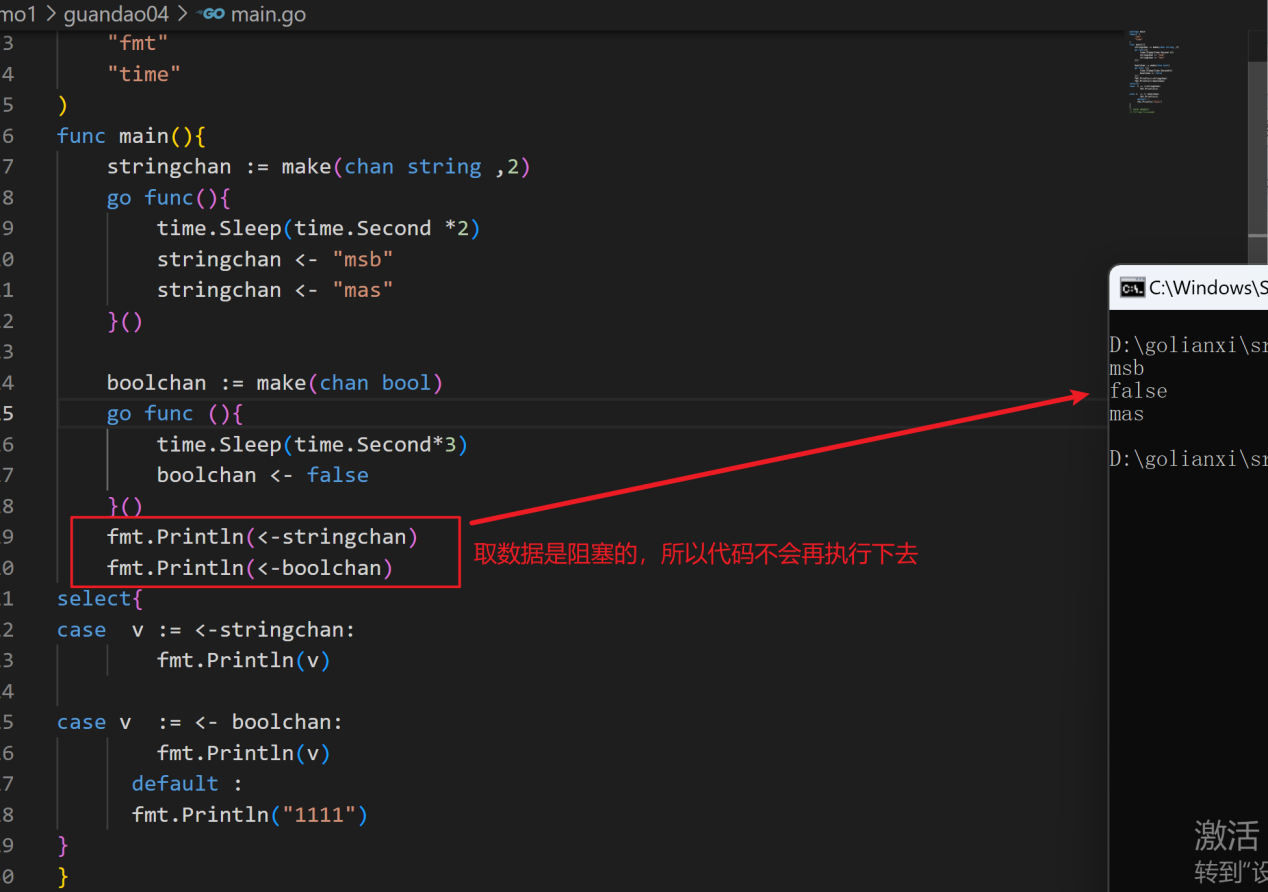
Ps：case后面必须进行的是io操作，不能是等值，随机去选择一个io操作

Ps：为防止select被阻塞住，加入default



Ps：哪个程序的时间短，会优先进行

如果要读取里面的数据需要添加如下代码



Ps：case后面必须使用io操作

defer+recover的使用方法及使用原因

【1】问题原因：多个协程工作，其中某个协程出现panic，导致程序崩溃

【2】解决办法：利用defer+recover捕获panic并进行处理，即使协程出现问题，主线程仍然不受影响继续执行

