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

**Recommended usage：**

class Job implements Runnable {

@Override

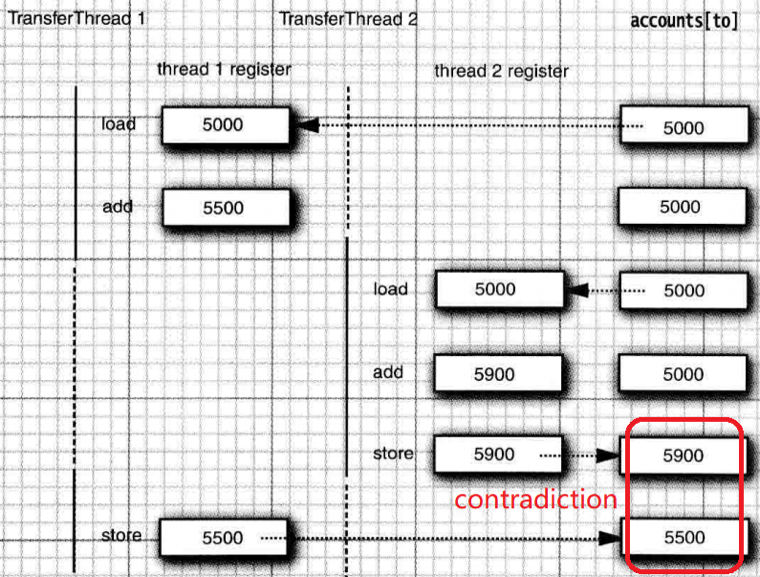
public void run() { //线程执行函数

// TODO

} //线程的终止：run()正常结束，或异常未捕获。

}

new Thread(new Job()).start();



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。

****

**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
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。

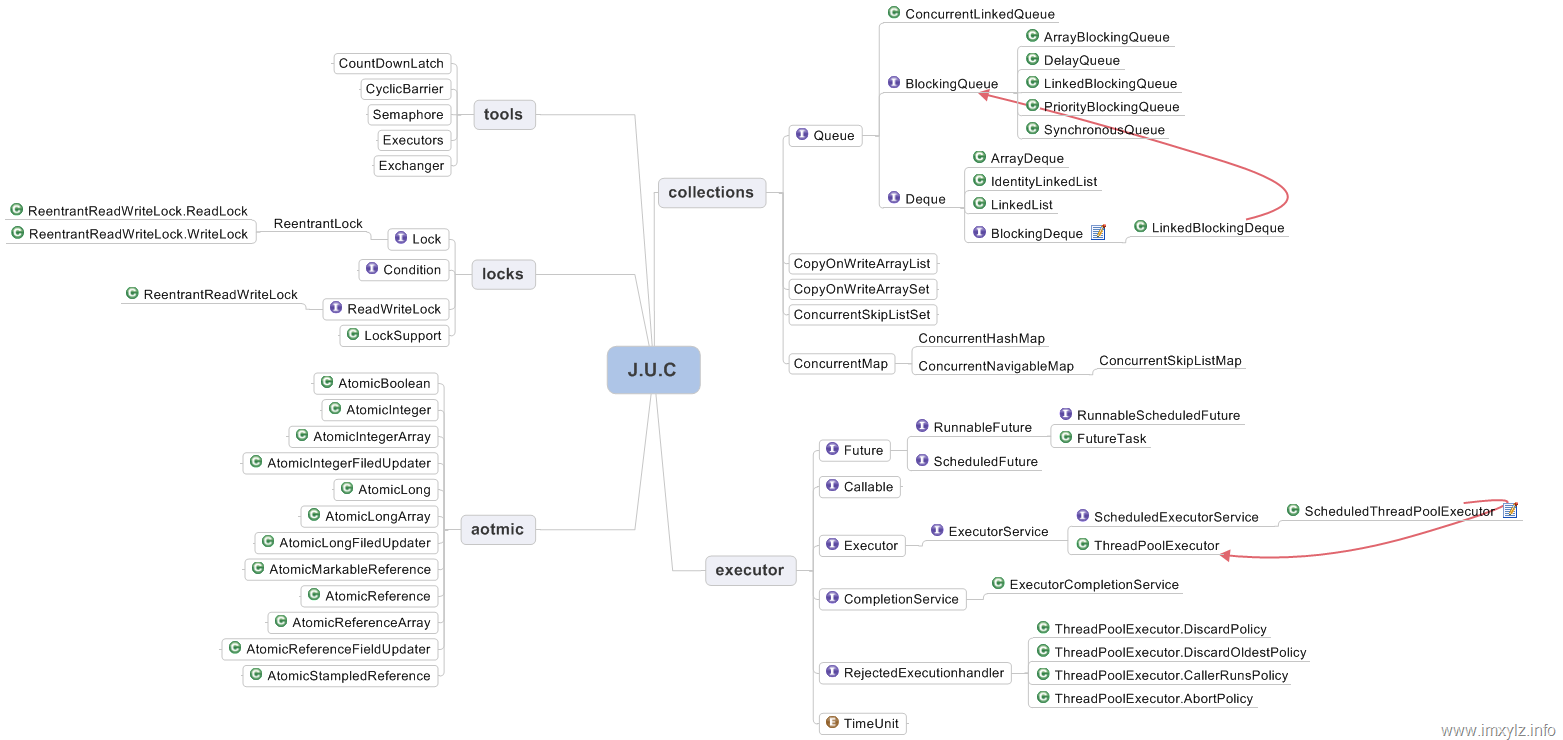
**wait/notifyAll【团队，Condition】**

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. 场景：线程间的协作、通信。



***原子类***

1. 基本类型
2. 数组
3. 对象的引用
4. 对象的属性

***锁***

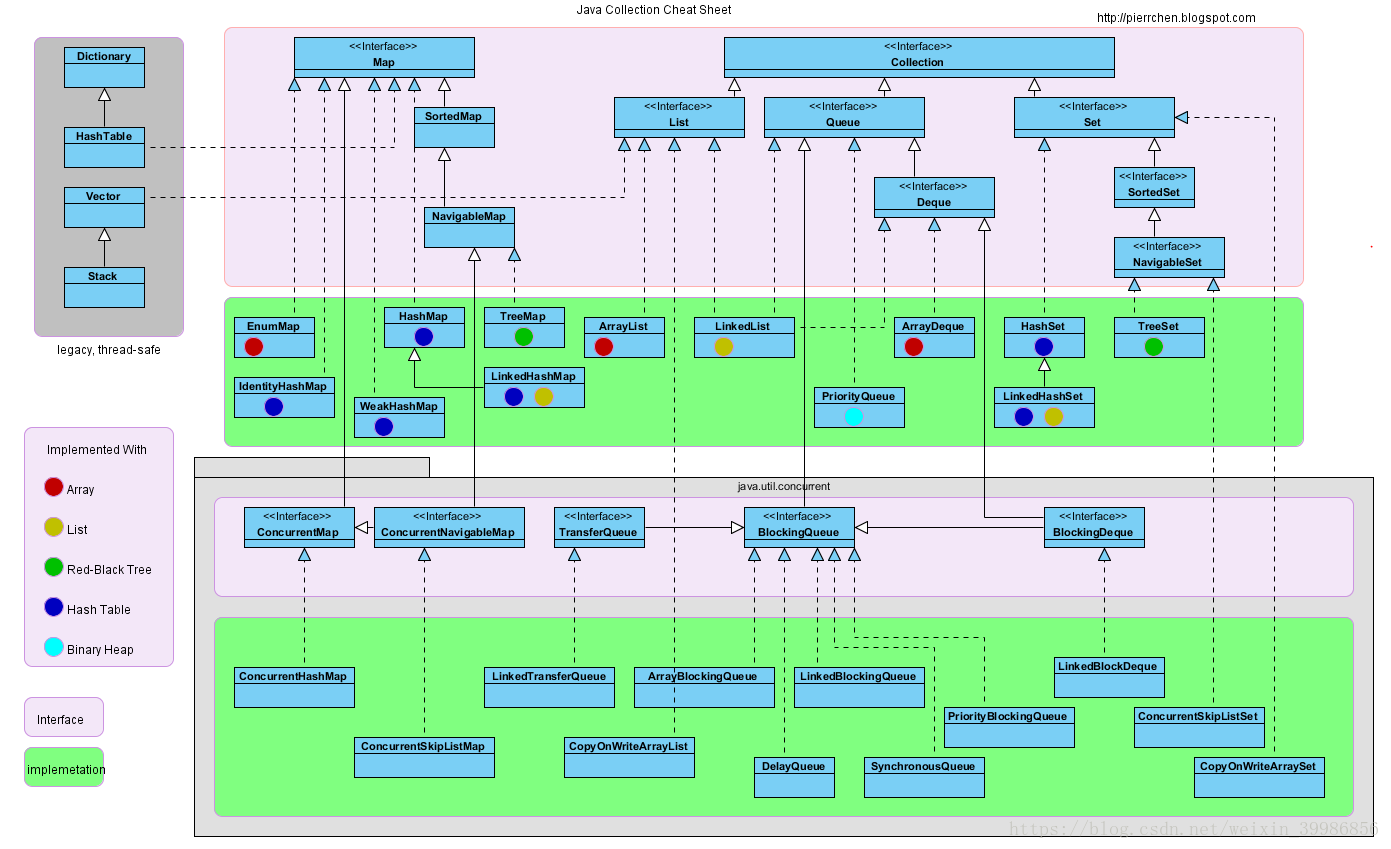
1. AbstractQueuedSynchronizer

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(可重入，指一个线程对同一资源的重复加锁)
3. ReentrantReadWriteLock：readLock排斥写操作，writeLock排斥读写操作。
4. Semaphore​(int permits, boolean fair)信号量

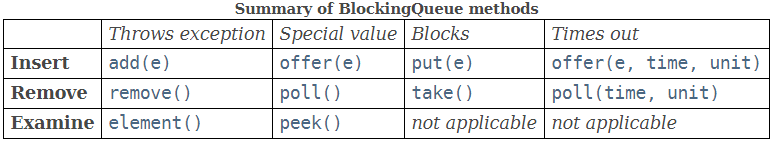
***集合***



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

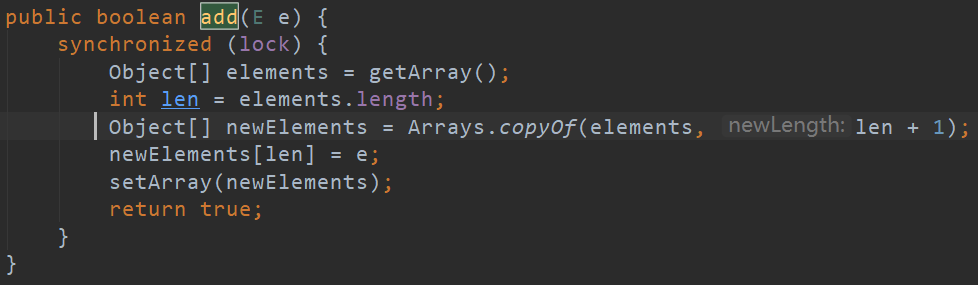
LinkedBlockingQueue链表

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

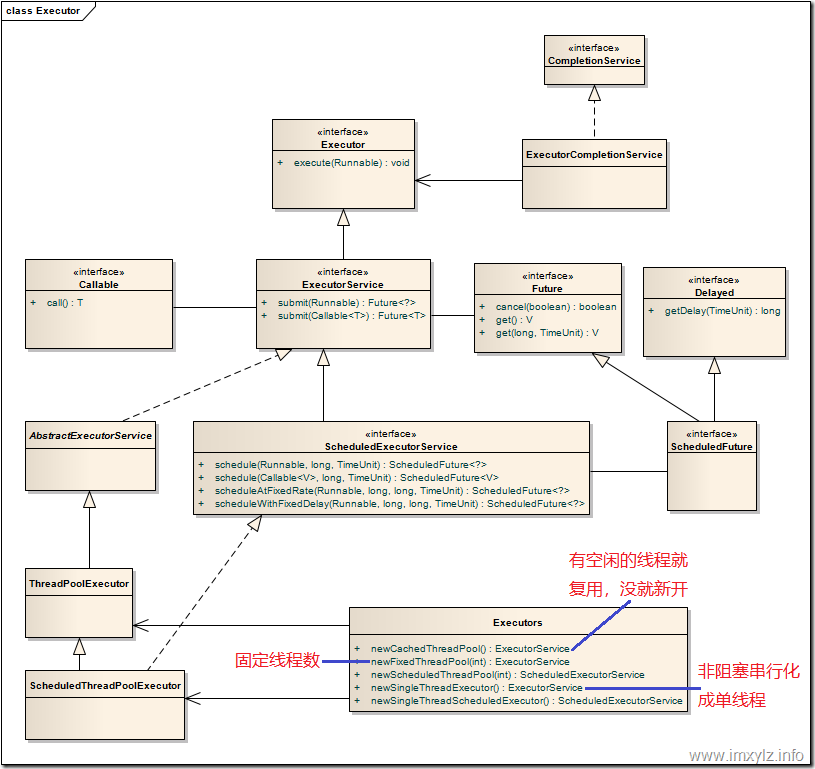
SynchronousQueue一次插入必须跟着一次删除，否则无法继续

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



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

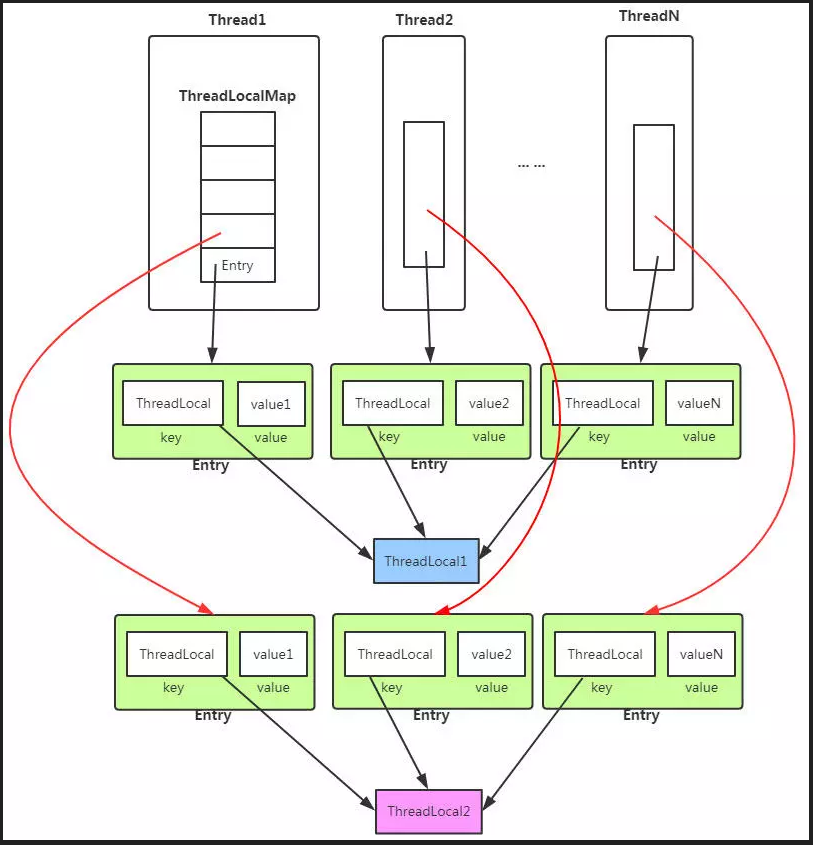
***线程池***



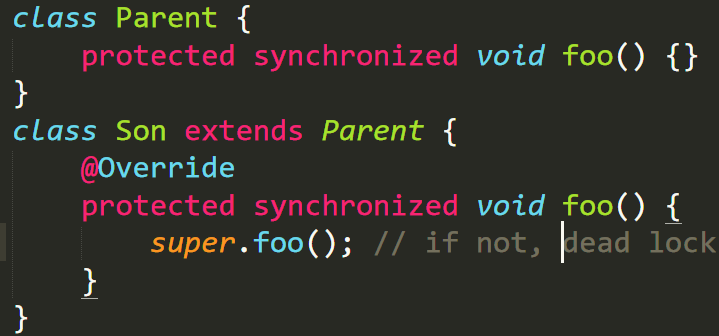
细化

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：不用了，请调用remove()从而调用expungeStaleEntry()，来避免挖坑。



1. volatile：表明该变量的改写(跳过CPU cache读写内存)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. (JDK8)Long[Accumulator](https://docs.oracle.com/javase/10/docs/api/java/util/concurrent/atomic/LongAccumulator.html)在高并发下冲突更少，但不精确。若保证精确，则使用AtomicLong
3. 1个condition若对应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()
4. 设置超时：允许线程打破死锁。
5. 无状态的线程，一定是线程安全的。
6. 初始化锁，防止初始化多次。
7. 为什么需要可重入？避免同一个线程的资源不必要受阻。



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.