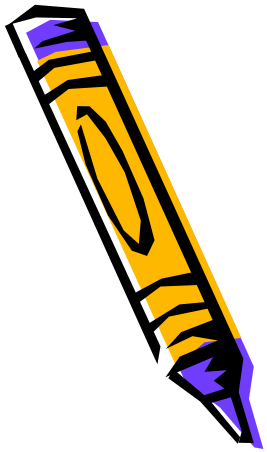
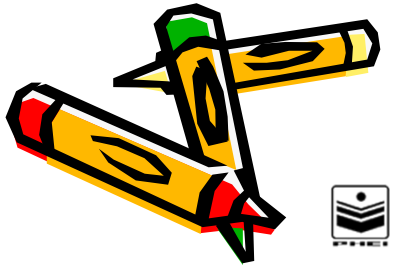
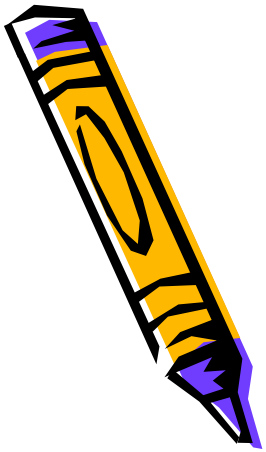


第7章 多线程



第7章 多线程

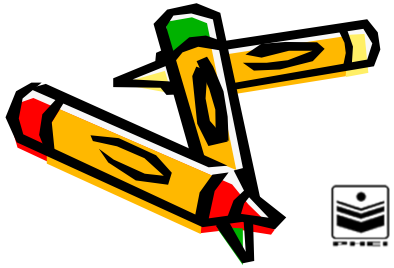


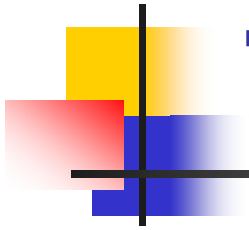
内容和要求：

1. 理解进程与线程概念，掌握创建、管理和控制Java线程对象的方法。
2. 了解并发执行的多线程间存在的各种关系，掌握实现线程互斥和线程同步方法。

重点：创建Java线程对象，改变线程状态，设置线程优先级以控制线程调度。

难点：线程互斥，线程同步。





7.1 操作系统中的进程与线程

- 7.1.1 进程
- 7.1.2 线程
- 7.1.3 并发程序设计

7.1.1 进程

进程的定义和属性

进程（**process**）是一个可并发执行的具有独立功能的程序（**program**）关于某个数据集合的一次执行过程，也是操作系统进行资源分配和保护的基本单位。

① 结构性
② 共享性
③ 动态性
④ 独立性
⑤ 并发性
⑥ 制约性

2. 进程的状态

① 就绪（**ready**）态
② 运行（**running**）态
③ 阻塞（**blocked**）态

7.1.1 进程

进程（**Process**，动态概念）是操作系统对一个正在运行的程序（Program，静态概念）的一种抽象，其是程序的一次动态执行过程，占用特定的地址空间。在一个系统上可以同时运行多个进程，而每个进程都好像在独占地使用硬件。而并发运行，则是说一个进程的指令和另一个进程的指令是交错执行的。操作系统保持跟踪进程运行所需的所有状态信息。在任何一个时刻，单处理器系统都只能执行一个进程的代码。当操作系统决定要把控制权从当前进程转移到某个新进程时，就会进行上下文切换，即保存当前进程的上下文、恢复新进程的上下文，然后将控制权传递到新进程。

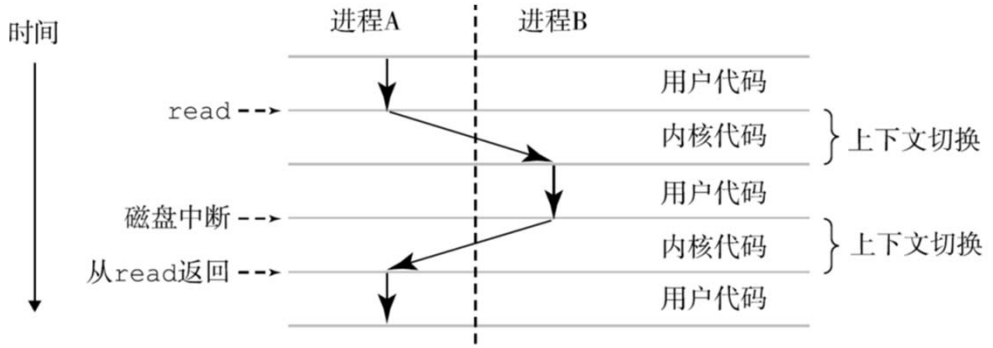


图 1-12　进程的上下文切换

7.1.1 进程

虚拟存储器是一个抽象概念，它为每个进程提供了一个假象，即每个进程都在独占地使用主存。每个进程看到的是一致的存储器，称为虚拟地址空间。图1-13 所示的是Linux进程的虚拟地址空间。在Linux 中，地址空间最上面的区域是为操作系统中的代码和数据保留的，这对所有进程来说都是一样的。地址空间的底部区域存放用户进程定义的代码和数据。

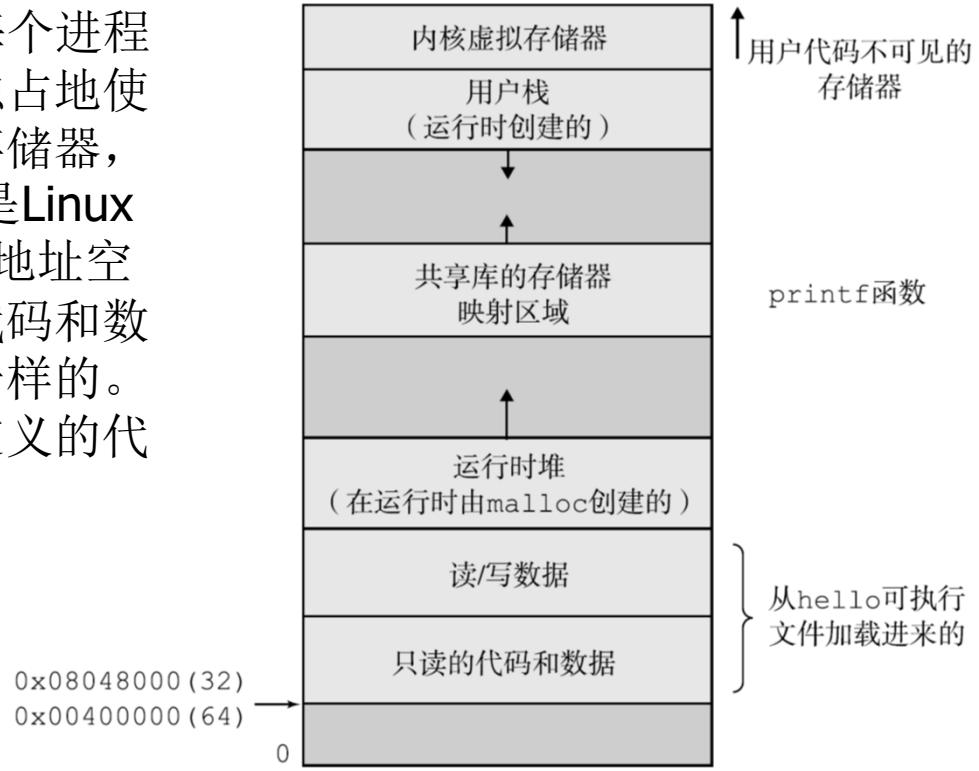


图 1-13　进程的虚拟地址空间

7.1.2 线程

1. 引入线程机制的动机和思路
2. 线程的定义和属性

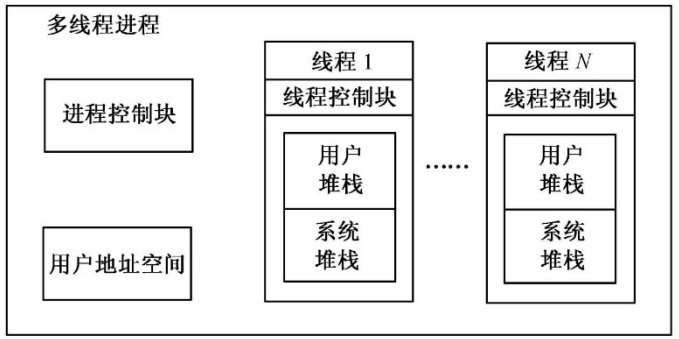
线程（**Thread**）是操作系统进程中能够独立执行的实体（控制流），是一个单一的连续控制流程，是处理器调度和分派的基本单位。

① 线程又被称为轻量级进程；

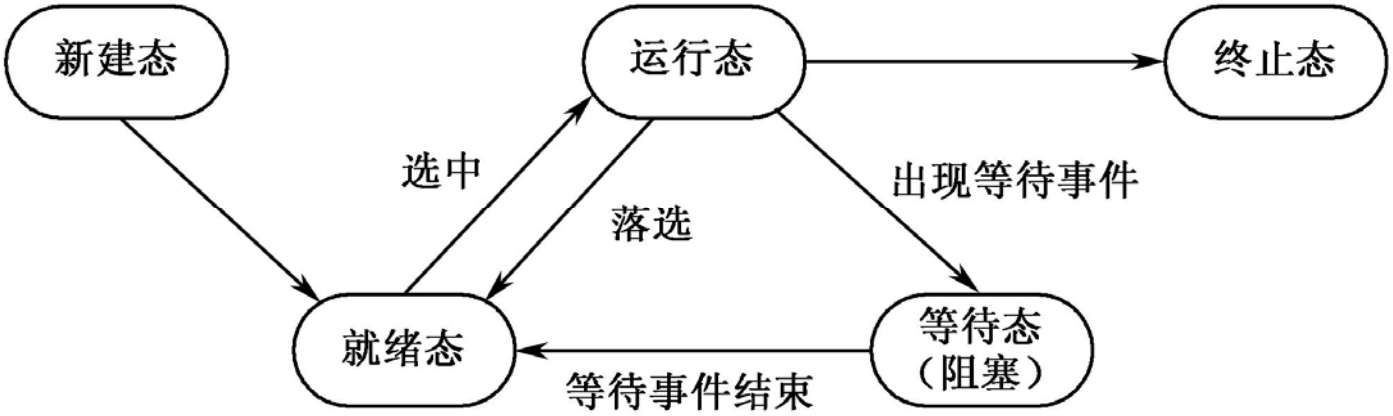
② 一个进程可拥有多个并行的线程；

③ 一个进程中的线程共享相同的内存单元/内存地址空间→可以访问相同的变量和对象，而且它们从同一堆中分配对象→通信、数据交换、同步操作；

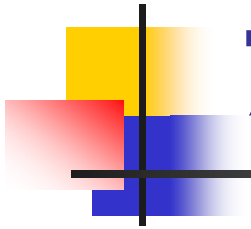
④ 由于线程间的通信是在同一地址空间上进行的，所以不需要额外的通信机制，这就使得通信更简便而且信息传递的速度也更快。



3. 线程的状态



4. 线程的并发性
5. 线程调度

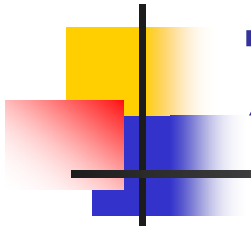


7.1.2 线程

尽管通常我们认为一个进程只有单一的控制流，但是在现代系统中，一个进程实际上可以由多个称为线程的执行单元组成，每个线程都运行在进程的上下文中，并共享同样的代码和全局数据。由于网络服务器对并行处理的需求，线程成为越来越重要的编程模型，因为多线程之间比多进程之间更容易共享数据，也因为线程一般来说都比进程更高效。当有多处理器可用的时候，多线程也是一种使程序可以更快运行的方法。

进程和线程的区别

区别	进程	线程
根本区别	作为资源分配的单位	CPU调度和执行的单位
开销	每个进程都有独立的代码和数据空间（进程上下文），进程间的切换会有较大的开销。	线程可以看成是轻量级的进程，同一类线程共享代码和数据空间，每个线程有独立的运行栈和程序计数器PC，线程切换的开销小。
所处环境	在操作系统中能同时运行多个任务（程序）。	在同一个应用程序中有多个顺序流同时执行。
分配内存	系统在运行的时候会为每个进程分配不同的内存区域	除了CPU之外，不会为线程分配内存（线程所使用的资源都是它们所属的进程的资源），一个进程内的各线程之间共享资源
包含关系	没有线程的进程是可以被看作单线程的，如果一个进程内拥有多个线程，则执行过程不是一条线的，而是多条线（线程）共同完成的。	线程是进程的一部分，所以线程有的时候被称为是轻量级进程。



7.2 Java的线程对象

- 7.2.1 Runnable接口与Thread类
- 7.2.2 线程对象的优先级
- 7.2.3 线程对象的生命周期
- 7.2.4 定时器与图形动画设计

7.2.1 创建线程的两种方式

- 在**Thread**子类覆盖的**run**方法中编写运行代码；
- 在传递给**Thread**对象的**Runnable**对象的**run**方法中编写代码。

总结：查看**Thread**类的**run()**方法的源代码，可以看到其实这两种方式都是在调用**Thread**对象的**run**方法，如果**Thread**类的**run**方法没有被覆盖，并且为该**Thread**对象设置了一个**Runnable**对象，该**run**方法会调用**Runnable**对象的**run**方法。

问题：如果在**Thread**子类覆盖的**run**方法中编写了运行代码，也为**Thread**子类对象传递了一个**Runnable**对象，那么，线程运行时的执行代码是子类的**run**方法的代码？还是**Runnable**对象的**run**方法的代码？

7.2.1 Runnable接口与Thread类

1. **Runnable接口**

```
public interface Runnable
{
    public abstract void run();
}
```

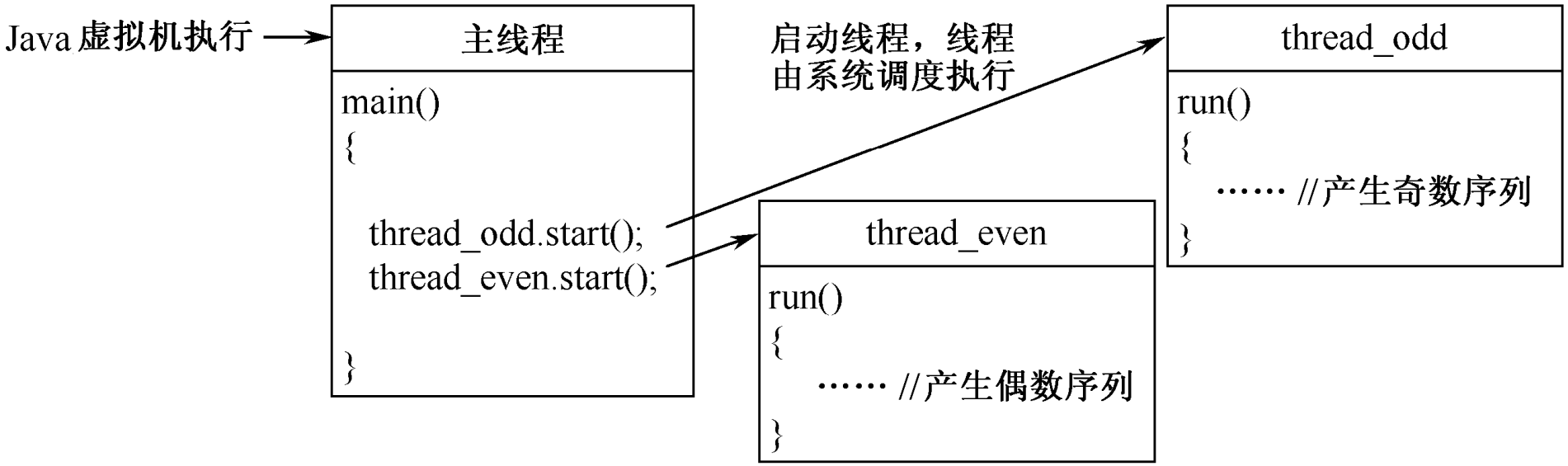
2. Thread线程类

```
public class Thread extends Object implements Runnable
{
    public Thread()                          //构造方法
    public Thread(String name)               //name指定线程名
    public Thread(Runnable target)   //target指定线程的目标对象
    public Thread(Runnable target, String name)

    public void run()                        //描述线程操作的线程体
    public final String getName()            //返回线程名
    public final void setName(String name)  //设置线程名
    public static int activeCount()          //返回当前活动线程个数
    public static Thread currentThread()   //返回当前执行线程对象
    public Sting toString()                  //返回线程的字符串信息
    public void start()                      //启动已创建的线程对象
}
```

【例7.1】 声明继承Thread类的奇数/偶数序列线程。

1. **main**是首先启动执行的线程
2. 两个线程交替运行

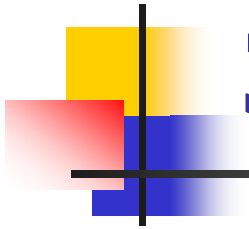


【例7.2】声明实现Runnable接口的奇数/偶数序列线程。

Thread类的**run()**方法声明如下：

```
public void run()              //描述线程操作的线程体
{
   if (target != null)
      target.run();            //执行目标对象的run()方法
}

Thread t1 = new Thread();     //t1的run()方法为空
Thread thread_odd = new Thread(target ,"奇数线程");
    //thread_odd实际执行target的run()方法
```



3. 两种创建线程方式的比较

（1） 继承线程**Thread**类

public class NumberThread extends Thread

（2） 实现**Runnable**接口

public class NumberRunnable implements Runnable

7.2.2 线程对象的优先级

1. **Thread**类中声明了**3**个表示优先级的公有静态常量：

```
public static final int MIN__PRIORITY=1        //最低优先级
public static final int MAX_PRIORITY=10        //最高优先级
public static final int NORM_PRIORITY=5        //默认优先级
```

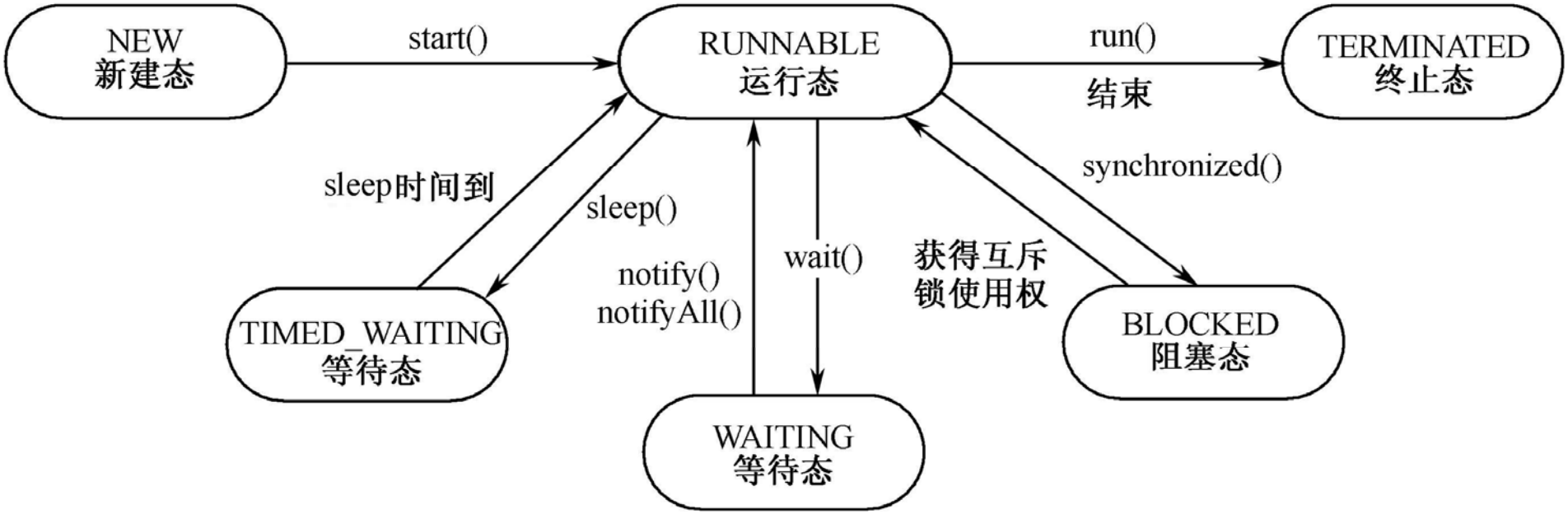
2. **Thread**类中与线程优先级有关的方法有以下**2**个：

```
public final int getPriority()                          //获得线程优先级
public final void setPriority(int newPriority)//设置线程优先级
```

7.2.3 线程对象的生命周期

1. **Thread.State类声明的线程状态**

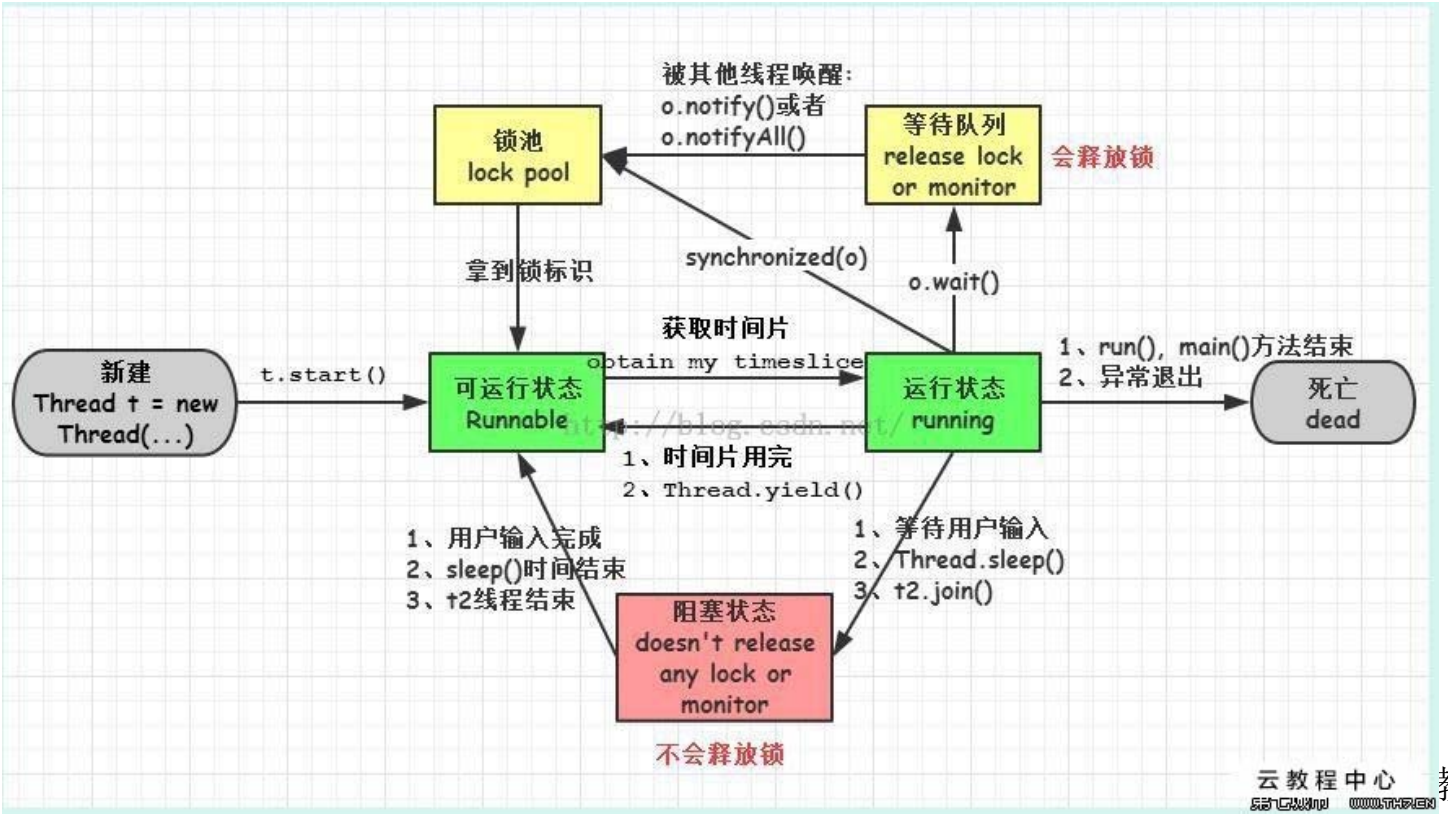
新建态、运行态、阻塞态和等待态、终止态



7.2.3 线程对象的生命周期

1. **Thread.State**类声明的线程状态

新建态、运行态、阻塞态和等待态、终止态



2. Thread类中改变和判断线程状态的方法

1. 线程启动

```
public void start()                    //启动线程对象
public final boolean isAlive()         //是否活动状态
```

2. 线程睡眠

```
public static void sleep(long millis) throws
    InterruptedException
```

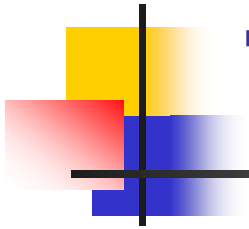
3. 线程中断

```
public void interrupt()                //设置中断标记
public boolean isInterrupted()         //判断是否中断
```

7.2.4 定时器与图形动画设计

```
public class Timer implements Serializable
{
   public Timer(int delay, ActionListener l)
   public void addActionListener(ActionListener l)
                                          //注册定时事件监听器
   public void setDelay(int delay)  //设置延时的时间间隔
   public void start()              //启动定时器
   public void stop()               //停止定时器
   public void restart()            //重新启动定时器
}
```

【例7.4】 弹弹球，使用定时器实现图形动画。

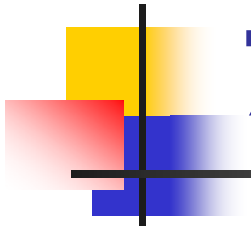


7.2.4 定时器的应用

问题：如何让一个事件2秒后发生，然后4秒后再发生，然后又2秒后发生，如此往复？

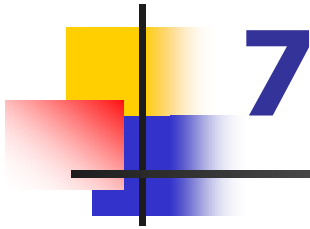
Timer类

TimerTask类



7.3 线程的同步机制

- 7.3.1 交互线程
- 7.3.2 线程间的竞争关系与线程互斥
- 7.3.3 线程间的协作关系与线程同步



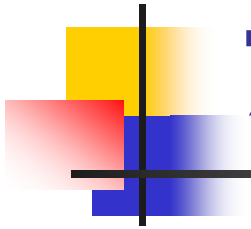
7.3.1 交互线程

1. 无关线程与交互线程

   无关的并发线程是指它们分别在不同的变量集合上操作。

   交互的并发线程是指它们共享某些变量。

2. 并发执行的交互线程间存在与时间有关的错误



7.3.1 交互线程

线程安全问题（代码的原子性：有个线程来执行我的时候，别的线程就不能来执行我）（多个线程在操作共享的数据；操作共享数据的线程代码有多条。当一个线程在执行操作共享数据的多条代码过程中，其它线程参与了运算，就会导致线程安全问题的产生。）

【例7.5】银行账户的存/取款线程设计。

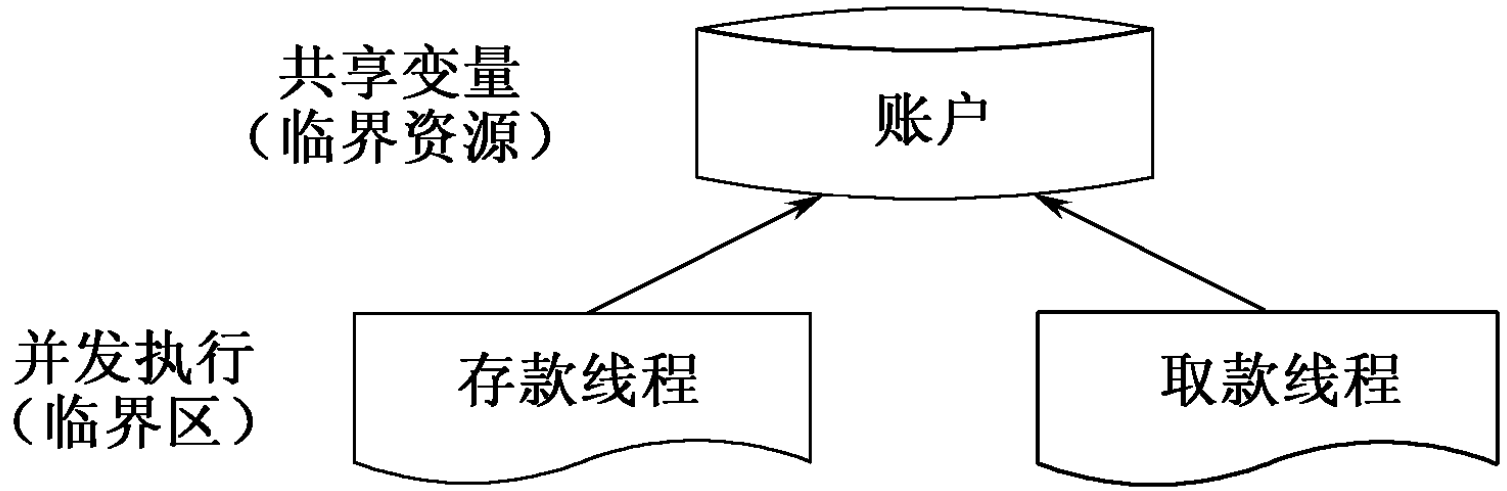


图7.7 并发线程共享临界资源

- 运行结果不惟一，取决于线程调度
- 线程执行被打断时出现错误

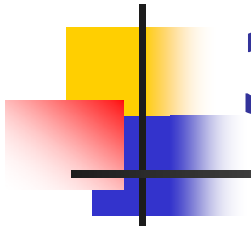
2. 线程互斥和临界区管理

操作系统对共享一个变量的若干线程进入各自临界区有以下3个调度原则：

① 一次至多一个线程能够在它的临界区内。

② 不能让一个线程无限地留在它的临界区内。

③ 不能强迫一个线程无限地等待进入它的临界区。特别地，进入临界区的任一线程不能妨碍正等待进入的其他线程的进展。



3. Java的线程互斥实现

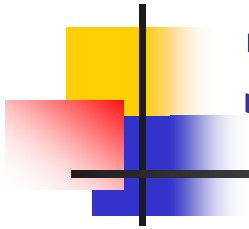
1. 同步语句

   synchronized (对象)

   语句

2. 同步方法

   synchronized 方法声明

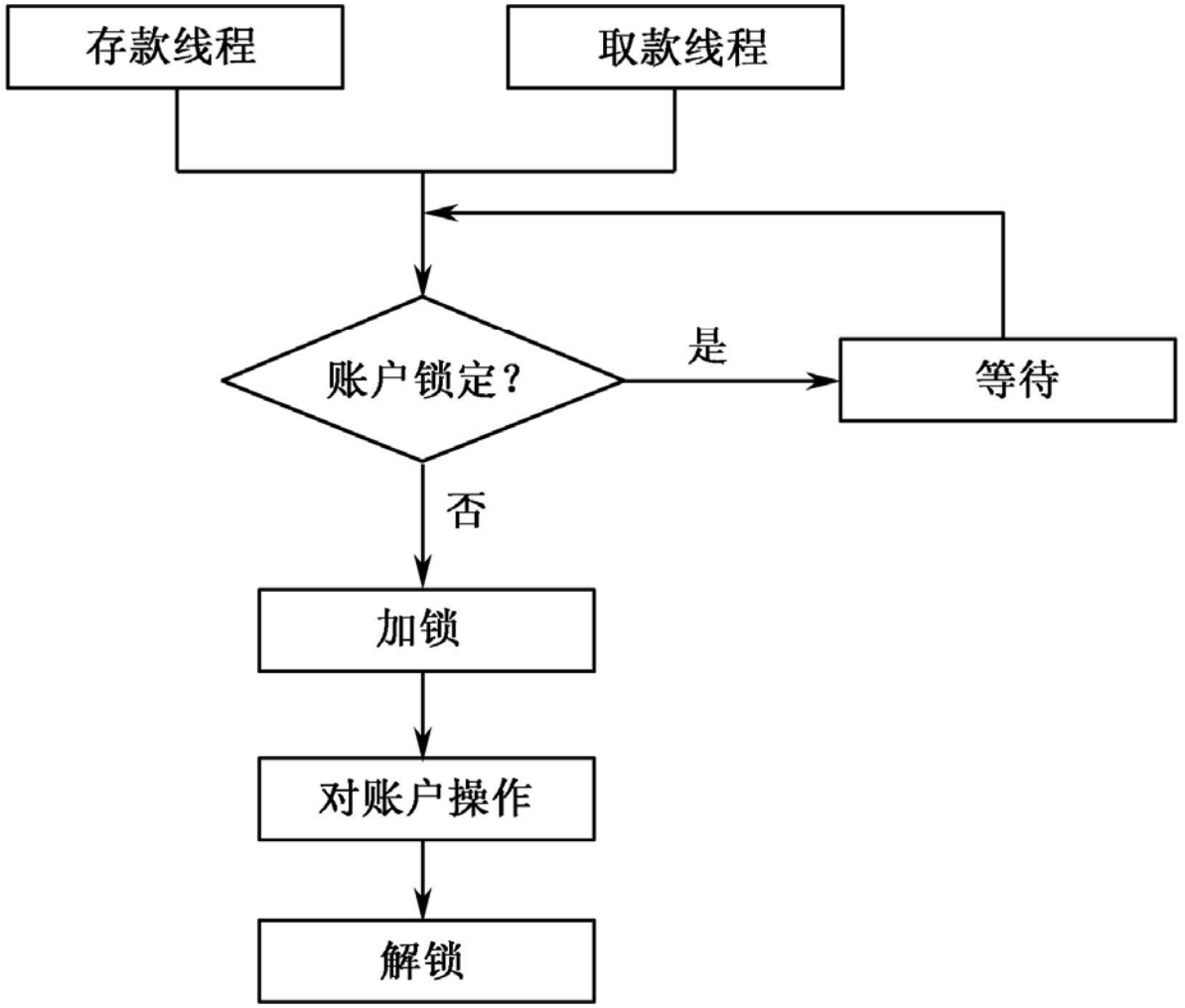


3. Java的线程互斥实现

- 使用**synchronized**代码块及其原理（一段代码或两段代码被两个线程执行时要互斥，则需用**synchronized**代码块包围起来）
- 使用**synchronized**方法
- 分析静态方法所使用的同步监视器对象（锁）是什么？

【例7.6】 互斥的存/取款线程设计。

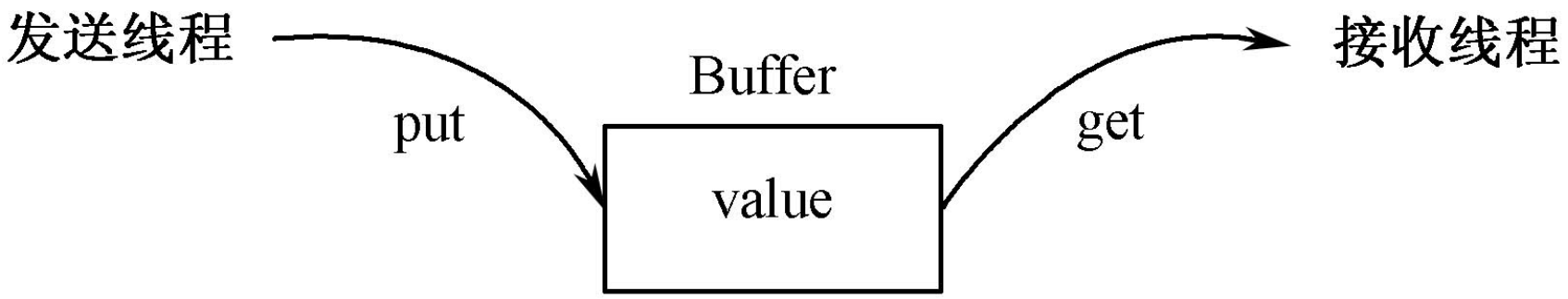
图7.9 带互斥锁的并发线程执行流程



7.3.3 线程间的协作关系与线程同步

1. 线程间的协作关系

【例7.7】 发送线程与接收线程。



2. 线程同步

3. 线程同步机制

① 背景

② 设置信号量

③ 线程根据信号量状态而执行

4. Java的线程通信方法

```
public final void wait() throws
    InterruptedException                    //等待
public final native void notify();
                                    //唤醒一个等待线程
public final native void notifyAll();
                                    //唤醒所有等待线程
```

2. 线程同步

wait与notify实现线程间的通信

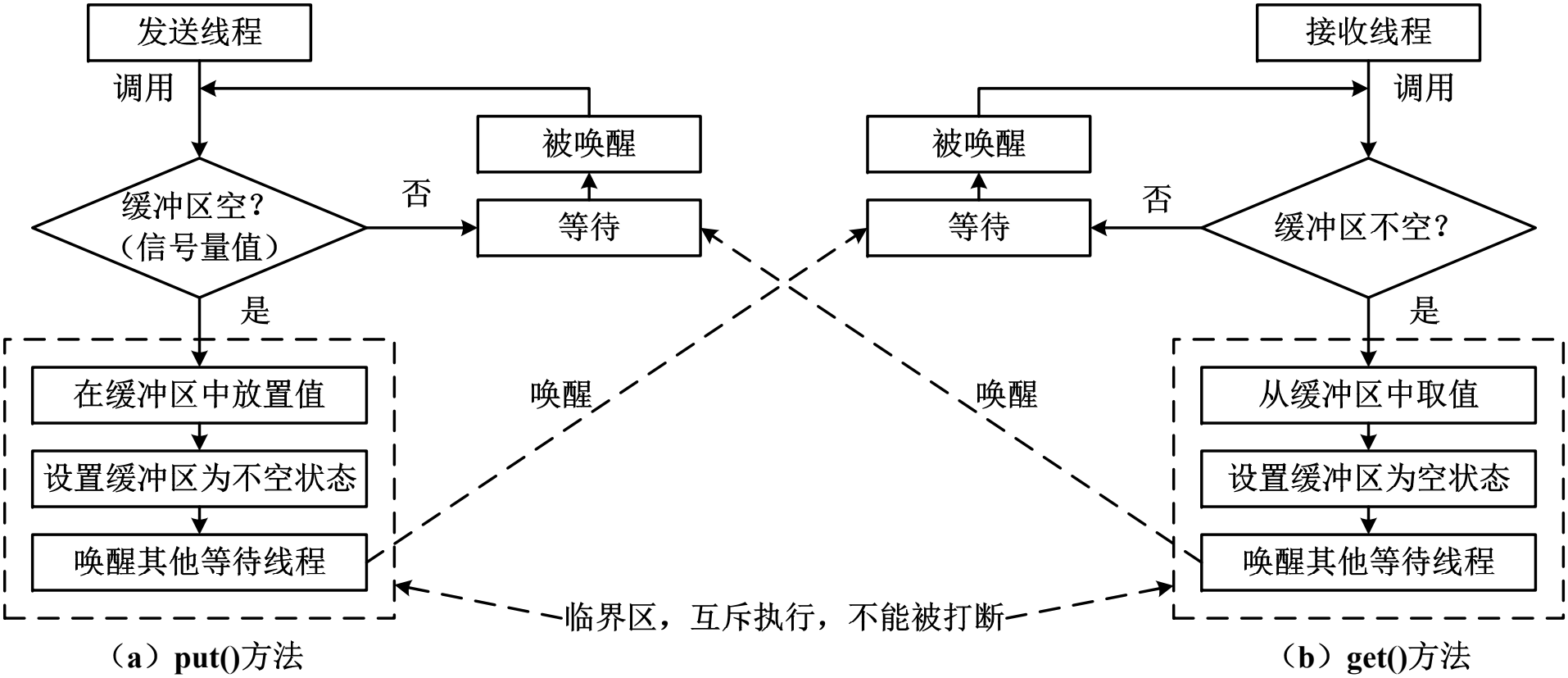
问题：子线程循环**10**次，接着主线程循环**100**次，接着又回到子线程循环**10**次，接着再回到主线程又循环**100**次，如此循环**50**次，请写出程序。

设计提示：要用到共同数据（包括同步锁）或共同算法的若干个方法应该归在同一个类身上，这种设计体现了高内聚和程序的健壮性。

存在虚假唤醒问题：**while()**代替**if()**

【例7.8】采用信号量和同步方法使发送线程与接收线程同步运行。

图7.11 发送线程与接收线程同步执行流程



（a）put()方法　　（b）get()方法

7.4.1 线程范围内共享数据

线程范围内共享变量的概念和作用（**ThreadLocal**）

- **ThreadLocal**的作用与目的：用于实现线程内部的数据共享，即对于相同的程序代码，多个模块在同一个线程中运行时要共享一份数据，而在另外线程中运行时又共享另外一份数据。
- 每个线程调用全局**ThreadLocal**对象的**set**方法，就相当于往其内部的**map**中增加一条记录，**key**分别是各自的线程，**value**是各自的**set**方法传递进去的值。在线程结束时可以调用**ThreadLocal.clear()**方法，这样会更快释放内存，不调用也可，因为线程结束后也可以自动释放相关的**ThreadLocal**变量。
- **ThreadLocal**的应用场景：如**Struct2**的**ActionContext**，同一段代码被不同的线程调用运行时，该代码操作的数据是每个线程各自的状态和数据，对于不同的线程来说，**getContext**方法拿到的对象都不相同，对同一个线程来说，不管调用**getContext**方法多少次和在哪个模块中**getContext**方法，拿到的都是同一个。
- 实现对**ThreadLocal**变量的封装，让外界不要直接操作**ThreadLocal**变量。对于基本类型的数据的封装，这种应用很少见；对于对象类型的数据封装，较常见，即让某个类针对不同线程分别创建一个独立的实例对象。

总结：一个**ThreadLocal**代表一个变量，故其中只能放一个数据，有两个变量都要线程范围内共享，则要定义两个**ThreadLocal**对象。可以先定义一个对象来装多个要线程内共享的变量，然后在**ThreadLocal**中存储这个对象。

7.4.2多个线程访问共享对象和数据的方式

问题：设计4个线程，其中两个线程每次对j增加1，另外两个线程对j每次减少1。写出程序。

A．如果每个线程执行的代码相同，可以使用同一个Runnable对象，这个Runnable对象中有那个共享数据；

B．如果每个线程执行的代码不同，这时候需要用不同的Runnable对象，有如下两种方式来实现这些Runnable对象之间的数据共享：

- 将共享数据封装在另外一个对象中，然后将这个对象逐一传递给各个Runnable对象。每个线程对共享数据的操作方法也分配到那个对象身上完成，这样容易实现针对该数据进行的各个操作的互斥与通信。
- 将这些Runnable对象作为某一个类中的内部类，共享数据作为这个外部类中的成员变量，每个线程对共享数据的操作方法也分配给外部类，以便实现对共享数据进行的各个操作的互斥与通信，作为内部类的各个Runnable对象调用外部类的这些方法。
- 上面两种方式的组合：将共享数据封装在另外一个对象中，每个线程对共享数据的操作方法也分配到那个对象身上去完成，对象作为这个外部类中的成员变量或方法中的局部变量，每个线程的Runnable对象作为外部类中的成员内部类或局部内部类。
- 总之，要同步互斥的几段代码最好是分别放在几个独立的方法中，这些方法再放到同一个类中，这样比较容易实现它们之间的同步互斥和通信。

实验7 线程设计

- 目的：使用线程设计并发执行程序。
- 要求：
  ① 掌握创建、管理和控制**Java**线程对象的方法；
  ② 掌握实现线程互斥和线程同步的方法。
- 重点：创建**Java**线程对象，改变线程状态，设置线程优先级以控制线程调度。
- 难点：线程互斥，线程同步。