12-2 Iterator 的基本使用

一、取得 Iterator

函式	說明
`v.begin()`	指向第一個元素
`v.end()`	指向最後一個元素的下一位（不可解參考！）
`v.rbegin()`	反向，指向最後一個元素
`v.rend()`	反向，指向第一個元素的前一位

> v.end() 不是最後一個元素，是「越界哨兵(Sentry)」，只能用來判斷是否走完，不能 *v.end()！

```
vector<int> v = {10, 20, 30, 40, 50};

vector<int>::iterator it;   // 宣告 iterator
it = v.begin();             // 指向第一個元素

cout << *it << "\n";        // 取值：10
++it;                       // 向前移一位
cout << *it << "\n";        // 20
it += 2;                    // 向前移兩位（vector iterator 支援）
cout << *it << "\n";        // 40
```

二、使用 Iterator 走訪 vector

方法一：傳統 iterator 寫法

缺點是 iterator 的宣告很長，撰寫或閱讀都會很困擾。

```
for (vector<int>::iterator it = v.begin(); it != v.end(); ++it) {
    cout << *it << " ";
}
// 輸出：10 20 30 40 50
```

方法二：auto（C++11 起推薦）

自 C++11 起，auto 可以幫我們自動推導變數的型別。以下面程式碼為例，若 v 的型別是 `vector<int>`，則由 `v.begin()` 可以推導出 it 的型別為 `vector<int>::iterator`，這讓我們輕鬆不少。

```
for (auto it = v.begin(); it != v.end(); ++it) {
    cout << *it << " ";
}
```

方法三：範圍式 for（底層也是 iterator）

auto 和 範圍式for(range-based for loop) 在競程中都很常使用，因為非常簡潔方便。

```
for (int x : v) {
    cout << x << " ";
}
```

> 學 iterator 時建議先練習方法一，理解底層原理

三、反向走訪

vector 還有一組 `r` 開頭的 iterator，`rbegin()`、`rend()`。可以用來反向走訪。

```
vector<int> v = {10, 20, 30, 40, 50};
for (auto it = v.rbegin(); it != v.rend(); ++it) {
    cout << *it << " ";
}
// 輸出：50 40 30 20 10
```

四、Iterator 的算術運算

由於 `vector` 是具備隨機存取特性的容器，其 iterator 屬於隨機存取迭代器（Random Access Iterator），支援加減法運算。

```
vector<int> v = {10, 20, 30, 40, 50};
auto it = v.begin();

auto it2 = it + 3;          // 指向第 4 個元素
cout << *it2 << "\n";       // 40
```

```
cout << it2 - it << "\n";   // 兩個 iterator 的距離：3
```

> ⚠ 並非所有容器都支援 +、-。像 list 的 iterator 就只能 ++、--。這是 vector 的優勢之一。

五、修改元素

Iterator 不只能讀，也能寫：

```
vector<int> v = {1, 2, 3, 4, 5};
for (auto it = v.begin(); it != v.end(); ++it) {
    *it *= 2;   // 每個元素乘以 2
}
// v 變成 {2, 4, 6, 8, 10}
```

容器的 iterator 通常還會有一組配套的 const iterator，用來確保我們在走訪的過程中只能讀取，無法修改容器內的元素。

與 `begin()`、`end()` 配套的是 `cbegin()`、`cend()`。

與 `rbegin()`、`rend()` 配套的是 `crbegin()`、`crend()`。

若程式碼嘗試使用 const iterator 來修改容器內的元素，會被編譯器擋下來。

```
for (auto it = v.cbegin(); it != v.cend(); ++it) {
    // *it = 0;  // 編譯錯誤！
    cout << *it << " ";
}
```

六、插入與刪除

vector 的 insert 和 erase 都使用 iterator 指定位置：

```
vector<int> v = {1, 2, 4, 5};

// 在 index 2 插入 3
v.insert(v.begin() + 2, 3);
// v = {1, 2, 3, 4, 5}

// 刪除 index 1 的元素
v.erase(v.begin() + 1);
// v = {1, 3, 4, 5}
```

```
// 刪除 index 1 到 3（不含 3）
v.erase(v.begin() + 1, v.begin() + 3);
// v = {1, 5}
```

思考一個問題：如果在 erase 之前就有一些 iterator 指向這個 vector 裡的某些元素。那麼在 erase 之後，這些 iterator 還有用嗎？

Revision #2
★Created 2026-06-15 00:05:03 UTC by huihui
Updated 2026-06-15 01:49:22 UTC by huihui