最近开始学习 Go 语言，在看到其内存模型的时候一度懵逼了，主要是碰到关于信道的其中两句概念整理以为冲突了，先整理自己的理解如下。

原文

- A send on a channel happens before the corresponding receive from that channel completes.
- A receive from an unbuffered channel happens before the send on that channel completes.

从字面意思来看：

1. 发送到信道的动作发生在对应的接收动作完成之前（这个是针对带缓冲区域和不带缓冲区域的）
2. 针对不带缓冲的信道，从信道接收的动作要发生在发送数据到信道动作完成之前。

一开始怎么也无法理解，然后查到这个，特别注意到了 completes 这个词。这篇对话很长，英语渣实在看不下去了，先如下理解了，等以后慢慢摸索

关于自己的理解

前一句话很容易搞懂，既然要获取数据，那么肯定是要先把数据发送到信道了才能从信道获取，主要是后一句话，要明确不带缓冲区域就意味着这个变量保存不了数据（我是这么理解的），接收动作和发送动作是需要同时进行的。

比如这段代码：

```
package main

import "fmt"

var c = make(chan int32)
var s int32 = 1

func f(){
  s = <-c
}

func main(){
```

```
  go f()
  c <- 1234567890
  fmt.Println(s)
}
```

输出的是 1234567890 (01001001100101100000001011010010) 这个值。我理解的发送和接收图解说明：

```
  int32 是 32bits 的长度，下面则是这个信道可以接收的数据量
                 31                                                  0
                 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
                  0 1 0 0 1 0 0 1 1 0                                  0101100000001011010010 <---send
                <比如发送到信道的时候，才发送了高位的 10 个字，接收动作就需要开始了>
  01001 <---recv             0 0 1 1 0 0 1 0 1 1 0                    0000001011010010 <---send
```

那么归纳一下就是带缓冲区域的 channel 只要有值就可以获取，否则无法获取报错。不带缓冲区域的一定要同时发送和获取，否则报错。

还遇到了这样子的一个问题，代码如下：

```
package main

import (
    "fmt"
    "time"
)

var c = make(chan int)
var s int = 1

func f(){
    s = <-c
}

func main(){
    go f()
    time.Sleep(time.Duration(1000)*time.Nanosecond)
    c <- 1234567890123
```

```
    fmt.Println(s)
}
```

输出结果是 1 如果调整数值 1000 为 600 则可以正常输出 1234567890123 这个值根据不同执行环境有区别。我觉得可能是等待超时了，暂时先不去理会