9-1 vector

1. 為什麼需要 vector？

1.1 傳統陣列的限制

在 C++ 中，我們熟悉的傳統陣列有一個根本的問題：**大小必須在編譯時決定，且無法改變。**

```
int scores[100];   // 固定 100 格，多了浪費，少了不夠用
```

傳統陣列的痛點：

- 宣告時必須給定大小（或使用 `new`，但要自己管理記憶體）
- 不知道目前有幾個元素（需要額外的變數追蹤）
- 插入、刪除元素需要手動搬移資料
- 忘記 `delete[]` 就記憶體洩漏(memory leak)

```
// 傳統做法：又長又容易出錯
int* scores = new int[100];  // 動態跟作業系統要一塊記憶體
int count = 0;

scores[count++] = 90;  // 注意這裡的 count++，count 值先被評估，再遞增 1
scores[count++] = 85;

cout << "We have " << count << " items in scores[]" << endl;

for(int i=0; i<count; i++)
{
    cout << "scores[" << i << "] = " << scores[i] << endl;
}

// 用完還要記得把記憶體還作業系統
delete[] scores;
```

1.2 vector 是什麼？

`vector` 是 C++ **標準模板庫（STL）**中的一種**動態陣列容器**。

它的核心優勢：

- 大小可以**自動擴張**，不需要手動管理記憶體
- 提供豐富的**成員函數**，操作方便
- 支援與陣列相同的**下標存取** `[]`
- 離開作用域後**自動釋放記憶體**

1.3 圖解：記憶體動態擴張

```
初始狀態（capacity = 1）：
┌───┐
│ 3 │
└───┘
  size=1, capacity=1

push_back(7)，空間不足 → 自動擴張為 capacity=2：
┌───┬───┐
│ 3 │ 7 │
└───┴───┘
  size=2, capacity=2

push_back(5)，空間不足 → 自動擴張為 capacity=4：
┌───┬───┬───┬───┐
│ 3 │ 7 │ 5 │   │
└───┴───┴───┴───┘
```

```
  size=3, capacity=4（預留空間）

push_back(1)，空間足夠 → 直接寫入：
┌───┬───┬───┬───┐
│ 3 │ 7 │ 5 │ 1 │
└───┴───┴───┴───┘
  size=4, capacity=4
```

重點：size 是目前元素數量，capacity 是已分配的記憶體空間。vector 通常以倍增方式擴張，以攤平擴張的成本。

Revision #6
★Created 26 April 2026 11:44:09 by huihui
Updated 27 April 2026 07:01:30 by huihui