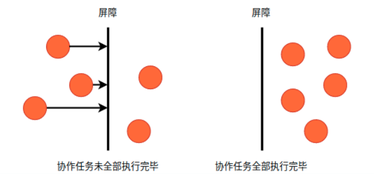
goroutine的同步

屏障: WaitGroup

WaitGroup用于协同多个goroutine并发工作，在语义上与内存屏障类似: 屏障允许每个goroutine等待，直到所有的合作goroutine全部完成后继续运行



其实就是fork join模型的一种实现方式，全部任务被分割以后，进行并行处理，全部处理完毕后再进行汇总

```go
func main() {
    wg := &sync.WaitGroup{}

    for i:= 0; i < 10; i++ {
        wg.Add(1)
        go func(wg *sync.WaitGroup, value int) {
            defer wg.Done()  // 使用defer,确保计数器正确减1
            fmt.Printf("Process value: %d\n", value)
        }(wg, i)
    }

    wg.Wait() // 等待所有协作任务结束
    fmt.Println("All tasks done")
}
```

如果熟悉Java或者是POSIX线程的话，这段代码看着其实并不陌生，与Java CyclicBarrier在使用方法上非常类似

互斥量: sync.Mutex

竞态(Race Condition)

产生竞态条件的根本原因在于多个goroutine对同一个共享变量同时进行修改，并且修改并不是以原子的方式进行，那么就会产生数据不一致的问题

解决竞态问题的方式通常有两种，一种是使所有的goroutine串行化执行，另一种方式就是对变量的修改使其原子化的进行，在修改期间，其余goroutine不可对其进行修改

在实际应用中，使所有goroutine串行化执行代价过于庞大，所以会使用互斥量的方式使得多个goroutine对变量的修改串行化，而不是所有goroutine的串行化

```go
lock := &sync.Mutex{}

func UpdateSomething(lock *sync.Mutex) {
    /* 其它语句 */
    lock.Lock()
    defer lock.Unlock()
    /* 对共享变量的修改 */
}
```

同POSIX线程互斥量一样，主要使用两个方法: Lock()与Unlock()

对互斥量的解锁可放置于defer中，也可置于对共享变量的更新之后，后者能缩短互斥量的持有时间

读写锁: sync.RWMutex

读写锁是对互斥量的一种优化，当变量未添加写锁时，读取操作可并行执行。当变量添加写锁/读锁时，读取操作/写入操作将会被阻塞，直到写入操作/读取操作完成

sync.RWMutex相较于sync.Mutex仅多了2个方法而已

- RLock — 只读锁的添加
- RUnlock — 只读锁的释放

需要注意的是，RWMutex仅在绝大多数goroutine都在获取读锁且竞争比较激烈时才有优势，其内部实现比普通的Mutex更为复杂，所以当锁争抢并不严重时，其效率会差于Mutex

```go
func Read(lock *sync.RWMutex) {
    lock.RLock()
    defer lock.RUnlock()
    /* 只读 */
}

func Write(lock *sync.RWMutex) {
    lock.Lock()
    defer lock.Unlock()
    /* 读+写, 或者只写 */
}
```

一次性初始化: sync.Once

在POSIX线程中，提供了pthread_once()调用来实现一次性初始化

*int pthread_once(pthread_once_t *once_control, void (*init)(void));*

无论多少个线程对pthread_once调用了多少次，只会执行一次init参数所指向的函数

sync.Once与pthread_once的作用相同，多个goroutine在调用sync.Once时，只会执行一次所执行的函数

在sync.Once内部，存在一个布尔变量和一个互斥量(Mutex)，互斥量用于保护该布尔值以及客户端的数据结构

sync.Once常常用于一次性加载配置文件，实现Go语言中的"单例模式"

```go
type Config struct {
    AppName string
}

var (
    once sync.Once
    config *Config
)

func GetConfig() *Config {
    /* 方法Do是Once的唯一方法 */
    once.Do(func() {
        config = &Config{}
        /* 初始化相关配置信息 */
        config.AppName = os.Getenv("appName")
    })
    return config
}
```