第十五章
多執行緒

本章學習目標

- 認識執行緒
- 學習如何建立執行緒
- 學習如何管理執行緒
- 認識執行緒的同步處理

15.1 認識執行緒

「多執行緒」的機制可以同時執行多個程式區塊。

app15_1 是 run() 執行緒的範例：

```
01  // app15_1, run() 執行緒的範例
02  class CTest
03  {
04     private String id;
05     public CTest(String str)   // 建構元，設定資料成員 id
06     {
07        id=str;
08     }
09     public void run()    // run() method
10     {
11        for(int i=0;i<4;i++)
12        {
13           for(int j=0;j<100000000;j++);  // 空迴圈，用來拖慢 14 行執行的速度
14           System.out.println(id+" is running..");
15        }
16     }
17  }
```

```
18
19  public class app15_1
20  {
21     public static void main(String args[])
22     {
23        CTest dog=new CTest("doggy");
24        CTest cat=new CTest("kitty");
25        dog.run();
26        cat.run();
27     }
28  }
```

```
/* app15_1 OUTPUT-------
doggy is running..
doggy is running..
doggy is running..
doggy is running..
kitty is running..
kitty is running..
kitty is running..
kitty is running..
-------------------------*/
```

第 25 行用 dog 物件呼叫 run() method 的執行結果

第 26 行用 cat 物件呼叫 run() method 的執行結果

啟動執行緒

要啟動執行緒，必須先準備好下列兩件事情：

(1) 此類別必須是延伸自 Thread 類別，使自己成為它的子類別。

(2) 執行緒的處理必須撰寫在 run() method 內。

要使一類別可啟動執行緒，必須用下列的語法來撰寫：

```
class 類別名稱 extends Thread  // 從 Thread 類別延伸出子類別
{
     類別裡的資料成員;
     類別裡的 method;
     修飾子 run()       // 改寫 Thread 類別裡的 run() method
     {
        以執行緒處理的程序;
     }
}
```

格式 15.1.1
執行緒定義語法

以上述的觀念來重新撰寫 app15_1，使它可以同時啟動多個執行緒：

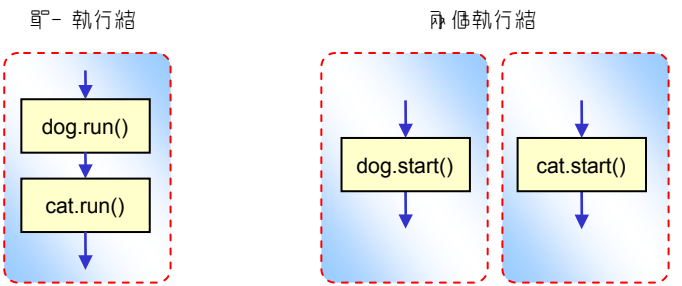
```
01  // app15_2, 啟動執行緒的範例
02  class CTest extends Thread  // 從 Thread 類別延伸出子類別 CTest
03  {
04     private String id;
05     public CTest(String str)   // 建構元，設定成員 id
06     {
07        id=str;
08     }

09     public void run()    // 改寫 Thread 類別裡的 run() method
10     {
11        for(int i=0;i<4;i++)
12        {
13           for(int j=0;j<100000000;j++);// 空迴圈,用來拖慢 14 行執行的速度
14           System.out.println(id+" is running..");
15        }
16     }
17  }
18
19  public class app15_2
20  {
21     public static void main(String args[])
```

```
22      {
23         CTest dog=new CTest("doggy");
24         CTest cat=new CTest("kitty");
25         dog.start();   // 注意是呼叫 start(),而不是 run()
26         cat.start();   // 注意是呼叫 start(),而不是 run()
27      }
28   }
```

```
/* app15_2 OUTPUT-------
kitty is running..   ——— 第 26 行用 cat 物件呼叫 start() method
doggy is running..   ——— 第 25 行用 dog 物件呼叫 start() method
kitty is running..
kitty is running..
doggy is running..
kitty is running..
doggy is running..
doggy is running..
-------------------------*/
```

下圖為單一執行緒與兩個執行緒的執行流程比較：



app15_1.java

執行完 dog.run() 之後，再執行 cat.run()

app15_2.java

由 start() 啟動執行緒，因而 dog 和cat物件的run() 會同時被執行

15.2 實作 Runnable 介面來建立執行緒

- ✓ 如果類別本身已經繼承了某個父類別，可以利用實作 Runnable 介面的方式建立執行緒。
- ✓ Runnable 介面裡宣告了抽象的 run() method，因此把處理執行緒的程式碼放在 run() 裡就可以建立執行緒。

下面的實例說明了 Runnable 介面的使用：

```
01  // app15_3,實作 Runnable 介面來建立執行緒
02  class CTest implements Runnable  // 由 CTest 類別實作 Runnable 介面
03  {
04     private String id;
05     public CTest(String str)   // 建構元，設定物件 id
06     {
07        id=str;
08     }
09     public void run()   // 詳細定義 runnable() 介面裡的 run() method
10     {
11        for(int i=0;i<4;i++)
12        {
13           for(int j=0;j<100000000;j++);// 空迴圈,用來拖慢 14 行執行的速度
14           System.out.println(id+" is running..");
15        }
16     }
17  }
18
19  public class app15_3
20  {
21     public static void main(String args[])
```

```
22      {
23         CTest dog=new CTest("doggy");
24         CTest cat=new CTest("kitty");
25         Thread t1=new Thread(dog);  // 呼叫 Thread 類別的物件 t1
26         Thread t2=new Thread(cat);  // 呼叫 Thread 類別的物件 t2
27         t1.start(); // 用 t1 啟動執行緒
28         t2.start(); // 用 t2 啟動執行緒
29      }
30   }
```

```
/* app15_3 OUTPUT-------
kitty is running..   ——— 第 28 行用 t2 物件呼叫 run() method
doggy is running..   ——— 第 27 行用 t1 物件呼叫 run() method
kitty is running..
kitty is running..
doggy is running..
kitty is running..
doggy is running..
doggy is running..
------------------------*/
```

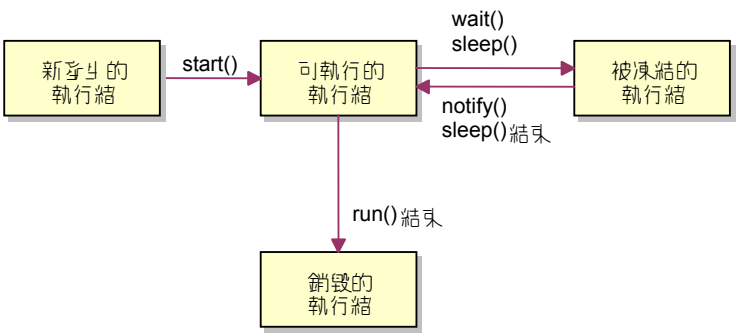
15.3 執行緒的管理

15.3.1 執行緒的生命週期

每一個執行緒，在其產生和銷毀之前，均會處於下列四種狀態之一：

- ✓ 新產生的（newly created）
- ✓ 可執行的（runnable）
- ✓ 被凍結的（blocked）
- ✓ 銷毀的（dead）

執行緒狀態的轉移與 method 之間的關係可用下圖來表示：



新產生的執行緒

用 new Thread() 建立物件時。

可執行的狀態

當 start() method 啟動執行緒時，執行緒便進入可執行的狀態。

被凍結的狀態

發生下列的事件時，凍結狀態的執行緒便產生：

1. 該執行緒呼叫物件的 wait() method。
2. 該執行緒本身呼叫 sleep() method。
3. 該執行緒和另一個執行緒 join() 在一起。

銷毀的狀態

當 run() method 執行結束，或是由執行緒呼叫它的 stop() method 時。

15.3.2 建立執行緒之實作步驟

下面是 sleep() method 的使用範例：

```
01  // app15_4, sleep() method 的示範
02  class CTest extends Thread  // 從 Thread 類別延伸出子類別
03  {
04     private String id;
05     public CTest(String str)   // 建構元，設定成員 id
06     {
07        id=str;
08     }
09     public void run()    // 改寫 Thread 類別裡的 run() method
10     {
11        for(int i=0;i<4;i++)
12        {
13           try
14           {
15              sleep((int)(1000*Math.random()));
16           }
17           catch(InterruptedException e){}
18           System.out.println(id+" is running..");
19        }
20     }
```

sleep() method 必須寫在 try-catch 區塊裡

```
21   }
22
23   public class app15_4
24   {
25      public static void main(String args[])
26      {
27         CTest dog=new CTest("doggy");
28         CTest cat=new CTest("kitty");
29         dog.start();
30         cat.start();
31      }
32   }
```

```
/* app15_4 OUTPUT-------
kitty is running..   ——— 第 30 行用 cat 物件呼叫 start() method
doggy is running..   ——— 第 29 行用 dog 物件呼叫 start() method
kitty is running..
kitty is running..
doggy is running..
kitty is running..
doggy is running..
doggy is running..
------------------------*/
```

15.3.3 等待執行緒

在兩個執行緒啟動之後，再加上一行列印字母的範例，結果會怎樣呢？

```
01  // app15_5,執行緒排程的設計(-)
02  // 將 app15_4 的CTest 類別置於此處
03  public class app15_5
04  {
05     public static void main(String args[])
06     {
07        CTest dog=new CTest("doggy");
08        CTest cat=new CTest("kitty");
09        dog.start();     // 用 dog 物件來啟動執行緒
10        cat.start();     // 用 cat 物件來啟動執行緒
11        System.out.println("main() method finished");
12     }
13  }
```

```
/* app15_5 OUTPUT-------
main() method finished
doggy is running..
kitty is running..
doggy is running..
doggy is running..
doggy is running..
kitty is running..
kitty is running..
kitty is running..
---------------------*/
```

下面是執行緒排程設計的範例：

```
01  // app15_6, 執行緒排程的設計(二)
02  // 將 app15_4 的 CTest 類別置於此處
03  public class app15_6
04  {
05     public static void main(String args[])
06     {
07        CTest dog=new CTest("doggy");
08        CTest cat=new CTest("kitty");
09
10        dog.start();     // 啟動 dog 執行緒
11        try
12        {
13           dog.join();   // 阻制 dog 執行緒結束後才能往下執行
14           cat.start();  // 啟動 cat 執行緒
15           cat.join();   // 阻制 cat 執行緒結束後才能往下執行
16        }
17        catch(InterruptedException e){}
18        System.out.println("main() method finished");
19     }
20  }
```

join() 必須寫在 try-catch 區塊裡

```
/* app15_6 OUTPUT-------
doggy is running..
doggy is running..
doggy is running..
doggy is running..
kitty is running..
kitty is running..
kitty is running..
kitty is running..
main() method finished
-------------------------*/
```

先執行 dog 執行緒

再執行 cat 執行緒

最後再執行第 18 行的敘述

15.4 同步處理

如果兩個執行緒共用一個變數，且其中一個執行緒在 run() method 還沒結束前，另一個執行緒已開始啟動，可能會造成錯誤，如下面的範例：

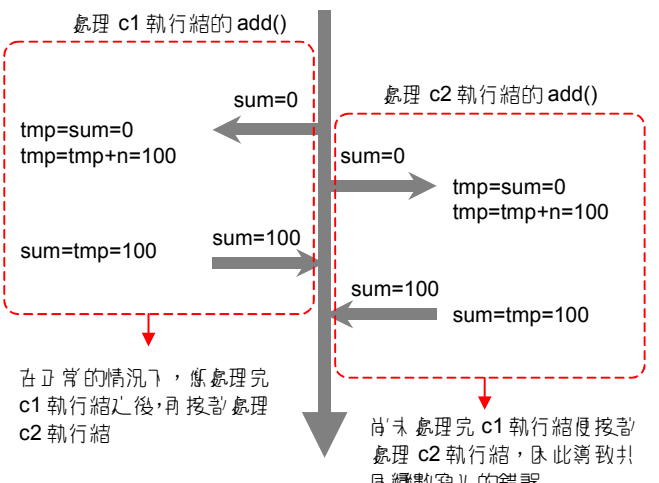
```
01  // app15_7, 沒有同步處理的執行緒
02  class CBank
03  {
04     private static int sum=0;
05     public static void add(int n)
06     {
07        int tmp=sum;
08        tmp=tmp+n;   // 累加提款總額
09        try
10        {
11           Thread.sleep((int)(1000*Math.random()));  // 小睡 0~1 秒鐘
12        }
13        catch(InterruptedException e){}
14        sum=tmp;
15        System.out.println("sum= "+sum);
16     }
17  }

18  class CCustomer extends Thread // CCustomer 類別,繼承自 Thread 類別
```

```
19   {
20      public void run()              // run() method
21      {
22         for(int i=1;i<=3;i++)
23            CBank.add(100);         // 將 100 元分三次滙入
24      }
25   }
26   public class app15_7
27   {
28      public static void main(String args[])
29      {
30         CCustomer c1=new CCustomer();
31         CCustomer c2=new CCustomer();
32         c1.start();
33         c2.start();
34      }
35   }
```

```
/* app15_7 OUTPUT----(沒有加synchronized的執行結果)
sum= 100
sum= 100
sum= 200
sum= 300
sum= 200
sum= 300
-----------------------*/
```

下圖顯示了這次錯誤的原因：



利用 synchronized 關鍵字即可修正前敘的錯誤：

```
05   public synchronized static void add(int n)
06   {
         ... ...
17   }
```

在 add() method 之前加上 synchronized 關鍵字

程式執行的結果應如下所示：

```
/* app15_7 OUTPUT----(加上 synchronized 的執行結果)
sum= 100
sum= 200
sum= 300
sum= 400
sum= 500
sum= 600
-----------------------*/
```


-The End-