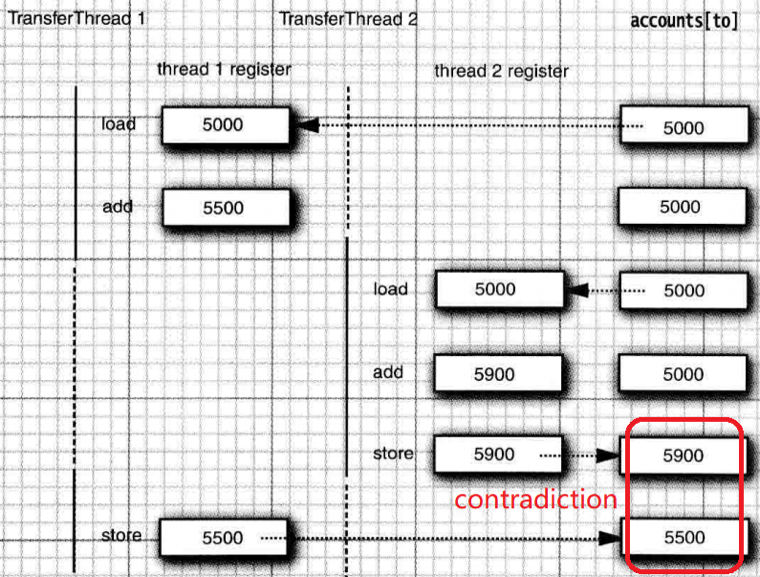
*Multithread: A thread is a thread of execution in a program.*



抢占式调度系统给每一个可运行线程一个时间片来执行任务。当时间片用完，操作系统剥夺该线程的运行权，并给另一个线程运行机会。当选择下一个线程时，操作系统考虑线程的优先级。多处理器的机器上，多线程并行。但如果线程的数目大于处理器支持的数目，调度器依然采用时间片机制。当thread进行同步阻塞等待时(jdbc，socket)，thread.status=RUNNABLE。因为jvm层面的状态，不细化到os层面的waiting状态。BLOCKED指无法进入synchronized代码块。

**Property:**

1. priority
2. daemon thread, 辅助用户线程，随时会结束。The Java Virtual Machine exits when the only threads running are all daemon threads.
3. UncaughtExceptionHandler用途：线程内部的异常抛出外部处理
4. ThreadGroup

**Problem：**(万恶之源：多线程共享同一块存储空间)

共享且可变的数据/资源的竞争和一致性问题

**Solution:** (线程安全：多线程运行，不存在二义性，符合预期。)

思路：代码/过程的原子性和时序先后依赖问题

1. Lock --> resource(data/object) 【同步阻塞等待的思路】

任何时刻只有一个线程进入临界区。一旦一个线程封锁了锁对象，其他任何线程都无法通过lock语句。当其他线程调用lock时，它们被阻塞，直到第一个线程unlock锁对象。

FAULT: deadlock、starvation、livelock

1. volatile + CompareAndSwap乐观锁：写入时，检查数据来源是否被其他线程改写，若改变，则重取数据重新操作一遍；若不变，则写入。

**synchronized【个人，Lock】**

1. 修饰非static函数时，synchronized(this){}。锁定函数所在的对象。
2. 修饰static函数时，synchronized(.class){}。锁定函数所在的类。

**wait/notifyAll【团队，Condition】**

1. This method wait() causes the current thread to place itself in the wait set for this object and then to relinquish any and all synchronization claims on this object. Note that only the locks on this object are relinquished; any other objects on which the current thread may be synchronized remain locked while the thread waits.阻塞，解锁，等待激活通知。
2. The current thread must own this object's monitor lock. 因为wait()和notify()之间是通过对象的同步锁关联起来的。这也是notify(), wait()等函数定义在Object中，而不是Thread中的原因。
3. while (isOK(resource)) { // 避免重新激活后，资源的改变

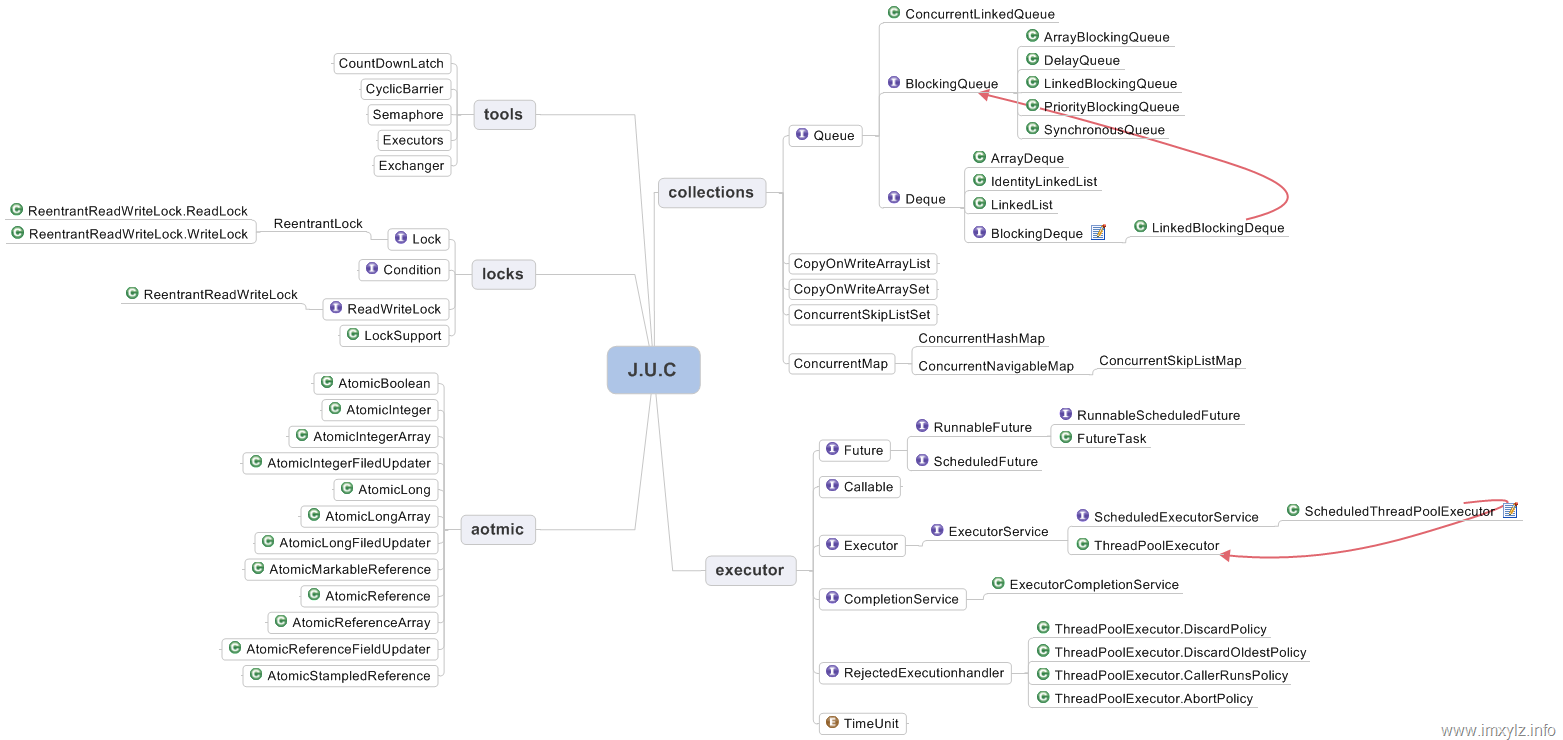
object.wait();

}

1. 若1个condition对应n个wait()：必须有>n次signal()，否则容易引发死锁。若只通知一次，建议使用signalAll()。
2. 场景：线程间的协作、通信。

**volatile**

1. 表明该变量的改写(写穿透CPU cache到内存)对所有线程可见，但不保证在该变量上的操作是原子性的。如果修饰数组，只表示引用可见，不保证数组的元素可见。
2. 重排序Reordering：允许编译器优化，调整指令执行顺序。



***原子类***

1. 基本类型
2. 数组
3. 对象的引用
4. 对象的属性

***锁***

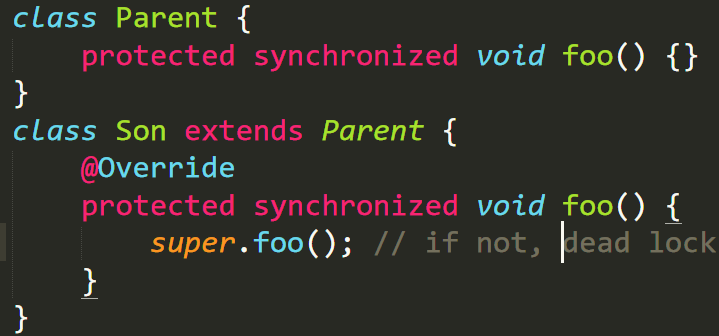
1. AbstractQueuedSynchronizer

a) 模式：exclusive mode, shared mode

b) 实现：tryAcquire()检查锁被占用的线程和次数，不符合的线程addWaiter()入队等待，acquireQueued()取队头并检查锁，线程往下执行，不符合的线程LockSupport.park(this)阻塞

1. A Condition instance is intrinsically bound to a lock. To obtain a Condition instance for a particular Lock instance use its newCondition() method.
2. ReentrantLock互斥锁

为什么需要可重入(同一个线程对同一资源的重复加锁)？



1. ReentrantReadWriteLock：readLock排斥写操作，writeLock排斥读写操作。
2. Semaphore​(int permits, boolean fair)信号量
3. CountDownLatch倒数闸，如：一篇文章划分5个线程词频统计，设定latchSize=5，每个线程完成后countDown一次，当latchSize=0，表示统计全部完成，可以开始汇总线程。
4. CyclicBarrier循环屏障，如：10个人跑3圈，等最后1个人跑完第1圈时，所有人才开始跑下一圈(第2圈)。If the current thread is not the last to arrive then it is disabled for thread scheduling purposes and lies dormant until one of the following things happens:

The last thread arrives; or

Some other thread interrupts the current thread; or

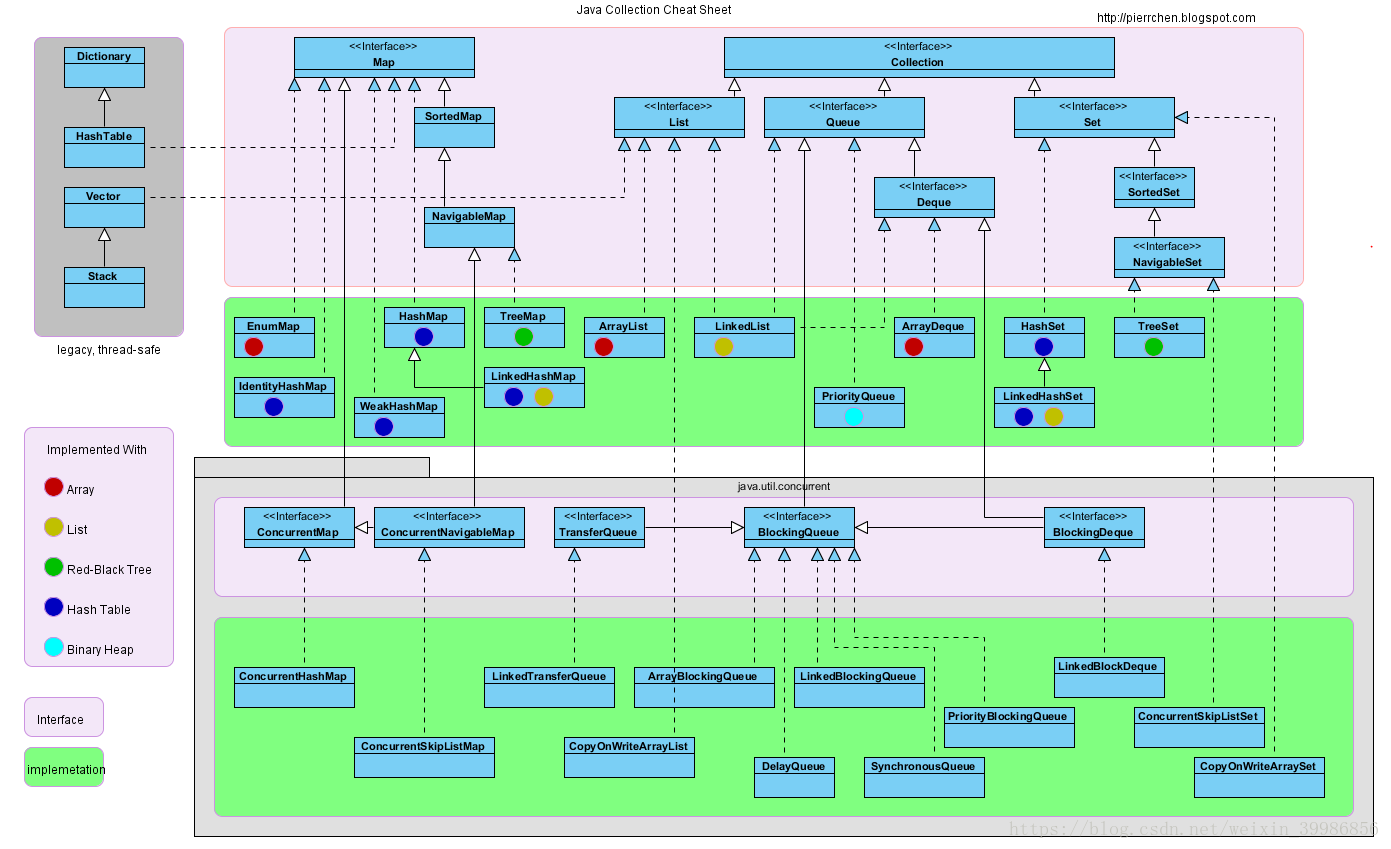
Some other thread interrupts one of the other waiting threads; or

Some other thread times out while waiting for barrier; or

Some other thread invokes reset() on this barrier.

1. 设置超时timeout：允许线程打破死锁。
2. 状态不变/无状态的线程，一定是线程安全的。线程封闭thread confinement：非线程安全的对象，仅在单线程内读写。实例封闭instance confinement：状态变量不超出对象的作用域，避免不可控的状态变化。

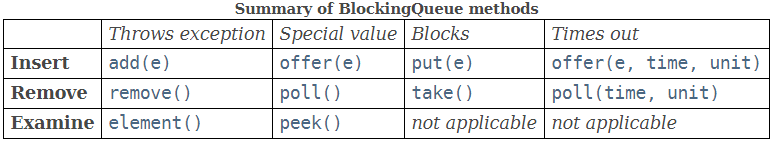
***集合***



1. 阻塞队列BlockingQueue，常用于生产者-消费者模型

·wait for the queue to become non-empty when retrieving an element

·wait for space to become available in the queue when storing an element.



ArrayBlockingQueue数组

LinkedBlockingQueue链表

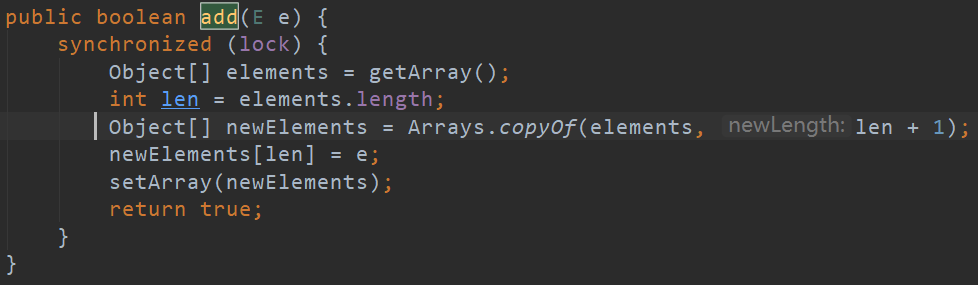
PriorityBlockingQueue优先级队列

DelayQueue过了延迟时间才能获取，如：缓存的失效状态，定时调度

SynchronousQueue一次插入必须跟着一次删除，否则无法继续

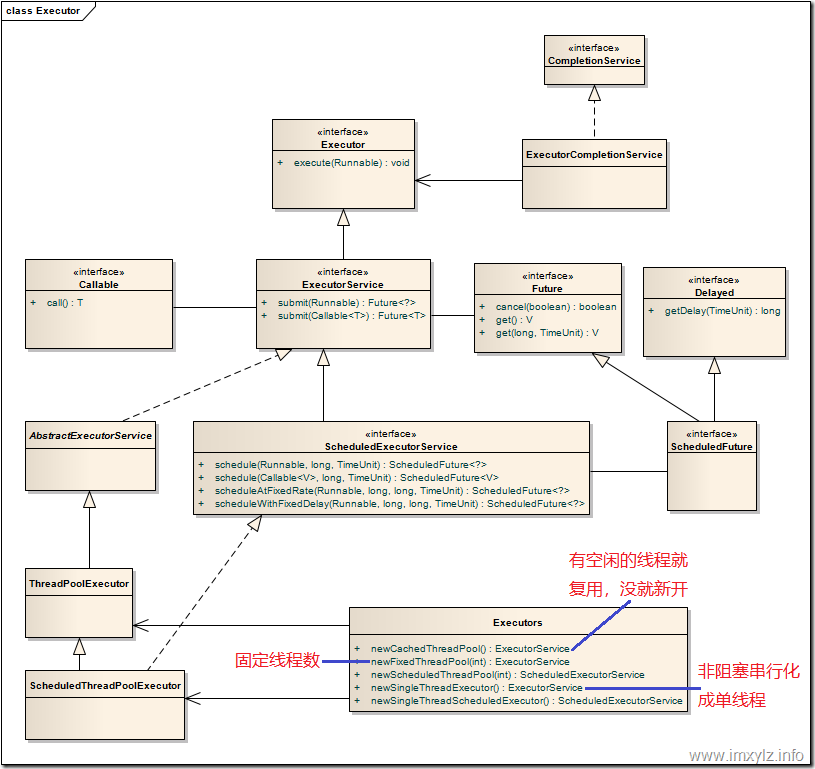
LinkedTransferQueue阻塞传递队列

1. CopyOnWriteArrayList (数据量大或写频繁，效率低下，原因如下)



1. CopyOnWriteArraySet：基于CopyOnWriteArrayList的addIfAbsent(E)
2. ConcurrentHashMap (JDK8的实现大改)

***线程池***



1. Executor框架：线程是任务异步执行的机制，Executor是任务调度的策略管理。
2. Runnable无返回值无异常的任务
3. Callable有返回值有异常的任务
4. Future异步结果. A Future represents the result of an asynchronous computation. The result can only be retrieved using method get when the computation has completed, blocking if necessary until it is ready. Once a computation has completed, the computation cannot be cancelled.
5. 执行任务

submit()：其实就是execute()，入队、线程start()，没有Future.get()操作

invokeAny()：Executes the given tasks, returning the result of one that has completed

invokeAll()：Executes the given tasks, returning a list of Futures holding their status and results when all complete or the timeout expires, whichever happens first.

1. 生命周期

运行 -> 关闭(不接受新任务，处理旧任务) --RejectedExecutionHandler--> 终止

1. 线程的取消与关闭

1) Java没有提供任何机制来安全终止线程，只有一种协作式中断。

2) 良好的线程设计，应根据中断状态，在合适的时候(取消点)结束任务。

interrupt() 设置中断状态

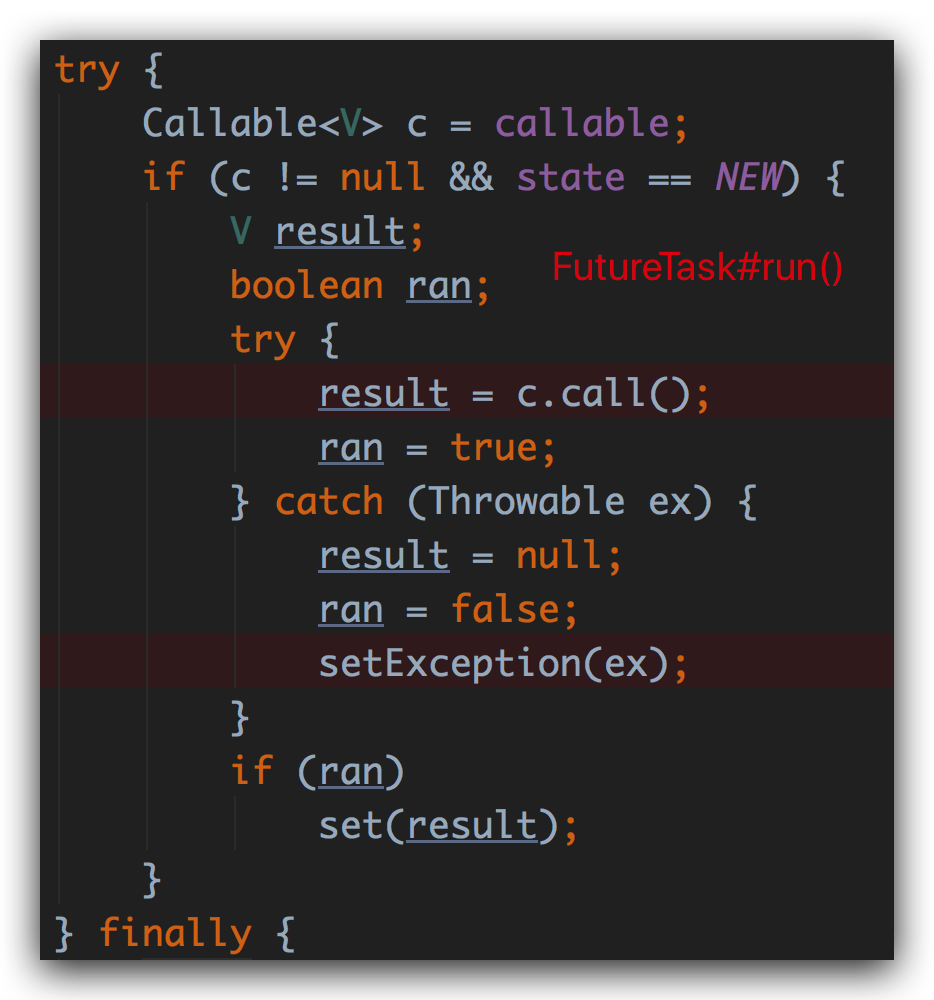
isInterrupted() 返回中断状态

interrupted() 返回中断状态，并清除

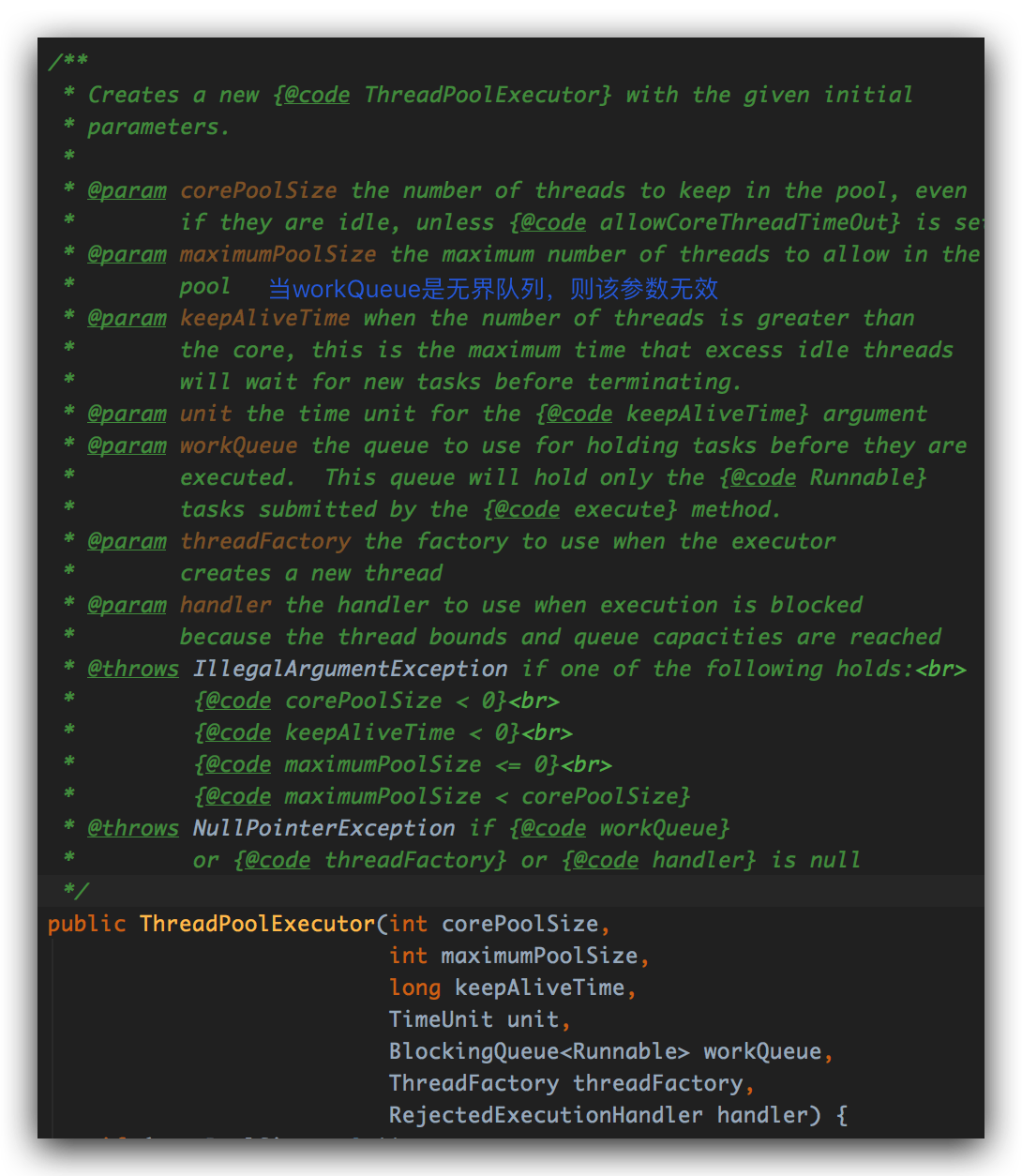
1. 线程池中的线程之异常处理

- execute(Runnable)：通过ThreadFactory绑定一个UncaughtExceptionHandler捕获异常

- submit(Callable)：通过Future.get捕获异常



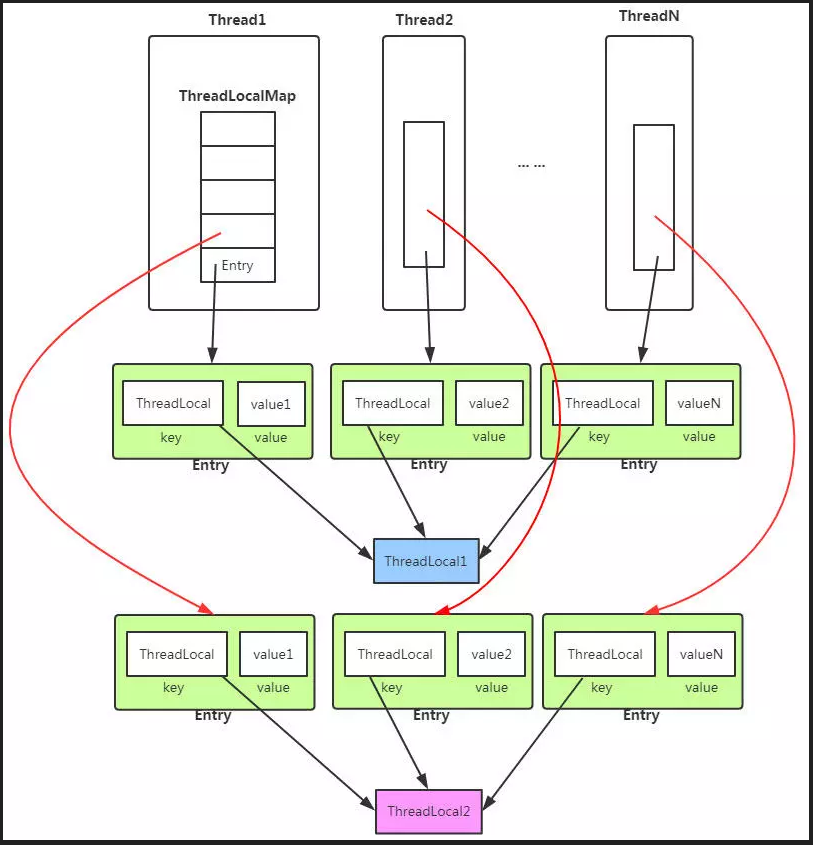
1. 通用线程池的构造函数



**细枝末节**

1. ThreadLocal类是修饰变量的，是在控制它的线程作用域，同一变量的不同线程副本。

建议：不再使用后，请调用remove()从而调用expungeStaleEntry()，来避免挖坑。



1. ThreadLocalRandom：利用ThreadLocal避免对同一个atom seed的竞争，适用于随机性要求不高、提高并发的场景。如果要求强随机性，需使用SecureRandom.