synchronized(this)、synchronized(\*.class)与synchronized(Object)的区别

synchronized 方法控制对类成员变量的访问：每个类实例对应一把锁，每个 synchronized 方法都必须获得调用该方法的类实例的锁方能执行，否则所属线程阻塞，方法一旦执行，就独占该锁，直到从该方法返回时才将锁释放，此后被阻塞的线程方能获得该锁，重新进入可执行状态。**这种机制确保了同一时刻对于每一个类实例，其所有声明为 synchronized 的成员函数中至多只有一个处于可执行状态（因为至多只有一个能够获得该类实例对应的锁），从而有效避免了类成员变量的访问冲突（只要所有可能访问类成员变量的方法均被声明为 synchronized）。**

在 Java 中，不光是类实例，每一个类也对应一把锁，这样我们也可将类的静态成员函数声明为 synchronized ，以控制其对类的静态成员变量的访问。

synchronized 方法的缺陷：若将一个大的方法声明为synchronized 将会大大影响效率，典型地，若将线程类的方法 run() 声明为 synchronized ，由于在线程的整个生命期内它一直在运行，因此将导致它对本类任何 synchronized 方法的调用都永远不会成功。当然我们可以通过将访问类成员变量的代码放到专门的方法中，将其声明为 synchronized ，并在主方法中调用来解决这一问题，但是 Java 为我们提供了更好的解决办法，那就是 synchronized 块。

synchronized 块：通过 synchronized关键字来声明synchronized 块。语法如下：

synchronized(syncObject) {

//允许访问控制的代码

}

synchronized 块是这样一个代码块，其中的代码必须获得对象 syncObject （如前所述，可以是类实例或类）的锁方能执行，具体机制同前所述。由于可以针对任意代码块，且可任意指定上锁的对象，故灵活性较高。

# synchronized同步

**synchronized关键字既可以修饰方法，也可以修饰代码块。不管哪种，synchronized都需要有对应的监视对象即锁对象。**

**方法分为静态方法和实例方法。**

**当synchronized直接修饰方法式，对应的锁对象是默认的，修饰静态方法其锁对象为该类的class对象，对该类的所有的实例都是可以获取的，等价于synchronized(Xxx.class)；修饰实例方法时，其锁对象为this，即当前实例方法的对象,等价于synchronized(this)。**

**除了Xxx.class和this之外，synchronized的锁对象可以是任意的，一般为了简单起见，设置一个1字节的字节数组。**

**private static byte[] lock = new byte[1];**

**synchronized(lock){}**

# synchronized(this)与synchronized(\*.class)的区别

Java语法规定，任何线程执行同步方法、同步代码块之前，必须先获取**对应的监视器**。并且监听器this 和 \*\*.class 是不同的；this 是对用方法的对象本身；class 是该类本身(只有监听器相同锁才会起作用)。

这样理解：利用this作为监视器，this指代类的当前实例，一般类会有多个实例，如果多个线程拥有不同的实例，则允许同时执行同步代码块或同步方法，但是内部的共享也仅限于当前实例的变量；但是对于\*.class来说，类是唯一的，\*.class也是唯一的，所以无论该类实例多少，在一个时刻只能有一个线程执行该同步代码块，因为类的静态变量时所有实例共享的。

对于同步方法来说，实例方法的监视器就是this，静态方法的监视器就是\*.class。

对于synchronized(xxx){}的监视器可以任意选择，这里选择this还是\*.class需要考虑下。