泛型程式設計 與 C++ 標準程式庫

generic programming and the C++ Standard Library

一開始，讓我們考慮泛型程式設計領域裡頭挑選出來的幾個題目。這些難題的焦點放在如何有效使用 templates, iterators, algorithms，以及如何使用並擴充標準程式庫的設施。然後，這些想法會漂亮地導引出下一個章節，分析撰寫 exception-safe templates 時所謂的異常安全性（exception safety）。

條款 1：Iterators 困難度：7

每一位手上正在使用標準程式庫的程式員，都必須知道以下這些常見（或不是那麼常見）的 iterator 錯誤運用。你可以找出幾個錯誤？

以下程式至少有四個與 iterator 相關的問題。你可以找出幾個？

```
int main()
{
  vector<Date> e;
  copy ( istream_iterator<Date>(cin),
         istream_iterator<Date>(),
         back_inserter( e ) );
  vector<Date>::iterator first =
       find( e.begin(), e.end(), "01/01/95" );
  vector<Date>::iterator last =
       find( e.begin(), e.end(), "12/31/95" );
  *last = "12/30/95";
  copy( first,
        last,
        ostream_iterator<Date>( cout, "\n" ) );
```

```
    e.insert( --e.end(), TodaysDate() );
    copy( first,
          last,
          ostream_iterator( cout, "\n" ) );
}
```

解答

```
int main()
{
  vector<Date> e;
  copy ( istream_iterator<Date>(cin),
         istream_iterator<Date>(),
         back_inserter( e ) );
```

目前爲止一切正確。Date class 的作者提供了一個萃取函式（extractor function），型式爲 operator>>(istream&, Date&)，以便 istream_iterator<Date> 得以用來從 cin stream 讀取 Dates 資料。上述的 copy 演算法會把 Dates 裝填到 vector 內。

```
    vector<Date>::iterator first =
         find( e.begin(), e.end(), "01/01/95" );

    vector<Date>::iterator last =
         find( e.begin(), e.end(), "12/31/95" );

    *last = "12/30/95";
```

錯誤：上一行可能是不合法的，因爲 last 可能是 e.end()，因而不是一個可提領（dereferenceable）的 iterator。

如果找不到目標，find() 演算法會將其第二引數（也就是用以指示範圍尾端者）傳回。本例之中如果 "12/31/95" 不在 e 之內，那麼 last 便等於 e.end()，也就是指向 container 最後一個元素的下一位置，那不是個有效的 iterator。

```
    copy( first,
          last,
          ostream_iterator<Date>( cout, "\n" ) );
```

錯誤：這可能是不合法的，因爲 [first, last) 可能不是一個有效範圍；因爲 first 有可能在 last 之後。

舉個例子，如果 "01/01/95" 不在 e 之內而 "12/31/95" 在 e 之內，那麼 iterator last 將指向符合 "12/31/95" 的那個 Date object，而其位置將於 iterator first 之前；此時的 first 指向 e 的最後元素的下一位置。然而，copy 演算法要求 first 必須指向 last 之前，也就是說 [first, last) 必須是一個有效範圍（valid range）。

除非你用的是一個有檢驗能力的標準程式庫，可以偵測出像這樣的問題，否則很可能在 copy() 演算法運算期間或運算之後，出現一個難以診斷的 core dump（譯註：因程式錯誤而形成的一個錯誤狀態檔）。

```
    e.insert( --e.end(), TodaysDate() );
```

錯誤之一：--e.end() 可能是不合法的。

理由很簡單，但有一點隱晦：一般的 C++ 標準程式庫實際上都是以一個 Date* 來實作出 vector<Date>::iterator，而 C++ 語言並不允許你修改內建型別的暫時物件。例如以下程式碼並不合法：

```
Date* f();  // 函式傳回一個 Date*
p = --f();  // 錯誤。但如果寫爲 "f() - 1" 就可以。
```

幸運的是，我們知道 vector<Date>::iterator 是個隨機存取（random access）的 iterator，所以修改成這樣並不會損失效率：

```
    e.insert( e.end() - 1, TodaysDate() );
```

錯誤之二：如果 e 是空的，任何人企圖取得「e.end() 之前一個位置的 iterator」，不論是寫成 --e.end() 或是寫成 e.end()-1，都不是有效的 iterator。

```
    copy( first,
          last,
          ostream_iterator( cout, "\n" ) );
}
```

錯誤：first 和 last 可能不是有效的 iterators。

vector 的成長係呈塊狀（chunks）成長，所以它不需要在你每次安插新元素時重新配置緩衝區。然而有時候 vector 呈滿載狀態，這時候再加入新元素進去，便會觸發記憶體重新配置。

本例經過 `e.insert()` 之後，vector 可能成長也可能沒有成長，意味其記憶體可能移動也可能沒有移動。由於這種不確定性，我們必須視現有的任何 iterators 都不再有效。本例之中如果記憶體真的移動了，那麼 `copy()` 會再次產生出難以診斷的 core dump。

設計準則：絕對不要提領（dereference）一個無效的 iterator。

摘要：使用 iterators 時，務必清楚以下四點：

1. 有效的數值：這個 iterator 可以提領嗎？如果你寫 *e.end()，絕對是個錯誤。
2. 有效的壽命：這個 iterator 被使用時還有效嗎？或是它已經因爲某些操作而變得無效了。
3. 有效的範圍：一對 iterators 是否組成一個有效範圍？是否 first 真的在 last 之前（或相等）？是否兩者指向同一個 container？
4. 不合法的操作行爲：程式碼是否企圖修改內建型別的暫時物件，像 --e.end() 這樣？（幸運的是編譯器通常能夠捕捉這種錯誤。至於「指向 class 型別」（而非內建型別）的 iterator，程式庫作者往往會允許這種事情發生，爲的是語法層面上的方便性）