线程同步

Java 中允许多线程并行访问，同一时间段内多个线程同时完成各自的操作。

多个线程同时操作**同一个共享数据**时，可能会导致数据不准确的问题。

使用线程同步可以解决上述问题。

可以通过 synchronized 关键字修饰方法实现线程同步，每个 Java 对象都有一个内置锁，内置锁会保护使用 synchronized 关键字修饰的方法，要调用该方法就必须先获得锁，否则就处于阻塞状态。

非线程同步

```
package com.southwind.test;

public class Account implements Runnable {

  private static int num;

  @Override
  public void run() {
    // TODO Auto-generated method stub
    //1.num++操作
    num++;
    //2.休眠1毫秒
    try {
      Thread.currentThread().sleep(1000);
    } catch (InterruptedException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
    //3.打印输出
    System.out.println(Thread.currentThread().getName()+"是当前的第"+num+"位访问");
  }

}
```

线程同步

```
package com.southwind.test;

public class Account implements Runnable {

  private static int num;

  @Override
```

```
  public synchronized void run() {
    // TODO Auto-generated method stub
    //1.num++操作
    num++;
    //2.休眠1毫秒
    try {
      Thread.currentThread().sleep(1000);
    } catch (InterruptedException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
    //3.打印输出
    System.out.println(Thread.currentThread().getName()+"是当前的第"+num+"位访问");
  }

}
```

```
package com.southwind.test;

public class Test {
  public static void main(String[] args) {
    Account account = new Account();
    Thread t1 = new Thread(account,"张三");
    Thread t2 = new Thread(account,"李四");
    t1.start();
    t2.start();
  }
}
```

synchronized 关键字可以修饰实例方法，也可以修饰静态方法，两者在使用的时候是有区别的。

```
package com.southwind.test;

public class SynchronizedTest {

  public static void main(String[] args) {
    for(int i = 0; i < 5;i++) {
      Thread thread = new Thread(new Runnable() {

        @Override
        public void run() {
          // TODO Auto-generated method stub
          SynchronizedTest.test();
        }
      });
      thread.start();
    }
```

```
  }

  /**
   * 先输出start...
   * 间隔1s
   * 再输出end...
   * 输出start...
   * ...
   */
  public synchronized static void test() {
    //1.输出start
    System.out.println("start......");
    //2.休眠
    try {
      Thread.currentThread().sleep(1000);
    } catch (InterruptedException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
    //3.输出end
    System.out.println("end......");
  }

}
```

synchronized 修饰非静态方法

```
package com.southwind.test;

public class SynchronizedTest2 {
  public static void main(String[] args) {
    for(int i=0;i<5;i++) {
      Thread thread = new Thread(new Runnable() {

        @Override
        public void run() {
          // TODO Auto-generated method stub
          SynchronizedTest2 synchronizedTest2 = new SynchronizedTest2();
          synchronizedTest2.test();
        }
      });
      thread.start();
    }
  }

  public synchronized void test() {
    System.out.println("start......");
    try {
      Thread.currentThread().sleep(1000);
```

```
    } catch (InterruptedException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }
    System.out.println("end......");
  }
}
```

给实例方法（非静态方法）添加 synchronized 关键字并不能实现线程同步。

线程同步的本质是锁定多个线程所共享的资源，synchronized 还可以修饰代码块，会为代码块加上内置锁，从而实现同步。

```
package com.southwind.test;

public class SynchronizedTest3 {

  public static void main(String[] args) {
    for(int i=0;i<5;i++) {
      Thread thread = new Thread(new Runnable() {

        @Override
        public void run() {
          // TODO Auto-generated method stub
          SynchronizedTest3.test();
        }
      });
      thread.start();
    }
  }

  public static void test() {

    synchronized (SynchronizedTest3.class) {
      System.out.println("start...");
      try {
        Thread.currentThread().sleep(1000);
      } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
      }
      System.out.println("end...");
    }

  }

}
```

如何判断线程同步或是不同步?

找到关键点：锁定的资源在内存中是一份还是多份？一份大家需要排队，线程同步，多份（一人一份），线程不同步。

无论是锁定方法还是锁定对象，锁定类，只需要分析这个方法、对象、类在内存中有几份即可。

对象一般都是多份

类一定是一份

方法就看是静态方法还是非静态方法，静态方法一定是一份，非静态方法一般是多份