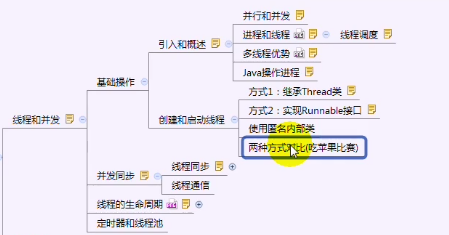
Day 14 线程



**01并行和并发**

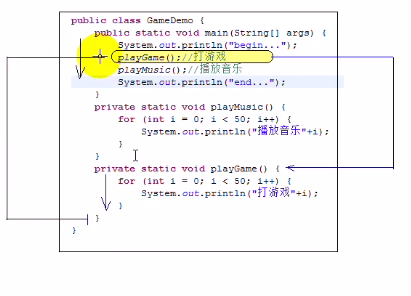
单线程：



串行执行

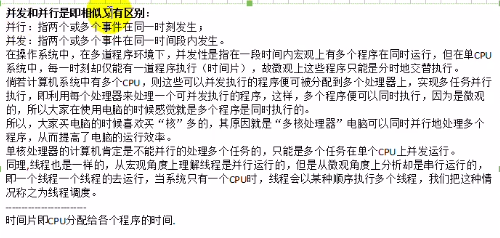
需求：边运行游戏变听歌？

顺序结构：



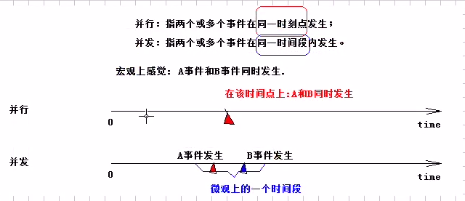
按照流程顺序执行

要解决上述问题需要使用多进程（多个应用程序同时进行，运行的程序之间几乎不会影响）或者多线程（多核cpu可以实现类似进程的操作，单核cpu可以模拟类似同时执行两个程序，实际不是，单核CPU实际是并发）：



并行比并发更加微观，并行指的是真正的同时，并发指的是一个微观的时间段。

并发涉及到线程调度



在单核cpu中在一个时刻只能有一个线程在计算，因此不能实现并行只能实现并发而是分时运行；而在多核cpu上是有可能实现并行的。

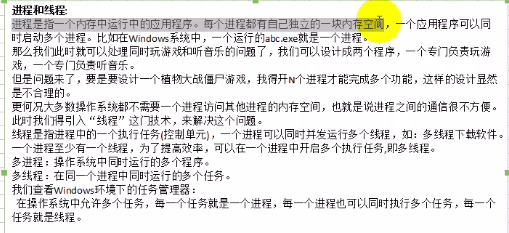


单核cpu在多个进程之间切换

单核只能有并发的概念，为分时运行，多核才有并行的概念

并行强调时间点，并发强调时间段

**02理解进程和线程**



进程的数据是私有的，都有自己的独立的内存空间，从而进程间的通信就需要特殊的处理；

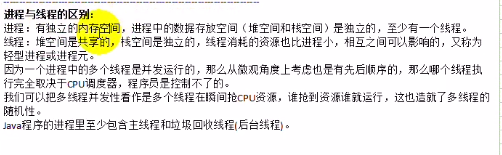
而同一个进程的多个线程之间共享进程的资源，指的是内存堆中的资源。

开启进程消耗资源非常高，进程之间的通信的实现很麻烦。

线程就是进程中的一个执行单元，也是进程实际的执行单元

多任务系统指运行多个线程。

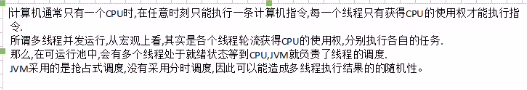
一个进程至少有一个线程，即至少有一个线程



进程在堆中的数据在线程之间是共享的，各个线程的栈中的资源是私有的。

创建线程的资源消耗比进程低。

在pc中进程的调度是由cpu来调度的，在java程序中线程的调度是由jvm来调度的。

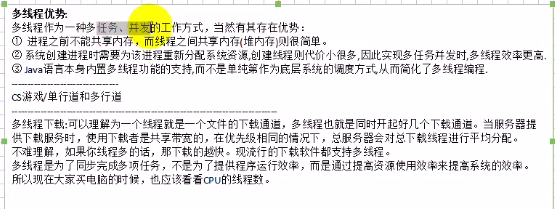


单cpu-某个时刻-单条计算机指令

调度类别：抢占式调度和分时调度（时间分片，如将1秒分为很多份，分别交给不同的进程执行命令）

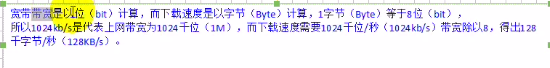
Java程序的进程至少有主线程（main）和垃圾回收线程（后台线程——守护线程）。

**03多线程的优势**

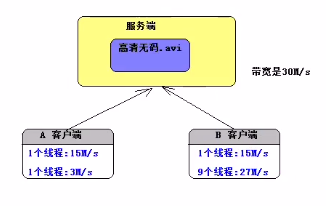


多线程是为了同时完成多项任务，通过提高资源利用率来提高系统的效率，而并非为了程序运行的效率，多线程并不一定会使得程序运行的效率增加。

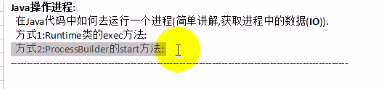
共享内存；创建成本低；在java程序中多线程是由jvm来调度的，而不是由cpu来调度的。



多线程下载（使得某个进程抢占更多的下载资源从而使得该进程下载的速度更快），但是计算机的总带宽是不变的，只是在不同的进程之间分配：



**04创建进程操作**



多线程并不一定提高程序的运行效率，而是提高了资源的使用率。

//方式1：使用runtime的exec方法  
Runtime rt=Runtime.getRuntime();  
//rt.exec("NotePad");//打开记事本  
  
//方式2：使用ProcessBuilder的start方法  
ProcessBuilder pb=new ProcessBuilder("NotePad");  
pb.start();

**05使用 继承方式创建线程并启动**

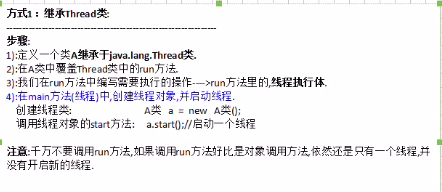


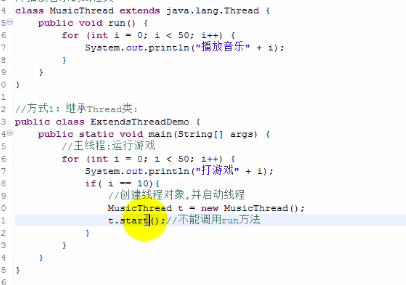
线程类：

Java.lang.thread

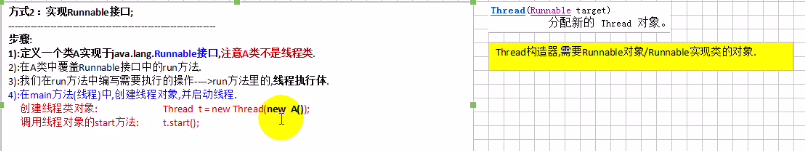
Thread和其子类才能成为线程类

Main是一个主线程，由jvm调用，在main方法中调用的线程是由main生成的线程。





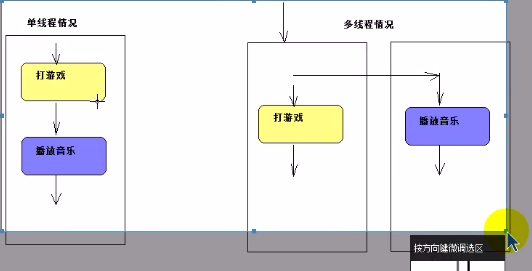
**06使用实现方式创建并启动线程**



启动线程需要Thread对象,只有该类和该类的子类才是线程类，才能开启线程。



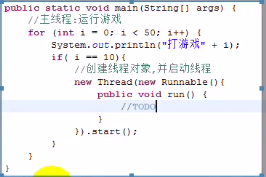
子线程创建后具有和主线程同样的级别。即使主线程执行完毕，也要等待子线程执行完毕程序才算执行完成。



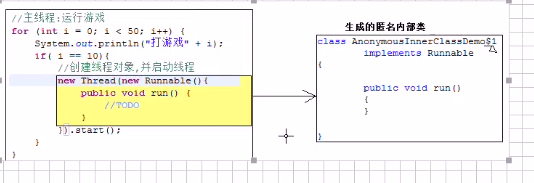
**07使用匿名内部类创建并启用线程**

匿名内部类只适用于某个类只使用一次的情况。

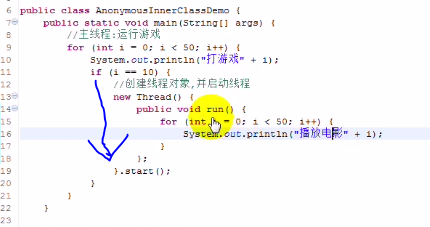
使用接口的形式（较多）



编译完后：

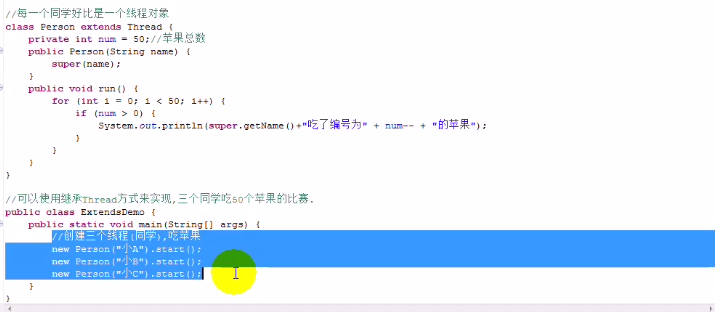


使用类的方式，相当于定义了一个匿名的子类

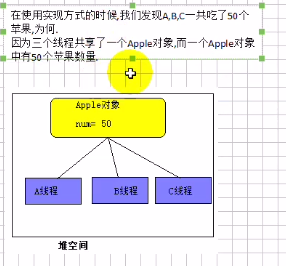


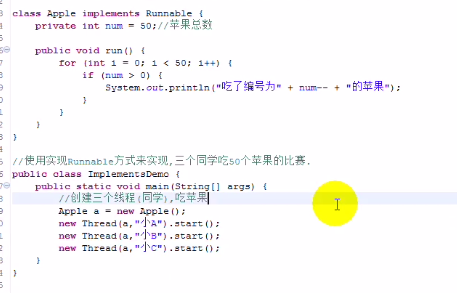
**08案例分析-吃苹果比赛**

**09使用继承方式**

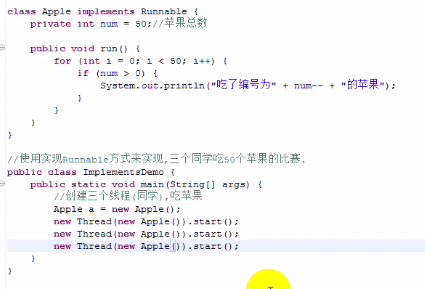


**10使用实现方式**





下面的方式是错误的，三个线程使用三个各自的对象，三个堆空间不共享：



获取当前线程对象的引用，Thread. currentThread ()。

**11分析继承方式和实现方式的区别**

继承方式

1. 单继承
2. 操作上继承方式更简单，获取线程名字也简单（操作上更简单）
3. 从多线程共享同一个非静态资源上分析，继承方式不能做到

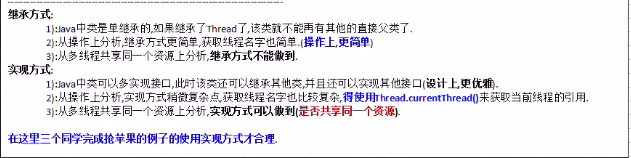
实现方式

1. 多实现

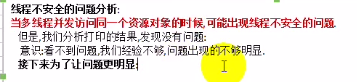
2） 从操作上分析，实现方式稍微复杂点

3) 从多线程共享同一个资源（通过同一个资源创建多个线程对象）上分析，实现方式可以做到

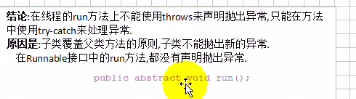
以后做的时候大多以实现的方式来做线程。



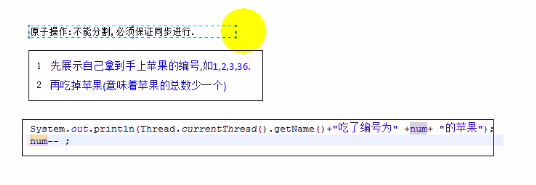
**12线程不安全问题分析**



使用Thread.sleep()，当前线程休息，让其他线程去抢资源（经常用来模拟网络延迟）







要解决上述多线程并发访问多个资源的安全问题，解决方案：保证打印苹果编号和苹果总数减一操作同步完成（某些操作应该做成一个事务），即为一个原子操作

方法一：同步代码块

方法二：同步方法

方法三：锁机制（lock）--悲观锁和乐观锁

**13线程同步-同步代码块**

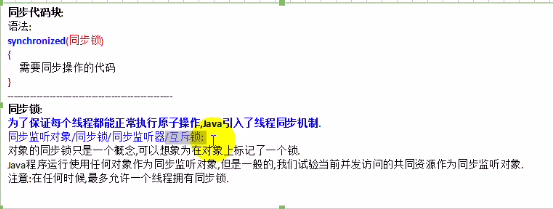
语法：

Synchronized(同步锁)

{

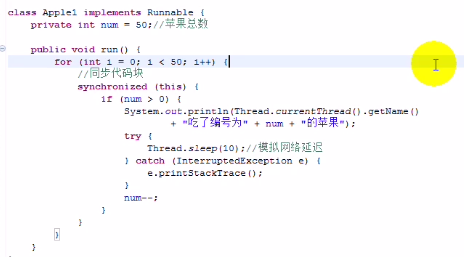
需要同步操作的代码块

}



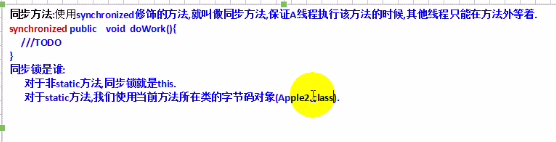
锁可以理解为在对象上标记了一下，如果其他线程看到该锁有标记就等待直到锁释放标记消失，java几乎任何对象都可以作为同步监听对象

谁拿到锁谁就能执行代码块，其他的等待。



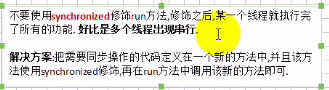
**14线程同步-同步方法**

使用synchronized修饰的方法，就叫做同步方法，保证A线程执行该方法的时候其他线程只能在方法外等着。

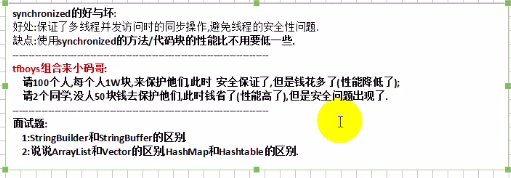


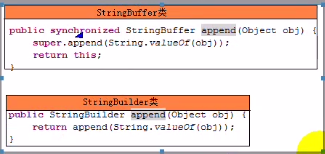
对于非static方法，同步锁就是this

对于static方法，同步锁为当前方法的字节码对象



**15 synchronized的好与坏**



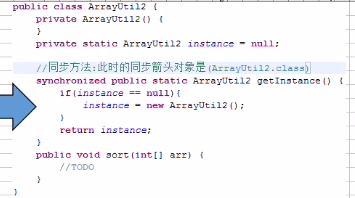


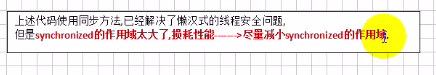
尽量减少synchronized的作用域，它会影响性能

**16单例模式的懒加载**

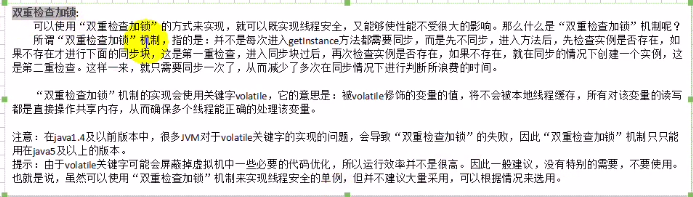


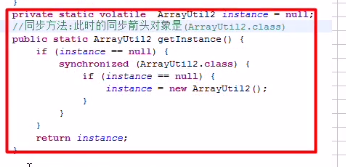
**懒加载存在线程不安全问题**





Double lock check

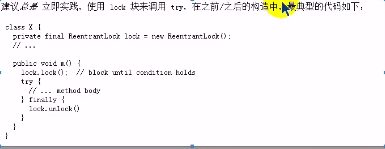


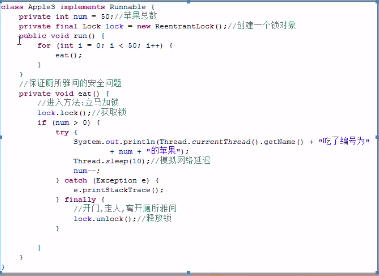


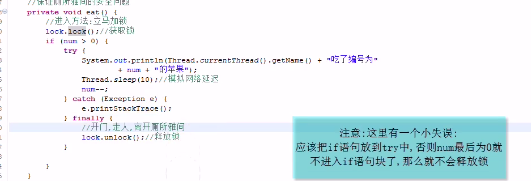
推荐使用饿汉式

懒汉式需要在同步锁上加volatile关键字

17线程同步-同步锁







18今日小结