sync.Mutex互斥锁

可以用一个容量只有1的channel来保证最多只有一个goroutine在同一时刻访问一个共享变量，来实**sync.Mutex**的效果

```
var (
    sema    = make(chan struct{}, 1) // a binary semaphore guarding balance
    balance int
)

func Deposit(amount int) {
    sema <- struct{}{} // acquire token
    balance = balance + amount
    <-sema // release token
}

func Balance() int {
    sema <- struct{}{} // acquire token
    b := balance
    <-sema // release token
    return b
}
```

如何避免go中重复加锁导致的死锁问题？

通过Mutex 互斥访问共享变量balance ：

```
import "sync"

var (
    mu      sync.Mutex // guards balance
    balance int
)
```

```
func Deposit(amount int) {
    mu.Lock()
    balance = balance + amount
    mu.Unlock()
}

func Balance() int {
    mu.Lock()
    b := balance
    mu.Unlock()
    return b
}
```

考虑一下下面的Withdraw函数。成功的时候，它会正确地减掉余额并返回true。但如果银行记录资金对交易来说不足，那么取款就会恢复余额，并返回false。

```
// NOTE: not atomic!
func Withdraw(amount int) bool {
    Deposit(-amount)
    if Balance() < 0 {
        Deposit(amount)
        return false // insufficient funds
    }
    return true
}
```

函数终于给出了正确的结果，但是还有一点讨厌的副作用。当过多的取款操作同时执行时，balance可能会瞬时被减到0以下。这可能会引起一个并发的取款被不合逻辑地拒绝。
这里的问题是取款不是一个原子操作：它包含了三个步骤，每一步都需要去获取并释放互斥锁，但任何一次锁都不会锁上整个取款流程。
理想情况下，取款应该只在整个操作中获得一次互斥锁。但是下面这样的尝试是错误的：

```
func Withdraw(amount int) bool {
    mu.Lock()
    defer mu.Unlock()
    Deposit(-amount)
    if Balance() < 0 {
        Deposit(amount)
        return false // insufficient funds
    }
    return true
}
```

Deposit会调用mu.Lock()第二次去获取互斥锁，但因为mutex已经锁上了，而无法被重入，这会导致程序死锁，没法继续执行下去，Withdraw会永远阻塞下去。

一个通用的解决方案是将一个函数分离为多个函数，比如我们把Deposit分离成两个：一个不导出的函数deposit，这个函数假设锁总是会被保持并去做实际的操作，另一个是导出的函数Deposit，这个函数会调用deposit，但在调用前会先去获取锁。同理我们可以将Withdraw也表示成这种形式：

```
func Withdraw(amount int) bool {
    mu.Lock()
    defer mu.Unlock()
    deposit(-amount)
    if balance < 0 {
        deposit(amount)
        return false
    }
    return true
}

func Deposit(amount int) {
    mu.Lock()
    defer mu.Unlock()
    deposit(amount)
}
```

```
func Balance() int {
    mu.Lock()
    defer mu.Unlock()
    return balance
}

func deposit(amount int) { balance += amount }
```