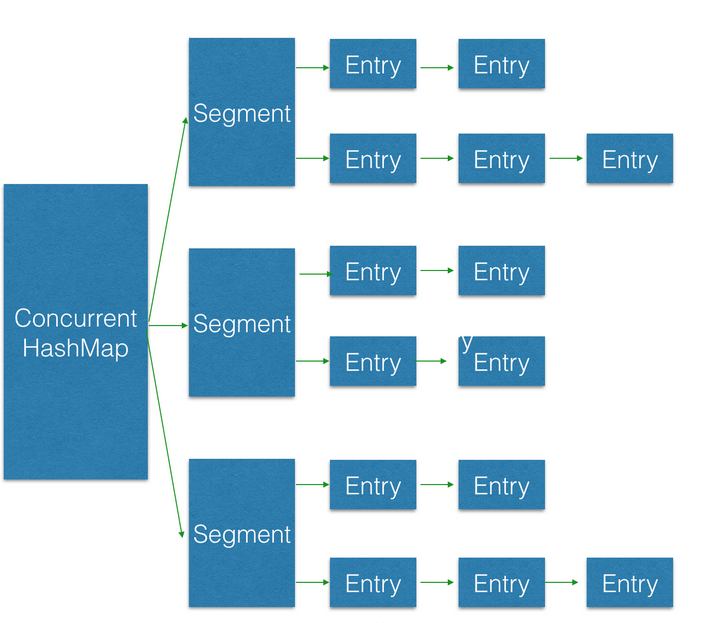
# ****ConcurrentHashMap****

HashMap线程不安全,HashTable效率低下 诞生此类

ConcurrentHashMap使用分段锁技术，将数据分成一段一段的存储，然后给每一段数据配一把锁



内部分为很多Segmaent(继承ReentrantLock),每个Segment拥有一把锁, 包含多个Entry

对于key 需要三次hash

1：hash第一次得到h1

2：h1高位hash得到h2确定元素存放在哪个Segment

3：对h1进行hash 得到h3 确定存放的entry

初始化参数

1：entry默认16个

2：负载因子0.75 元素个数大于0.75\*最大容量 rehash 扩容

3：并发级别 决定Segment个数=并发级别最靠近2的n次方 eg 16=2的4次 级别13

Put操作(自旋锁) Get操作(乐观锁)

Size操作

需要遍历所有的Segment 可以lock所有的后统计 效率太低

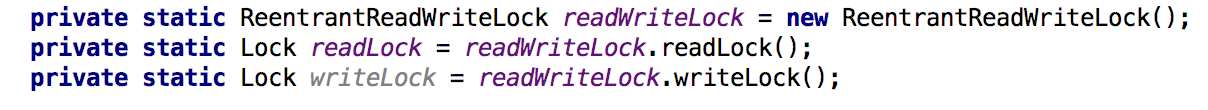
高效率：给三次机会

不锁住所有的Segment统计三次

如果相邻两次获取的Segment的modcount全部相同 则无修改 直接返回

如果不相同 证明被修改了 则对所有的Segment加锁 统计结束后释放锁

# ReadWriteLock读写锁



# CountDownLatch和CyclicBarrier

<https://www.cnblogs.com/xiaorenwu702/p/3977833.html>

# 线程池

## newFixedThreadPool

固定线程数量线程池,当调度任务无空闲线程时,任务暂存任务队列中

## newSingleThreadExecutor

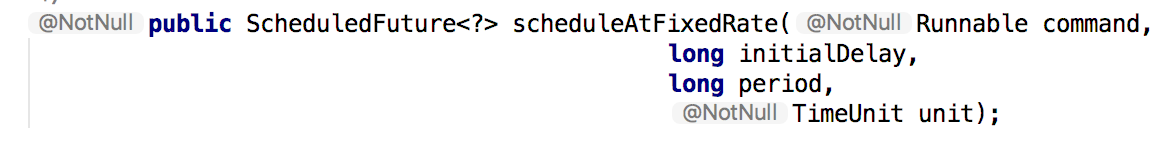
单线程的线程池,任务执行按照先进先出的顺序

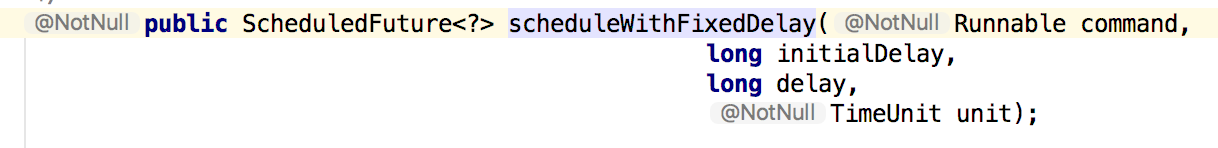
## newCachedThreadPool

可以动态调整线程数量的线程池，执行任务会优先复用可复用的线程，若可用线程都在执行任务，则创建新的线程执行新的任务，执行完毕之后，线程归还线程池

## newSingleThreadScheduledExecutor

大小为1的线程池,拓展了定时间执行某任务的功能,例如延迟,周期执行任务





以上两者都会创建一个周期性任务,第一次任务都开始于初始延时

前者:后续任务按照初始延时+n\*间隔参数执行

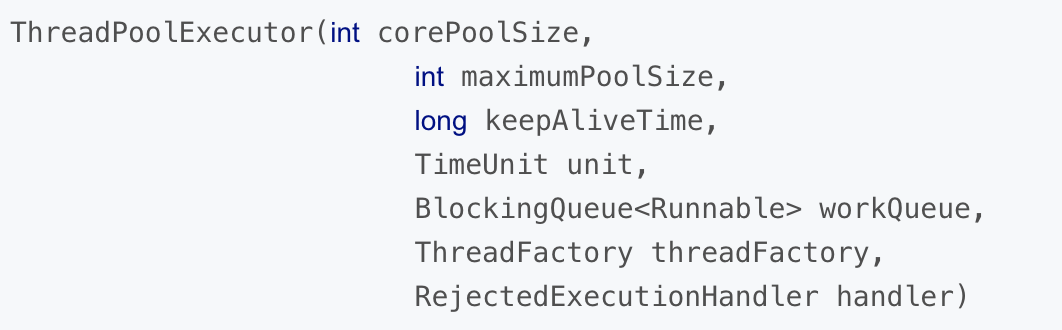
后者:后续任务以上一次任务结束时间+间隔参数执行

## newScheduledThreadPool

同上,但是可以指定线程数量,

## 线程池内部实现原理 (重点)

通过TheradPoolExecutor实现



参数解释

corePoolSize : 核心线程数

maximumPoolSize : 线程池的最大线程数

workQueue :任务队列,已经提交但是尚未被执行的任务

threadFactory : 线程工厂

handler: 拒绝策略

### workQueue 任务队列

直接提交的队列 SynchronousQueue

有界的队列 ArrayBolckingQueue

无界队列 LinkedBlockingQueue

优先级队列 PriorityBlockQueue

### RejectedExecutorHandler 拒绝策略

AbortPolicy策略 该策略直接抛出异常，阻止系统工作

CallerRunsPolicy策略

只要线程池未关闭，该策略直接在调用者线程中运行当前被丢弃的任务。显然这样不会真的丢弃任务，但是，会影响调用线程性能

DiscardOledestPolicy策略

丢弃最老的一个请求任务，也就是丢弃一个即将被执行的任务，并尝试再次提交当前任务。

DiscardPolicy策略

默默的丢弃无法处理的任务，不予任何处理。

### 扩展线程池

