# Java多執行緒

何謂單工、多工、雙向單工、雙向多工…？

電腦工作都要經過CPU處理，早期電腦只有一個CPU，現在多核心，可以完成多工處理。針對個別不需要複雜運算的工作(如影像處理、聲音串流…)，可以使用另一顆低階的CPU處理，以分散CPU的工作。

多工硬體必須配合多工軟體才能達到多工效能。單工硬體的執行速度夠快時，可以利用軟體在不同時間分別處理不同工作，當速度快到使用者無法察覺時，就可達到多工的效果，稱為分時多工。

多執行緒屬於分時多工，多工的執行不是依程式的順序。執行緒屬於程式，執行緒必須在程式中執行，執行緒結束，程式可繼續執行；程式結束，則執行緒必結束。

1. 繼承Thread類別
2. 實作Runnable介面

## 繼承Thread類別

package org.jinni;

class MyThread extends Thread{

private String name;

public MyThread(String name){

this.name = name;

}

@override

public void run(){

for (int x=0; x<20; x++){

System.out.println(this.name+" "+x);

}

}

}

run()是執行緒的啟動方法。通過物件直接呼叫。

public class ThreadDemo{

public static void main(String[] args){

MyThread t1 = new MyThread("執行緒1");

MyThread t2 = new MyThread("執行緒2");

MyThread t3 = new MyThread("執行緒3");

MyThread t4 = new MyThread("執行緒4");

MyThread t5 = new MyThread("執行緒5");

t1.run();

t2.run();

t3.run();

t4.run();

t5.run();

}

}

沒有實際啟動執行緒，所以結果仍依序執行。

t1.start();

此時執行緒才會同步執行，執行緒間的順序是依搶資源方式安排。

## 實作Runnable介面

public interface Runnable{

public void run();

}

透過Runnable介面啟動執行緒。

Thread類別的構造方法：Thread(Runnable target)表示Thread接收的引數是Runnable子類別物件。所以執行緒使用Runnable介面時，可使用Thread的start()方法啟動。

class MyThread implements Runnable{

private String name;

public MyThread(String name){

this.name = name;

}

@override

public void run(){

for (int x=0; x<30; x++){

System.out.println(this.name+" "+x);

}

}

}

public class ThreadDemo{

public static void main(String[] args){

MyThread t1 = new Thread("執行緒1");

MyThread t2 = new Thread("執行緒2");

MyThread t3 = new Thread("執行緒3");

MyThread t4 = new Thread("執行緒4");

MyThread t5 = new Thread("執行緒5");

New Thread(t1).run();

New Thread(t2).run();

New Thread(t3).run();

New Thread(t4).run();

New Thread(t5).run();

}

}

## 啟動方法比較

執行緒使用Thread有主體類別單繼承的束縛

別最好繼承抽象類別或實作介面。

## 共用資料

### 資料獨佔

class MyThread extends Thread{

private int ticket=5;

@Override

public void run(){

for (int x=0; x<30; x++){

if (this.ticket > 0){

System.out.println("賸餘票數："+ this.ticket--);

}

}

}

}

public class ThreadDemo{

public static void main(String[] args){

MyThread t1 = new MyThread();

MyThread t2 = new MyThread();

MyThread t3 = new MyThread();

t1.start();

t2.start();

t3.start();

}

}

### 資料分享

#### 使用extends Thread

class MyThread extends Thread{

private int ticket=5;

@Override

public void run(){

for (int x=0; x<30; x++){

if (this.ticket > 0){

System.out.println("賸餘票數："+ this.ticket--);

}

}

}

}

public class ThreadDemo{

public static void main(String[] args){

MyThread t = new MyThread();

New Thread(t).start();

New Thread(t).start();

New Thread(t).start();

}

}

#### 使用implements Runnable

class MyThread implements Runnable{

private int ticket=15;

@Override

public void run(){

for (int x=0; x<30; x++){

if (this.ticket > 0){

System.out.println("賸餘票數："+ this.ticket--);

}

}

}

}

public class ThreadDemo{

public static void main(String[] args){

MyThread t = new Thread();

New Thread(t).start();

New Thread(t).start();

New Thread(t).start();

}

}

## 執行緒

執行緒命名不重複，命名後不修改。

構造子：public Thread(Runnable taget, String name)

命名方法：public final void setName(String name)

取得名稱方法：public final String getName()

取得目前執行緒名稱方法：public static Thread currentThread()

class MyThread extends Thread{

@Override

public void run(){

for (int x=0; x<30; x++){

System.out.println("執行中的執行緒："+Thread.currentThread().getName());

}

}

}

public class ThreadDemo{

public static void main(String[] args){

MyThread t = new MyThread();

New Thread(t, "執行緒1").start();

New Thread(t).start();

New Thread(t, "執行緒2").start();

New Thread(t).start();

}

}

未命名執行緒程式自動產生特定名稱。

class MyThread extends Thread{

@Override

public void run(){

System.out.println("執行中的執行緒："+Thread.currentThread().getName());

}

}

public class ThreadDemo{

public static void main(String[] args){

MyThread t = new MyThread();

New Thread(t, "執行緒").start();

t.run();

}

}

main()就是一個執行緒。

## 執行緒休眠

public static void sleep(long millis)throws InterruptedException

class MyThread extends Thread{

@Override

public void run(){

for (int x=0; x<30; x++){

try{

Thread.sleep(1000);

} catch (InterruptedException e){

e.printStackTrace();

}

System.out.println("執行中的執行緒："+Thread.currentThread().getName()+" x="+x);

}

}

}

public class ThreadDemo{

public static void main(String[] args){

MyThread t = new MyThread();

New Thread(t,"執行緒1").start();

New Thread(t,"執行緒2").start();

New Thread(t,"執行緒3").start();

New Thread(t,"執行緒4").start();

New Thread(t,"執行緒5").start();

}

}

## 執行緒的優先等級

優先等級高者先被執行。

public static final int MAX\_PRIORITY = 10

public static final int NORM\_PRIORITY = 5

public static final int MIN\_PRIORITY = 1

class MyThread extends Thread{

@Override

public void run(){

System.out.println("執行中的執行緒："+Thread.currentThread().getName());

}

}

public class ThreadDemo{

public static void main(String[] args){

MyThread t = new MyThread();

Thread t1 = new Thread(t, "執行緒1");

Thread t2 = new Thread(t, "執行緒2");

Thread t3 = new Thread(t, "執行緒3");

t1.start();

t2.start();

t3.start();

}

}

public class ThreadDemo{

public static void main(String[] args){

MyThread t = new MyThread();

Thread t1 = new Thread(t, "執行緒1");

Thread t2 = new Thread(t, "執行緒2");

Thread t3 = new Thread(t, "執行緒3");

t1.setPriority(Thread.MIN\_PRIORITY);

t1.setPriority(Thread.MAX\_PRIORITY);

t1.start();

t2.start();

t3.start();

}

}

取得執行緒的優先等級：Thread.currentThread().getPriority()

public class ThreadDemo{

public static void main(String[] args){

System.out.println(Thread.currentThread().getPriority());

}

}

## 同步

class MyThread extends Thread{

private int ticket=50;

@Override

public void run(){

for (int x=0; x<30; x++){

if (this.ticket > 0){

System.out.println(Thread.currentThread().getName()+"賸餘票數："+ this.ticket--);

}

}

}

}

public class ThreadDemo{

public static void main(String[] args){

MyThread t = new MyThread();

New Thread(t,"售票員1").start();

New Thread(t,"售票員2").start();

New Thread(t,"售票員3").start();

New Thread(t,"售票員4").start();

}

}

處理期間，有其他執行緒通過賸餘票數檢查，因此造成溢賣。

class MyThread extends Thread{

private int ticket=50;

@Override

public void run(){

for (int x=0; x<30; x++){

if (this.ticket > 0){

try{

Thread.sleep(300);

}catch(InterruptedException e){

e.printStackTrace();

}

System.out.println(Thread.currentThread().getName()+"賸餘票數："+ this.ticket--);

}

}

}

}

防止此一錯誤發生的方法：

1. 同步(synchronized)程式塊
2. 同步方法

### 同步程式

synchronized (this){

if (this.ticket > 0){

try{

Thread.sleep(300);

}catch(InterruptedException e){

e.printStackTrace();

}

System.out.println(Thread.currentThread().getName()+"賸餘票數："+ this.ticket--);

}

}

### 同步方法

class MyThread extends Thread{

private int ticket=50;

@Override

public void run(){

for (int x=0; x<30; x++){

this.sale();

}

}

public synchronized void sale(){

if (this.ticket > 0){

try{

Thread.sleep(300);

}catch(InterruptedException e){

e.printStackTrace();

}

System.out.println(Thread.currentThread().getName()+"賸餘票數："+ this.ticket--);

}

}

}

## 鎖死

交互作用的執行緒都使用同步，可能造成互相等待對方的情形。

Object提供三個執行緒防鎖死的方法：

1. 等待：public final void wait() throws InterruptedException;
2. 喚醒第一個：public final void notify(); 依程式順序執行。
3. 喚醒全部：public final void notifyAll(); 全部喚醒後，依優先等級執行。