synchronized 回顾

作者：ChenforCode
原文链接：https://ld246.com/article/1607413249333
来源网站：链滴

Synchronized

- Synchronized的作用：保证在同一时刻只有一个线程执行该段代码，保证并发安全
- 对象锁
  - 方法锁。锁定一个方法，锁对象是this对象
  - 同步代码块锁，锁定一个代码块，锁对象是自己指定的
- 类锁：虽然叫类锁，其实是Class对象的锁
  - 加到static方法上
  - synchronized(xxx.class)
- 几种同步情况，判断是否同步，就看多个线程访问的代码的锁是不是同一把就可以了。
  - 两个线程访问同一个对象的同步方法——由于同步方法的锁对象是一个this，串行访问
  - 两个线程访问两个对象不同的同步方法——对象不同，互不干扰，并行访问
  - 两个线程访问Synchronized修饰的static方法，都是类锁，串行
  - 访问同一个对象的同步方法与非同步方法——非同步就没有锁，并行
  - 访问同一个对象的不同同步方法——都是同一个this，串行
  - 同时访问静态syn和非静态syn——不是同一把锁，并行
  - 一个线程在syn中抛出异常，另外一个进程会立即进入。
- 总结
  - 对象锁，只有访问到同一个对象，才会同步。例如a.synfunc和a.syncunc或者a.synfunc和a.synfnd都会锁住。
    - 类锁，这个类产生的所有对象都公用这一把锁，访问任何一个对象都会同步。例如访问a.synf

nc和b.syncunc或者a.synfunc或者b.synfund都会被锁住

- 想要锁住一个代码，就必须让锁，是唯一的！

```
public class SynchronizeTest {
    //第一种，使用一个互斥对象锁住代码块；它可以这么理解，多个线程想访问互斥代码块，必须拿mutex对象的锁。
    // 然后，后面的所有类型都可以用第一种类型来解释
    Object mutex = new Object();

    public void methodA() {
        synchronized (mutex) {
            //codeAAAAAAAAAAAAAAAAAA
        }
    }

    //第二种，锁住整个非静态method
    public synchronized void methodB() {
        //codeBBBBBBBBBBBBBBBB
    }
    //第二种相当于用this对象来锁住方法
    public void methodBB(){
        synchronized (this){
            //codeBBBBBBBBBBBBBBBB
        }
    }

    //第三种，使用this锁住一个代码块，这个其实和第二种一样
    public void methodC() {
        synchronized (this) {
            //codeCCCCCCCCCCCCCCCCCC
        }
    }

    //第四种，锁住整个静态method
    public static synchronized void methodD() {
        //codeDDDDDDDDDDDDDDDDDD
    }
    //第四种相当于用类对象锁住代码块
    public static void methodDD(){
        synchronized (SynchronizeTest.class){
            //codeDDDDDDDDDDDDDDDDDD
        }
    }
}
```