GO语言学习笔记

## sync/atomic库，保证原子操作的库，并发编程里面很有用，只支持整形，和整形指针

sync/atomic 库提供了原子操作的支持，原子操作直接有底层CPU硬件支持，因而一般要比基于[**操作系统**](http://lib.csdn.net/base/operatingsystem)API的锁方式效率高些。本文对 sync/atomic 中的基本操作进行一个简单的介绍。

很简单的原子操作的库函数，如增加，取值，比较后赋值，直接赋值，赋值后返回旧值

### 原子增值

用于对变量值进行原子增操作，并返回增加后的值。

e.g.

var sum uint32 = 100

var wg sync.WaitGroup

for i := 0; i < 50; i++ {

wg.Add(1)

go func() {

defer wg.Done()

//sum += 1 //1

atomic.AddUint32(&sum, 1) //2

}()

}

wg.Wait()

fmt.Println(sum)1234567891011121312345678910111213

通过对比//1与//2的结果，可以很清楚的看到原子操作起到的作用。使用//1时，可以看到sum的值是不定的，取决于sum的同步访问情况；使用//2时，结果是确定而且正确的，同一时间只有一个goroutine修改sum。

函数原型：

atomic.AddUint32(addr \*uint32, delta uint32) uint32

atomic.AddUint64(addr \*uint64, delta uint64) uint64

atomic.AddInt32(addr \*int32, delta int32) int32

atomic.AddInt64(addr \*int64, delta int64) int64

atomic.AddUintptr(addr \*uintptr, delta uintptr) uintptr1234512345

### CAS

先比较变量的值是否等于给定旧值，等于旧值的情况下才赋予新值，最后返回新值是否设置成功。

e.g.

var sum uint32 = 100

var wg sync.WaitGroup

for i := uint32(0); i < 100; i++ {

wg.Add(1)

go func(t uint32) {

defer wg.Done()

atomic.CompareAndSwapUint32(&sum, 100, sum+1)

}(i)

}

wg.Wait()

fmt.Println(sum)123456789101112123456789101112

可以看到sum的值只改变了一次，只有当sum值为100的时候，CAS才将sum的值修改为了sum+1。

函数原型：

atomic.CompareAndSwapUint32(addr \*uint32, old, new uint32) bool

atomic.CompareAndSwapUint64(addr \*uint64, old, new uint64) bool

atomic.CompareAndSwapInt32(addr \*int32, old, new int32) bool

atomic.CompareAndSwapInt64(addr \*int64, old, new int64) bool

atomic.CompareAndSwapUintptr(addr \*uintptr, old, new uintptr) bool

atomic.CompareAndSwapPointer(addr \*unsafe.Pointer, old, new unsafe.Pointer) bool123456123456

### 原子导出值

导出变量当前的值。

函数原型：

atomic.LoadUint32(addr \*uint32) uint32

atomic.LoadUint64(addr \*uint64) uint64

atomic.LoadInt32(addr \*int32) int32

atomic.LoadInt64(addr \*int64) int64

atomic.LoadUintptr(addr \*uintptr) uintptr

atomic.LoadPointer(addr \*unsafe.Pointer) unsafe.Pointer123456123456

### 原子导入值

赋予变量新值，而不管它原来是什么值。

函数原型：

atomic.StoreUint32(addr \*uint32, val uint32)

atomic.StoreUint64(addr \*uint64, val uint64)

atomic.StoreInt32(addr \*int32, val int32)

atomic.StoreInt64(addr \*int64, val int64)

atomic.StoreUintptr(addr \*uintptr, val uintptr)

atomic.StorePointer(addr \*unsafe.Pointer, val unsafe.Pointer)123456123456

### 原子交换值

赋予变量新值，同时返回变量的旧值。

函数原型：

atomic.SwapUint32(addr \*uint32, new uint32) old uint32

atomic.SwapUint64(addr \*uint64, new uint64) old uint64

atomic.SwapInt32(addr \*int32, new int32) old int32

atomic.SwapInt64(addr \*int64, new int64) old int64

atomic.SwapUintptr(addr \*uintptr, new uintptr) old uintptr

atomic.SwapPointer(addr \*unsafe.Pointer, new unsafe.Pointer) old unsafe.Pointer