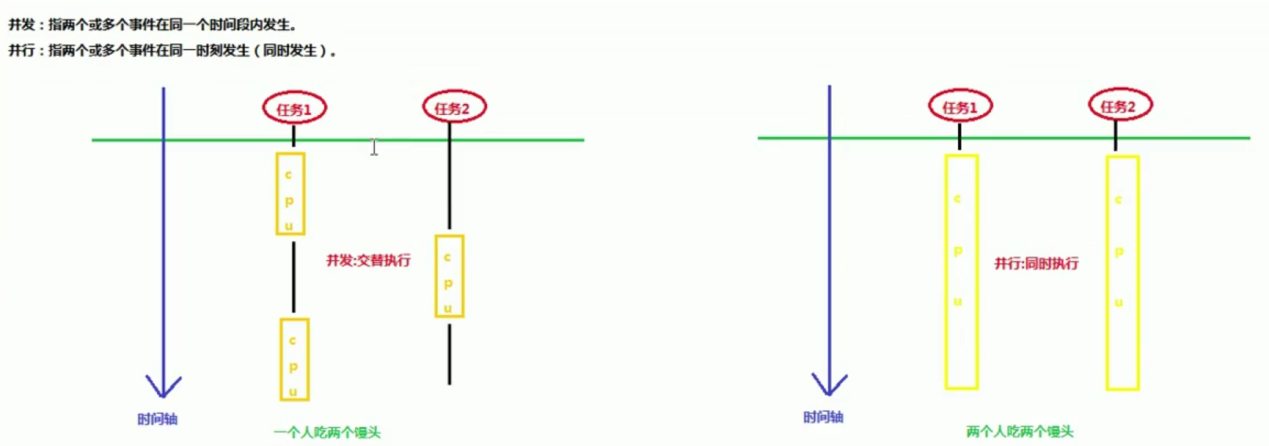
线程实现方式

1.并发与并行

导言：之前我们的程序都是一步一步执行的，要是我们想设计一个程序，可以边听歌，边打游戏，该怎么做呢？

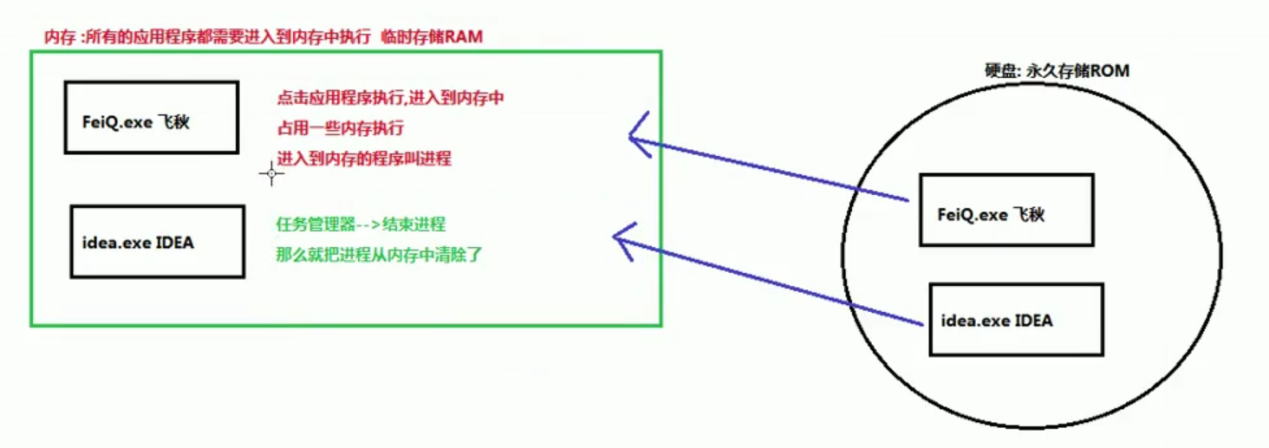
并发：指两个或多个事件在**同一个时间段**内发生。

并行：指两个过多个事件在**同一时刻**发生（同时发生）。



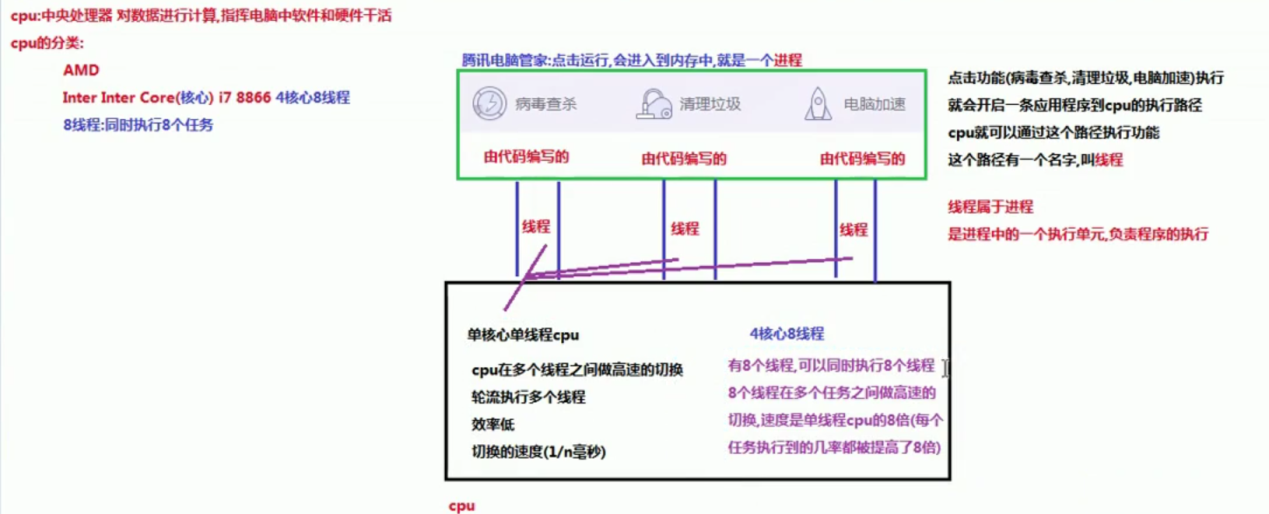
2.线程和进程

进程：是指一个内存中运行的应用程序。（进入到内存中的程序）



线程：是进程中的一个执行单元，负责当前进程中程序的执行，一个进程中至少有一个线程。一个进程中是可以有多个线程的，这个应用程序也可以称之为**多线程程序**。

简而言之：一个程序运行后至少有一个进程，一个进程中可以包含多个线程。



3.线程调度

分时调度：所有线程轮流使用CPU的使用权，平均分配每个线程占用CPU的时间。

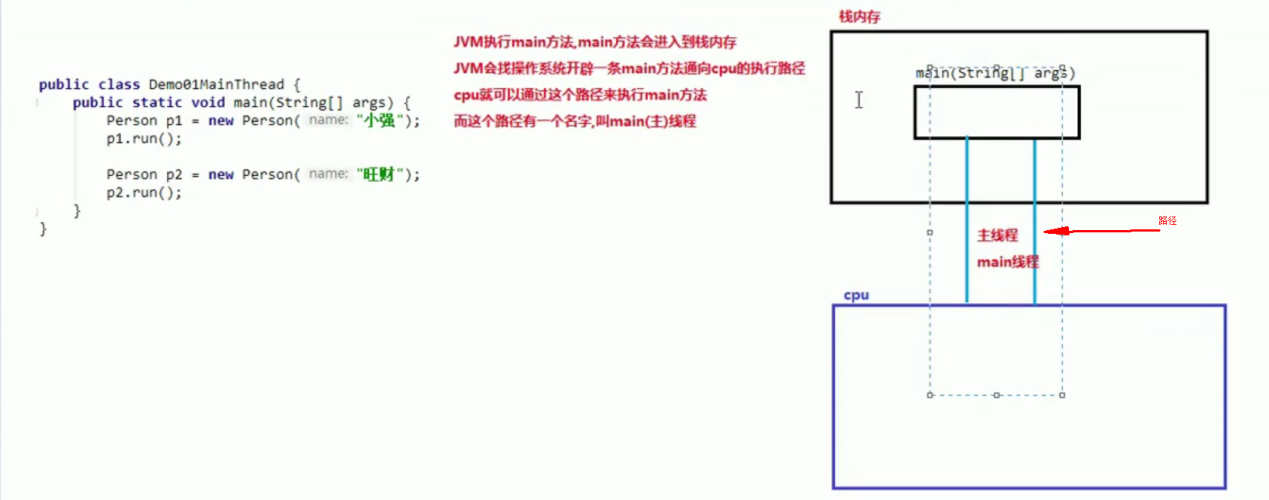
抢占式调度：优先让优先级高的线程使用CPU,如果线程的优先级相同，那么会随机选择一个（线程随机性），java使用的为抢占式调度。

4.主线程

主线程：执行主方法（main()）的线程。

单线程程序：java程序中只有一个线程

执行从main方法开始，从上到下依次执行。



5.创建**多线程**的**Java程序**

**1、第一种方式：创建Thread类的子类**

-java.lang.Thread类：是描述线程的类，所以我们想要实现多线程程序，就必须**继承**Thread类。

实现步骤：

1、创建一个Thread类的子类。

2、**重写**Thread类中的run()方法，**设置线程任务（开启线程要做什么）**

3、创建Thread类的子类对象。

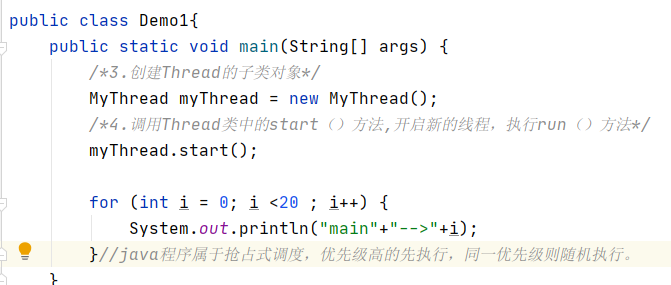
4、调用Thread类中的方法start()方法，开启新的线程，执行run()方法。

-void **start()**：使该线程开始执行，Java虚拟机调用该线程的***run()***方法。

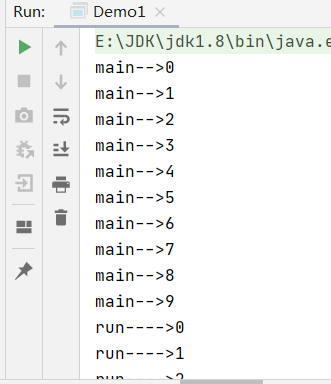
结果是两个线程并发地运行，当前线程（main线程）和另一个线程（创建的新线程，执行其run()方法）。

多次启动一个线程是非法的，特别是当线程已经结束执行后，不能再重新启动。

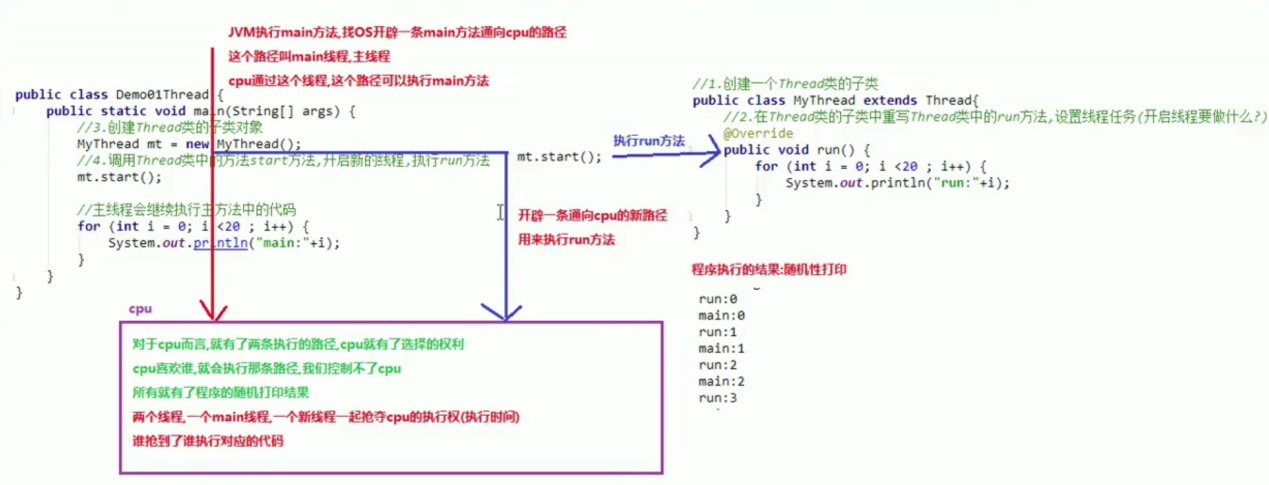
Java程序属于抢占式调度，哪个线程的优先级高，哪个线程就优先执行，同一优先级，则随机选择一个执行。



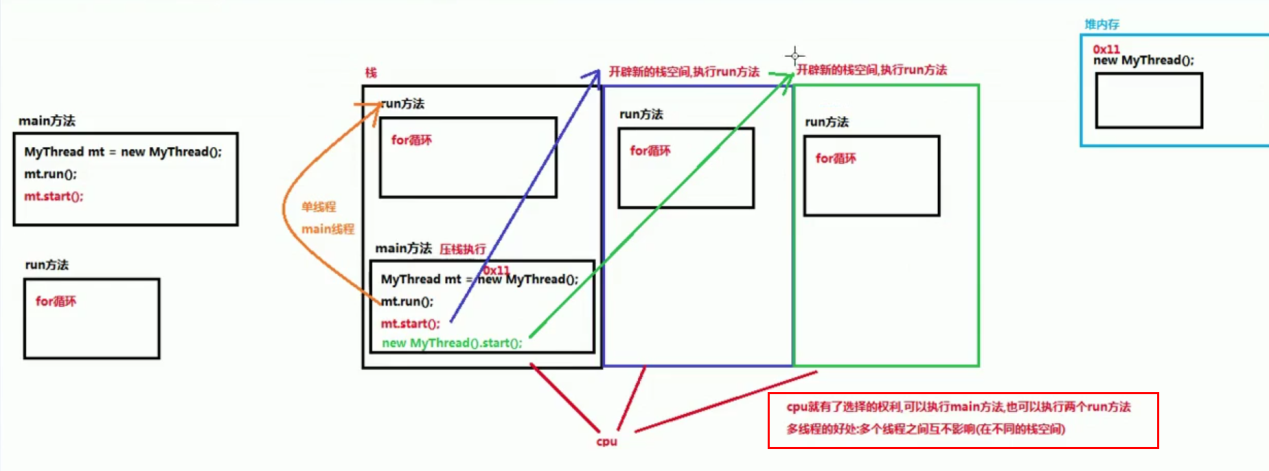
注意：这里如果是myThread.run()和myThread.start()，两者是有区别的。



（以上程序）多线程原理\_随机性打印结果分析图



多线程原理\_多线程内存图解



多线程的好处：在不同的栈空间，程序之间互不影响。

2、Thread类的**常用方法**

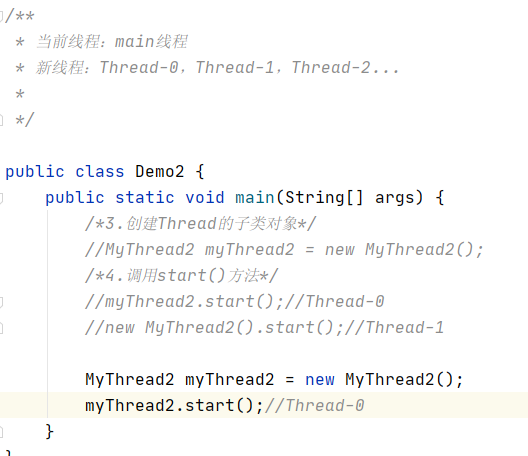
1、获取线程的名称：

1。使用Thread类中的getName()方法

-String getName()返回该线程的名字。

2。也可以先获取当前正在执行的线程，使用线程中的方法getName()。

-static Thread currentThread()返回对当前正在执行的线程对象的引用。



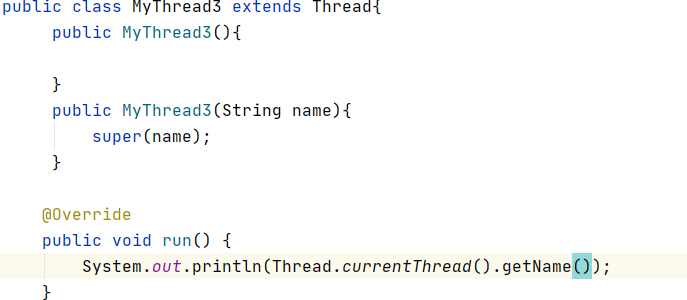
2、设置线程的名称：（了解）

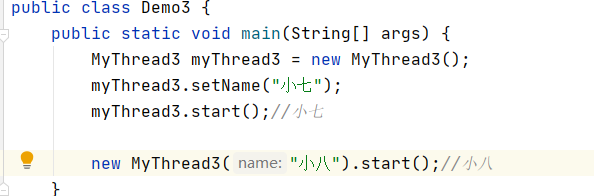
1.使用Thread类中的方法setName(名字);

-void setName(String name)改变线程名称，使之与参数name相同。

2.创建一个带参数的构造方法，参数传递线程的名称，调用父类的的带参构造方法把线程名称传递给父类，让父类（Thread）给子线程起一个名字。

-Thread(String name)分配新的Thread对象。

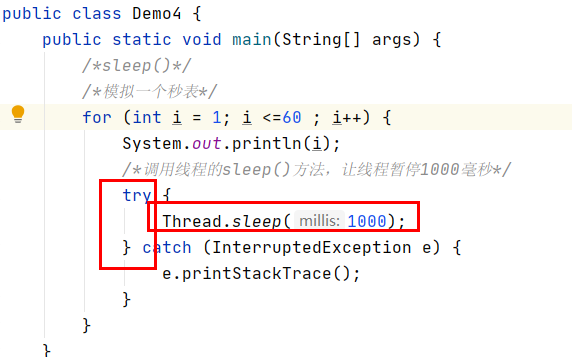




3、sleep

使当前正在执行的线程以指定的毫秒数暂停（**暂时停止执行**）。毫秒数**结束**之后，线程**继续执行**。

-public **static** void sleep(long millis);



**6.创建多线程程序的第二种实现方式**

实现Runnable**接口**

-java.lang.Runnable

Runnable接口应该由那些打算通过某一线程执行其实例的类来实现。类必须定义成为run的无参数方法。

-java.lang.**Thread**类的**构造方法**

Thread(**Runnable** target)：分配新的Thread对象。

Thread(**Runnable** target,String name):分配新的Thread对象。

实现步骤：

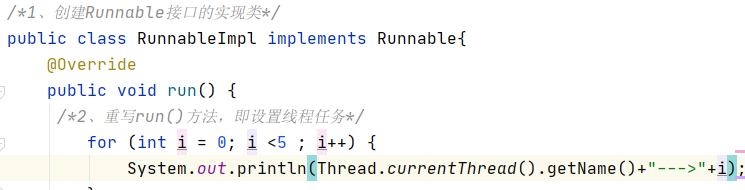
1.创建一个Runnable接口的实现类。

2.在实现类中重写Runnable接口的**run**方法，设置线程任务。

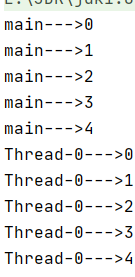
3.创建一个Runnable接口的实现类对象。

4.创建Thread类对象，**构造方法**中传递Runnable接口的实现类对象。

5.调用Thread类中的start方法，开启新的线程执行run方法。







7.Thread和Runnable的区别

即继承Thread类和实现Runnable接口创建多线程的区别。

实现Runnable接口创建多线程程序的**好处**：

1.避免了单继承的局限性。

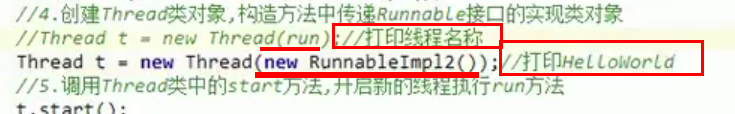
一个类只能继承一个类（一个人只能有一个亲爹），类继承了Thread类就不能继承其他类了。 但是实现了Runnable接口，还可以继承其他的类，实现其他的接口。

2.增强的程序的扩展性，降低了程序的耦合性（**解耦**）

实现Runnable接口的方式，**把设置线程任务和开启新线程进行了分离（解耦）**

**实现类中**，重写了run方法，用来设置线程任务

创建**Thread类对象**，调用start方法，用来开启新线程



不同的实现类，不同的线程任务。

匿名内部类方式实现线程的创建：

匿名：没有名字

内部类：写在其他类内部的类

匿名内部类的作用：简化代码

把子类继承父类，重写父类的方法，创建子类对象合一步完成。

把实现类实现接口，重写接口中的方法，创建实现类对象合一步完成。

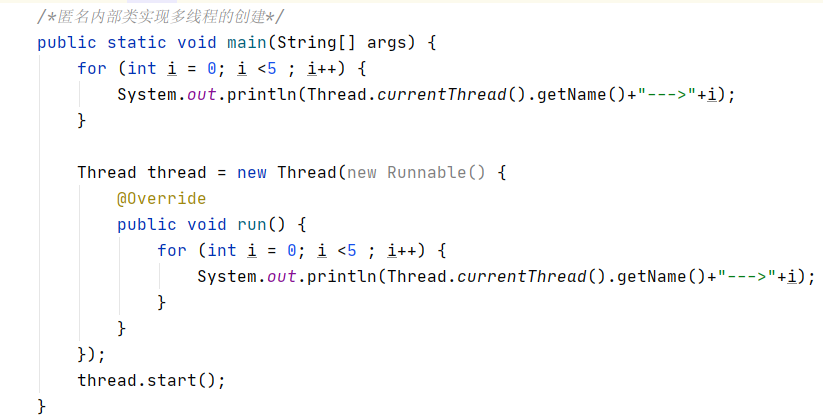
匿名内部类的最终产物：**子类/实现类对象**，而这个类没有名字

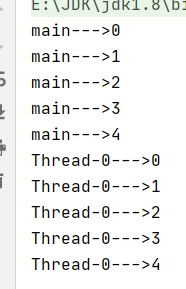
格式：

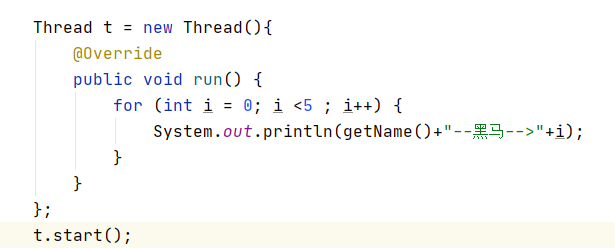
new 父类/接口（）{

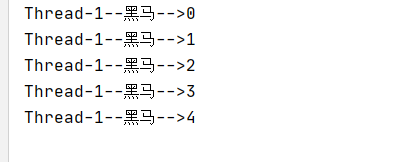
//重写父类/接口中的方法

};





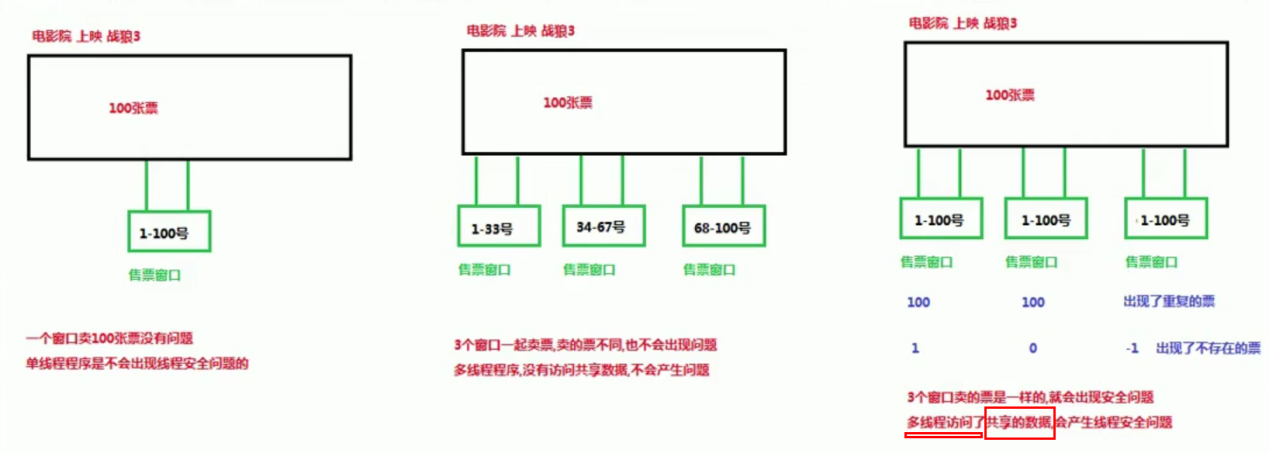




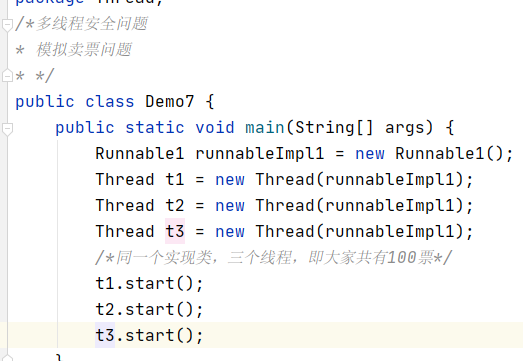
线程同步机制

1.线程安全问题

多线程访问了共享数据，产生线程安全问题。







2.线程安全问题产生的原理



需要注意的是：线程安全问题是不能产生的。

**3.解决线程安全问题的三种方式**

1.同步代码块

2.同步方法

3.锁机制（Lock）

**1、同步代码块**：**synchronized**关键字可以用于**方法中**的**某个区块中**，表示只对这个区块的资源实行互斥访问。

格式：

-synchronized(同步锁（即锁对象）){

需要同步操作的代码

/\*即可能出现线程安全问题的代码（访问了共享数据的代码）\*/

}

注意：

1.同步代码块中的锁对象，可以使用任意的对象

2.但是必须保证多个线程使用的锁对象是同一个

3.锁对象的作用：把同步代码块锁住，只让**一个线程**在**同步代码块**中**执行**



也可以直接传一个this进去，synchronized(this){}

同步技术的原理

同步技术实现的原理：使用了一个锁对象，这个锁对象叫同步锁，也叫对象锁，也叫对象监视器。

**总结：同步中的线程，没有执行完毕不会释放锁，同步外的线程没有锁进不去同步。**



2.1同步方法

使用步骤：

1.把访问了共享数据的代码抽取出来，放到一个方法中。

2.在方法上添加synchronized修饰符

格式: 修饰符 synchronized 返回值类型 方法名（参数列表）{}



**同步方法**的锁对象是谁？

是this。

2.2**静态同步方法**(在同步方法里加static)

静态的同步方法锁对象是谁？

不能是this。

this是创建对象之后产生的，静态方法优先于对象。所以说静态方法的锁对象是本类的class属性---》class文件对象（反射）

3.Lock锁（锁机制）

-java.util.concurrent.Locks.Lock**接口**

Lock实现提供了比使用synchronized方法和语句可获得更广泛的锁定操作。

Lock接口中的方法：

-void lock():获取锁。

-void unlock():释放锁。

-java.util.concurrent.Locks.ReentrantLock implements Lock接口

使用步骤：

1.在成员位置**创建一个ReentrantLock对象**

2.在可能会出现安全问题的代码**前**调用Lock接口中的方法**lock（）**获取锁

3. 在可能会出现安全问题的代码**后**调用Lock接口中的方法**unlock（）**释放锁

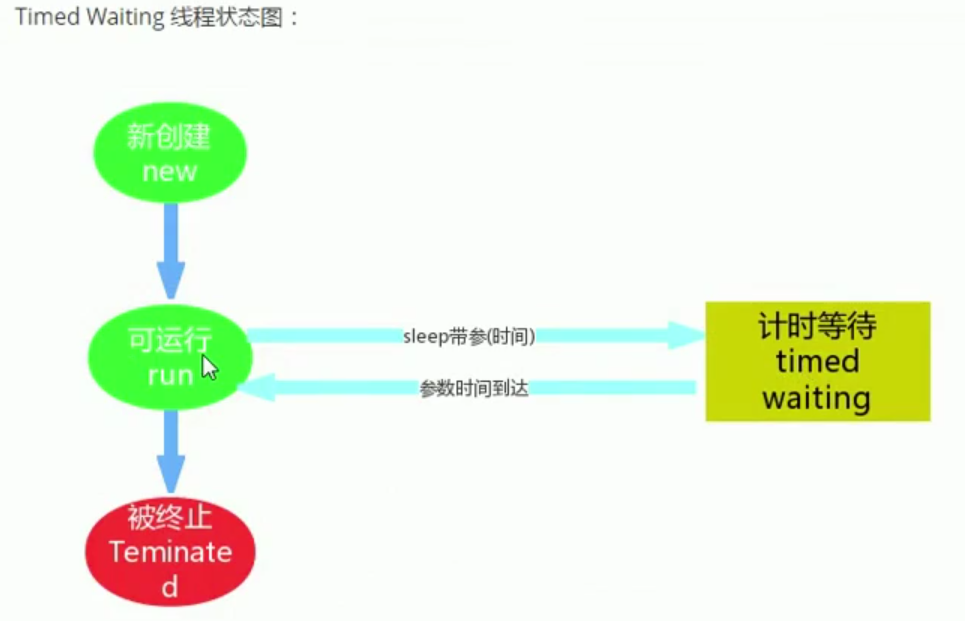


等待唤醒机制

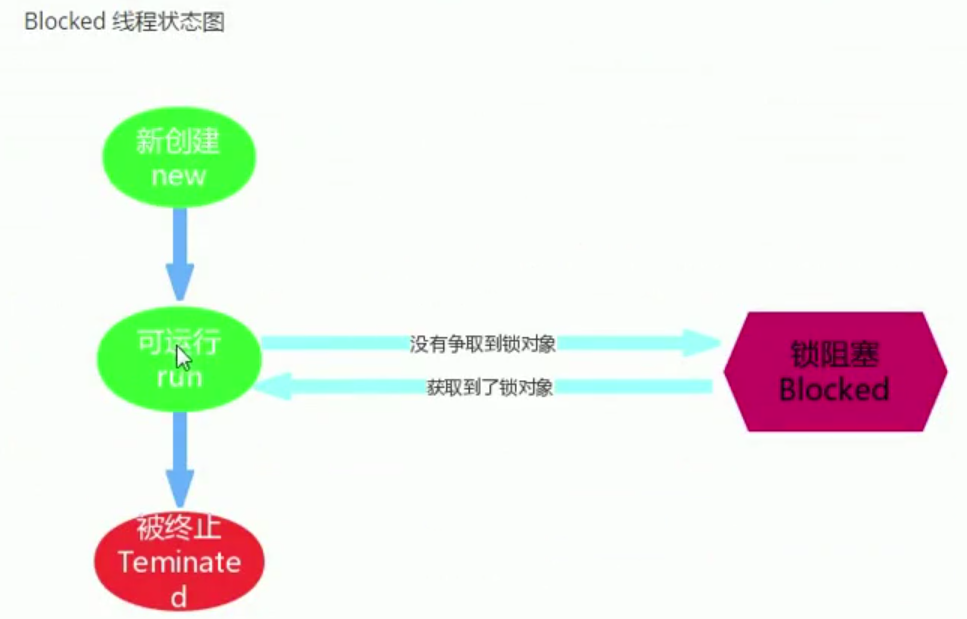
1.线程状态概述



1、计时等待状态



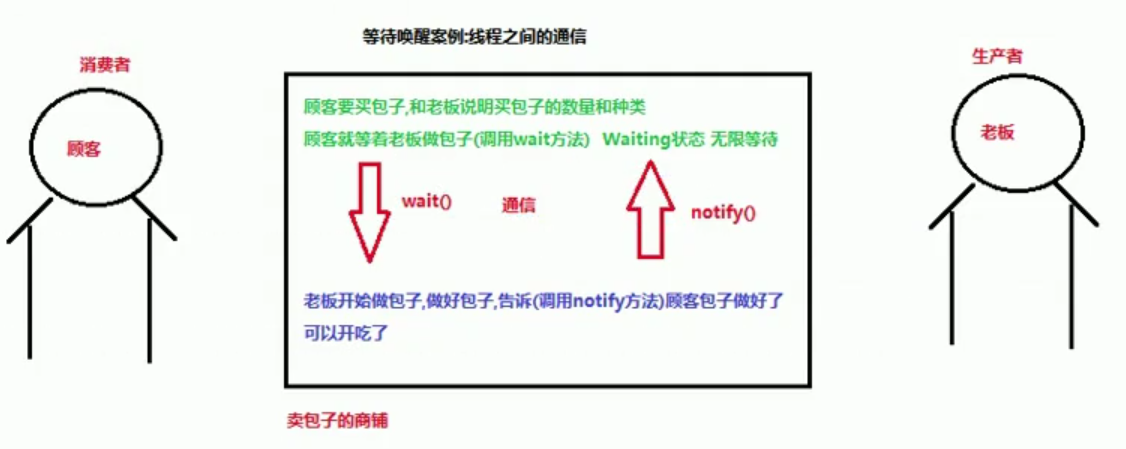
2.锁阻塞状态



3、Waiting（无限等待状态）

一个正在无限期等待另一个线程执行一个特别的（唤醒）动作的线程处于这一状态。

1.等待唤醒案例图（又叫线程之间的通信）



2.等待唤醒案例代码实现

-Object类中的方法

-void wait()

在其他线程调用此对象的notify()方法或者notifyAll()方法前，导致当前线程等待。

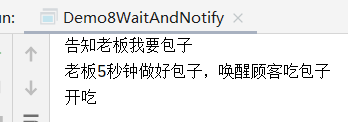
-void notify()

唤醒在此对象监视器上等待的单个线程。会继续执行wait()方法只有的代码。









进入到TimeWaiting（计时等待）有两种方式

1.使用sleep（long m）方法，在毫秒值结束后，线程睡醒进入到Runnable/Blocked状态。

2.使用wait（long m）方法，wait方法如果在毫秒值结束之后，还没有被notify唤醒，就会自动醒来， 线程睡醒进入到Runnable/Blocked状态。

唤醒的方法：

-void notify():唤醒在此对象监视器上等待的单个线程。如果有多个等待线程，随机唤醒一个。

-void notifyAll():唤醒在此对象监视器上等待的所有线程。如果有多个等待线程，全部唤醒。

等待唤醒机制



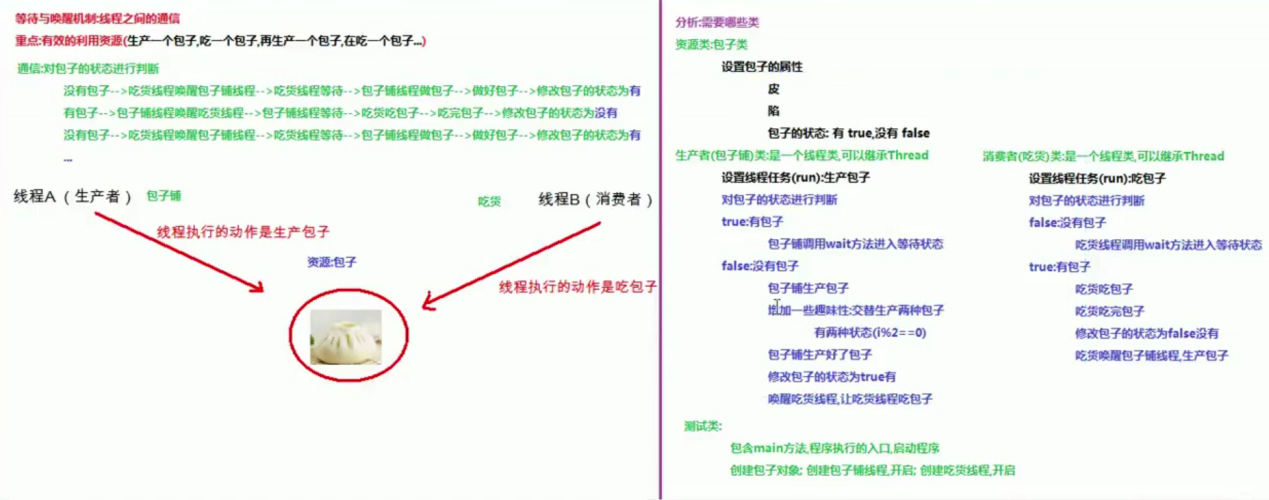
调用wait和notify方法需要注意的细节

1.wait方法和notify方法必须要由同一个锁对象调用。因为：对于的锁对象可以通过notify唤醒使用同一个锁对象调用的wait方法后的线程。

2.wait方法和notify方法是属于Object类的方法。因为：锁对象可以是任意对象，而任意对象的所属类都是继承了Object类的。

3.wait方法和notify方法必须要在同步代码块或者是同步函数中使用。因为：必须要通过锁对象调用这2个方法。

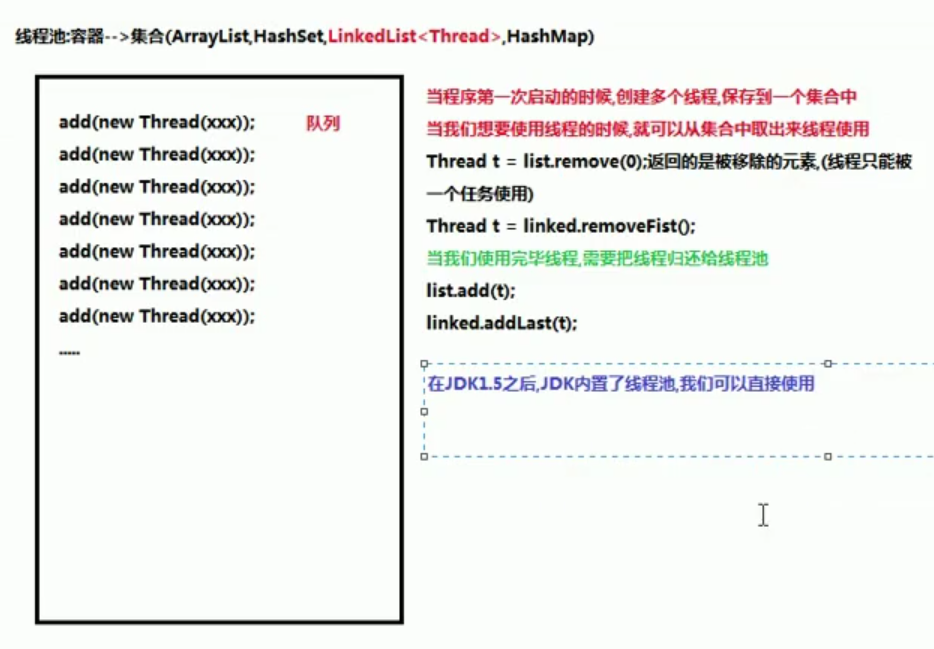
等待唤醒机制需求分析：



代码实现：

线程池

线程池的概念和原理



线程池的代码实现

