**Go语言实践4**

**Goroutine 协程**

1. **GPM调度模型**

* 并发和并行的区别：并发是指同时管理很多程序，重点在于调度的概念，可以做到一半，发现效率不高(比如要网络等待，或者等IO)就去执行另外的程序；而并行让多个程序运行在多个CPU上，重点在于能同时处理很多任务。如果机器只有一个CPU，实际上不能够并行执行程序，但是可以做到并发。
* go语言的并发调度逻辑，参考这篇博客<https://segmentfault.com/a/1190000015464889>

1. **第一个goroutine程序**



* **runtime.GOMAXPROCS(1)执行结果**

Q:0,Q:1,Q:2,Q:3,Q:4,Q:5,Q:6,Q:7,Q:8,Q:9,P:6,P:7,P:8,P:9,

解释：只有一个CPU时，P执行到5的时候会进入等待阻塞，调度器自动拉起Q继续执行，称之为**线程交换**，根据调度结果，Q执行到5的时候并没有再去拉起P，而是整个执行完Q再去执行P剩下的部分。

* **runtime.GOMAXPROCS(2)执行结果**

Q:0,Q:1,Q:2,Q:3,Q:4,Q:5,P:0,P:1,P:2,P:3,P:4,P:5,Q:6,Q:7,Q:8,Q:9,P:6,P:7,P:8,P:9,

解释：在各自CPU上按照顺序执行。

* **小结**

golang会自己调度并发，使得程序整体执行效率最高。当程序比较复杂时，**底层的执行逻辑实际不可控**，依赖cpu数量，并发执行的goruntine函数数量，以及golang不同版本的调度算法。

**哪些场景会阻塞线程，使goruntine的调度进行线程交换？**

A. I/O和网络调用：例如读文件，或者通过http协议发送消息

B. Channel 无缓冲通道相关操作 后详

C. 锁/原子操作：原子操作可以是一个步骤, 也可以是多个操作步骤, 但是其顺序不可以被打乱, 也不可以被切割而只执行其中的一部分。 将整个操作视作一个整体是原子性的核心特征

1. **锁和原子操作**

* **竞争状态**

如果多个goruntine在无限制的情况下，同时对一个变量进行读写操作，就称为竞争状态，此时会有潜在问题。golang在编译的时候可以自动进行竞争状态检测: go build -race

* **原子操作**
* 一般推荐++ +=等简单逻辑用原子操作

**互斥锁**

* 推荐复杂逻辑用互斥锁，注意一个锁对所有goruntine都生效，所以必要的时候可以定义多个锁，在不同位置生效，提高程序效率。
* 

1. **通道**

* **生产者消费者模型**

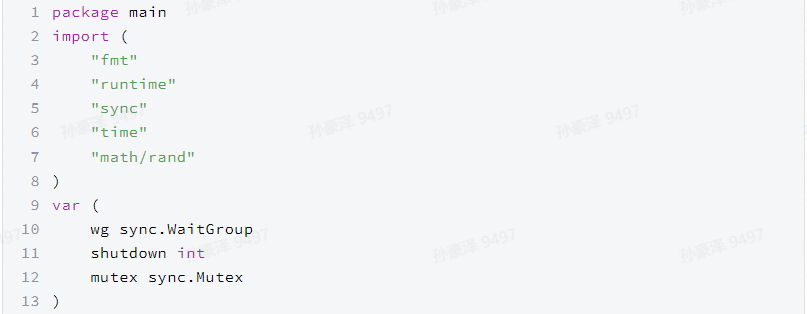
https://zhuanlan.zhihu.com/p/109138133

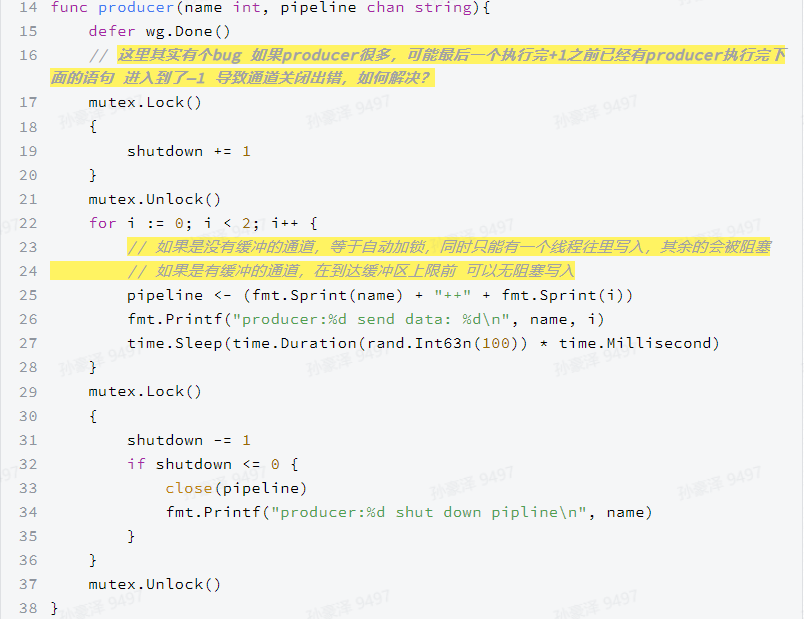
想象一个生产线，最终需要产品A，A依赖零件BC，C又依赖D。每个零件生产的速度不一样，因此需要安排不同数量的worker，来使得整个生产线上等待的时间最少。假如worker代表不同的goruntine，我们可以给不同的零件分配不同数量的协程，同时生产完毕后，将零件，也就是中间数据放入队列中（类似流水线），实现协程间的通讯

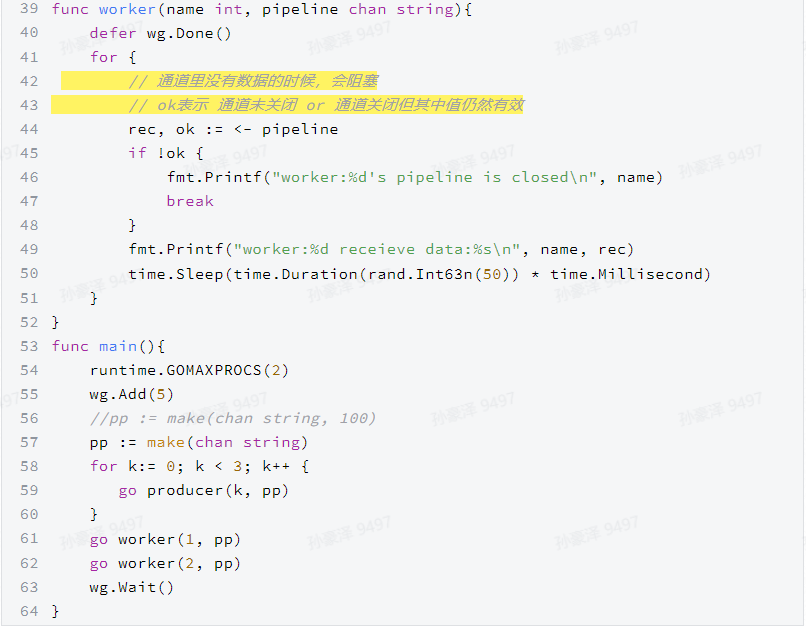
* **通道定义和操作**

1. unbuffered:=make(chan int) // 无缓冲区的channel
2. buffered:=make(chan int, 10) //有缓冲区大小为10的channel
3. buffered <- 123456 // 将123456 写入channel
4. a, ok := <-buffered // 尝试从channel获取值 ok表示channel是否关闭
5. a, ok := <-unbuffered // channel获取的时候会阻塞goruntine

* **实现一个生产者消费者程序**







* **通道关闭后，依然可以接受数据，但是不能再向其中发送数据(会报错)。如果缓冲区还有值，那么可以从关闭的通道继续读出相应的结果，并立即返回无阻塞。如果通道关闭且没有数据，会返回类型默认0值，此时需要ok字段去判断 是否通道关闭且无数据。**
* **当使用无缓冲通道时，运行结果为**

1. producer:0 send data: 0
2. worker:1 receieve data:1++0
3. worker:2 receieve data:0++0
4. producer:1 send data: 0 // 被阻塞 因此worker先收到数据后才进程切换输出
5. worker:1 receieve data:2++0
6. producer:2 send data: 0
7. worker:2 receieve data:0++1
8. producer:0 send data: 1
9. worker:2 receieve data:2++1
10. producer:2 send data: 1
11. producer:1 send data: 1
12. worker:1 receieve data:1++1
13. producer:1 shut down pipline
14. worker:2's pipeline is closed
15. worker:1's pipeline is closed

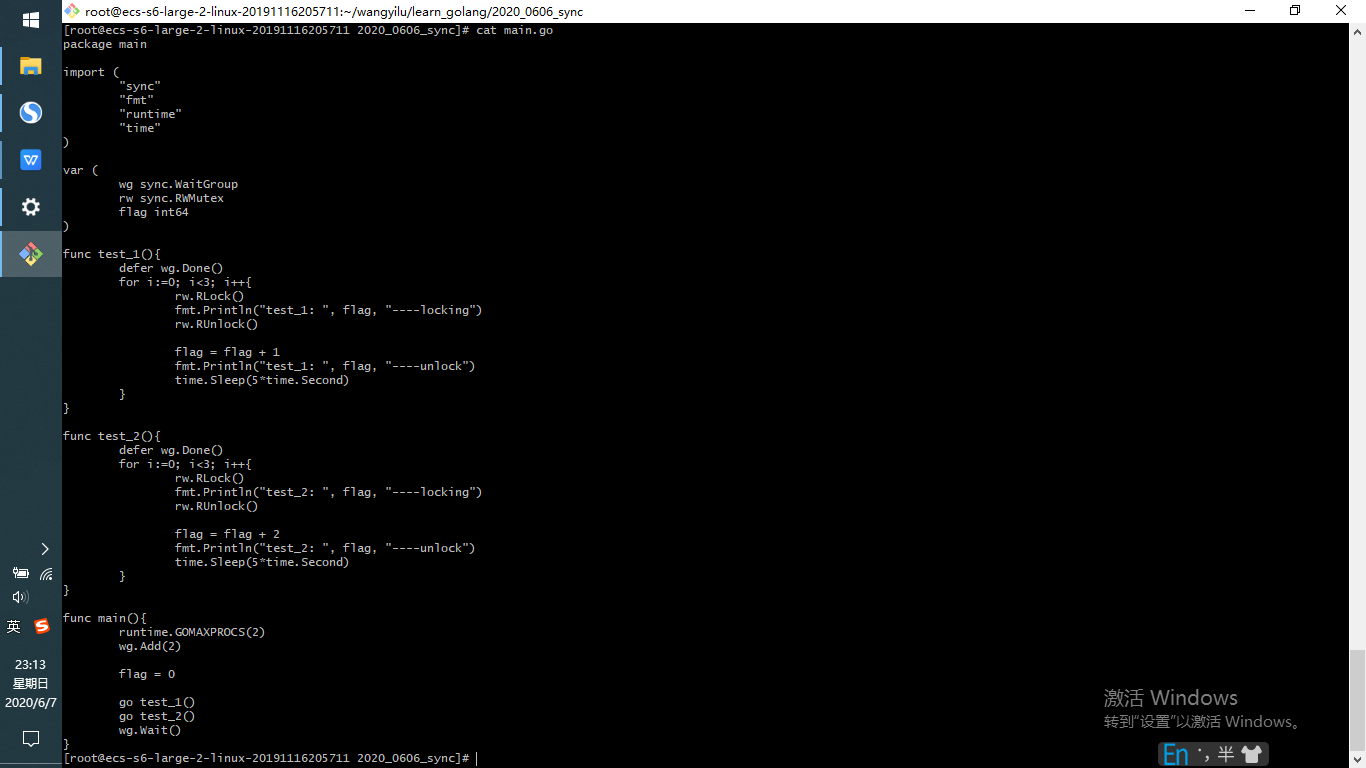
* **当使用有缓冲通道时，运行结果为**

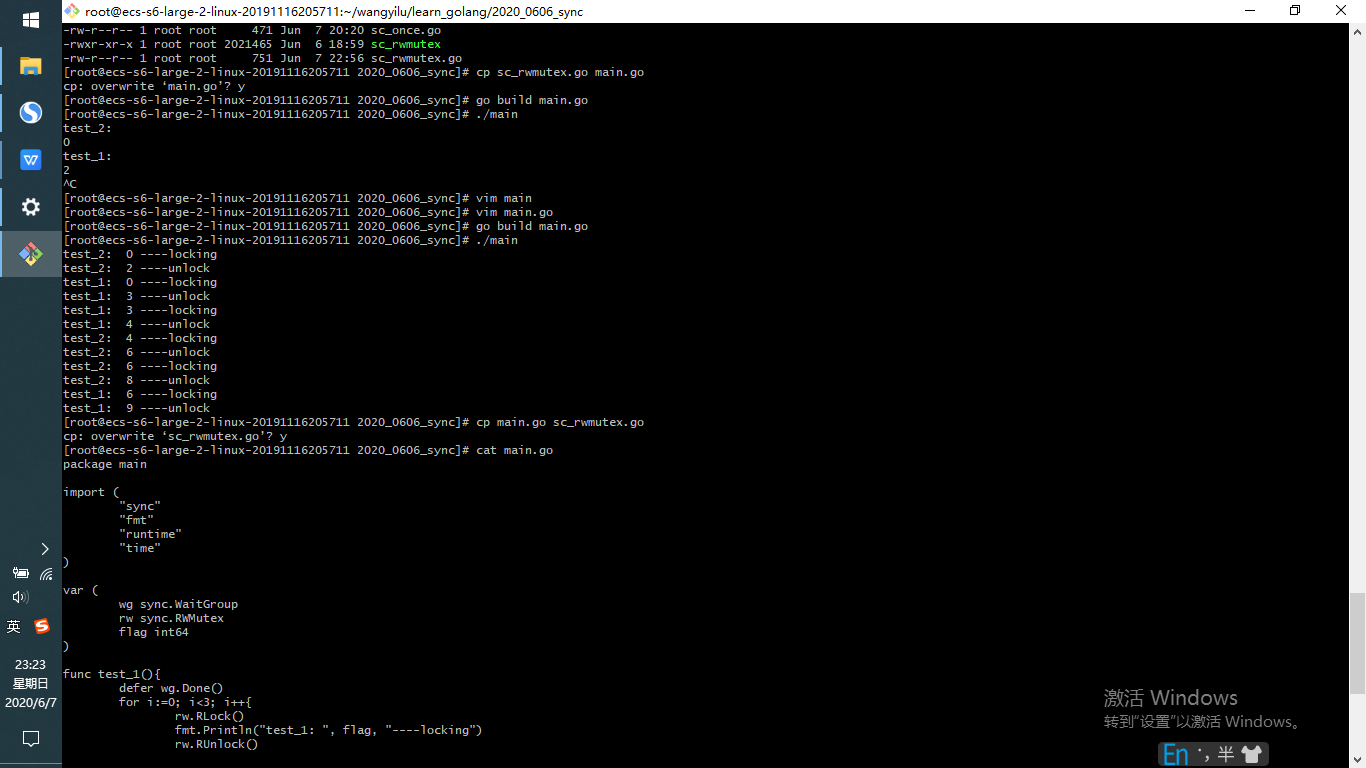
1. producer:0 send data: 0
2. producer:2 send data: 0
3. producer:1 send data: 0 // 未执行到sleep 写入无阻塞
4. worker:2 receieve data:0++0
5. worker:1 receieve data:2++0
6. worker:2 receieve data:1++0 //未执行到sleep 缓冲区数据量足够 读取无阻塞
7. producer:0 send data: 1
8. producer:1 send data: 1
9. worker:2 receieve data:0++1
10. worker:1 receieve data:1++1
11. producer:2 send data: 1
12. worker:2 receieve data:2++1
13. producer:2 shut down pipline
14. worker:1's pipeline is closed
15. worker:2's pipeline is closed

**课后练习**

1. 解决上面生产者消费者代码中的bug
2. 读sync包的文档，尝试使用其中的各功能，并梳理文档<https://golang.org/pkg/sync/>

sync包常用类型：<https://blog.csdn.net/cgl1079743846/article/details/90691127>

1. RWMutex
   * + 1. 功能简介
          1. 当有一个协程在读的时候，所有写的协程必须等到所有读的协程结束才可以获得锁进行写操作。
          2. 当有一个协程在读的时候，所有读的协程不受影响都可以进行读操作。
          3. 当有一个协程在写的时候，所有读、写的协程必须等到写的协程结束才可以获得锁进行读、写操作。
       2. 常用方法
          1. 
       3. 举例说明
          1. 
          2. 输出结果为



1. Once， 常用来控制某些函数只能被调用一次
2. Map它主要五个方法及其功能简介：
   * + 1. Store   存 key,value
       2. LoadOrStore   取&存-具体看代码
       3. Load   取key对应的value
       4. Range   遍历所有的key,value
       5. Delete   删除key,及其value