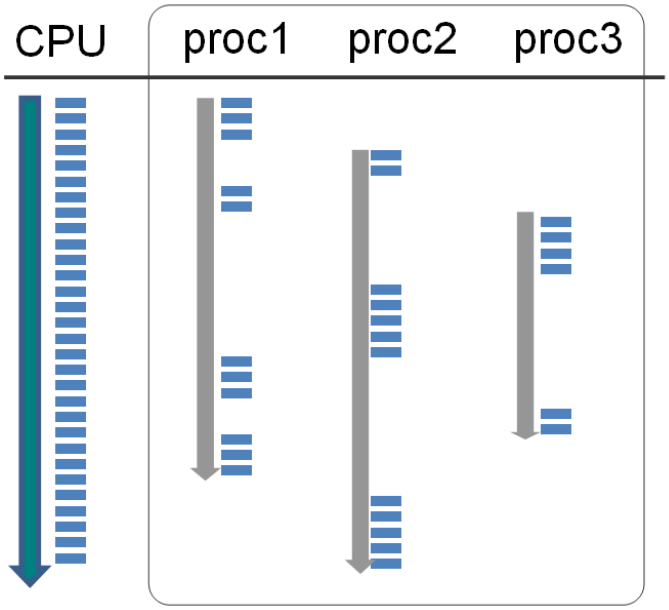
第 7 章　多執行緒程式設計

多執行緒程式設計

- 多個執行緒可以模擬出平行處理的效果
- 將一個CPU的執行時間切割為很小的單位，將這些單位分給多個行程去使用
- 模擬出多工（multi-task）的效果



main執行緒

- 一個可以執行的Java類別，在main這個特別的方法被執行時，也就擁有了作業系統的一個執行緒
- main方法的開始到結束
- 若設計一個賽馬的程式，就只能讓一匹馬從起點跑到終點

執行緒

- 執行緒的四種狀態
  - 執行中（Running）
    - 執行緒正在執行當中。
  - 暫停（Suspended）
    - 暫停已在執行當中的執行緒，但可以讓它繼續執行（resume）。
  - 被阻擋（Blocked）
    - 執行緒因某個特定的資源被其他執行緒占用時，會進入等待的狀態。
  - 終止（Terminated）
    - 執行緒被終止或停止，但無法再繼續執行。

Thread類別

- java.lang.Thread類別就讓設計者向作業系統取得額外的執行緒
- 在這個執行緒類別中，放入設計師想要做的工作
- 只需要繼承Thread類別即可讓新類別具備執行緒的能力

Thread類別的方法

回傳值	方法使用方式
String	getName()
	取得執行緒的名稱
void	run()
	執行緒的主要工作內容
void	start()
	啟動執行緒的方法
int	getPriority()
	取得執行緒的優先權值
boolean	isAlive()
	執行緒是否存活著？
void	join()
	等待執行緒結束，進入終止狀態
void	sleep(long millis)
	使執行緒進入睡眠狀態，傳入值為睡眠時間，單位是毫秒

兩匹馬假賽跑

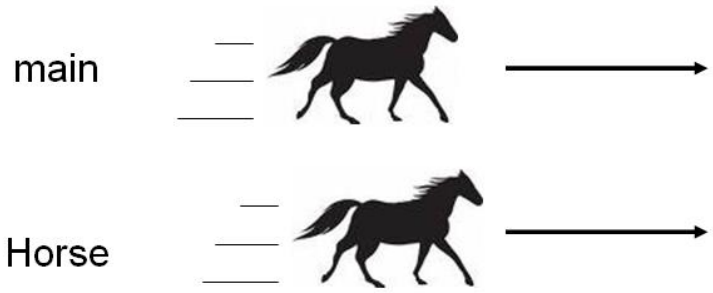
- 若只使用main方法來設計賽馬的程式，不能達到需求
- 利用for迴圈讓兩匹馬（整數h1與h2）從0加到5000，看那匹馬先到終點（5000）

```
03  public class RacingNG {
04      public static void main(String[] args) {
05          int h1 = 0;
06          int h2 = 0;
07          for (int i=0; i<5000; i++){
08              h1++;
09              h2++;
10              System.out.println("H1:"+h1);
11              System.out.println("H2:"+h2);
12          }
13      }
14  }
```

  - 執行結果，每一次執行一定是h1先到達終點

兩匹馬假賽跑(續)

- 每次執行結果都一樣，不符合實際需求
- 應該把h2的工作（從0加到5000）「搬到」另一個執行緒去處理
- 讓main執行緒與新執行緒一起執行



繼承Thread類別

- 新增一個Horse類別並繼承Thread
- 覆寫Thread的run()方法
- 將要執行的工作程式碼放在run()方法內

```
public class Horse extends Thread {

    public void run(){
        //程式碼

    }
}
```

產生執行緒並執行

- 將Horse類別建構出來後，再呼叫其方法start() ，即可啟動這個執行緒

```
1  package com.ch09;
2
3  public class Horse extends Thread{
4      //覆寫Thread方法run()
5      public void run(){
6          //由1跑到5000
7          int h = 0;
8          for (int i=0; i<5000; i++){
9              h++;
10             System.out.println(getName()+":"+ h);
11         }
12     }
13 }
```

產生執行緒並執行(續)

- 修改原程式Racing類別，在未開始執行迴圈之前，生出Horse類別的物件，再啟動這個執行緒

```
1  package com.ch09;
2
3  public class Racing {
4      public static void main(String[] args) {
5          int h1 = 0;
6          //產生Horse物件並啟動執行緒
7          Horse h2 = new Horse();
8          h2.start();
9          for (int i=0; i<5000; i++){
10             h1++;
11             System.out.println("H1:"+h1);
12         }
13     }
14 }
```

實作Runnable介面

- 有時候，一個已設計好的類別若已經繼承了其他父類別，而無法再繼承Thread類別
- 可以「實作Runnable介面」間接達成執行緒的設計

```
1  package com.ch09;
2
3  public class HorseRunnable implements Runnable {
4      public void run() {
5          int h = 0;
6          for (int i=0; i<5000; i++){
7              h++;
8              System.out.println(h);
9          }
10     }
11 }
```

啟動Runnable介面執行緒

- 產生執行緒類別的物件
  - HorseRunnable h3 = new HorseRunnable();
- 利用Thread類別建構子中的其中一個可接收參數Runnable物件的建構子
  - Thread thr = new Thread(h3);
- 呼叫Thread物件的start()方法
  - thr.start();

三匹馬的賽事

- 讓main方法專心處理產生執行緒與計算賽馬名次等工作
- 三匹馬都以Horse執行緒來執行
- 在main方法中產生三個Horse執行緒物件

```
1  package com.ch09;
2
3  public class Racing3 {
4      public static void main(String[] args) {
5          Horse h1 = new Horse();
6          Horse h2 = new Horse();
7          Horse h3 = new Horse();
8          h1.setName("h1");
9          h2.setName("h2");
10         h3.setName("h3");
11         h1.start();
12         h2.start();
13         h3.start();
14         System.out.println("main執行緒結束");
15     }
16 }
```

Thread類別的sleep方法

- sleep(long millis)
- sleep方法使用時必須以try…catch處理被意外中斷的例外InterruptedException

```
1  package com.ch09;
2
3  public class Horse extends Thread{
4      //覆寫Thread方法run()
5      public void run(){
6          try {
7              sleep(2000);
8              System.out.println(getName()+"到達終點");
9          } catch (InterruptedException e) {
10             System.out.println(getName()+"被中斷了");
11         }
12     }
13 }
```

問題

- 執行Racing3類別的結果如下（每次結果有可能不一樣）：

  main執行緒結束

  h1到達終點

  h3到達終點

  h2到達終點

- 可以讓main等待三個Horse執行緒都完成後，才結束嗎？

join()方法 - 等待

- Thread類別的方法join()，可以用來等待該執行緒完成

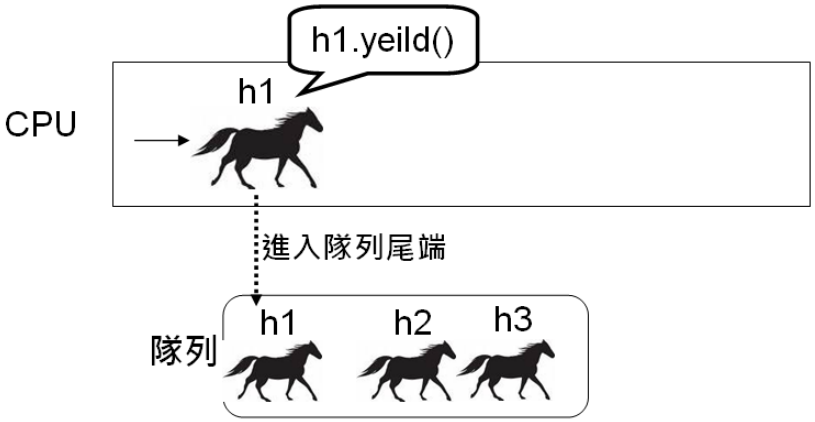
```
1  package com.ch09;
2
3  public class Racing3 {
4      public static void main(String[] args) {
5          Horse h1 = new Horse();
6          Horse h2 = new Horse();
7          Horse h3 = new Horse();
8          h1.setName("h1");
9          h2.setName("h2");
10         h3.setName("h3");
11         h1.start();
12         h2.start();
13         h3.start();
14         try {
15             h1.join();
16             h2.join();
17             h3.join();
18         } catch (InterruptedException e) {
19             System.out.println("執行緒被中斷");
20         }
21         System.out.println("main執行緒結束");
22     }
23 }
```

結算賽馬的名次 - Racing4

```
1  package com.ch09;
2
3  import java.util.Vector;
4
5  public class Racing4 {
6    public static void main(String[] args) {
7      Vector<RankHorse> rank = new Vector<RankHorse>();
8      RankHorse h1 = new RankHorse(rank);
9      RankHorse h2 = new RankHorse(rank);
10     RankHorse h3 = new RankHorse(rank);
11     h1.setName("h1");
12     h2.setName("h2");
13     h3.setName("h3");
14     h1.start();
15     h2.start();
16     h3.start();
17     try {
18       h1.join();
19       h2.join();
20       h3.join();
21     } catch (InterruptedException e) {
22       System.out.println("執行緒被中斷");
23     }
24     System.out.println(rank);
25     System.out.println("main執行緒結束");
26   }
27 }
```

管理執行緒

- 優先權值（Priority）
  - 為執行緒訂定「優先權值（Priority）」，優先權值較高的執行緒會較快進入執行的階段
- 執行緒的隊列（queue）
  - 執行緒的sleep()或yeild()方法被呼叫時，會暫停目前的工作，並進入執行緒的隊列（ready queue）排隊等待下一次CPU的執行單位



資源的鎖定

- 避免多個執行緒存取同一個資源
- Java語言利用「同步方法(method-level)」與「同步區塊(block-level)」這兩個方式
- 同步方法
  - 為一個類別的方法加上synchronized修飾字
  - 限定在同一時間，只能由此一個物件使用這個方法

```
public synchronized void thunder(){
    //程式碼
}
```

資源的鎖定(續)

- 同步區塊
  - 限定只能由單一執行緒能執行某段程式碼
  - 必需指定一個需要同步的物件

```
synchronized( 物件 ){
  程式碼
  程式碼
}
```

伺服器的多執行緒

- 以main執行緒為每個客戶端程式產生執行緒，以處理個別的需求。
- 設計一個執行緒，可分別處理每個客戶端的需求

```
1  package com.ch09;
2
3  import java.io.IOException;
4  import java.net.ServerSocket;
5  import java.net.Socket;
6
7  public class EchoServer {
8    public static void main(String[] args) {
9      try {
10       ServerSocket server = new ServerSocket(3333);
11       while (true) {
12         System.out.println("接受連線中");
13         Socket socket = server.accept();
14         //交由後續設計的Echo處理執行緒
15         EchoThread echo = new EchoThread(socket);
16         echo.start();
17       }
18
19     } catch (IOException e) {
20       e.printStackTrace();
21       System.out.println("伺服器窗口發生錯誤");
22     }
23   }
24 }
```

設計EchoThread執行緒

- 初期設計

```
1 package com.ch09;
2
3 public class EchoThread extends Thread {
4   public void run() {
5
6   }
7 }
```

- 設計一建構子，接收java.net.Socket