多线程同步内部如何实现的

wait/notify, synchronized, ReentrantLock

自旋

```
volatile int status=0;//标识---是否有线程在同步块-----是否有线程上锁成功
void lock(){

 while(!compareAndSet(0,1)){
 }
 //lock

}

void unlock(){
  status=0;
}

boolean compareAndSet(int except,int newValue){
  //cas操作,修改status成功则返回true
}
```

缺点：耗费cpu资源。没有竞争到锁的线程会一直占用cpu资源进行cas操作，假如一个线程获得锁后要花费Ns处理业务逻辑，那另外一个线程就会白白的花费Ns的cpu资源

思路：让得不到锁的线程让出CPU

yield+自旋

```
volatile int status=0;
void lock(){
 while(!compareAndSet(0,1)){
 yield();//自己实现
 }
 //lock

}
void unlock(){
  status=0;
}
```

要解决自旋锁的性能问题必须让竞争锁失败的线程不空转,而是在获取不到锁的时候能把cpu资源给让出来，yield()方法就能让出cpu资源，当线程竞争锁失败时，会调用yield方法让出cpu。自旋+yield的方式并没有完全解决问题，当系统只有两个线程竞争锁时，yield

是有效的。需要注意的是该方法只是当前让出cpu，有可能操作系统下次还是选择运行该线程，比如里面有2000个线程，想想会有什么问题？

sleep+自旋

```
volatile int status=0;
void lock(){
 while(!compareAndSet(0,1)){
 sleep(10);
 }
 //lock

}
void unlock(){
  status=0;
 }
```

sleep的时间为什么是10？这么控制呢？就是你是调用者其实很多时候你也不知道这个时间是多少？

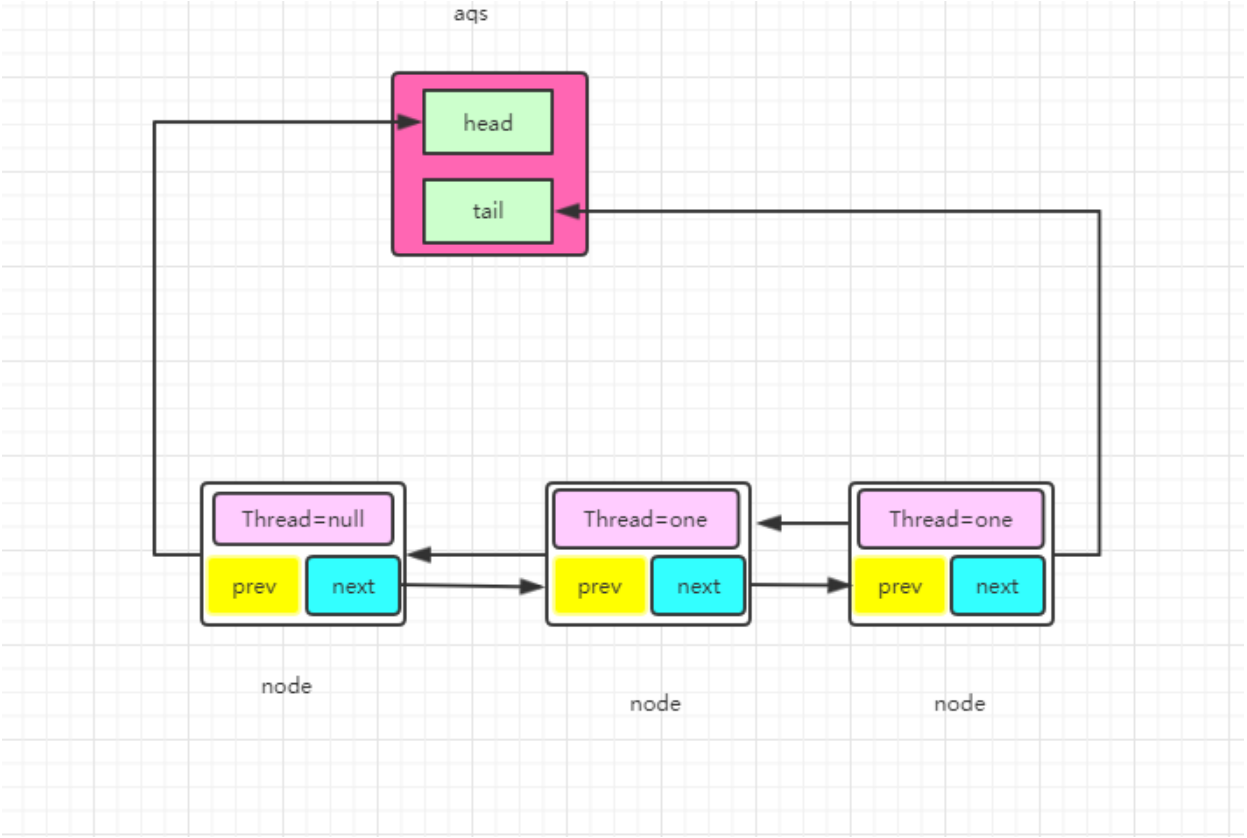
park+自旋

```
volatile int status=0;
Queue parkQueue;//集合 数组 list

void lock(){
 while(!compareAndSet(0,1)){
 //
 park();
 }
 //lock 10分钟
  。。。。。。
 unlock()
}

void unlock(){
 lock_notify();
}

void park(){
 //将当期线程加入到等待队列
 parkQueue.add(currentThread);
 //将当期线程释放cpu 阻塞
 releaseCpu();
}
```

```
void lock_notify(){
 //得到要唤醒的线程头部线程
 Thread t=parkQueue.header();
 //唤醒等待线程
 unpark(t);
}
```

AQS

```
private transient volatile Node head; //对首
private transient volatile Node tail;//对尾
private volatile int state;//锁状态，加锁成功则为1，重入+1 解锁则为0
```



NODE

```
public class Node{
 volatile Node prev;
 volatile Node next;
 volatile Thread thread;
}
```

```
final void lock() {
 acquire(1);
```

```
}
```

1标识加锁成功之后改变的值

```
public final void acquire(int arg) {
 if (!tryAcquire(arg) &&
 acquireQueued(addWaiter(Node.EXCLUSIVE), arg))
 selfInterrupt();
}
```

公平锁的加锁过程的代码

```
tryAcquire(arg)



protected final boolean tryAcquire(int acquires) {
 final Thread current = Thread.currentThread();
 int c = getState();
 if (c == 0) {//没人占用锁--->我要去上锁----1、锁是自由状态
 if (!hasQueuedPredecessors() &&
 compareAndSetState(0, acquires)) {
 setExclusiveOwnerThread(current);
 return true;
 }
 }
 else if (current == getExclusiveOwnerThread()) {
 int nextc = c + acquires;
 if (nextc < 0)
 throw new Error("Maximum lock count exceeded");
 setState(nextc);
 return true;
 }
 return false;
}


hasQueuedPredecessors 排队---怎么排
```