###### 进程与线程的区别

1.进程

狭义定义:进程是正在运行的程序的实例（an instance of a computer program that is being executed）。

广义定义：进程是一个具有一定独立功能的程序关于某个数据集合的一次运行活动。它是操作系统动态执行的基本单元，在传统的操作系统中，进程既是基本的分配单元，也是基本的执行单元。

进程的概念主要有两点：第一，进程是一个实体。每一个进程都有它自己的地址空间，一般情况下，包括文本区域（text region）、数据区域（data region）和堆栈（stack region）。文本区域存储处理器执行的代码；数据区域存储变量和进程执行期间使用的动态分配的内存；堆栈区域存储着活动过程调用的指令和本地变量。第二，进程是一个“执行中的程序”。程序是一个没有生命的实体，只有处理器赋予程序生命时（操作系统执行之），它才能成为一个活动的实体，我们称其为进程。

2.线程

线程，有时被称为轻量级进程(Lightweight Process，LWP），是程序执行流的最小单元。一个标准的线程由线程ID，当前指令指针(PC），寄存器集合和堆栈组成。

###### yield()方法

Java线程调度的一点背景

在各种各样的线程中，Java虚拟机必须实现一个有优先权的、基于优先级的调度程序。这意味着Java程序中的每一个线程被分配到一定的优先权，使用定义好的范围内的一个正整数表示。优先级可以被开发者改变。即使线程已经运行了一定时间，Java虚拟机也不会改变其优先级

优先级的值很重要，因为Java虚拟机和下层的操作系统之间的约定是操作系统必须选择有最高优先权的Java线程运行。所以我们说Java实现了一个基于优先权的调度程序。该调度程序使用一种有优先权的方式实现，这意味着当一个有更高优先权的线程到来时，无论低优先级的线程是否在运行，都会中断(抢占)它。这个约定对于操作系统来说并不总是这样，这意味着操作系统有时可能会选择运行一个更低优先级的线程。(我憎恨多线程的这一点，因为这不能保证任何事情)

注意Java并不限定线程是以时间片运行，但是大多数操作系统却有这样的要求。在术语中经常引起混淆：抢占经常与时间片混淆。事实上，抢占意味着只有拥有高优先级的线程可以优先于低优先级的线程执行，但是当线程拥有相同优先级的时候，他们不能相互抢占。它们通常受时间片管制，但这并不是Java的要求。

理解线程的优先权

接下来，理解线程优先级是多线程学习很重要的一步，尤其是了解yield()函数的工作过程。

记住当线程的优先级没有指定时，所有线程都携带普通优先级。

优先级可以用从1到10的范围指定。10表示最高优先级，1表示最低优先级，5是普通优先级。

记住优先级最高的线程在执行时被给予优先。但是不能保证线程在启动时就进入运行状态。

与在线程池中等待运行机会的线程相比，当前正在运行的线程可能总是拥有更高的优先级。

由调度程序决定哪一个线程被执行。

t.setPriority()用来设定线程的优先级。

记住在线程开始方法被调用之前，线程的优先级应该被设定。

你可以使用常量，如MIN\_PRIORITY,MAX\_PRIORITY，NORM\_PRIORITY来设定优先级

当一个线程调用yield()方法时,表示该线程告诉java虚拟机它乐意让出自己的CPU使用权给其他的线程。

使用yield()方法需要注意的几点

(1)Yield是一个静态的原生(native)方法

(2)Yield告诉当前正在执行的线程把运行机会交给线程池中拥有相同优先级的线程。

(3)Yield不能保证使得当前正在运行的线程迅速转换到可运行的状态

(4)它仅能使一个线程从运行状态转到可运行状态，而不是等待或阻塞状态

**package** com.yin.demo.tij.chapter21\_1;

/\*\*

\* yield()方法使用示例

\*

\* **@author** yixunzhi

\* **@createdTime** 2015年8月21日下午5:58:49

\*

\*/

**public** **class** demo01 {

**public** **static** **void** main(String[] args) {

Thread t1 = **new** Thread(**new** Producer());

Thread t2 = **new** Thread(**new** Consumer());

t2.start();

t1.start();

}

}

**class** Producer **implements** Runnable {

**public** **void** run() {

**for** (**int** i = 0; i < 5; i++) {

System.*out*.println("I'm producer " + i);

Thread.*yield*();

}

}

}

**class** Consumer **implements** Runnable{

**public** **void** run() {

**for** (**int** i = 0; i < 5; i++) {

System.*out*.println("I'm consumer " + i);

Thread.*yield*();

}

}

}

###### Sleep()方法

上面代码执行多次会发现结果是不固定的。但是可以发现yield()不能保证线程交替执行。

###### Join()方法