**理解进程和线程的概念**

一个运行的程序就是一个进程，线程是进程中的一个小任务，一个进程可以有多个线程同时进行。例如运行的360杀毒软件是一个进程，可以同时开启电脑清理、木马查杀等多个线程；迅雷软件可以并行下载多个文件等。

一个进程中通常都有多个线程在执行任务。

Cpu运行程序的原理

单核cup，只有一个运算核心，某一时刻只能执行一个任务（线程），当多个程序（进程）在电脑上同时运行时，实际是cup在运行的多个程序（进程）的线程下快速的进行切换，给我们的感觉是多个程序在同时运行。

双核cpu，有两个运算核心，某一时刻能同时执行两个任务（线程）

多个进程同时运行时，进程下的线程需要cpu来分别切换执行，此时就存在一个不同的线程占用cpu运算时间的问题：

1. 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，比较公平。
2. 抢占式调度：优先让优先级高的线程使用cpu，如果优先级相同会随机选择一个，java中使用的是抢占式调度（通俗理解为线程也分vip和非vip）

**创建线程的两种方式**

一：继承Thread类方式，重写run方法

1. public 子线程类 extends Thread{

run(){

//任务代码

}

}

2.子线程类 mt = new 子线程类();

3.mt.start()

二．任务类实现Runable接口，实现run方法

1.public task implements Runable{

Run(){

//任务代码

}

}

2.Thread thread = new Thread(new task());

3.Thread.start();

**Ps：代码中一般使用匿名内部类用上面的两种方式快速创建一个线程对象**

*/\*\*  
 \* 匿名内部类：快速创建一个子类对象或一个接口的实现类对象  
 \* 格式：  
 \* new 父类(){  
 \* 重写方法(){}  
 \* }  
 \* new 接口(){  
 \* 实现方法(){}  
 \* }  
 \*  
 \* 使用匿名内部类创建线程对象如下  
 \*/*public class MainThread {  
 public static void main(String[] args) {  
 //匿名内部类 继承方式  
 new Thread(){  
 @Override  
 public void run() {  
 for (int i=0; i<200; i++){  
 System.*out*.println(getName() + "-" + i);  
 }  
 }  
 }.start();  
  
 //匿名内部类 接口实现方式  
 new Thread(new Runnable() {  
 @Override  
 public void run() {  
 for (int i=0; i<200; i++){  
 System.*out*.println(Thread.*currentThread*().getName() + "-" + i);  
 }  
 }  
 }).start();  
 }  
}

两种创建方式的比较：

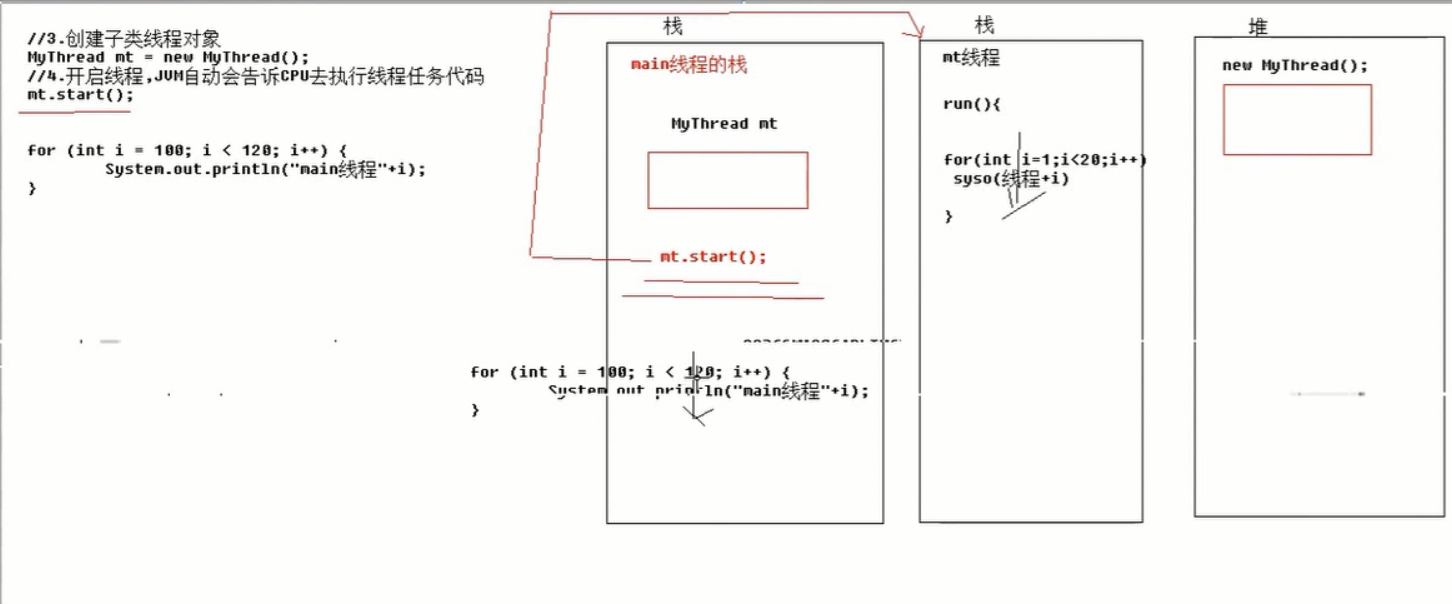
从耦合性分析：第一种方式线程和任务是紧密联系在一起的，耦合性高；第二种方式线 程是线程，任务是任务，他们之间没有必然联系，给一个线程传递哪个任务那么该 线程就执行哪个任务，耦合性低，故常用第二种方式创建。

从可扩展性分析：第一种方式由于是继承Thread，故不能再继承别的类

第二种方式由于是实现接口，故可以再继承其他类，可扩展性好

**其实，多线程程序并不能提高程序的运行速度（反而会降低），但是能提高程序的运行效率，大大提高cpu的使用效率。**

**开启线程后的栈内存图（每一个线程都对应一个栈）**



**线程安全问题：**多个线程同时执行同一段代码操作共享数据时，会存在线程安全问题，解决方法就是给任务代码加锁，同步锁，加锁的方式有**三种**

1. 给任务代码块加锁，同步代码块

synchronized (this){  
 if (count > 0){  
 System.*out*.println(Thread.*currentThread*().getName()+"卖出第" + count + "票");  
 count--;  
 }  
}

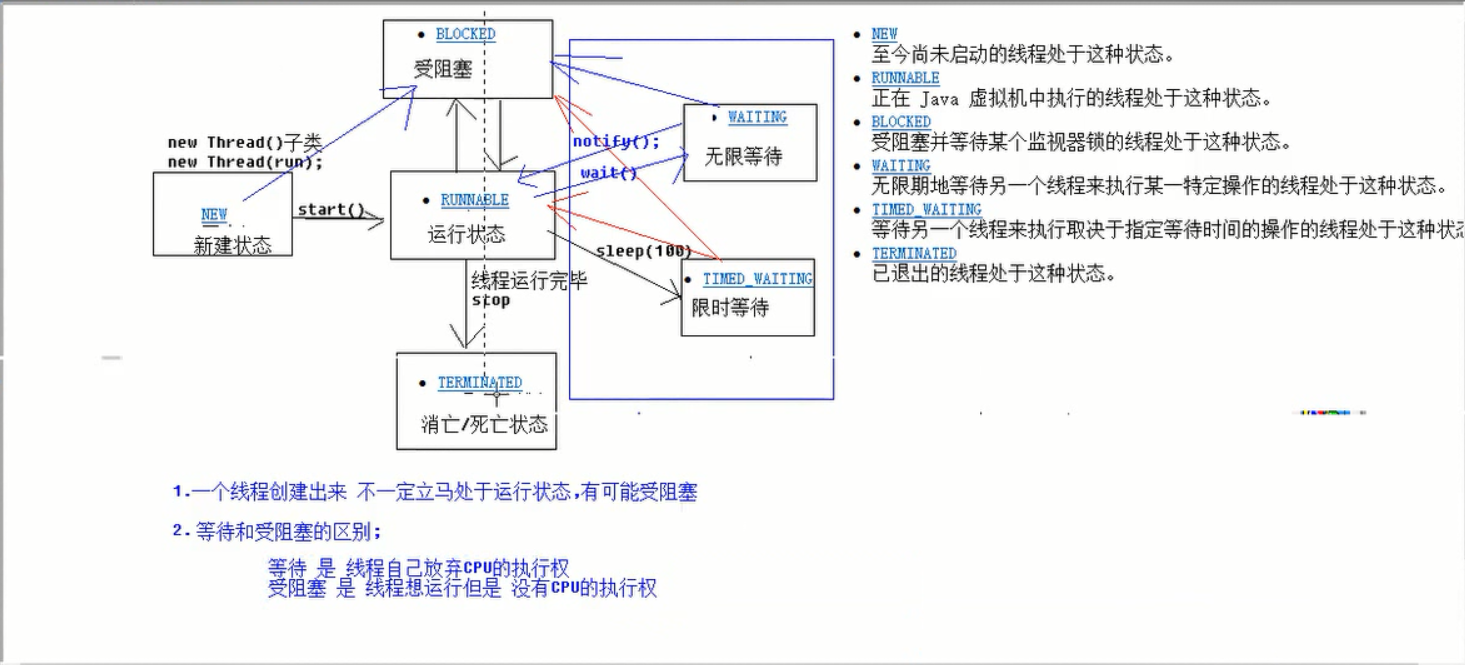
1. 给任务方法加锁，同步方法

public synchronized void sale(){  
 if (count > 0){  
 System.*out*.println(Thread.*currentThread*().getName()+"卖出第" + count + "票");  
 count--;  
 }  
}

1. Lock接口对象加锁

public class SaleTicketRunable3 implements Runnable {  
 //共享数据 100张票 此处三个线程同时执行卖票任务  
 private int count = 100;  
 //创建锁对象  
 private Lock lock = new ReentrantLock();  
  
 @Override  
 public void run() {  
 //执行卖票任务  
 while (true){  
 lock.lock();  
 if (count > 0){  
 System.*out*.println(Thread.*currentThread*().getName()+"卖出第" + count + "票");  
 count--;  
 }  
 lock.unlock();  
 }  
 }  
}

线程状态：6种状态



**线程之间的通信**：等待与唤醒机制，回顾包子铺和吃货实例

**线程池：实现线程的复用**

程序中如果并发的线程数量比较多，并且每个线程执行一个耗时很短的任务就结束了，这样频繁的创建线程会大大降低系统的效率，因为创建和销毁线程都要消耗时间。

线程池的作用就是使得线程可以复用，执行完一个任务后不被销毁，继续去执行下一个任务。

*/\*\*  
 \* jdk1.5之后可以利用线程池对象来拿到线程执行任务  
 \* 线程池对象是由Excetors工厂类创建  
 \*/*public class MainTest {  
 public static void main(String[] args) {  
 //创建有三个线程的线程池对象  
 ExecutorService executorService = Executors.*newFixedThreadPool*(3);  
 executorService.submit(new Runable1());  
 executorService.submit(new Runable2());  
 }  
}