

Section 13-2 Constructor

建構子

◆ 1. 建構子是什麼？

建構子是一種特殊的**成員函式**，在 **物件被建立時自動呼叫**，負責：

- 初始化資料成員（class 的變數）
- 可以進行額外的準備工作（如開啟檔案、配置記憶體等）

✴ 特性

- 名字和 class 名稱一樣
- 沒有傳回值（甚至不能寫 void）
- 可以有參數（稱為「有參建構子」）

◆ 2. 建構子的種類

種類	說明
預設建構子（Default Constructor）	不帶參數
有參建構子（Parameterized Constructor）	帶參數，可自定義初始化方式
拷貝建構子（Copy Constructor）	用另一個同型別物件初始化
移動建構子（Move Constructor）	C++11 引入，用於資源移轉
委派建構子（Delegating Constructor）	一個建構子呼叫另一個建構子

◆ 3. 解構子是什麼？

解構子是物件「被銷毀時自動呼叫」的函式，負責：

- 清理資源（釋放記憶體、關閉檔案）

✴ 特性

- 名字為 ~類別名稱
- 沒有參數

- 不能 overload (過載)
- 每個 class 最多只能有一個解構子

◆ 4. 實作範例

範例：Box 類別

```
#include <iostream>
using namespace std;

class Box {
private:
    int length;
    int width;

public:
    // 預設建構子
    Box() {
        length = 1;
        width = 1;
        cout << "預設建構子被呼叫" << endl;
    }

    // 有參建構子
    Box(int l, int w) {
        length = l;
        width = w;
        cout << "有參建構子被呼叫" << endl;
    }

    // 拷貝建構子
    Box(const Box &b) {
        length = b.length;
        width = b.width;
        cout << "拷貝建構子被呼叫" << endl;
    }

    // 解構子
```

```

~Box() {
    cout << "解構子被呼叫" << endl;
}

int area() {
    return length * width;
}

};

int main() {
    Box b1;          // 呼叫預設建構子
    Box b2(3, 4);    // 呼叫有參建構子
    Box b3 = b2;     // 呼叫拷貝建構子
    cout << "b3 面積: " << b3.area() << endl;
    return 0;
}

```

◆ 5. 額外補充：初始化列表（Initializer List）

💡 建議用法：

```
Box(int l, int w) : length(l), width(w) {}
```

優點：

- 初始化順序更清楚
- 對 `const` 或 `reference` 成員是**唯一可行方式**
- 效能較佳，因為直接初始化而不是先 `default` 後賦值

◆ 使用初始化列表的好處詳細解析


✅ 1. 初始化順序更清楚（Initialization order is clearer）

◆ 誤解：你以為「初始化列表的順序」就是變數初始化的順序

其實不然！成員變數的初始化順序是根據 宣告順序，不是你在初始化列表裡寫的順序！

🧠 範例說明：

```
class Test{
private:
    int a;
    int b;
public:
    Test() : b(20), a(10) {} // 雖然 b 在 a 前，但實際上 a 先被初始化
};
```

 實際的初始化順序：

```
// 根據宣告順序（不是初始化列表順序）
// 實際執行：
// a = 10; // b = 20;
```

使用初始化列表可以幫你：

- 清楚掌握變數是何時初始化的（尤其當有依賴關係時）
- 避免一種 bug：你用未初始化的值初始化另一個變數

2. 對 `const` 或 `reference` 成員是唯一可行方式

 `const` & `reference` 無法在建構子本體中被賦值！

因為它們只能被初始化一次，而這個初始化只能透過初始化列表完成。

 範例說明：

```
class Example{
private:
    const int x;
    int& y;
public:
    //  正確做法：用初始化列表
    Example(int a, int& b) : x(a), y(b) {}

    //  錯誤做法：會編譯錯誤
    // Example(int a, int& b) {
    //     x = a; //  const 無法在本體內賦值
    //     y = b; //  reference 也無法重指向
    // }
```

```
//}
};
```

✅ 3. 效能較佳：直接初始化 > 預設建構後再賦值

🌱 什麼是「預設後再賦值」？

當你在建構子裡寫這樣：

```
Box::Box(int l, int w) {
    length = l;
    width = w;
}
```

其實這段程式做了：

1. 呼叫 `length` 和 `width` 的預設建構子（先初始化）
2. 再用 `l` 和 `w` 賦值一次 ⇒ 變成兩次操作

🌱 而這樣才是最有效率的作法：

```
Box::Box(int l, int w) : length(l), width(w) {}
```

只會呼叫一次初始化，直接用 `l` 和 `w` 來建構 `length` 和 `width`

📌 小結比較

方法	操作次數	可用於 const / reference	效能	推薦？
初始化列表	✅ 只初始化一次	✅ 可以	✅ 快	✅ 強烈推薦
在建構子本體賦值	❌ 初始化一次 + 賦值一次	❌ 不行	❌ 慢	❌ 不建議（除非必要）

😎 延伸話題：預設參數 vs 初始化列表

「預設參數」其實是 C++ 建構子常見的設計方式之一，它跟「初始化列表」是兩種不同的概念，但可以一起搭配使用。

一、什麼是預設參數？

建構子的預設參數允許你在「不給某些參數」的情況下仍然能呼叫建構子。

範例：





```
class Box {
private:
    int length;
    int width;
public:
    // 有預設參數的建構子
    Box(int l = 1, int w = 1) {
        length = l;
        width = w;
    }
};
```

你可以這樣使用：

```
Box b1;    // 使用預設值 (1, 1)
Box b2(5); // 使用 (5, 1)
Box b3(5, 6); // 使用 (5, 6)
```

這樣就不用多寫一個「預設建構子」了，程式碼也比較簡潔！

二、預設參數 vs 初始化列表 — 有何不同？

比較點	預設參數	初始化列表
是什麼	函式參數可以不給	成員變數初始化語法
能否用來初始化 const /reference 成員？	 不行	 可以
關注點	使用者呼叫建構子時 想不想給參數	內部初始化方式（初始化 vs 賦值）
可否一起使用？	 可以	 可以

三、範例比較：預設參數搭配初始化列表

```
class Person {
private:
    const string name;
    int age;

public:
    // 使用預設參數 + 初始化列表
    Person(string n = "Unknown", int a = 0)
        : name(n), age(a) {
        cout << "建構子被呼叫 name = " << name << ", age = " << age << endl;
    }
};
```

使用方式：

```
Person p1;           // name="Unknown", age=0
Person p2("Alice"); // name="Alice", age=0
Person p3("Bob", 25); // name="Bob", age=25
```

小陷阱提示

1. 不要在函式宣告和實作都寫預設參數
會造成「預設值重複定義」錯誤：

```
// 錯誤寫法
class Person {
public:
    Person(string n = "Unknown");
    // 預設參數寫這裡就好
};

Person::Person(string n = "Unknown") { // ❌ 再寫一次會錯 ...
}
```

2. 使用預設參數不等於使用初始化列表

預設參數只是在「呼叫建構子時提供預設值」，如果你在建構子裡用「=」去設定成員變數，那麼還是會多一次預設建構。

```
class A {
private:
    int x;

public:
    // 使用預設參數，但在建構子裡用 = 指派
    A(int val = 10) {
        x = val; // !這裡是「先預設建構 x，再指派」
    }
};
```

🧠 這段在背後其實是：

```
int x; // ← 系統先幫你做「預設初始化（沒內容）」
x = val; // ← 再把 val 指派進去
```

這個過程會產生：

- 一次預設初始化
- 一次指定值 → 雙重成本

✅ 使用「初始化列表」的寫法比較高效：

```
class A {
private:
    int x;
public:
    A(int val = 10) : x(val) { // ← 這裡直接用 val 初始化 x（建構時就決定了）
    }
};
```

🔧 這裡是「一次性初始化」，省去了不必要的多一次 assignment。

🔧 技術底層（尤其對 class 類型的成員很重要）

如果成員變數是物件（不是 int、double 這種 primitive type），差別就更大了！

```
class Student {
    private:
        string name;
    public:
        Student(string name_val = "Unknown") {
            name = name_val; // !string(name) 被預設建構一次，再 assign
        }
};
```

VS

```
class Student {
    private:
        string name;
    public:
        Student(string name_val = "Unknown") : name(name_val) {
            // ✅ name 直接使用 name_val 建構，不會有預設建構 + assign 的動作
        }
};
```

✅ 總結比較

名稱	功能	範例用途	可以合用嗎
預設參數	讓建構子使用者可以不給所有參數	Person(string name = "Unknown")	✅ 可以
初始化列表	更有效率、更正確地初始化成員	: name(n), age(a)	✅ 可以

😎 延伸話題：Constructor Overload vs 預設參數

Constructor Overloading（建構子多載）

你可以寫「多個建構子」，只要它們參數型別或數量不同，C++ 編譯器就會根據你傳入的參數來選擇呼叫哪一個。

✅ 範例：

```
class Box {  
private:  
    int length, width;  
  
public:  
    Box() {  
        length = 1;  
        width = 1;  
    }  
  
    Box(int l) {  
        length = l;  
        width = 1;  
    }  
  
    Box(int l, int w) {  
        length = l;  
        width = w;  
    }  
};
```

使用方法：

```
Box b1;    // 呼叫 Box()  
Box b2(5); // 呼叫 Box(int)  
Box b3(5, 6); // 呼叫 Box(int, int)
```

💡 「寫了三個版本」來應付不同的初始化需求。

📦 缺點：如果參數多或成員變數多，就會寫很多重複的建構子。

Default Parameters（預設參數）

你也可以只寫一個建構子，讓它用預設參數處理沒給的情況。

✓ 範例：

```
class Box {
    private:
        int length, width;
    public:
        Box(int l = 1, int w = 1) {
            length = l;
            width = w;
        }
};
```

使用方式：

```
Box b1;    // 預設 l=1, w=1
Box b2(5); // l=5, w=1
Box b3(5, 6); // l=5, w=6
```

💡 這樣就不用寫三個建構子了，更簡潔！

🧠 什麼時候用哪一個？

情況	建議用法
每個初始化情況都需要不同邏輯處理	✓ 用 overloading
只是「參數數量不同，邏輯都一樣」	✓ 用 default parameters