CH. 14

集合與泛型 (COLLECTIONS & GENERICS)

HORAZON

C#程式設計

陣列 (ARRAY) 的缺點

回顧一下陣列：

```
int[] scores = new int[5]; // 長度固定為 5
```

1. **長度固定**：建立後就不能改變。如果玩家撿到第 6 個道具怎麼辦？
2. **功能較少**：要刪除中間某個元素很麻煩 (需要移動後面所有元素)。

在實際開發中，我們更常使用 **集合 (Collections)**。

LIST<T>：動態陣列

`List<T>` 是 C# 中最常用的集合，可以把它想像成「會自動長大的陣列」。

`<T>` 代表 **泛型 (Generic)**，意思是「你可以指定裡面要裝什麼型別」。

```
using System.Collections.Generic; // 必須引用

// 建立一個裝字串的 List
List<string> items = new List<string>();

// 新增資料 (Add)
items.Add("紅藥水");
items.Add("藍藥水");
```

現在長度是 2。如果再 Add，長度會自動變 3。

LIST<T> 常用操作

```
List<string> enemies = new List<string>() { "史萊姆", "哥布林" };

// 1. 取得資料 (跟陣列一樣用索引)
string e = enemies[0];

// 2. 取得數量 (是 Count 不是 Length)
int count = enemies.Count;

// 3. 插入資料 (Insert)
enemies.Insert(0, "魔王"); // 插在最前面

// 4. 移除資料 (Remove)
enemies.Remove("史萊姆"); // 刪除特定物件
enemies.RemoveAt(0);      // 刪除第 0 個
```

DICTIONARY<KEY, VALUE>：字典

如果你想透過「名字」或「ID」來找資料，而不是透過索引 (0, 1, 2...)，就使用 **字典**。
它是由 **鍵 (Key)** 與 **值 (Value)** 組成的對應表。

```
// Key是字串 (學號), Value是整數 (分數)
Dictionary<string, int> scores = new Dictionary<string, int>();

// 新增 (Key 不能重複！)
scores.Add("A001", 90);
scores.Add("A002", 80);

// 取值 (透過 Key)
int s = scores["A001"]; // 90
```

#Dictionary 的特性

- Key **必須唯一**：不能有兩個 "A001"。
- **無序**：字典裡的資料順序是不固定的。
- **快速查找**：就算有一萬筆資料，透過 Key 找資料依然非常快 (接近 O(1))。

常用方法：

```
if (scores.ContainsKey("A003")) {
    // 檢查有沒有這個 Key，避免報錯
}

scores.Remove("A001"); // 移除
```

FOREACH 迴圈

遍歷集合最簡單的方法是 `foreach`。

```
List<string> names = new List<string> { "Alice", "Bob", "Cat" };

foreach (string name in names)
{
    Console.WriteLine(name);
}
```

對於 Dictionary：

```
foreach (KeyValuePair<string, int> pair in scores)
{
    Console.WriteLine($"{pair.Key}: {pair.Value}");
}
```

泛型 (GENERICS) 的好處

為什麼要寫 `<string>` 或 `<int>`？
這稱為 **泛型**。

1. **型別安全**：編譯器會幫你檢查，你不能把 `int` 放進 `List<string>`。
2. **效能更好**：避免了裝箱/拆箱 (Boxing/Unboxing) 的效能損耗。
3. **程式碼重用**：`List` 這個類別只要寫一次，就能給各種類別使用。

總結：該用哪個？

需求	選擇
長度固定，追求極致效能	**陣列** (Array)
長度不固定，需要頻繁增刪	List<T> (最常用)
需要透過 ID / **名稱** 來找資料	Dictionary<Key, Value>
只是先排個隊 (先進先出)	Queue<T>
堆疊 (後進先出)	Stack<T>

課程結束

接下來，帶著這些知識進入 UNITY 的世界吧！

恭喜你完成了 C# 基礎課程！