**线程**

1. java的mian线程：

每一个程序开始运行时，至少有一个线程。

1. Java默认的线程就是main线程（java基础部分的程序中main()函数启动的线程）：
2. 除了java默认的线程，我们还可以继承来自Thread类来建立的线程对象。而Thread类建立的线程对象则从run()方法开始执行。

Eg：

先让一个类来继承Thread类，然后构造这个类的run方法(属于重写)，run方法中可以调用Thread中的各种方法，最后在main()方法中建立新类的对象（线程），对象通过调用start()方法让线程对象开始运行。

\* 线程常用方法介绍

\* 线程的创建：Thread()

\* Thread(String name)

\* Thread(Runnabal target)

\* Thread(Runnabal target,String name)

\*线程的方法：

\* 启动：void start();

\* 休眠：static void sleep(long millis);

\* static void sleep(long millis,int nanos);

\* 使其他的线程等待当前线程终止：

\* void join();

\* void join(long millis);

\* void join(long millis,int nanos);

\* 使当前运行线程释放处理器资源：

\* static void yield();

设定线程名称：

Void setName();

显示线当前程数目：

Void activeCount();

\*获取线程引用：

\* static Thread currentThread();

Eg：**public** **class** A **extends** Thread {

**private** String name;

**public** A(String name1)

{

name=name1;

}

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

{

**for**(**int** i=0;i<100;++i)

{

System.***out***.println(Thread.*currentThread*().getName()+"　"+"购买彩票: "+(**int**)((Math.*random*()\*10)+1)+"号");

**try** {

**if**(++i%10==0)

{

Thread.*sleep*(5000);//休眠5秒

}

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

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

// **TODO** Auto-generated method stu

A a1=**new** A("金玉宝");//第一个线程

a1.start();//启动第一个线程

A a2=**new** A("张宽");//第二个线程

a2.start();//启动第二个线程

}

}

1. **利用Runnable接口建立多线程**

利用接口可以实现多重继承的目的，当我们想要建立的多线程的类已经继承了其他的类，很显然此时就不可以使用继承Thread类实现多线程，此时就需要使用Runnable接口实现多重继承以及多线程。

与直接继承的区别：  
建立线程对象时存在区别：

Thread t=new classname();

或者：

Classname c=new classname();

Thread t=new Thread(c);

因为是通过接口继承的，就必须用接口来实例化对象，通过接口实例化的对象来调用接口里面的方法。、

Eg:

**package** 接口实现福利彩票;

/\*\*

\* 用一个实现类B

\* **@author** Administrator

\*

\*/

**class** B {

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

// **TODO** Auto-generated method stub

C a=**new** C("金玉宝");

Thread t=**new** Thread(a);//建立线程对象

C a1=**new** C("张宽");

Thread t1=**new** Thread(a1);//建立线程对象

t.start();

t1.start();

}

}

**class** C **implements** Runnable{

**private** String name;

**public** C(String name1)

{

name=name1;

}

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

{

**for**(**int** i=0;i<100;i++)

{

**int** j=0;

System.***out***.println(Thread.*currentThread*().getName()+" "+"购买彩票:"+(**int**)((Math.*random*()\*10)+1)+"号");

**if**((++j)%10==0)

{

**try** {

Thread.*sleep*(50);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

}

}

***注意*：**对于循环类的一定要记住使用**System.out*.flush()*;**

舒心缓冲池（释放资源）

4．线程的存活与中断

使用方法void isAlive()&&void interupt();

Eg;

**class** B {

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

// **TODO** Auto-generated method stub

C a=**new** C("金玉宝");

Thread t=**new** Thread(a);//建立线程对象

C a1=**new** C("张宽");

Thread t1=**new** Thread(a1);//建立线程对象

t.start();

t1.start();

t.setName("金玉宝");

t1.setName("张宽");

System.***out***.println("当前的线程总数为："+Thread.*activeCount*());

**if**(t.isAlive())

{

System.***out***.println("中断线程："+Thread.*currentThread*().getName());

t.*interrupted*();

}

**if**(t1.isAlive())

{

System.***out***.println("中断线程："+Thread.*currentThread*().getName());

t1.*interrupted*();

}

}

}

**class** C **implements** Runnable{

**private** String name;

**public** C(String name1)

{

name=name1;

}

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

{

**for**(**int** i=0;i<100;i++)

{

**int** j=0;

System.***out***.println(Thread.*currentThread*().getName()+" "+"购买彩票:"+(**int**)((Math.*random*()\*10)+1)+"号");

**System.*out*.flush();**

**if**((++j)%10==0)

{

**try** {

Thread.*sleep*(50);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

}

}

**//两个类并列书写则没有public classname表达**

1. **设定线程的优先权：**

**每个线程都会设定一个优先权值，权值范围为：1~10；1是最低优先权值。**

**Thread类中声明权值的方法：**

**Void setprioority();**

**几个权值常数：**

**Thread.NORM\_PRIORITY; 5;**

**Thread.Max\_PRIORITY; 10;**

**Thread.Min\_PRIORITY; 1;**

**权值关乎：使用CPU资源的优先顺序以及占用CPU资源的多少；、**

**Eg:**

**public** **class** B **implements** Runnable

{

**private** **long** counter;

**public** **boolean** flag;

**public** B()

{

counter =0;

flag=**true**;

}

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

{

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**while**(flag)

{

++counter;

}

}//while不是方法，必须写在方法中

**public** **void** Ting()

{

flag=**false**;

}

**public** **long** Counter()

{

**return** counter;

}

}

**public** **class** A {

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

// **TODO** Auto-generated method stub

B b1=**new** B();

B b2=**new** B();

B b3=**new** B();

Thread t=**new** Thread(b1);

Thread t1=**new** Thread(b2);

Thread t2=**new** Thread(b2);

t.setName("金");

t1.setName("玉");

t2.setName("宝");

//System.out.println("总的线程数目："+Thread.activeCount());

System.***out***.println("main的优先权值为："+Thread.*currentThread*().getPriority());

System.***out***.println("t的优先权值为："+t.getPriority());

System.***out***.println("t1的优先权值为："+t1.getPriority());

System.***out***.println("t2的优先权值为："+t2.getPriority());

t.start();

t1.start();

t2.start();//只会执行方法run()中的

t.setPriority(10);

t2.setPriority(7);

t1.setPriority(3);

//设定优先权值

**try** {

Thread.*sleep*(1800);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.***out***.println(t.getName()+"执行："+b1.Counter()+"次");

System.***out***.println(t1.getName()+"执行："+b2.Counter()+"次");

System.***out***.println(t2.getName()+"执行："+b3.Counter()+"次");

}

}

1. **线程方法同步：**

**使用同一个类产生的多个线程对象，可以分别执行对象中的方法；当类中的方法设定为synchronized时，在同一时刻只能有一个线程对象执行这个方法，其他想要执行这个方法的线程，必须等到他空闲时才能执行。**

**Eg:  
public** **class** A **implements** Runnable{

**private** String name;

**public** A(String name1)

{

name=name1;

}

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

{

show();

}

**public** **synchronized** **void** show()

{

System.***out***.println(name);

}

//使用了 synchronized修饰的方法，在同一时间只能让一个线程执行，排队执行

}

**public** **class** B {

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

// **TODO** Auto-generated method stub

A a1=**new** A("jin");

A a2=**new** A("yu");

A a3=**new** A("bao");

Thread t1=**new** Thread(a1);

Thread t2=**new** Thread(a2);

Thread t3=**new** Thread(a3);

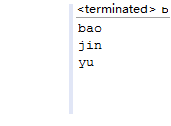
t3.start();

t1.start();

t2.start();

}

}



**Eg:(避免同步的运用)**

**package** 排队同步例子;

**public** **class** A **implements** Runnable{

**private** static **long** sum=0;**//声明为静态变量的重要性**

**private** String name;

**public** A(String name1)

{

name=name1;

}

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

{

**for**(**int** i=1;i<=5;++i)

{

add(name,1000);

}

}

**public** **synchronized** **void** add(String name,**long** m)

{

sum+=m;

System.***out***.println(name+"存入"+"m"+"总款"+sum);

}

}

**public** **class** B {

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

// **TODO** Auto-generated method stub

A a1=**new** A("金");

A a2=**new** A("玉");

A a3=**new** A("宝");

Thread t1=**new** Thread(a1);

Thread t2=**new** Thread(a2);

Thread t3=**new** Thread(a3);

t1.start();

System.***out***.println("线程总数："+Thread.*activeCount*());

t2.start();//此时t1还在执行，t2已经启动，进入待命状态，里面的语句还未进行执行

System.***out***.println("线程总数："+Thread.*activeCount*());

t3.start();

System.***out***.println("线程总数："+Thread.*activeCount*());

}

}

**//多个对象公用一个变量（此变量声明为一个静态变量）**

**注意：**

1. **如果没有使用synchronized定义的方法，则避免不了同步；**
2. **使用静态变量：static,否则会出现三个5000；**

**知识点：**

**类中的静态变量：（含有静态变量的类必须独立于函数之外，就像在声明全局变量要在函数之外声明一样）。**

**使用含有静态变量的类所建立的对象，彼此可以共同存取同一个静态变量，例如想要设计一个记录这个类一共产生了多少个对象**

1. **线程间的通信：**

Wait()方法和notify()方法可以建立线程之间的通信机制。前者可以使线程进入等待状态，后者可以唤醒第一个进入等待状态的线程，另外notify（）方法可以唤醒所有处于等待状态的线程。

当要设计具有通信机制的线程时，通常会将线程分为读取和写入两各个部分，这样读写之间就可以互相通信，每当有数据可读的时候，读取线程就进入等待状态，等待写入线程来唤醒，反之亦然。

1. **利用join()方法顺序显示**

**public** **class** A **extends** Thread{

**public** **int** b;

**public** A(**int** a1)

{

b=a1;

}

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

{

**for**(**int** i=1;i<=5;++i)

{

**int** m=i\*b;

System.***out***.println(i+"\*"+b+"的倍数："+m);

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

}

**public** **class** B {

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

// **TODO** Auto-generated method stub

A a1=**new** A(5);

A a2=**new** A(7);

A a3=**new** A(9);

// Thread t1=new Thread(a1);

// Thread t2=new Thread(a2);

// Thread t3=new Thread(a3);

a1.start();

**try** {

a1.join();

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

a2.start();

**try** {

a2.join();

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

a3.start();

**try** {

a3.join();

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

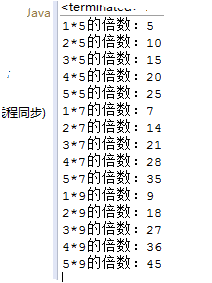
e.printStackTrace();

}

}

}

**结果显示：**



**编程练习：**

**1.**

**public** **class** B **implements** Runnable {

**public** **int** i;

**public** B(**int** i1)

{

i=i1;

}

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

{

**for**(**int** r=1;r<2000;++r)

{

**int** s=r\*i;

System.***out***.println(r+"\*"+i+"的倍数为："+s);

System.***out***.flush();

}

}

//public void Ting()

//{

// a1=false;

//}

}

**public** **class** A {

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

// **TODO** Auto-generated method stub

B b1=**new** B(7);

B b2=**new** B(9);

**boolean** a=**true**;

**long** start=System.*currentTimeMillis*();

// b1.run1();

// b2.run1();**//不用多线程需要的时间为：54微秒**

Thread t1=**new** Thread(b1);

Thread t2=**new** Thread(b2);

System.***out***.println("主线程的名称："+Thread.*currentThread*().getName());

t1.setName("7的倍数");

t2.setName("9的倍数");

System.***out***.println("t1线程的名称："+t1.getName());

System.***out***.println("t2线程的名称："+t2.getName());

t1.start();

t2.start();

**if**(a)

{

**Thread.*sleep*(100);//使主线程休眠**

**if**(t1.isAlive())

{

System.***out***.println("t1还在执行");

// b1.Ting();

}

**if**(t2.isAlive())

{

System.***out***.println("t2还在执行");

t2.interrupt();

}

**long** end=System.*currentTimeMillis*();

System.***out***.println("时间："+(end-start));**//使用多线程用时在5微秒之内**

}

System.***out***.println("总的线程数目："+Thread.*activeCount*());

}

}