**1、1.同步方法 (静态方法锁住类对象，其它方法锁住实例对象)**

    即有synchronized关键字修饰的方法。

    由于java的每个对象都有一个内置锁，当用此关键字修饰方法时，

    内置锁会保护整个方法。在调用该方法前，需要获得内置锁，否则就处于阻塞状态。

**2.同步代码块 （锁住标记的对象）**

    即有synchronized关键字修饰的语句块。

    被该关键字修饰的语句块会自动被加上内置锁，从而实现同步

### 3.使用重入锁实现线程同步

在JavaSE5.0中新增了一个java.util.concurrent包来支持同步。ReentrantLock类是可重入、互斥、实现了Lock接口的锁， 它与使用synchronized方法和快具有相同的基本行为和语义，并且扩展了其能力。  
     ReenreantLock类的常用方法有：  
         **ReentrantLock() : 创建一个ReentrantLock实例   
         lock() : 获得锁**

**代码  
         unlock() : 释放锁**    注：ReentrantLock()还有一个可以创建公平锁的构造方法，但由于能大幅度降低程序运行效率，不推荐使用

**4.使用特殊域变量(volatile)实现线程同步（无法保证原子性的，写到这儿来只是强调一下，它只保证了可见性和有序性。）**

    a.volatile关键字为域变量的访问提供了一种免锁机制，

    b.使用volatile修饰域相当于告诉[虚拟机](http://www.2cto.com/os/xuniji/" \t "_blank)该域可能会被其他线程更新，

    c.因此每次使用该域就要重新计算，而不是使用寄存器中的值

d.volatile不会提供任何原子操作，它也不能用来修饰final类型的变量

e.

对于volatile修饰的变量，jvm虚拟机**只是保证从主内存加载到线程工作内存的值是最新的。**

您只能在有限的一些情形下使用 volatile 变量替代锁。要使 volatile 变量提供理想的线程安全，必须同时满足下面两个条件：

* **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。**

**可以使用的场景：**

**将 volatile 变量作为状态标志使用（和lock差不多，但是volatile 赋值true本来这种就是原子操作，所以就刚好可以使用。）**

5.**volatile（保证可见性和有序性）**的一个重要作用就是和**CAS结合，保证了原子性**

最简单的比如i++,用volatile可以保证取得的值是最新的，而cas操作可以保证你修改前后的值只+1，而不会覆盖掉别的线程已经修改过的值，如果别的线程已经修改过，CAS会自动不修改的。

但是没有**volatile时，原子性不保证可见性。**

**CAS改完可能值还在缓存里，不会马上把**工作内存中被修改后的值 写回 主内存。

**CAS只解决了比较和更新的原子性的问题，要保证可见性，需要加锁或者是用volatile修饰变量。**

1. 首先，声明共享变量为volatile；
2. 然后，使用CAS的原子条件更新来实现线程之间的同步；
3. 同时，配合以volatile的读/写和CAS所具有的volatile读和写的内存语义来实现线程之间的通信。

**一句话：volatile变量的读/写和CAS可以实现线程之间的通信。把这些特性整合在一起，就形成了整个concurrent包得以实现的基石。**