

你我都是多執行緒者，因為我們可以同時做多件事

多執行緒 Multi-threading

簡介

目前我們寫的應用程式 Program 中，程式執行順序:一個指令做完，才接著做下一個指令。

然而，你有沒看到:

Web 瀏覽器可以同時下載檔案，同時播放音樂，同時顯示網頁內容。

你如何寫一個程式同時能做多個工作?

哪你就要學--多執行緒啦!

在一個應用程式中，同時執行多個行程(稱之為“thread” “線程”)。

多工與多執行緒不同有何不同?

多工是指作業系統在同一個時間執行多個應用程式

多執行緒是指在一個應用程式中同時執行多個行程

Thread 程式撰寫方法有兩種：

1. 繼承 Thread 類別(較容易)

啟動方式: 呼叫 物件方法 start()

```
Machine1 m1 = new Machine1();
```

2. 實作 Runnable 介面

沒有提供 start()方法

使用方式必須透過 Thread 類別的建構子：

Thread(new RunnableClass()) 傳入或是初始化一個 Runnable 物件

```
Thread m2 = new Thread(new Machine2());  
m2.start();
```

Runnable 介面 長得怎樣？怎麼介面是這麼簡單？

```
public interface Runnable {  
    public void run();  
}
```

為何需要執行緒？順序執行與多執行緒執行有何不同？

```
package thread;  
  
import java.util.Scanner;  
import javafx.application.Application;  
import javafx.application.Platform;  
import javafx.event.ActionEvent;  
import javafx.event.EventHandler;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.scene.layout.StackPane;  
import javafx.stage.Stage;  
  
public class Lab20_ThreadDemoJavaFXMain extends Application {  
  
    @Override  
    public void start(Stage primaryStage) {  
  
        //為何需要執行緒？  
        //順序執行:等待輸入  
        System.out.println("請輸入第 1 個數:");  
        Scanner input = new Scanner(System.in);
```

```
String msg = input.next();
System.out.println("第 1 個數輸入值:" + msg);

//執行緒執行:等待輸入
new Thread(new Runnable() {
    @Override
    public void run() {
        System.out.println("請輸入第 2 個數:");
        Scanner input = new Scanner(System.in);
        String msg = input.next();
        System.out.println("第 2 個數輸入值:" + msg);
    }
}).start();

//順序執行:無窮迴圈
int i = 0;
while (i < 10000) {
    System.out.println("looping");
}

//執行緒執行:無窮迴圈
new Thread(new Runnable() {
    @Override
    public void run() {
        while (true) {
            System.out.println("looping");
        }
    }
}).start();

//產生第 1 個視窗
Button btn = new Button();
btn.setText("第 1 個視窗按鈕");
btn.setOnAction(new EventHandler<ActionEvent>() {

    @Override
    public void handle(ActionEvent event) {
        System.out.println("Hello World!");
    }
});
```

```

        }
    });

    StackPane root = new StackPane();
    root.getChildren().add(btn);

    Scene scene = new Scene(root, 300, 250);

    primaryStage.setTitle("第 1 個視窗");
    primaryStage.setScene(scene);
    primaryStage.show();

    //產生第 2 個視窗 另外一個執行緒
    Stage myStage = new Stage();
    Button btn2 = new Button("額外產生的第 2 個視窗按鈕");
    StackPane root2 = new StackPane();
    root2.getChildren().add(btn2);
    btn2.setOnAction(new EventHandler<ActionEvent>() {

        @Override
        public void handle(ActionEvent event) {
            System.out.println("你好!");
            //Platform.exit();
        }
    });

    Scene scene2 = new Scene(root2, 300, 250);

    myStage.setTitle("第 2 個視窗");
    myStage.setScene(scene2);
    myStage.show();

    System.out.println("多少個執行緒:" + Thread.activeCount());

}

public static void main(String[] args) {
    launch(args);
}

```

```

    }
}

```

簡單的實例

第 1 種寫法:繼承 Thread 類別

```

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

//第一種寫法:繼承 Thread 類別
class Machine1 extends Thread {
    @Override
    public void run() {
        try {
            for (int i = 0; i <= 10; i++) {
                System.out.printf("Machine1 processes #%d\n", i);
                sleep(1000);
            }
            System.out.println("Machine1 is shutdown.");
        } catch (InterruptedException e) {
            System.out.println("Machine1 is interrupted!");
        }
    }
}

```

第 2 種寫法:實作 Runnable 介面

```

實作 Runnable 介面
class Machine2 implements Runnable {
    @Override

```

```

public void run() {
    try {
        for (int i = 0; i < 10; i++) {
            System.out.printf("Machine2 processes #%d with 0.5 second.\n", i);
            Thread.sleep(500);
        }
        System.out.println("Machine2 is shutdown.");
    } catch (InterruptedException e) {
        System.out.println("Machine2 is interrupted!");
    }
}
}

```

執行方式：

```

public class ThreadDemo1 {

    public static void main(String[] args) {

        //剛開始主程式本身是一個主要執行緒
        System.out.println("多少個執行緒:"+Thread.activeCount());

        Machine1 m1 = new Machine1();
        Thread m2 = new Thread(new Machine2());

        //執行方式 1
        m1.start();
        m2.start();

        //執行方式 2: 匿名執行緒物件
        new Machine1().start();
        new Thread(new Machine2()).start();

        //主要執行緒之下有 2 個執行緒
        System.out.println("多少個執行緒:"+Thread.activeCount());
    }
}

```

```
//執行方式 3 :
ExecutorService executor = Executors.newCachedThreadPool();
executor.execute(m1);
executor.execute(m2);
executor.shutdown();//關掉執行者 程式才會結束
}
}
```

執行方式 3 解說：

```
ExecutorService executor = Executors.newCachedThreadPool();
executor.execute(m1);
executor.execute(m2);
executor.shutdown();//關掉執行者 程式才會結束
```

(1)宣告 executor 是一個專門用來 執行 執行緒的 介面 ExecutorService (ExecutorService 介面 作為一個 資料型別 之用)

(2)呼叫 Executors 類別的靜態方法 newCachedThreadPool() 此方法回傳一個 ExecutorService 物件

(3) executor 就是一個可以執行 執行緒的物件了，這些執行緒放在一個 "游泳池"內 等著被執行

進階技術:

● Synchronized 同步(synchronized method)

當同時有多個執行緒要存取方法同一份資料(通常是某個變數)時，先到者優先使用，後到者必須持續等待前一個執行緒執行完離開後，釋放其 lock 才可存取。