### 9. volatile关键字和synchornized关键字

volatile是Java提供的一种轻量级的同步机制，在并发编程中，它也扮演着比较重要的角色。同synchronized相比（synchronized通常称为重量级锁），volatile更轻量级，相比使用synchronized所带来的庞大开销，倘若能恰当的合理的使用volatile，自然是美事一桩。

**可见性：**一个线程修改共享变量时，其他线程能够立即知道这个修改；

使用Volatile关键字将解决线程之间可见性, 强制线程每次读取该值的时候都去“主内存”中取值

9.1 volatile是什么，适用场合及其作用（建议了解一下原理）

* 对变量的写操作不依赖于当前值。
* 该变量没有包含在具有其他变量的不变式中。

9.2 synchorinzed的作用及使用方式（建议了解一下原理）

9.3 volatile和synchornized的关系

1.volatile本质是在告诉jvm当前变量在寄存器（工作内存）中的值是不确定的，需要从主存中读取；synchronized则是锁定当前变量，只有当前线程可以访问该变量，其他线程被阻塞住；

2.volatile仅能使用在变量级别；synchronized则可以使用在变量、方法、和类级别的；

3.volatile仅能实现变量的修改可见性，不能保证原子性；而synchronized则可以保证变量的修改可见性和原子性；

4.volatile不会造成线程的阻塞；synchronized可能会造成线程的阻塞；

5.volatile标记的变量不会被编译器优化；synchronized标记的变量可以被编译器优化。

**10. synchornized和Reentrantlock的区别**

① 底层实现上来说，synchronized 是JVM层面的锁，是Java关键字，通过monitor对象来完成（monitorenter与monitorexit），对象只有在同步块或同步方法中才能调用wait/notify方法，ReentrantLock 是从jdk1.5以来（java.util.concurrent.locks.Lock）提供的API层面的锁。

② **是否可手动释放：**

synchronized 不需要用户去手动释放锁，synchronized 代码执行完后系统会自动让线程释放对锁的占用； ReentrantLock则需要用户去手动释放锁，如果没有手动释放锁，就可能导致死锁现象。一般通过lock()和unlock()方法配合try/finally语句块来完成，使用释放更加灵活。

③ **是否可中断**

synchronized是不可中断类型的锁，除非加锁的代码中出现异常或正常执行完成； ReentrantLock则可以中断，可通过trylock(long timeout,TimeUnit unit)设置超时方法或者将lockInterruptibly()放到代码块中，调用interrupt方法进行中断。

④  **是否公平锁**

synchronized为非公平锁 ReentrantLock则即可以选公平锁也可以选非公平锁，通过构造方法new ReentrantLock时传入boolean值进行选择，为空默认false非公平锁，true为公平锁。

⑤ **锁是否可绑定条件Condition**

synchronized不能绑定； ReentrantLock通过绑定Condition结合await()/singal()方法实现线程的精确唤醒，而不是像synchronized通过Object类的wait()/notify()/notifyAll()方法要么随机唤醒一个线程要么唤醒全部线程。

⑥ **锁的对象**

synchronzied锁的是对象，锁是保存在对象头里面的，根据对象头数据来标识是否有线程获得锁/争抢锁；ReentrantLock锁的是线程，根据进入的线程和int类型的state标识锁的获得/争抢。

**12、为什么 ArrayList 的 elementData 加上 transient 修饰？**

http://www.javashuo.com/article/p-moiudvqi-gy.html

# synchronized保证线程安全的原理（jvm层面）

<https://blog.csdn.net/cckkpp/article/details/88595419>

10、ConcurrentHashMap 底层具体实现知道吗？实现原理是什么？

![图示

描述已自动生成]()

在JDK1.7版本中，ConcurrentHashMap的数据结构是由一个Segment数组和多个HashEntry组成，Segment数组的意义就是将一个大的table分割成多个小的table来进行加锁，也就是上面的提到的锁分离技术，而每一个Segment元素存储的是HashEntry数组+链表，这个和HashMap的数据存储结构一样

JDK1.8的实现已经摒弃了Segment的概念，而是直接用Node数组+链表+红黑树的数据结构来实现，并发控制使用Synchronized和CAS来操作，整个看起来就像是优化过且线程安全的HashMap，虽然在JDK1.8中还能看到Segment的数据结构，但是已经简化了属性，只是为了兼容旧版本

说明：ConcurrentHashMap的数据结构（数组+链表+红黑树），桶中的结构可能是链表，也可能是红黑树，红黑树是为了提高查找效率。

3、comparable 和 comparator的区别？

两种方法各有优劣， 用Comparable 简单， 只要实现Comparable接口的对象直接就成为一个可以比较的对象，但是需要修改源代码。 用Comparator 的好处是不需要修改源代码， 而是另外实现一个比较器， 当某个自定义的对象需要作比较的时候，把比较器和对象一起传递过去就可以比大小了， 并且在Comparator 里面用户可以自己实现复杂的可以通用的逻辑，使其可以匹配一些比较简单的对象，那样就可以节省很多重复劳动了。

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

5、TreeMap 和 TreeSet 在[排序](file:////jump/super-jump/word%3fword=%25E6%258E%2592%25E5%25BA%258F" \t "_blank)时如何比较元素？Collections 工具类中的 sort()方法如何比较元素？

TreeSet要求存放的对象所属的类必须实现Comparable接口，该接口提供了比较元素的compareTo()方法，当插入元素时会回调该方法比较元素的大小。TreeMap要求存放的键值对映射的键必须实现Comparable接口从而根据键对元素进行排序。Collections工具类的sort方法有两种重载的形式，第一种要求传入的待排序容器中存放的对象必须实现Comparable接口以实现元素的比较；第二种不强制性的要求容器中的元素必须可比较，但是要求传入第二个参数，参数是Comparator接口的子类型（需要重写compare方法实现元素的比较），相当于一个临时定义的排序规则，其实就是通过接口注入比较元素大小的算法，也是对回调模式的应用（Java中对函数式编程的支持）