# 线程与进程区别

每个正在系统上运行的程序都是一个进程。每个进程包含一到多个线程。线程是一组指令的集合，或者是程序的特殊段，它可以在程序里独立执行。也可以把它理解为代码运行的上下文。所以线程基本上是轻量级的进程，它负责在单个程序里执行多任务。通常由操作系统负责多个线程的调度和执行。

使用线程可以把占据时间长的程序中的任务放到后台去处理，程序的运行速度可能加快，在一些等待的任务实现上如用户输入、文件读写和网络收发数据等，线程就比较有用了。在这种情况下可以释放一些珍贵的资源如内存占用等等。

如果有大量的线程,会影响性能，因为操作系统需要在它们之间切换，更多的线程需要更多的内存空间，线程的中止需要考虑其对程序运行的影响。通常块模型数据是在多个线程间共享的，需要防止线程死锁情况的发生。

总结:进程是所有线程的集合，每一个线程是进程中的一条执行路径。

# 为什么要使用多线程？

# 多线程应用场景？

答:主要能体现到多线程提高程序效率。

举例: 迅雷多线程下载、数据库连接池、分批发送短信等。

# 多线程创建方式

## 第一种继承Thread类 重写run方法

|  |
| --- |
| 代码: |
| **/\*\***  **\***  **\* @classDesc: 功能描述:(创建多线程例子-Thread类 重写run方法)**  **\* @author: 余胜军**  **\* @version: v1.0**  **\* @copyright:上海每特教育科技有限公司**  **\*/**  **class CreateThread extends Thread {**  **// run方法中编写 多线程需要执行的代码**  **publicvoid run() {**  **for (inti = 0; i< 10; i++) {**  **System.*out*.println("i:" + i);**  **}**  **}**  **}**  **publicclass ThreadDemo {**  **publicstaticvoid main(String[] args) {**  **System.*out*.println("-----多线程创建开始-----");**  **// 1.创建一个线程**  **CreateThread createThread = new CreateThread();**  **// 2.开始执行线程 注意 开启线程不是调用run方法，而是start方法**  **System.*out*.println("-----多线程创建启动-----");**  **createThread.start();**  **System.*out*.println("-----多线程创建结束-----");**  **}**  **}** |

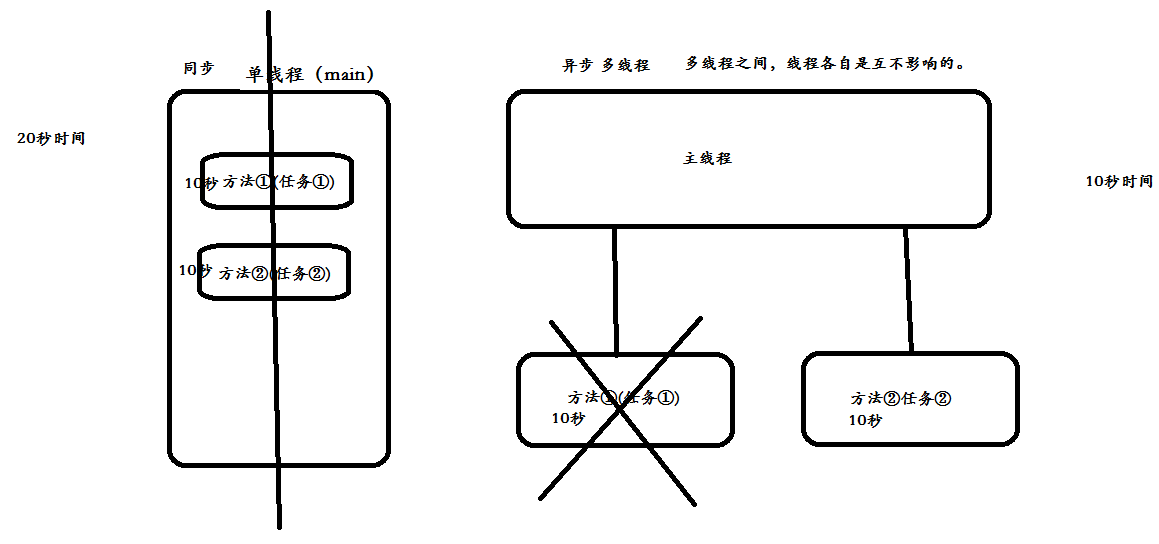
**允许结果:**



**调用start方法后，代码并没有从上往下执行，而是有一条新的执行分之**

**。**

**注意：画图演示多线程不同执行路径。**



## 第二种实现Runnable接口,重写run方法

代码:

|  |
| --- |
| **/\*\***  **\***  **\* @classDesc: 功能描述:(创建多线程例子-Thread类 重写run方法)**  **\* @author: 余胜军**  **\* @version: v1.0**  **\* @copyright:上海每特教育科技有限公司**  **\*/**  **class CreateRunnable implements Runnable {**  **@Override**  **publicvoid run() {**  **for (inti = 0; i< 10; i++) {**  **System.*out*.println("i:" + i);**  **}**  **}**  **}**  **/\*\***  **\***  **\* @classDesc: 功能描述:(实现Runnable接口,重写run方法)**  **\* @author: 余胜军**  **\* @version: v1.0**  **\* @copyright:上海每特教育科技有限公司**  **\*/**  **publicclass ThreadDemo2 {**  **publicstaticvoid main(String[] args) {**  **System.*out*.println("-----多线程创建开始-----");**  **// 1.创建一个线程**  **CreateRunnable createThread = new CreateRunnable();**  **// 2.开始执行线程 注意 开启线程不是调用run方法，而是start方法**  **System.*out*.println("-----多线程创建启动-----");**  **Thread thread = new Thread(createThread);**  **thread.start();**  **System.*out*.println("-----多线程创建结束-----");**  **}**  **}** |

## 第三种使用匿名内部类方式

|  |
| --- |
| **System.*out*.println("-----多线程创建开始-----");**  **Thread thread = new Thread(new Runnable() {**  **public void run() {**  **for (int i = 0; i< 10; i++) {**  **System.*out*.println("i:" + i);**  **}**  **}**  **});**  **thread.start();**  **System.*out*.println("-----多线程创建结束-----");** |

## 使用继承Thread类还是使用实现Runnable接口好？

使用实现实现Runnable接口好，原因实现了接口还可以继续继承，继承了类不能再继承。

## 启动线程是使用调用start方法还是run方法？

开始执行线程 注意 开启线程不是调用run方法，而是start方法

调用run知识使用实例调用方法。

# 获取线程对象以及名称

|  |  |
| --- | --- |
| **常用线程api方法** | |
| start() | 启动线程 |
| currentThread() | 获取当前线程对象 |
| getID() | 获取当前线程ID Thread-编号  该编号从0开始 |
| getName() | 获取当前线程名称 |
| sleep(long mill) | 休眠线程 |
| Stop（） | 停止线程, |
| **常用线程构造函数** | |
| Thread（） | 分配一个新的 Thread 对象 |
| Thread（String name） | 分配一个新的 Thread对象，具有指定的 name正如其名。 |
| Thread（Runable r） | 分配一个新的 Thread对象 |
| Thread（Runable r, String name） | 分配一个新的 Thread对象 |

# 守护线程

Java中有两种线程，一种是用户线程，另一种是守护线程。

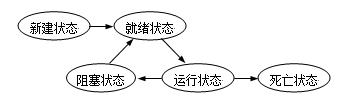
用户线程是指用户自定义创建的线程，主线程停止，用户线程不会停止

守护线程当进程不存在或主线程停止，守护线程也会被停止。

使用setDaemon(true)方法设置为守护线程

|  |
| --- |
| \*\*  \*  \* 什么是守护线程? 守护线程 进程线程(主线程挂了) 守护线程也会被自动销毁.  \*  \* **@classDesc**: 功能描述:(守护线程)  \* **@author**: 余胜军  \* **@createTime**: 2017年8月20日 下午8:55:58  \* **@version**: v1.0  \* **@copyright**:上海每特教育科技有限公司  \*/  **public** **class** DaemonThread {  **public** **static** **void** main(String[] args) {  Thread thread = **new** Thread(**new** Runnable() {  @Override  **public** **void** run() {  **while** (**true**) {  **try** {  Thread.*sleep*(100);  } **catch** (Exception e) {  // **TODO**: handle exception  }  System.***out***.println("我是子线程...");  }  }  });  thread.setDaemon(**true**);  thread.start();  **for** (**int** i = 0; i < 10; i++) {  **try** {  Thread.*sleep*(100);  } **catch** (Exception e) {  }  System.***out***.println("我是主线程");  }  System.***out***.println("主线程执行完毕!");  }  } |

# 多线程运行状态



 线程从创建、运行到结束总是处于下面五个状态之一：新建状态、就绪状态、运行状态、阻塞状态及死亡状态。

## 新建状态

   当用new操作符创建一个线程时， 例如new Thread(r)，线程还没有开始运行，此时线程处在新建状态。 当一个线程处于新生状态时，程序还没有开始运行线程中的代码

## 就绪状态

一个新创建的线程并不自动开始运行，要执行线程，必须调用线程的start()方法。当线程对象调用start()方法即启动了线程，start()方法创建线程运行的系统资源，并调度线程运行run()方法。当start()方法返回后，线程就处于就绪状态。

     处于就绪状态的线程并不一定立即运行run()方法，线程还必须同其他线程竞争CPU时间，只有获得CPU时间才可以运行线程。因为在单CPU的计算机系统中，不可能同时运行多个线程，一个时刻仅有一个线程处于运行状态。因此此时可能有多个线程处于就绪状态。对多个处于就绪状态的线程是由[Java](http://lib.csdn.net/base/java" \t "_blank" \o "Java 知识库)运行时系统的线程调度程序(*thread scheduler*)来调度的。

## 运行状态

当线程获得CPU时间后，它才进入运行状态，真正开始执行run()方法.

## 阻塞状态

    线程运行过程中，可能由于各种原因进入阻塞状态:  
        1>线程通过调用sleep方法进入睡眠状态；  
        2>线程调用一个在I/O上被阻塞的操作，即该操作在输入输出操作完成之前不会返回到它的调用者；  
        3>线程试图得到一个锁，而该锁正被其他线程持有；  
        4>线程在等待某个触发条件；

## 死亡状态

有两个原因会导致线程死亡：  
  1) run方法正常退出而自然死亡，  
   2) 一个未捕获的异常终止了run方法而使线程猝死。  
  为了确定线程在当前是否存活着（就是要么是可运行的，要么是被阻塞了），需要使用isAlive方法。如果是可运行或被阻塞，这个方法返回true； 如果线程仍旧是new状态且不是可运行的， 或者线程死亡了，则返回false.

# join()方法作用

当在主线程当中执行到t1.join()方法时，就认为主线程应该把执行权让给t1

## 需求:

创建一个线程，子线程执行完毕后，主线程才能执行。

|  |
| --- |
| Thread t1 = **new** Thread(**new** Runnable() {  @Override  **public** **void** run() {  **for** (**int** i = 0; i < 10; i++) {  **try** {  Thread.*sleep*(10);  } **catch** (Exception e) {  }  System.***out***.println(Thread.*currentThread*().getName() + "i:" + i);  }  }  });  t1.start();  // 当在主线程当中执行到t1.join()方法时，就认为主线程应该把执行权让给t1  t1.join();  **for** (**int** i = 0; i < 10; i++) {  **try** {  Thread.*sleep*(10);  } **catch** (Exception e) {  }  System.***out***.println("main" + "i:" + i);  } |

## 优先级

现代操作系统基本采用时分的形式调度运行的线程，线程分配得到的时间片的多少决定了线程使用处理器资源的多少，也对应了线程优先级这个概念。在JAVA线程中，通过一个int priority来控制优先级，范围为1-10，其中10最高，默认值为5。下面是源码（基于1.8）中关于priority的一些量和方法。

|  |
| --- |
| **class PrioritytThread implements Runnable {**  **public void run() {**  **for (int i = 0; i < 100; i++) {**  **System.*out*.println(Thread.*currentThread*().toString() + "---i:" + i);**  **}**  **}**  **}**  **/\*\***  **\***  **\* @classDesc: 功能描述:(Join方法)**  **\* @author: 余胜军**  **\* @createTime: 2017年8月20日 下午9:23:30**  **\* @version: v1.0**  **\* @copyright:上海每特教育科技有限公司**  **\*/**  **public class ThreadDemo4 {**  **public static void main(String[] args) {**  **PrioritytThread prioritytThread = new PrioritytThread();**  **Thread t1 = new Thread(prioritytThread);**  **Thread t2 = new Thread(prioritytThread);**  **t1.start();**  **// 注意设置了优先级， 不代表每次都一定会被执行。 只是CPU调度会有限分配**  **t1.setPriority(10);**  **t2.start();**    **}**  **}** |

## Yield方法

Thread.yield()方法的作用：暂停当前正在执行的线程，并执行其他线程。（可能没有效果）

yield()让当前正在运行的线程回到可运行状态，以允许具有相同优先级的其他线程获得运行的机会。因此，使用yield()的目的是让具有相同优先级的线程之间能够适当的轮换执行。但是，实际中无法保证yield()达到让步的目的，因为，让步的线程可能被线程调度程序再次选中。

结论：大多数情况下，yield()将导致线程从运行状态转到可运行状态，但有可能没有效果。

# 作业题

## 现… 在有T1、T2、T3三个线程，你怎样保证T2在T1执行完后执行，T3在T2执行完后执行

代码:

|  |
| --- |
| public class JoinThreadDemo02 {  public static void main(String[] args) {  Thread t1 = new Thread(new Runnable() {  public void run() {  for (int i = 0; i < 20; i++) {  System.*out*.println("t1,i:" + i);  }  }  });  Thread t2 = new Thread(new Runnable() {  public void run() {  try {  t1.join();  } catch (Exception e) {  // TODO: handle exception  }  for (int i = 0; i < 20; i++) {  System.*out*.println("t2,i:" + i);  }  }  });  Thread t3 = new Thread(new Runnable() {  public void run() {  try {  t2.join();  } catch (Exception e) {  // TODO: handle exception  }  for (int i = 0; i < 20; i++) {  System.*out*.println("t3,i:" + i);  }  }  });  t1.start();  t2.start();  t3.start();  }  } |

# 面试题

1. **进程与线程的区别？**

答:进程是所有线程的集合，每一个线程是进程中的一条执行路径，线程只是一条执行路径。

1. **为什么要用多线程？**

答:提高程序效率

1. **多线程创建方式？**

答:继承Thread或Runnable 接口。

1. **是继承Thread类好还是实现Runnable接口好？**

答:Runnable接口好，因为实现了接口还可以继续继承。继承Thread类不能再继承。

1. **你在哪里用到了多线程？**

答:主要能体现到多线程提高程序效率。

举例:分批发送短信、迅雷多线程下载等。