synchronized的四种用法

1）修饰方法：synchronized关键字不能被继承，也就是子类重写父类的方法后没有同步的特性

```
public synchronized void method()
{
    // todo
}
```

2）修饰一个代码块：

1.一个线程访问一个对象中的synchronized(this)同步代码块时，其他试图访问该对象的线程将被阻塞

2.当一个线程访问对象的一个synchronized(this)同步代码块时，另一个线程仍然可以访问该对象中的

非synchronized(this)同步代码块。

3）修饰一个静态方法

4）修饰一个类：

Synchronized还可作用于一个类，用法如下:

```
class ClassName {
   public void method() {
      synchronized(ClassName.class) {
         // todo
      }
   }
}
```

总结：

1. 修饰一个代码块，被修饰的代码块称为同步语句块，其作用的范围是大括号{}括起来的代码，作用的对象是调用这个代码块的对象；

2. 修饰一个方法，被修饰的方法称为同步方法，其作用的范围是整个方法，作用的对象是调用这个方法的对象；

3. 修饰一个静态的方法，其作用的范围是整个静态方法，作用的对象是这个类的所有对象；

4. 修饰一个类，其作用的范围是synchronized后面括号括起来的部分，作用主的对象是这个类的所有对象。

无论synchronized关键字加在方法上还是对象上，如果它作用的对象是非静态的，则它取得的锁是对象；如果synchronized作用的对象是一个静态方法或一个类，则它取得的锁是类的

锁，该类所有的对象使用同一把锁

Reentrantlock实现原理

ReentrantLock主要利用CAS+AQS队列来实现。它支持公平锁和非公平锁，两者的实现类似。

源码里有一个名为Sync的内部类，继承了AQS。然后里面还有2个内部类又直接继承了Sync，分别为FairSync以及NofairSync，它们分别代表公平锁以及非公平锁的实现

```
public ReentrantLock(boolean fair) {
    sync = fair ? new FairSync() : new NonfairSync();
}
```

默认非公平锁

```
public ReentrantLock() {
    sync = new NonfairSync();
}
```

ReentrantLock的基本实现可以概括为：先通过CAS尝试获取锁。如果此时已经有线程占据了锁，那就加入AQS队列并且被挂起。当锁被释放之后，排在同步队列队首的线程会被唤醒，然后CAS再次尝试获取锁。

锁实现原理

volatile：与jvm的内存模型有关，每个线程都有一个工作内存，在修改后马上刷新到主内存中，工作内存与主内存之间有个缓存一致协议（MESI），其他处理器会通过嗅探在总线上传播的数据来检查自己工作区域的值是否过期，过期就要设置为无效状态，下次使用的话要去主内存中进行数据更新

Synchronized：基于进入退出Monitor对象来实现方法同步和代码块同步。任何对象都有一个monitor与之关联，并且当一个monitor被持有后，它将处于锁定状态。

monitorenter

每一个对象都有一个monitor，一个monitor只能被一个线程拥有。当一个线程执行到monitorenter指令时会尝试获取相应对象的monitor，获取规则如下：

- 如果monitor的进入数为0，则该线程可以进入monitor，并将monitor进入数设置为1，该线程即为monitor的拥有者。
- 如果当前线程已经拥有该monitor，只是重新进入，则进入monitor的进入数加1，所以synchronized关键字实现的锁是可重入的锁。
- 如果monitor已被其他线程拥有，则当前线程进入阻塞状态，直到monitor的进入数为0，再重新尝试获取monitor。

monitorexit

只有拥有相应对象的monitor的线程才能执行monitorexit指令。每执行一次该指令monitor进入数减1，当进入数为0时当前线程释放monitor，此时其他阻塞的线程将可以尝试获取该monitor。

执行到monitorenter指令的时候，将会去尝试获取对象所对应的monitor的所有权