# 一：lock和synchronized的同步区别与选择

**区别如下：**   
1. lock是一个接口，而synchronized是java的一个关键字，synchronized是内置的语言实现；（具体实现上的区别在《Java虚拟机》中有讲解底层的CAS不同，以前有读过现在又遗忘了。）   
2. synchronized在发生异常时候会自动释放占有的锁，因此不会出现死锁；而lock发生异常时候，不会主动释放占有的锁，必须手动unlock来释放锁，可能引起死锁的发生。（所以最好将同步代码块用try catch包起来，finally中写入unlock，避免死锁的发生。）   
3. lock等待锁过程中可以用interrupt来终端等待，而synchronized只能等待锁的释放，不能响应中断；   
4. lock可以通过trylock来知道有没有获取锁，而synchronized不能；   
5. Lock可以提高多个线程进行读操作的效率。（可以通过readwritelock实现读写分离）

**5.Lock和synchronized的选择**

　　总结来说，Lock和synchronized有以下几点不同：

　　1）Lock是一个接口，而synchronized是Java中的关键字，synchronized是内置的语言实现；

　　2）synchronized在发生异常时，会自动释放线程占有的锁，因此不会导致死锁现象发生；而Lock在发生异常时，如果没有主动通过unLock()去释放锁，则很可能造成死锁现象，因此使用Lock时需要在finally块中释放锁；

　　3）Lock可以让等待锁的线程响应中断，而synchronized却不行，使用synchronized时，等待的线程会一直等待下去，不能够响应中断；

　　4）通过Lock可以知道有没有成功获取锁，而synchronized却无法办到。

　　5）Lock可以提高多个线程进行读操作的效率。

　　在性能上来说，如果竞争资源不激烈，两者的性能是差不多的，而当竞争资源非常激烈时（即有大量线程同时竞争），此时Lock的性能要远远优于synchronized。所以说，在具体使用时要根据适当情况选择。

synchronized是java中的一个关键字，也就是说是Java语言内置的特性。那么为什么会出现Lock呢？

　　在上面一篇文章中，我们了解到如果一个代码块被synchronized修饰了，当一个线程获取了对应的锁，并执行该代码块时，其他线程便只能一直等待，等待获取锁的线程释放锁，而这里获取锁的线程释放锁只会有两种情况：

　　1）获取锁的线程执行完了该代码块，然后线程释放对锁的占有；

　　2）线程执行发生异常，此时JVM会让线程自动释放锁。

　　那么如果这个获取锁的线程由于要等待IO或者其他原因（比如调用sleep方法）被阻塞了，但是又没有释放锁，其他线程便只能干巴巴地等待，试想一下，这多么影响程序执行效率。

因此就需要有一种机制可以不让等待的线程一直无期限地等待下去（比如只等待一定的时间或者能够响应中断），通过Lock就可以办到。

再举个例子：当有多个线程读写文件时，读操作和写操作会发生冲突现象，写操作和写操作会发生冲突现象，但是读操作和读操作不会发生冲突现象。

　　但是采用synchronized关键字来实现同步的话，就会导致一个问题：

　　如果多个线程都只是进行读操作，所以当一个线程在进行读操作时，其他线程只能等待无法进行读操作。

　　因此就需要一种机制来使得多个线程都只是进行读操作时，线程之间不会发生冲突，通过Lock就可以办到。

　　另外，通过Lock可以知道线程有没有成功获取到锁。这个是synchronized无法办到的。

　　总结一下，也就是说Lock提供了比synchronized更多的功能。但是要注意以下几点：

　　1）Lock不是Java语言内置的，synchronized是Java语言的关键字，因此是内置特性。Lock是一个类，通过这个类可以实现同步访问；

　　2）Lock和synchronized有一点非常大的不同，采用synchronized不需要用户去手动释放锁，当synchronized方法或者synchronized代码块执行完之后，系统会自动让线程释放对锁的占用；而Lock则必须要用户去手动释放锁，如果没有主动释放锁，就有可能导致出现死锁现象。

## 二.java.util.concurrent.locks包下常用的类

　　下面我们就来探讨一下java.util.concurrent.locks包中常用的类和接口。

**1.Lock**

　　首先要说明的就是Lock，通过查看Lock的源码可知，Lock是一个接口：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | public interface Lock {      void lock();      void lockInterruptibly() throws InterruptedException;      boolean tryLock();      boolean tryLock(long time, TimeUnit unit) throws InterruptedException;      void unlock();      Condition newCondition();  } |

下面来逐个讲述Lock接口中每个方法的使用，lock()、tryLock()、tryLock(long time, TimeUnit unit)和lockInterruptibly()是用来获取锁的。unLock()方法是用来释放锁的。newCondition()这个方法暂且不在此讲述，会在后面的线程协作一文中讲述。

　　在Lock中声明了四个方法来获取锁，那么这四个方法有何区别呢？

　　首先lock()方法是平常使用得最多的一个方法，就是用来获取锁。如果锁已被其他线程获取，则进行等待。

　　由于在前面讲到如果采用Lock，必须主动去释放锁，并且在发生异常时，不会自动释放锁。因此一般来说，使用Lock必须在try{}catch{}块中进行，并且将释放锁的操作放在finally块中进行，以保证锁一定被被释放，防止死锁的发生。通常使用Lock来进行同步的话，是以下面这种形式去使用的：

Lock lock = ...;

lock.lock();

try{

    //处理任务

}catch(Exception ex){

}finally{

    lock.unlock();   //释放锁

}

而用synchronized修饰的话，当一个线程处于等待某个锁的状态，是无法被中断的，只有一直等待下去。

## 三.锁的相关概念介绍

　　在前面介绍了Lock的基本使用，这一节来介绍一下与锁相关的几个概念。

**1.可重入锁**

　　如果锁具备可重入性，则称作为可重入锁。像synchronized和ReentrantLock都是可重入锁，可重入性在我看来实际上表明了锁的分配机制：基于线程的分配，而不是基于方法调用的分配。举个简单的例子，当一个线程执行到某个synchronized方法时，比如说method1，而在method1中会调用另外一个synchronized方法method2，此时线程不必重新去申请锁，而是可以直接执行方法method2。

　　看下面这段代码就明白了：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | class MyClass {      public synchronized void method1() {          method2();      }      public synchronized void method2() {      }  } |

 　　上述代码中的两个方法method1和method2都用synchronized修饰了，假如某一时刻，线程A执行到了method1，此时线程A获取了这个对象的锁，而由于method2也是synchronized方法，假如synchronized不具备可重入性，此时线程A需要重新申请锁。但是这就会造成一个问题，因为线程A已经持有了该对象的锁，而又在申请获取该对象的锁，这样就会线程A一直等待永远不会获取到的锁。

　　而由于synchronized和Lock都具备可重入性，所以不会发生上述现象。

**2.可中断锁**

　　可中断锁：顾名思义，就是可以相应中断的锁。

　　在Java中，synchronized就不是可中断锁，而Lock是可中断锁。

　　如果某一线程A正在执行锁中的代码，另一线程B正在等待获取该锁，可能由于等待时间过长，线程B不想等待了，想先处理其他事情，我们可以让它中断自己或者在别的线程中中断它，这种就是可中断锁。

　　在前面演示lockInterruptibly()的用法时已经体现了Lock的可中断性。

**3.公平锁**

　　公平锁即尽量以请求锁的顺序来获取锁。比如同是有多个线程在等待一个锁，当这个锁被释放时，等待时间最久的线程（最先请求的线程）会获得该所，这种就是公平锁。

　　非公平锁即无法保证锁的获取是按照请求锁的顺序进行的。这样就可能导致某个或者一些线程永远获取不到锁。

　　在Java中，synchronized就是非公平锁，它无法保证等待的线程获取锁的顺序。

　　而对于ReentrantLock和ReentrantReadWriteLock，它默认情况下是非公平锁，但是可以设置为公平锁。

**4.读写锁**

　　读写锁将对一个资源（比如文件）的访问分成了2个锁，一个读锁和一个写锁。

　　正因为有了读写锁，才使得多个线程之间的读操作不会发生冲突。

　　ReadWriteLock就是读写锁，它是一个接口，ReentrantReadWriteLock实现了这个接口。

　　可以通过readLock()获取读锁，通过writeLock()获取写锁。

**Java并发编程：volatile关键字解析**

**原子性和可见性**

1,volatile是一个类型修饰符，他是被设计用来修饰被不同线程访问和修改的变量，被volatile定义的变量，系统每次用到他时都是直接从对应的内存当中读取的，而不是利用缓存。

**volatile关键字就是为了保证内存数据的可见性**

**但是volatile和Synchronized具有不同姓：**

**1，相对于synchronized是一种较为轻量级的同步策略**

**2，synchronized具有互斥性，也就是两个线程同时抢一个锁，当一个线程进去了，另外一个线程就进不去，只有等一个线程出来了，另外一个线程才能进去**

**3，volatile就不具备“互斥性”如果有一个线程在访问共享数据，另外一个线程也可以同时访问共享数据，只不过所有的访问数据都在主存中完成而已**

**4，volatile不能保证变量的原子性http://www.importnew.com/18126.html**

当程序在运行过程中，会将运算需要的数据从主存复制一份到CPU的高速缓存当中，那么CPU进行计算时就可以直接从它的高速缓存读取数据和向其中写入数据，当运算结束之后，再将高速缓存中的数据刷新到主存当中。举个简单的例子，比如下面的这段代码：

|  |  |
| --- | --- |
| 1 | i = i + 1; |

当线程执行这个语句时，会先从主存当中读取i的值，然后复制一份到高速缓存当中，然后CPU执行指令对i进行加1操作，然后将数据写入高速缓存，最后将高速缓存中i最新的值刷新到主存当中。