第14节课内容总结

什么是线程安全?

多线程操作共享数据的时候不会出现意想不到的结果就叫线程安全，否则，就是线程不安全。

原子属性是线程安全的吗?

原子属性只能保障set `或者` get的读写安全，但我们在使用属性的时候，往往既有set又有get，所以说原子属性并不是线程安全的。

自旋锁和互斥锁的区别

自旋锁: 在访问被锁的资源的时候，调用者线程不会休眠，而是不停循环在那里，直到被锁资源释放锁。（忙等）

互斥锁: 在访问被锁资源时，调用者线程会休眠，此时cpu可以调度其他线程工作。直到被锁的资源释放锁。然后再唤醒休眠线程。（闲等）

自旋锁的优点在于，因为自旋锁不会引起调用者线程休眠，所以不会进行线程调度，cpu时间片轮转等一些耗时的操作。所以如果能在很短的时间内获得锁，自旋锁的效率远高于互斥锁。

自旋锁缺点在于，自旋锁一直占用CPU，在未获得锁的情况下，一直自旋，相当于死循环，会一直占用着CPU，如果不能在很短的时间内获得锁，这无疑会使CPU效率降低。`而且自旋锁不能实现递归调用`。

自旋锁优先级反转的bug

当多个线程有优先级的时候，如果一个优先级低的线程先去访问某个数据，此时使用自旋锁进行了加锁，然后一个优先级高的线程又去访问这个数据，那么优先级高的线程因为优先级高会一直占着CPU资源，此时优先级低的线程无法与优先级高的线程争夺 CPU 时间，从而导致任务迟迟完不成、锁无法释放。

由于自旋锁本身存在的这个问题，所以苹果在iOS10以后已经废弃了OSSpinLock。

`也就是说除非大家能保证访问锁的线程全部都处于同一优先级，否则 iOS 系统中的自旋锁就不要去使用了。`

NSCondition存在的虚假唤醒

当线程从等待已发出信号的条件变量中醒来，却发现它等待的条件不满足时，就会发生虚假唤醒。之所以称为虚假，是因为该线程似乎无缘无故地被唤醒了。但是虚假唤醒不会无缘无故发生：它们通常是因为在发出条件变量信号和等待线程最终运行之间，另一个线程运行并更改了条件。线程之间存在竞争条件，典型的结果是有时，在条件变量上唤醒的线程首先运行，赢得竞争，有时它运行第二，失去竞争。

在许多系统上，尤其是多处理器系统上，虚假唤醒的问题更加严重，因为如果有多个线程在条件变量发出信号时等待它，系统可能会决定将它们全部唤醒，将每个signal( )唤醒一个线程视为broadcast( )唤醒所有这些，从而打破了信号和唤醒之间任何可能预期的 1:1 关系。如果有 10 个线程在等待，那么只有一个会获胜，另外 9 个会经历虚假唤醒。

读写锁

读写锁的目的：

- 多读单写:在同一时刻可以被多条线程进行读取数据的操作，但是在同一时刻只能有一条线程在写入数据。
- 读写互斥:在同以时刻，读和写不能同时进行。

用GCD实现读写锁：

```
//写
-(void)lg_write:(NSDictionary *)dic {
    dispatch_barrier_async(self.iQueue, ^{
        [self.dataDic setDictionary:dic];
    });
}

//读
-(NSString *)lg_read {

    __block NSString *ret;
    dispatch_sync(self.iQueue, ^{
        ret = self.dataDic[@"name"];
    });
    NSLog(@"%@",ret);
    return ret;
}
```