**多线程**

# 进程与线程

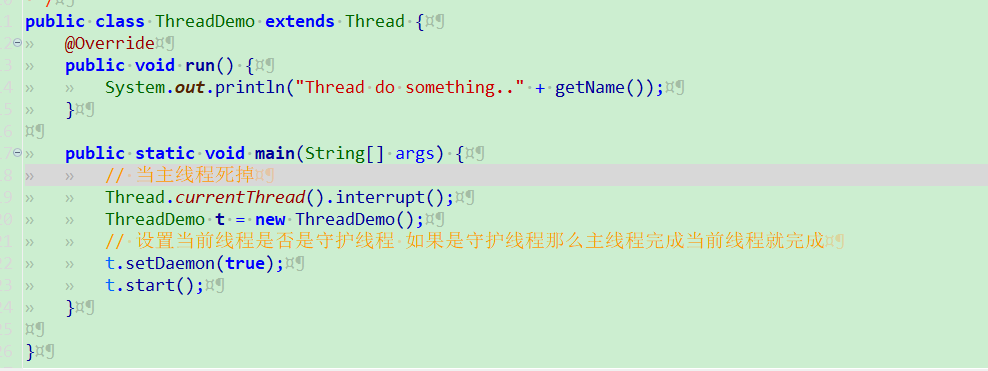
线程是进程运行的最小单位， 一个进程中可以有多个线程。 具体可以看任务管理器

# Java 线程的创建

Java 线程 一般是由 Thread 的 start 方法启动；

其中实现 无返回接口 Runabble， 有返回接口 Callable 接口都是实现线程任务。

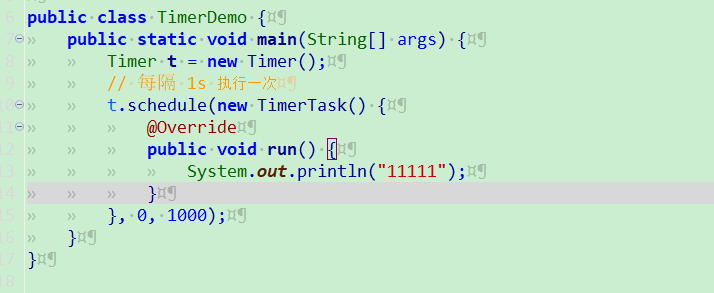
## Thread



## Callable



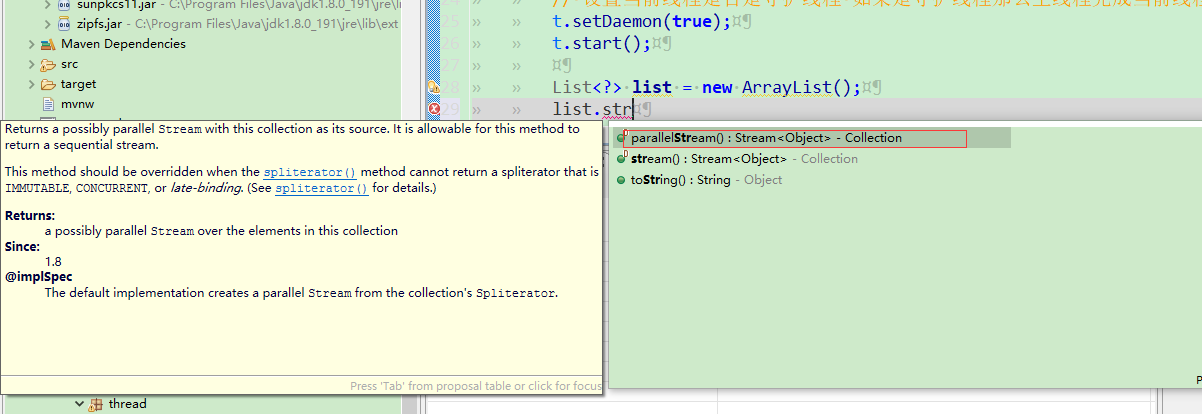
# 定时任务



# 线程池 ThreadPoolExecutor 参照源码

里边有 blockIngQueue 阻塞队列

# Lamada 表达式串行



# Java 中的锁

Yield(); 当前线程 让出cpu 该方法 由 cpu 调用 让出的 线程继续争用 锁 可能还是让出线程的 争取到该锁

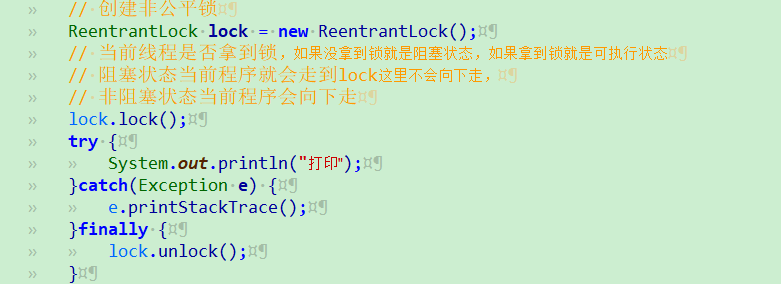
## 1.1 Synchronized

调用操作系统的锁 1.6 之前 性能低下

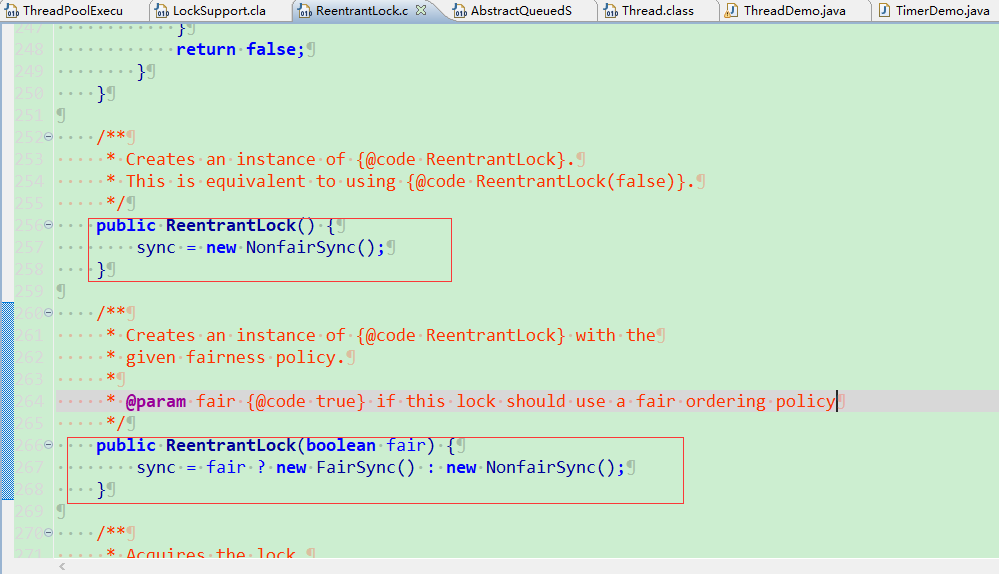
ConcurrentHashMap 在1.7 之前用的是 reentrantLock，1.8 之后用的是 synchronized

## Reentrantlock

用法

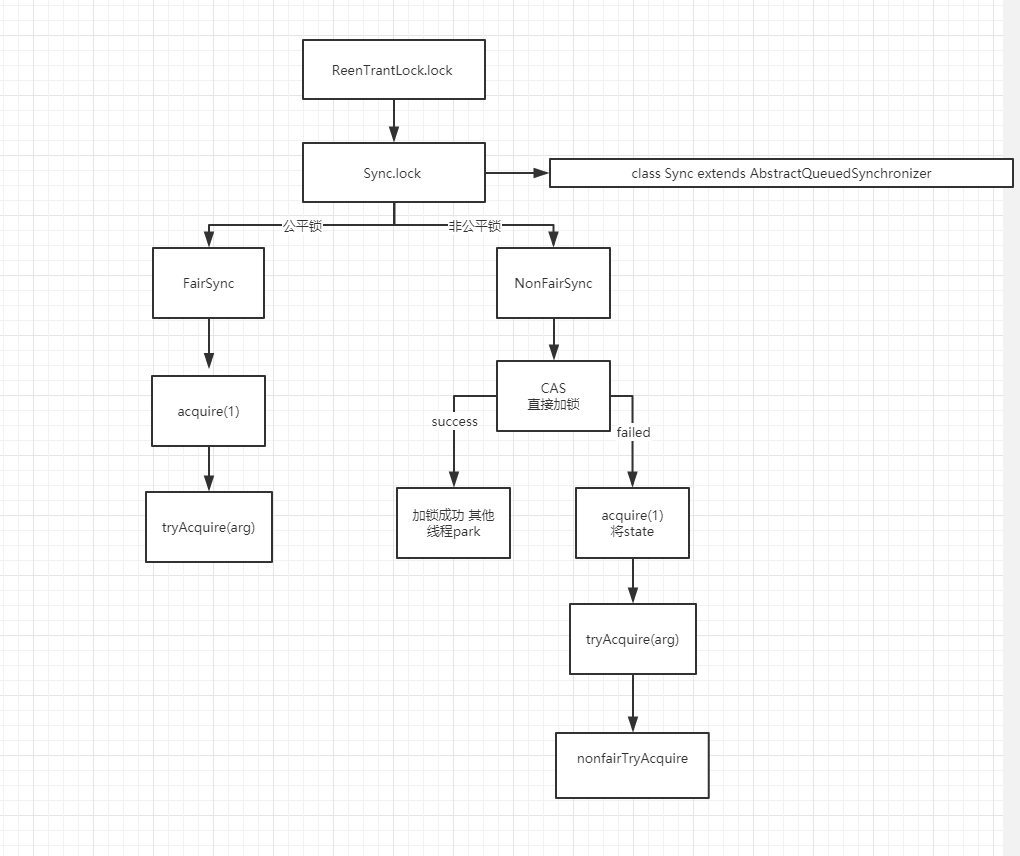


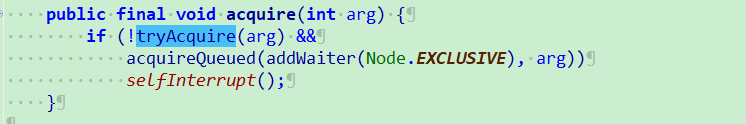
**默认构造为非公平锁，根据构造也可以实现公平锁**



**其中的CAS 操作，** compareAndSwapInt

结构图





**如果抢占锁失败调用Thread 的interrupt 方法进行阻塞**

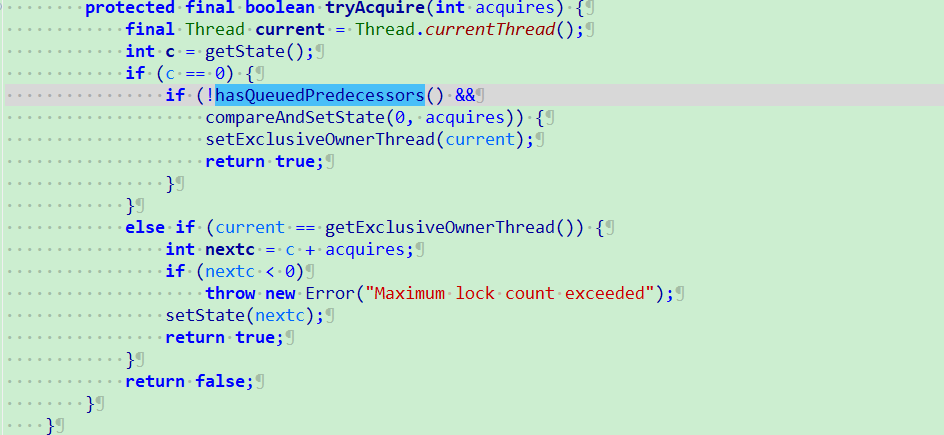
**为什么用ReentrantLock？**

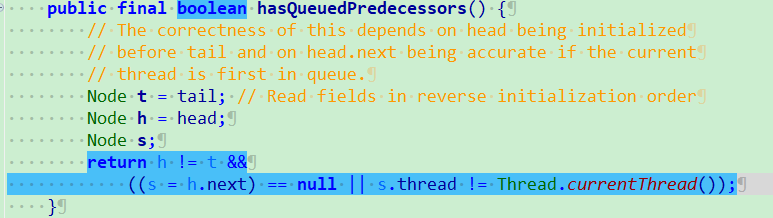
**在1.6 之前没有优化 Synchronized 性能低，reentrantlock 性能比较高**

**轻量级锁，没有调用 os 的park unPark；**

**交替执行， 一个线程 执行完 另一个线程执行，在第一个线程执行完成之前没有第二个线程来进行执行没有有队列； 假如 第一个线程执行过程中第二个线程执行，放入队列之中等待执行**

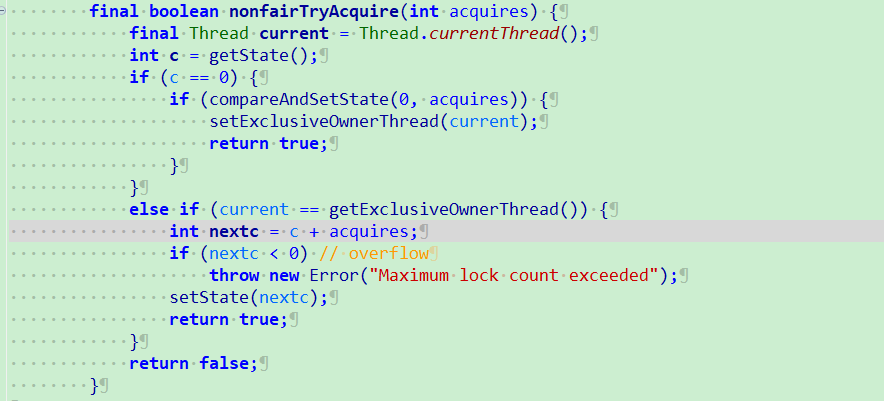
### FairSync 公平锁





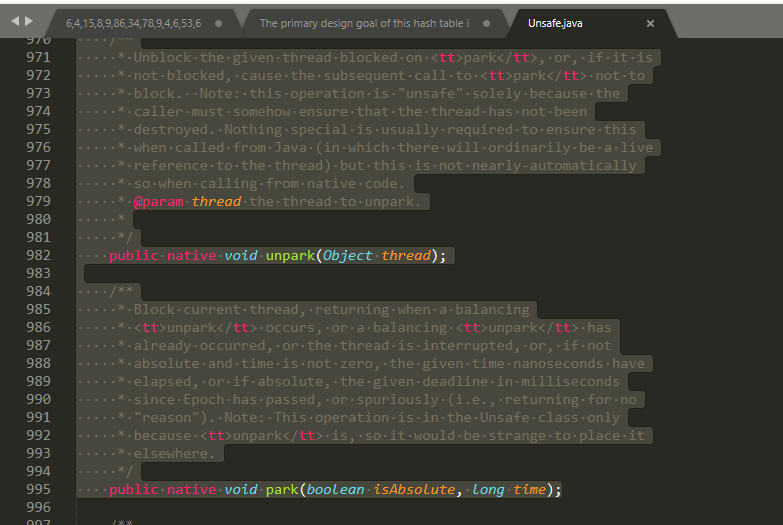
公平锁的情况下，如果锁是自由状态，如果队列为空的话不需要排队 做CAS操作，如果队列不为空并且只有一个的话不加锁，如果队列不为空，并且值大于1的情况下，判断第二个元素是不是当前独占锁的线程，

### NonFairSync 非公平锁



非公平锁，如果当前锁是自由状态， 那么直接进行CAS操作 然后将当前线程就是独占线程； 如果当前锁是非自由状态 并且独占线程是当前线程那么将当前状态加1；

### 1.2.3 Unsafe



#### 1.2.3.1 Park 自旋 实现锁

实现思路

1. 理解

多线程共享同一资源，当前资源为可见性资源（如果一个线程对资源进行更改所有线程都能见到）； 当其他线程看到当前资源状态改变后，就开始阻塞，等当前使用线程执行完毕后，将状态改变，让阻塞资源进行争夺；

例：公厕使用；当厕所有人使用的时候将厕所门从里边上锁，挂出正在使用的牌子；当第二个人也要使用的时候，排队等候第一个人使用完毕， 第一个人使用完毕后将锁打开，正在使用牌子摘掉第二个人开始使用，依次类推；

1. 简单实现
2. volatile关键字， 该关键字修饰变量为确保变量的可见性，多线程可见，需要注意的是，是需要原子性操作
3. 创建一个类，其中有原子性操作的可见的变量（volatile 修饰），设置变量的状态表示是否资源为占用状态，判断该属性是否被占用，如果是占用状态，其余线程调用park（）方法让出资源变成阻塞状态，其中加个队列，将其余线程放进去；如果当前执行任务执行完成， 将变量还原在调用unPark 方法，同时将队列中的第一个线程让其执行。

#### UnPark（Thread t）

唤醒 park 的线程

### 1.2.4 LockSupport

使用 UnSafe 类中的 native（本地操作系统的） park、 unPark 进行加锁、解锁的支持。

## AQS AbstractQueuedSynchronizer

Class Node{

Node tail；

Node head；

Int state；

Thread thread；

}

AQS入队操作， 现在有线程 T1， T2， T3 三个线程需要操作同一资源，当T1 线程抢到锁之后还未执行完成，T2，T3线程请求也来了；这个时候就需要入队操作；

