CH. 8

陣列 (ARRAY)

HORAZON

C#程式設計

什麼是陣列 (ARRAY)?

- 陣列是一種**資料結構**，用來儲存**多個**相同類型的資料。
- 就像是一個置物櫃，每個格子都有編號，可以放一樣的東西。

為什麼需要陣列？

如果不使用陣列，儲存 5 個學生的成績需要宣告 5 個變數：

`score1`, `score2`, `score3`, `score4`, `score5` ... (太麻煩了！)

使用陣列，只需要一個變數：`scores`。

宣告與建立陣列

語法：資料型別[] 陣列名稱 = new 資料型別[長度];

```
// 建立一個可以放 5 個整數的陣列
int[] numbers = new int[5];

// 建立一個可以放 3 個字串的陣列
string[] names = new string[3];
```

直接初始化值：

```
// 宣告時直接給值，系統會自動判斷長度為 3
int[] scores = { 80, 90, 75 };

// 另一種寫法
string[] fruits = new string[] { "Apple", "Banana" };
```

陣列結構示意

假設宣告 `int[] array = { 2, 4, 6, 8, 10 };`

索引 (Index)	0	1	2	3	4
數值 (Value)	2	4	6	8	10

- **陣列長度 (Length)**：5 (共有 5 個元素)
- **索引範圍**：0 ~ 4 (最後一個元素的索引是 `Length - 1`)

存取陣列元素

- 陣列的索引 (Index) 是從 0 開始的。
- 第一個元素的索引是 `0`，第二個是 `1`，以此類推。

```
int[] data = { 10, 20, 30 };

Console.WriteLine(data[0]); // 印出 10
Console.WriteLine(data[2]); // 印出 30

data[1] = 99; // 修改第二個元素的值為 99
Console.WriteLine(data[1]); // 印出 99
```

> [!WARNING]
> 若存取超過範圍的索引 (例如 `data[3]`)，會發生 IndexOutOfRangeException 錯誤。

陣列長度 (LENGTH)

使用 `.Length` 屬性可以取得陣列的長度 (元素個數)。

```
string[] cars = { "Volvo", "BMW", "Ford", "Mazda" };

Console.WriteLine(cars.Length); // 印出 4
```

這在搭配迴圈使用時非常有用，可以確保不會超出範圍。

使用迴圈走訪陣列 (FOR)

最傳統的方式，利用索引值來存取。

```
int[] numbers = { 2, 4, 6, 8, 10 };

for (int i = 0; i < numbers.Length; i++)
{
    Console.WriteLine($"索引 {i} 的值是：{numbers[i]}");
}
```

- 優點：可以知道目前的索引值，也可以修改陣列內容。
- 缺點：語法稍長。

使用迴圈走訪陣列 (FOREACH)

最簡潔的方式，直接依序取出元素。

```
string[] members = { "Alice", "Bob", "Charlie" };

foreach (string member in members)
{
    Console.WriteLine(member);
}
```

- 優點：語法簡單，不易出錯。
- 缺點：無法知道目前的索引值，且**不能**透過 foreach 修改陣列元素的值 (唯讀)。

常用陣列操作 (ARRAY 類別)

C# 提供了 `Array` 類別來協助處理陣列。

1. 排序 (Sort)

```
int[] nums = { 5, 1, 3, 2, 4 };
Array.Sort(nums); // 由小到大排序
// nums 變成 { 1, 2, 3, 4, 5 }
```

2. 搜尋 (IndexOf)

```
int index = Array.IndexOf(nums, 3);
// 找到 3 在陣列中的位置 (索引值)
```

二維陣列 (MULTIDIMENSIONAL ARRAY)

就像是 Excel 表格，有列 (Row) 和 欄 (Column)。

```
// 宣告一個 2x3 的二維陣列 (2列 3欄)
int[,] matrix = {
    { 1, 2, 3 },
    { 4, 5, 6 }
};

Console.WriteLine(matrix[0, 0]); // 印出 1 (第一列第一欄)
Console.WriteLine(matrix[1, 2]); // 印出 6 (第二列第三欄)
```

常用於地圖、棋盤遊戲等應用。

二維陣列結構示意

假設宣告 `int[,] data = new int[3, 4];` (3列 4欄)

索引	Col 0	Col 1	Col 2	Col 3
Row 0	`[0,0]`	`[0,1]`	`[0,2]`	`[0,3]`
Row 1	`[1,0]`	`[1,1]`	`[1,2]`	`[1,3]`
Row 2	`[2,0]`	`[2,1]`	`[2,2]`	`[2,3]`

存取方式：`data[列索引, 欄索引]`

例如：`data[1, 2]` 代表 **第 2 列、第 3 欄** 的資料。

總結

- 陣列是儲存**固定大小**、**相同型別**資料的容器。
- 索引從 0 開始。
- 使用 `Length` 取得長度。
- 使用 `for` 或 `foreach` 遍歷資料。
- 善用 `Array.Sort()` 等內建方法來處理資料。

常見範例 1：計算總和與平均

計算全班 5 位同學的數學成績總和與平均分數。

```
int[] scores = { 80, 95, 78, 92, 85 };
int sum = 0;

foreach (int score in scores)
{
    sum += score;
}

double average = (double)sum / scores.Length;

Console.WriteLine($"總分：{sum}");
Console.WriteLine($"平均：{average:F2}"); // 取小數點後兩位
```

常見範例 2：計算總和

題目：計算總和

請宣告一個整數陣列包含 `10, 20, 30, 40, 50`，並使用 `foreach` 計算總和。

```
int[] numbers = { 10, 20, 30, 40, 50 };
int sum = 0;

foreach (int num in numbers)
{
    sum += num;
}

Console.WriteLine($"總和為：{sum}"); // 150
```

常見範例 3：尋找最大值與最小值

找出陣列中的最大值與最小值。

```
int[] nums = { 10, 5, 40, 30, 20 };

int max = nums[0]; // 假設第一個數是最大的
int min = nums[0]; // 假設第一個數是最小的

foreach (int n in nums)
{
    if (n > max) max = n;
    if (n < min) min = n;
}

Console.WriteLine($"最大值：{max}");
Console.WriteLine($"最小值：{min}");
```

常見範例 4：選擇排序法

```
int[] nums = { 10, 5, 40, 30, 20 };

for (int i = 0; i < nums.Length - 1; i++)
{
    int minIndex = i;
    for (int j = i + 1; j < nums.Length; j++)
    {
        if (nums[j] < nums[minIndex])
        {
            minIndex = j;
        }
    }
    // 交換最小值
    int temp = nums[i];
    nums[i] = nums[minIndex];
    nums[minIndex] = temp;
}

Console.WriteLine(string.Join(", ", nums));
```

常見範例 3：選擇排序法

用程式碼模擬從小到大排序的過程。

```cs
int[] nums = { 10, 5, 40, 30, 20 };

for (int i = 0; i < nums.Length - 1; i++)
{
int minIndex = i;
for (int j = i + 1; j < nums.Length; j++)
{
if (nums[j] < nums[minIndex])
{
minIndex = j;
}
}
// 交換最小值
```

資料結構與演算法簡介

- **資料結構 (Data Structure)**：
  - 用來儲存與組織資料的方式。
  - 就像是：**置物櫃**、**書架**、**排隊隊伍**。
  - 範例：**陣列 (Array)**。
- **演算法 (Algorithm)**：
  - 解決問題的明確步驟與流程。s
  - 就像是：**食譜**、**組裝說明書**。
  - 範例：**排序 (Sort)**、**搜尋 (Search)**。