第一課：STL 的歷史與設計哲學

第一課：STL 的歷史與設計哲學

1.1 什麼是 STL？

STL 全名是 **Standard Template Library（標準模板庫）**，它是 C++ 標準函式庫中最核心的部分之一。
簡單來說，STL 提供了一套**經過高度優化、可重複使用**的程式元件，讓你不必從頭實作常見的資料結構（如動態陣列、鏈結串列、二元樹）和演算法（如排序、搜尋、複製）。

```
#include <iostream>
#include <vector>
#include <algorithm>

int main() {
// 不用自己寫動態陣列 - 用 vector
std::vector<int> numbers = {5, 2, 8, 1, 9};
// 不用自己寫排序 - 用 sort
std::sort(numbers.begin(), numbers.end());
// 不用自己寫搜尋 - 用 find
auto it = std::find(numbers.begin(), numbers.end(), 8);
if (it != numbers.end()) {
std::cout << "找到了: " << *it << std::endl;
}
return 0;
}
```

輸出：

```
找到了: 8
```

在你過去寫 C 語言時，如果需要一個可以動態增長的陣列，你可能要自己用 `malloc`、`realloc` 來管理記憶體。有了 STL，這些底層細節都被封裝好了。

1.2 STL 的歷史

起源：Alexander Stepanov 的願景

STL 的創造者是 **Alexander Stepanov**（亞歷山大·斯捷潘諾夫），一位出生於蘇聯的電腦科學家。

時間軸：

```
1970年代 Stepanov 開始思考「泛型編程」的概念
↓
1979年 在莫斯科開始研究抽象演算法
↓
1985年 移居美國，在 GE 研究中心繼續研究
↓
1987年 在 Bell Labs 與 Andrew Koenig 合作
↓
1992年 加入 HP 實驗室，與 Meng Lee 合作開發 STL
↓
1994年 向 C++ 標準委員會提案
↓
1994年7月 STL 被正式納入 C++ 標準草案
↓
1998年 隨 C++98 標準正式發布
```

關鍵時刻：1994 年的提案

1994 年 11 月，Stepanov 在聖地牙哥向 C++ 標準委員會展示了 STL。這是一個關鍵時刻——委員會只花了幾天就決定將 STL 納入標準。

這個決定之所以快速，是因為 STL 展現了一種**革命性的程式設計思維**：

「演算法不應該依賴於特定的資料結構，資料結構也不應該綁定特定的演算法。」

1.3 STL 的設計哲學

核心理念：泛型編程（Generic Programming）

STL 的設計哲學可以用一句話概括：

「用最少的程式碼，解決最多的問題。」

這是透過**泛型編程**達成的。讓我們看一個對比：

沒有泛型的世界（C 語言風格）

```
// 如果沒有泛型，你需要為每種型別寫一個排序函數
void sort_int(int* arr, int size) {
// 排序 int 的邏輯
```

```
}

void sort_double(double* arr, int size) {
// 排序 double 的邏輯（幾乎相同的程式碼）
}

void sort_string(char** arr, int size) {
// 排序 string 的邏輯（又是幾乎相同的程式碼）
}
```

有泛型的世界（STL 風格）

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <string>

int main() {
// 同一個 sort，處理所有型別
std::vector<int> ints = {3, 1, 4, 1, 5};
std::vector<double> doubles = {3.14, 1.41, 2.72};
std::vector<std::string> strings = {"cherry", "apple", "banana"};
std::sort(ints.begin(), ints.end());
std::sort(doubles.begin(), doubles.end());
std::sort(strings.begin(), strings.end());
std::cout << "排序後的整數: ";
for (int n : ints) std::cout << n << " ";
std::cout << std::endl;
std::cout << "排序後的浮點數: ";
for (double d : doubles) std::cout << d << " ";
std::cout << std::endl;
std::cout << "排序後的字串: ";
for (const auto& s : strings) std::cout << s << " ";
std::cout << std::endl;
return 0;
}
```

輸出：

```
排序後的整數: 1 1 3 4 5
```

```
排序後的浮點數: 1.41 2.72 3.14
排序後的字串: apple banana cherry
```

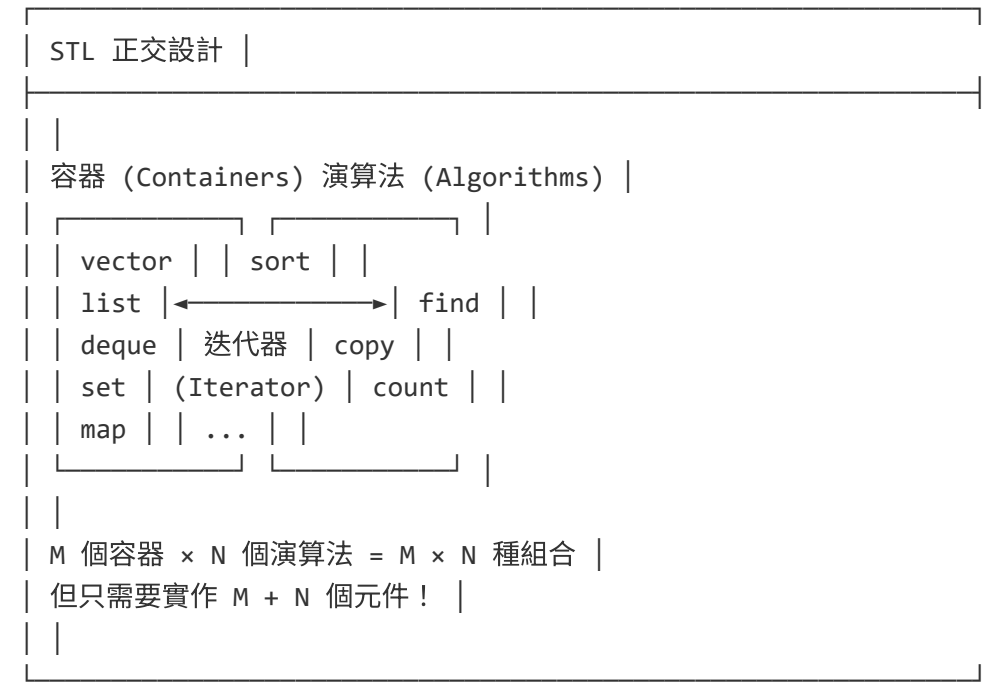
一個 `std::sort` 就能處理所有型別——這就是泛型編程的威力。

1.4 三個設計原則

Stepanov 在設計 STL 時，遵循了三個核心原則：

原則一：正交性（Orthogonality）

容器和**演算法**是獨立的兩個維度，透過**迭代器**連接。



假設有 10 個容器和 50 個演算法：

- **傳統做法**：需要實作 10 × 50 = 500 個函數
- **STL 做法**：只需要實作 10 + 50 = 60 個元件

這就是正交性帶來的效率。

原則二：效率優先（Efficiency）

STL 的設計目標是：

「泛型程式碼的效能應該與手寫特化程式碼相當。」

這意味著使用 `std::sort` 應該和你自己寫的最佳化排序一樣快。STL 透過 **template** 在編譯期展開，避免了執行期的效能損失。

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <chrono>

int main() {
const int SIZE = 1000000;
std::vector<int> data(SIZE);
// 填充隨機數據
for (int i = 0; i < SIZE; ++i) {
data[i] = rand();
}
// 測量 std::sort 的時間
auto start = std::chrono::high_resolution_clock::now();
std::sort(data.begin(), data.end());
auto end = std::chrono::high_resolution_clock::now();
auto duration = std::chrono::duration_cast<std::chrono::milliseconds>(end -
start);
std::cout << "排序 " << SIZE << " 個元素耗時: "
<< duration.count() << " 毫秒" << std::endl;
return 0;
}
```

可能的輸出：

```
排序 1000000 個元素耗時: 62 毫秒
```

`std::sort` 內部使用 **IntroSort**（混合排序），結合了 QuickSort、HeapSort 和 InsertionSort 的優點，是目前已知最有效率的通用排序演算法之一。

原則三：可組合性（Composability）

STL 的元件可以像樂高積木一樣組合：

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <iterator>

int main() {
std::vector<int> source = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

```
std::vector<int> destination;
// 組合多個 STL 元件：
// 1. copy_if - 演算法
// 2. back_inserter - 迭代器配接器
// 3. lambda - 函數物件
std::copy_if(
source.begin(), // 來源開始
source.end(), // 來源結束
std::back_inserter(destination), // 目標（自動擴展）
[](int n) { return n % 2 == 0; } // 條件：偶數
);
std::cout << "偶數: ";
for (int n : destination) {
std::cout << n << " ";
}
std::cout << std::endl;
return 0;
}
```

輸出：

```
偶數: 2 4 6 8 10
```

這段程式碼組合了：

- `copy_if` 演算法
- `back_inserter` 迭代器配接器
- Lambda 表達式作為判斷條件

1.5 STL 與傳統 C 風格的對比

讓我們用一個完整的例子來展示 STL 如何簡化程式碼：

任務：讀取數字、排序、找最大值、計算總和

C 風格寫法

```
#include <stdio.h>
#include <stdlib.h>

// 比較函數（給 qsort 用）
int compare(const void* a, const void* b) {
```

```
return (*(int*)a - *(int*)b);
}

int main() {
int* numbers = NULL;
int capacity = 4;
int size = 0;
// 動態配置
numbers = (int*)malloc(capacity * sizeof(int));
if (numbers == NULL) {
fprintf(stderr, "記憶體配置失敗\n");
return 1;
}
// 模擬輸入
int inputs[] = {5, 2, 8, 1, 9, 3, 7, 4, 6};
int input_count = sizeof(inputs) / sizeof(inputs[0]);
for (int i = 0; i < input_count; ++i) {
// 需要擴展嗎？
if (size >= capacity) {
capacity *= 2;
int* temp = (int*)realloc(numbers, capacity * sizeof(int));
if (temp == NULL) {
fprintf(stderr, "記憶體重新配置失敗\n");
free(numbers);
return 1;
}
numbers = temp;
}
numbers[size++] = inputs[i];
}
// 排序
qsort(numbers, size, sizeof(int), compare);
// 找最大值、計算總和
int max = numbers[0];
int sum = 0;
for (int i = 0; i < size; ++i) {
if (numbers[i] > max) max = numbers[i];
sum += numbers[i];
}
printf("排序後: ");
```

```
printf("%d ", numbers[i]);
}
printf("\n");
printf("最大值: %d\n", max);
printf("總和: %d\n", sum);
// 別忘了釋放記憶體！
free(numbers);
return 0;
}
```

STL 風格寫法

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <numeric>

int main() {
std::vector<int> numbers = {5, 2, 8, 1, 9, 3, 7, 4, 6};
// 排序
std::sort(numbers.begin(), numbers.end());
// 找最大值
int max = *std::max_element(numbers.begin(), numbers.end());
// 計算總和
int sum = std::accumulate(numbers.begin(), numbers.end(), 0);
std::cout << "排序後: ";
for (int n : numbers) {
std::cout << n << " ";
}
std::cout << std::endl;
std::cout << "最大值: " << max << std::endl;
std::cout << "總和: " << sum << std::endl;
return 0;
}
```

兩種寫法輸出相同：

```
排序後: 1 2 3 4 5 6 7 8 9
最大值: 9
總和: 45
```

對比分析

面向	C 風格	STL 風格
程式碼行數	~50 行	~20 行
記憶體管理	手動 malloc/realloc/free	自動
記憶體洩漏風險	高	極低
可讀性	需要理解指標操作	意圖清晰
維護成本	高	低

1.6 本課重點整理

```
┌──────────────────────────────────────────────────────────────────────┐
│ 第一課 重點整理 │
├──────────────────────────────────────────────────────────────────────┤
│ │
│ 1. STL = Standard Template Library │
│ → C++ 標準函式庫的核心部分 │
│ │
│ 2. 創造者：Alexander Stepanov │
│ → 1994 年被納入 C++ 標準 │
│ │
│ 3. 核心理念：泛型編程 │
│ → 一套程式碼，處理多種型別 │
│ │
│ 4. 三大設計原則： │
│ ① 正交性：容器與演算法獨立，透過迭代器連接 │
│ ② 效率優先：泛型不犧牲效能 │
│ ③ 可組合性：元件可自由組合 │
│ │
│ 5. STL 相比 C 風格： │
│ → 更少程式碼、更安全、更易讀、更易維護 │
│ │
```

└──┘

1.7 課後思考

在進入下一課之前，思考這個問題：
你在工作中使用 C++ 時，有沒有遇過需要自己管理動態陣列或實作排序的情況？如果有，那些程式碼是否可以用 STL 簡化？

準備好進入**第二課：泛型編程（Generic Programming）概念**了嗎？下一課我們會深入探討 template 如何讓 STL 的「一套程式碼處理多種型別」成為可能。