

侯捷觀點

上窮碧落下黃泉 源碼追蹤經驗談

剖析名家源碼，是讓自己技術躍升的捷徑。但是大系統的源碼非常龐大（Unix, Linux, Java, STL, MFC, VCL, Qt...），閱讀要有閱讀的方法。本文從動機、對象、前提、書籍、態度、工具、方法、瓶頸、價值、附加價值等方向加以討論。

讀者意見

讀者對我的印象，可能很大一部份是我寫了本《深入淺出 MFC》，剖析 MFC 的運作機制並簡化模擬了它們。的確，剖析與說理是我擅長的兩個項目。偶然的情況下，我起了經驗傳承的念頭，想寫這篇文章，於是在侯捷網站上貼出公告，藉此獲知讀者希望看到什麼。下面是幾封帶有具體提議的來信（節錄）以及我的簡單回覆：

- 能不能寫出一系列文章？我相信您有非常非常多和好的經驗（教訓☺），若只有一篇短文，則不解渴。
- 侯捷：我只打算談原則、談基本方法，這不需要一系列文章。
- (1). 如何開頭？從最開始的 .h 文件讀起？(2) 怎麼做前後連接工作？原碼中前後關係不少，你是怎麼做這些工作的？做堆成山的筆記嗎？(3) 如果你自己對程序中所用的算法（algorithms）並不熟悉，怎麼突破這個障礙？
- 侯捷：筆記一定要做，稍後詳述。演算法若不熟悉，比較困難，唯一的辦法是一一拆解，發揮想像力，設法搞懂它。搞不懂，就是遇上了瓶頸。
- 我也對如何閱讀大系統的源碼非常感興趣，如 Linux, GCC 以及一些開放源碼的系統，所以弟便抓了一套 RedHat 開發的可以閱讀大系統源碼的軟體叫做 Source Navigator，可是弟並沒耐性看完這套軟體的 manual，抓回來之後就丟

侯捷觀點

在一邊，沒有時間鑽研如何用它來看大系統的源碼（準備高考當中）。…希望能夠看到您出書寫到閱讀源碼相關的心得…

- 侯捷：有工具幫助是很好，沒工具也有沒工具的作法。
- 名家的代碼，簡練、優雅、富有彈性，對它進行剖析確實能夠增長自己的見識，提昇自己的功力，對於自己的實際工作起到很大的幫助。我覺得應該不僅僅停留在對這種已經成形的設計結果（源碼本身）的讚嘆，更應該探討這種優美設計的進化歷程，因為這些優美的源碼肯定是經過很多次的 **refactoring** 得到的。如果僅僅剖析已經存在的優美設計，很容易掩蓋怎樣進行設計的正确道路，可能造成一味的“模仿”，過度的 **up front design**。
- 侯捷：如果能夠反推回當初設計的原貌和演化歷程，當然最好。一般人可能沒有這樣的功力或時間。我以書寫為目的，探究技術的同時，儘可能做到這一點。技術的來龍去脈，乃我所謂之技術本質。不過我的設想與實際行動和你上述所言或有差異。
- D. E. knuth 在《*The Art of Computer Programming*》提到閱讀優秀的源代碼一直是被計算機科學教育所忽略的一個重要方面，Richard Stevens 也在《*Unix advanced programming*》一書封底提到，他認為學習程序的最好方法就是讀程序和寫程序。再至 K&R 這樣的泰斗，也都在不同地方強調過閱讀程序對於學習程序設計的重要性。…我多麼希望先生能就這一主題推向縱深寫出一本這方面的專著來…
- 侯捷：我只談原則和基本方法，這不構成一本書的份量。
- 我瀏覽過 MFC, Linux 源代碼，深感剖析名家源碼的確是使自己技術躍升的捷徑，但此間有很大的困難，名家源代碼一般極其龐大，內容涉及廣，很難把握其實質。我認為先要對所研究的源碼有一個高層抽象的把握，就像軟件工程的需求分析一樣，然後再逐步進入，一層一層具體化。
- 侯捷：的確，一定要對研究對象先有高階理解，不能矇著頭就栽入。
- STL 中大量的變量聲明，感覺稀奇古怪、密密麻麻，閱讀實在成問題，請問有什麼好的辦法呢？
- 侯捷：可能你看的是 Visual C++ 的 STL PJ 版本，那是可讀性最差的版本。換讀 GCC 附帶的 STL SGI 版本會好很多。

個人經驗

寫這篇文章，我拿什麼證明我有足夠的經驗給你參考？是這樣，我個人曾經深入

侯捷觀點

追蹤剖析 MFC 和 STL 源碼，並據以寫出《深入淺出 MFC》和《STL 源碼剖析》兩本書。對於 Java 和 Qt 源碼也有一點涉獵。Windows 源碼雖然沒看過（除了「那個人」，誰看過了），但對 Windows 系統內部的資料結構（用以管理 memory, processes, modules, threads...）以及 kernel APIs，倒是有幾番深刻理解（歸功於三本書，稍後詳述），亦曾經深刻剖析過 MZ/NE/PE 可執行檔格式，寫過一些（自用的）分析工具。

動機

任何事情都講求動機。動機強則成功率高，動機弱則失敗率高。下面幾種情況是剖析源碼的可能動機：

1. 像侯捷一樣以書寫、教育為目的。剖析源碼可以帶給我許多技術養份，又補充我退離編程第一線後的實戰磨練。屬於工作的一部份，和興趣結合，又能養家餬口，成功率最高，文件成果最豐碩。
2. 需要對某些 open source 進行改寫以量身訂製專用版本。這種情況最常見於 Linux 和 GCC。完全是一種職場工作，壓力很大，動機超強，成功率高。但有時間壓力，很難完善其說明文件；往往在專案結束後一段時間內，對如此大規模源碼的精神和實際面掌握度漸次消退，最後煙消雲散，只留下一絲絲模糊概念。
3. 嚮往華山論劍高手招式，希望學習名家風範，對技術有強烈的追求欲望。完全不是工作，只是一種學習。積極進取的學生可能是這一類。自由的時間加上學習的天賦使命，使學生時代成為剖析名家源碼的最佳時期，但年輕時期就具備足夠基礎與心性的人不很多。
4. 工作之外偶而發心——畢竟探看核心企求醍醐灌頂是每個科技人心中蛰居的渴望。由於本職工作的壓力，這一類型通常難以持久。

大系統源碼都十分龐大，值得剖析（根據我的價值判斷）的最小規模，大約是 C++ Standard Library。說它最小，你不妨親自看看有多大（就在你的 C++ 編譯器的 "INCLUDE" 目錄中）。因此，沒有強烈動機和縝密而系統化的措施，很難獲得真正有用的成果。半途而廢太可惜了，所以要嘛就下決心做到相當程度，有具體成果才罷手，要嘛乾脆別動，把時間拿去看恐龍展、拿破崙展、三星堆文物展、羅浮名畫展，或是和異性朋友培養感情，更有價值。

侯捷觀點

對象

取得名家源碼的機會很多。open source 不必說了，網絡上可以自由下載；其他諸如 classes library, framework...，多半採取白盒策略，也對使用者開放源碼¹。許多網絡社群組織也大方開放他們的成果。琳琅滿目的貨架上，什麼才是值得一探的寶貝呢？剖析源碼，時間與精力的投注很大，如果抱持「放進籃裡都是菜」的心態，一旦遇人不淑損失可就大了（別說你有的是時間）。

我個人認為，只有價值被百分之百認定的大型卓越作品，才值得剖析它，從中吸取深層技術養份。一樣米還養百樣人，哪來被百分百認定的大型卓越作品？唔，我說的是被你百分百認定，不是被百分之百的人認定。至於你認定錯誤，所學非人，虛擲歲月，那是你眼力差，調查不足，怨不得人。

前提

剖析源碼，並非學習語言的好方法 — 雖然你或許可以學到很好的語言運用。剖析源碼，也不是初學 OO 的好路線 — 雖然你或許可以學到很好的 OO 概念和實作。要知道，你現在是單騎入風塵，飄飄無所依，迎面撲來的是成千上萬如蝗蟲如夏蚊的程式碼。剖析一樣東西，必須先對它有一定程度的了解。假設你想剖析 MFC，為什麼會有這樣的念頭？因為你想徹底了解並掌握 MFC 的運行，這種需求一定是因為你想以 MFC 為基礎開發應用程式。那麼，不先寫幾個 MFC 應用程式觸發一點感覺，不宜貿貿然進入叢林深處 — 那兒有很多沼澤和蚊蚋。



¹ 這裡所說的開放源碼，和一般所謂的 open source 不同。前者只是將程式以源碼型式釋出，讓使用者得以觀察研究，或修改後用於自家產品。後者允許使用者在某種授權（例如 GPL, General Public License）之下做任意用途。

相同道理，閱讀 Qt 源碼之前，請先寫點 Qt 程式；閱讀 STL 源碼之前，請先學會使用 STL 並對 GP/template 有相當認識；閱讀 Java 源碼之前，請先學會撰寫 Java 程式並對 OO 有相當體會；閱讀 Linux 源碼之前，請先在 Linux 系統中玩一陣子並對記憶體管理、分離位址空間、檔案管理、驅動程式等系統知識有點準備。閱讀任何視窗系統上的任何 application framework，請先對該系統的訊息驅動機制做相當程度的了解。

書籍

永遠不要抱持「一切從輪子造起」的想法。捨棄別人的成果不用，走別人走過的路，犯別人犯過的錯，智者不為。

名家源碼剖析心得這一類書籍，屬於小眾市場，得遇一本應該感天謝地。你要的書到底存不存在，自己得仔細做點功課。當然，[書愈得好不好，也得你自己仔細做點功課](#)。www.amazon.com 是最好的書籍搜尋網站，打幾個關鍵字進去，用心鏈結瀏覽一下，花不了一個半天。MFC 方面，《MFC Internals》、《深入淺出 MFC》都是首選，STL 有《STL 源碼剖析》、《The C++ Standard Template Library》，Linux 方面可多了，蔚為大觀，絕對不愁找不到。想對 Windows 作業系統有深刻認識，應該看《Undocumented Windows》、《Windows Internals》、《Windows 95 System Programming SECRETs》²，印象中還有一本教你動手實現一個 Win32 作業系統的書。想看 GCC 源碼，應該先拿編譯器原理墊墊底，再找本《Building Your Own Compiler with C++》或《Crafting a Compiler with C》，能夠看看《How Debuggers Work》、《Linkers & Loaders》當然更好。

態世

真的，剖析源碼是一件大而艱鉅的工程。心理素質不好的人，不要嘗試。想像這樣的情境：「我走在廣袤的熱帶雨林中，濃密的樹冠連一絲陽光也透不進來。到處是黑漆漆的沼澤；蛇虺纏繞，蟲毒瘡癤。撲面而來盡是蚊蚋，群響如雷。碩大的

² 先前我曾說，我個人對 Windows 作業系統的內部結構有相當了解，便是得力於這三本書。非常非常棒的三本書——即使它們的出版年份分別是 1992,1993,1995，即使今天 Windows 作業系統已是 XP 當道。

蒼蠅毫無畏懼地在我臉上停留，舔舐我的臉孔並清理它們的腿毛。我想找一只魔戒，傳說中載上了它就擁有超人一等的力量，足以懾服眾生。但我不知道它在哪裡，連它的長像都不知道。每個疲憊不堪的夜晚，我夢見墜入暗無天日的泥淖，手忙腳亂地尋找一只針。呃，是的，一只鑲花針。極度疲倦中我入睡，極度無依中我醒來。日復一日。前面有三百六十五里路。每天都像行程伊始。聽說森林裡到處都是像我一樣的人……的冒號。」

語出何處？哦，是我的即興之作，博君一哂。沒有堅強信念，你走不出黑色森林。沒有適當的工具和方法，你也別想大海撈針。

工 具

下面是我用過的工具。由於我的最多經驗都在 MS Windows 環境下，剖析對象也都是 Windows 環境下的大型源碼，所以我所列的工具也就有某種侷限。然而任何人應該能夠從這裡面得到一些靈感。

1. grep

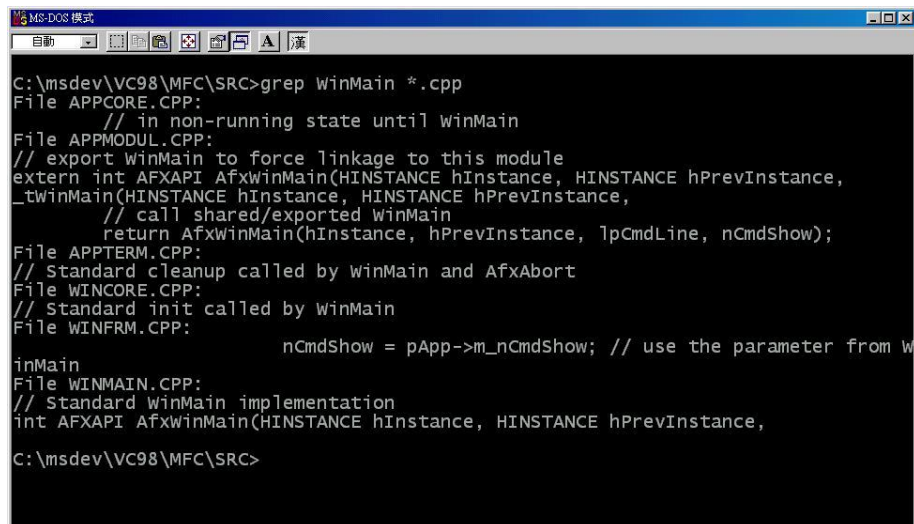
剖析 MFC 和 STL 源碼時，除了一般文字編輯器（我用老古董 PE2），我只使用一個工具：grep，這是源自 UNIX 的一個小小公用程式（utility），可以在一大堆檔案中找出某個字串的出現點。例如，我知道，任何 C/C++ Windows 程式不可能沒有 WinMain()，而 MFC 應用程式中沒有它的蹤影，因此我判斷一定被 MFC 包裝起來了。於是我想在茫茫大海中尋找 WinMain 落於何處，如圖 1。畫面第一行顯示我的動作是：

```
grep WinMain *.cpp
```

grep 為我找出 6 個出現有 WinMain 字樣的檔案，並列出每一個出現點的整行文字。從中我篩選出 APPMODUL.CPP 和 WINMAIN.CPP 做為下一個觀察目標。這樣我便有了很好的線索。如果希望搜尋目標擴及子目錄，grep 也辦得到，如圖 2，採用選項 “-d”。

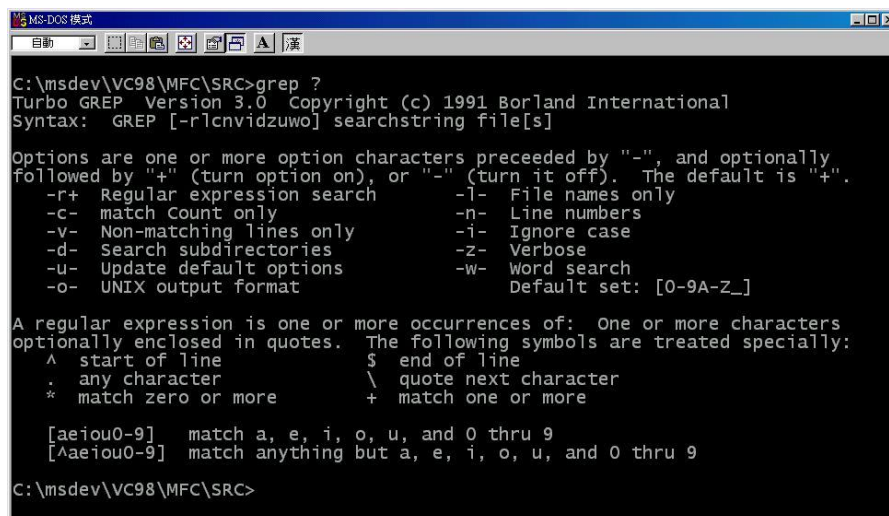
我手上這個 grep.exe 是 Borland 編譯器提供的版本。如果你沒有，可以到 <http://unxutils.sourceforge.net/> 下載一個同類工具（感謝 william 告訴我）。

侯捷觀點



```
MS-DOS 模式
C:\msdev\VC98\MFC\SRC>grep WinMain *.cpp
File APPCORE.CPP:
    // in non-running state until WinMain
File APPMODUL.CPP:
    // export WinMain to force linkage to this module
extern int AFXAPI AfxWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
    _twinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
    // call shared/exported WinMain
    return AfxWinMain(hInstance, hPrevInstance, lpCmdLine, nCmdShow);
File APPTERM.CPP:
    // Standard cleanup called by WinMain and AfxAbort
File WINCORE.CPP:
    // Standard init called by WinMain
File WINFRM.CPP:
    nCmdShow = pApp->m_nCmdShow; // use the parameter from WinMain
inMain
File WINMAIN.CPP:
    // Standard WinMain implementation
int AFXAPI AfxWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
C:\msdev\VC98\MFC\SRC>
```

圖 1. 以 grep 搜尋特定字串



```
MS-DOS 模式
C:\msdev\VC98\MFC\SRC>grep ?
Turbo GREP Version 3.0 Copyright (c) 1991 Borland International
Syntax: GREP [-rlcnvidzuw] searchstring file[s]

Options are one or more option characters preceeded by "-", and optionally
followed by "+" (turn option on), or "-" (turn it off). The default is "+".
-r+ Regular expression search      -l- File names only
-c- match Count only              -n- Line numbers
-v- Non-matching lines only       -i- Ignore case
-d- Search subdirectories            -z- Verbose
-u- Update default options        -w- Word search
-o- UNIX output format            Default set: [0-9A-Z_]

A regular expression is one or more occurrences of: One or more characters
optionally enclosed in quotes. The following symbols are treated specially:
^ start of line                      $ end of line
. any character                      \ quote next character
* match zero or more                 + match one or more

[aeiou0-9] match a, e, i, o, u, and 0 thru 9
[^aeiou0-9] match anything but a, e, i, o, u, and 0 thru 9
C:\msdev\VC98\MFC\SRC>
```

圖 2. 我手上的 grep 的全部功能選項。這是 Borland 編譯器提供的版本。

2. windiff

拿到 MFC7 源碼的那一天，我便立刻以 windiff 觀察兩個版本的差異。Windiff 是 VC 內附的小工具，方便觀察兩個檔案的差異，包括新增內容、刪減內容、修改內容等等，如圖 3。被觀察的兩個檔案內容置於同一個大視窗中，共同內容以白底黑字表現，紅色區域為第一檔案之獨特內容，黃色區域為第二檔案之獨特內容。左邊小視窗列出兩個檔案內容相異區域的映射圖，方便你掌握全局；藍點表示目前大視窗所觀察的區域在整個檔案的座落位置。其他標示及功能此處就不介紹了。利用這個工具，我輕易實證先前聽說的「MFC7 加強 Type-safe Message Maps」的實際作法：以相對安全的 `static_cast<>` 取代霸道的 C-style 強制轉型（如圖 3 所示）。

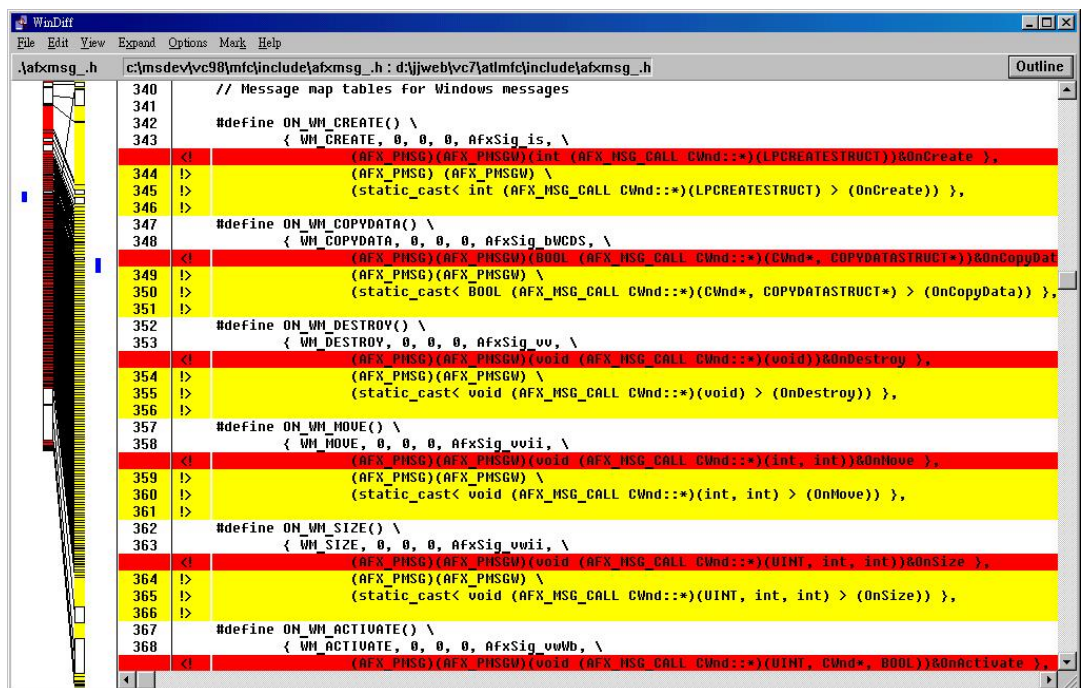


圖 3. windiff 可以讓使用者很方便地觀察兩個檔案之間的差異。

3. IDE debugger

做為技術文本書寫者，我一向不喜歡使用比讀者先進太多的工具。我喜歡鶴嘴鋤、丁字鎬、畚箕扁擔，因為我的讀者可能買不起昂貴的空壓機、怪手、樓蘭大吊車。如果我告訴讀者我以 BoundsChecker 或 SoftICE 觀察到某處有一塊記憶體洩漏，某處造成暫存器內容詭異，而我的讀者只能看著上述兩個高貴的名稱乾瞪眼，這有什麼意思？

不過，研究 MFC 而電腦內沒有安裝 VC++，實在是怪事一樁。VC++ 內建有除錯器，不好好利用就未免暴殄天物。剖析各種大型 libraries，你應該善用各種整合開發環境 (IDE) 中的除錯器，善用其 Breakpoint, Step Into, Step Over, Step Out 功能、善用其 CallStack 視窗和各種 Debug 視窗，如圖 4。這些功能讓你得以把程式的執行凍結放慢到一次一個指令，並在任何時刻觀察任何變數的現值及函式的呼叫順序。所謂「玩弄於股掌之間」差不多也就這樣子了。這些功能非 VC 獨有，每一種編譯器上的專業除錯器都有。

我模擬 MFC 做出 MFCLite3，撰寫過程中曾經遇到極隱微的臭蟲，如果不是除錯器的幫忙，協助我了解 MFCLite3 和 MFC 之間的差異，單只使用鶴嘴鋤丁字鎬和畚箕扁擔，我不敢想像需要花費多少額外的時間和精力。

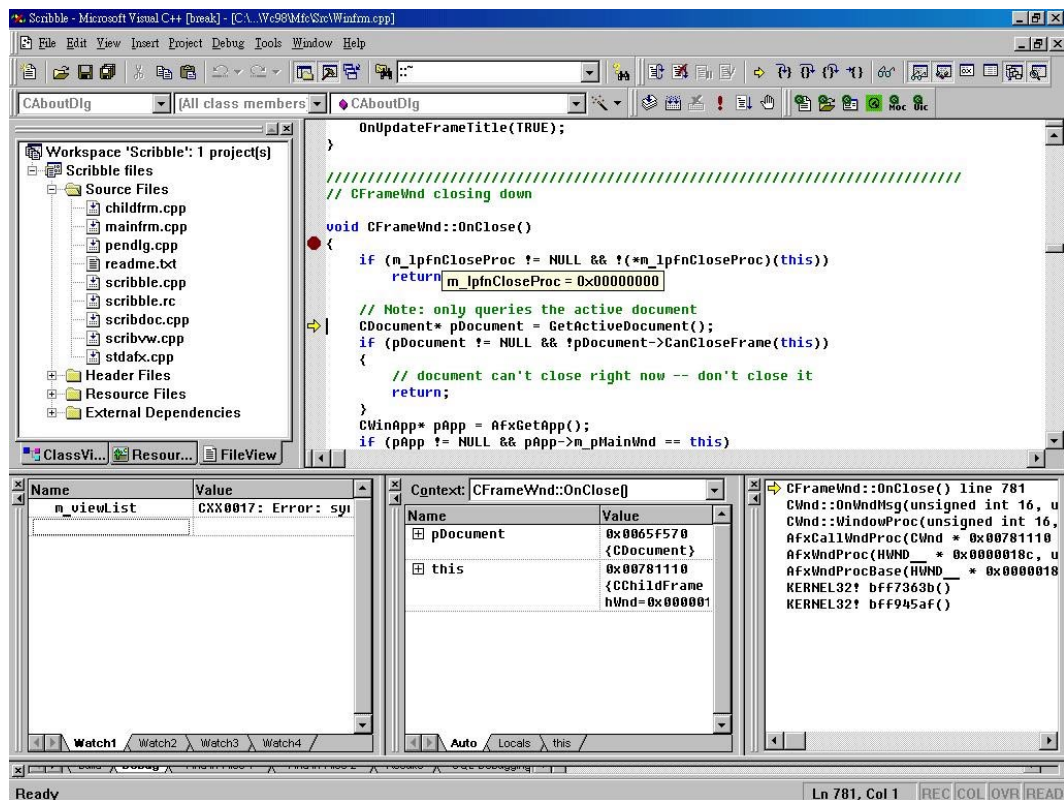


圖 4. VC 除錯器。左中視窗可觀察 classes, files, resources, 右中視窗可觀察源碼，右下角是 CallStack 視窗，左下角是 Watch 視窗（可觀察任何一個你設定的變數），下方正中央是目前執行脈絡下的區域變數視窗。將游標移至源碼視窗內的任何變數名稱上，其身旁便會出現現值，以小黃標籤框住。紅點表示中斷點，執行至中斷點後可選擇單步前進、進入函式、退出函式…等執行方式。每個視窗都可以隨意擺放，所以你的 VC 畫面可能和本圖不盡相同。所有這些功能並非 VC 獨有，每一種專業除錯器都有這些功能。

4. Spy++

觀察任何 Windows libraries，少不了需要 SPY++ 的幫忙。舉個例子，當我撰寫 MFCLite3 視窗/文件關閉系統時，我從 MFC 源碼中觀察到它處理了 WM_CLOSE 和 WM_DESTROY 和 WM_NCDESTROY，而我的 SDK 知識告訴我程式結束時還會發出 WM_QUIT。MFC 的某些處理方式（例如 MDI 視窗管理和 ::PostMessage() 同步行為）大大超出了 MFCLite3 的設定目標，因此我必須做些簡化，繞個彎在儘量逼

真的前提下模擬 MFC 行爲。首先我得確定視窗/文件關閉系統的所有相關訊息，這時候就用上了 SPY++。

SPY++ 是 VC 內附的一個工具，可以偵測四種東西：(1) Messages, (2) Windows, (3) Processes, (4) Threads，執行畫面如圖 5。

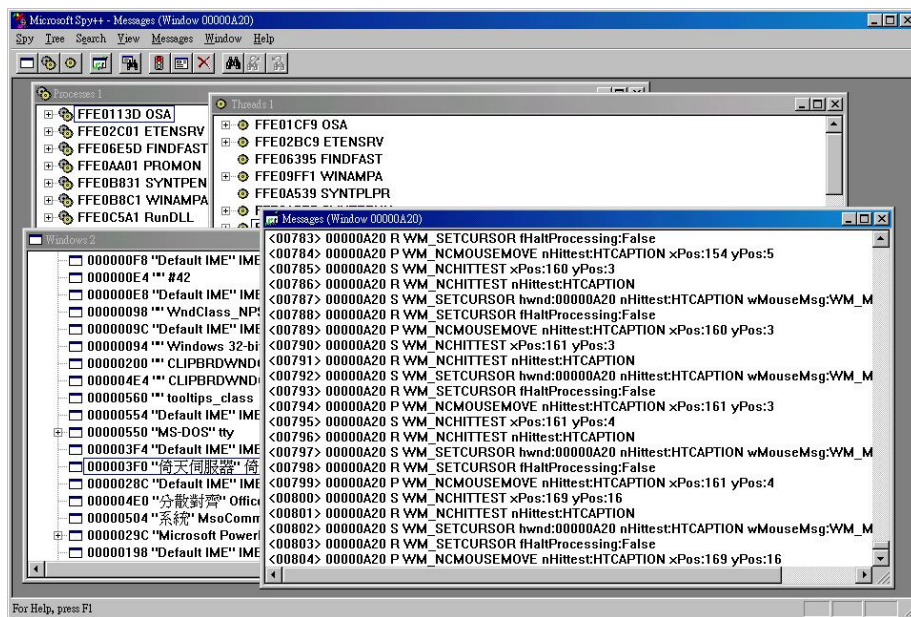


圖 5. SPY++ 執行畫面。四個子視窗分別展現 SPY++ 所能偵測的四種東西：(1) Messages, (2) Windows, (3) Processes, (4) Threads。

5. TDump

TDump 是 Borland 編譯器內附的一個工具，可用來觀察 MZ(DOS), NE(Win16), LE(VxD), PE(Win32), OMF(OBJ & .LIB) 等檔案格式，如圖 6。我在剖析 Windows 可執行檔格式時，曾大量倚重它來比對《Windows 95 System Programming SECRETS》第 8 章所示的資料結構。同類工具還有 VC 所附的 DUMPBIN.EXE 和 Matt Pietrek 所寫的 PEDUMP.EXE。

TDUMP 另有一個上述其他工具沒有的特性：可以傾印（dump）二進位檔內容。追蹤 MFC 源碼和撰寫 MFCLite3 時，我拿它來觀察文件格式，如圖 7。

```

MS-DOS - MORE
自動
Turbo Dump Version 5.0.16.12 Copyright (c) 1988, 2000 Inprise Corporation
Display of File SCRIBBLE.EXE

Old Executable Header
DOS File Size                A000h ( 40960. )
Load Image Size              450h ( 1104. )
Relocation Table entry count 0000h ( 0. )
Relocation Table address    0040h ( 64. )
Size of header record        0004h ( 4. )
Minimum Memory Requirement   0000h ( 0. )
Maximum Memory Requirement   FFFFh ( 65535. )
File load checksum           0000h ( 0. )
Overlay Number               0000h ( 0. )

Initial Stack Segment (SS:SP) 0000:00B8
Program Entry Point (CS:IP)  0000:0000

Portable Executable (PE) File
Header base: 000000D8
-- More --

```

圖 6 以 TDUMP 觀察 Win32 程式 (PE 格式)

```

MS-DOS
自動
D:\pic2\mfclite>tdump test.dat
Turbo Dump Version 3.1 Copyright (c) 1988, 1992 Borland International
Display of File TEST.DAT

000000: 08 00 FF FF 00 00 08 00 43 45 6C 6C 69 70 73 65 .....CEllipse
000010: 00 00 40 40 00 00 40 40 00 00 E0 40 00 00 A8 41 ..@...@...@...A
000020: FF FF 00 00 07 00 43 43 69 72 63 6C 65 00 00 A0 .....CCircle...
000030: 40 00 00 A0 40 00 00 E0 40 FF FF 00 00 09 00 43 @...@...@...C
000040: 54 72 69 61 6E 67 6C 65 00 00 00 00 00 00 00 00 Triangle.....
000050: 00 00 80 3F 00 00 00 00 00 00 00 00 00 00 80 3F ...?.....?
000060: FF FF 00 00 05 00 43 52 65 63 74 33 33 B3 40 9A .....CRect33.@.
000070: 99 D9 40 00 00 40 40 00 00 10 41 FF FF 00 00 07 ..@...@...A....
000080: 00 43 53 71 75 61 72 65 CD CC 4C 40 9A 99 89 40 .CSquare..L@...@
000090: 00 00 C0 40 FF FF 00 00 07 00 43 53 74 72 6F 6B ...@...@...CStrok
0000A0: 65 07 00 01 00 00 00 02 00 00 00 03 00 00 00 04 e.....
0000B0: 00 00 00 05 00 00 00 06 00 00 00 07 00 00 00 05 .....@...
0000C0: 80 00 00 00 00 00 00 00 00 00 00 60 40 00 00 00 .....@...
0000D0: 00 00 00 00 00 9A 99 A9 40 07 80 66 66 1E 41 66 .....@...ff.Af
0000E0: 66 1E 41 9A 99 99 40 CD CC 1C 41 00 00 00 00 00 f.A...@...A....

D:\pic2\mfclite>_

```

圖 7. 以 TDUMP 傾印 (dump) 二進位檔案內容。

6. Source Navigator

本文一開始，讀者來函中曾經提到 Source Navigator。我沒有用過這個工具，所以上網下載了一份試試。這個工具相當龐大，我還沒有足夠的動機去研究它。不過

侯捷觀點

從其執行畫面（圖 8）及功能表單觀之，大概是用來追蹤分析 C++ class library。如果真是這樣，我想編譯器所附的除錯器可以完全取代之。

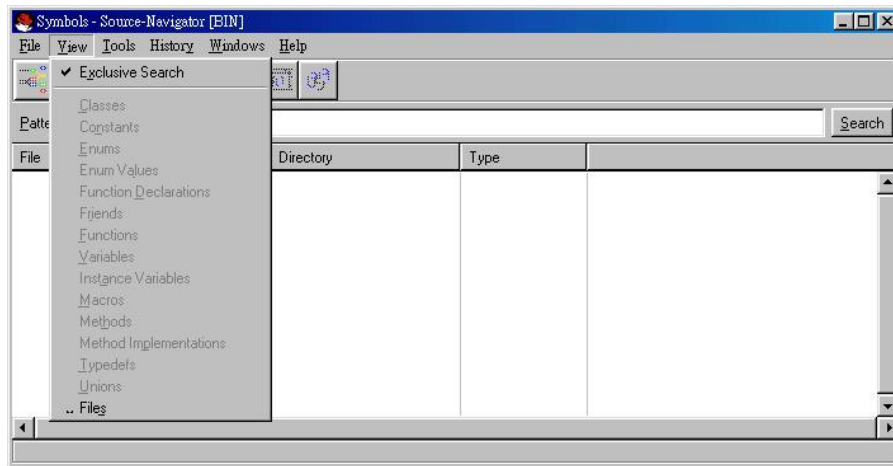


圖 8. Source Navigator 的執行畫面。

方法

萬事俱備，東風也有了，動手吧。首先你應該認識剖析對象的檔案組態。

檔案組態

到底你觀察的對象有哪些檔案，置於何處，首先你要掌握好。Java 程式源碼只有 .java 一種型態，C++ 程式源碼有實作檔（通常副檔名為 .CPP）和表頭檔（通常副檔名為 .H，或者無副檔名）兩種，分放不同的磁碟目錄內。

不同的系統，對於檔案命名肯定都有某種規則。閱讀源碼的過程中，對此必須留心記一記。以 MFC 為例，所有主軸核心類別的宣告放在 `afxwin.h`，訊息映射過程所需的訊息處理巨集定義於 `afxmsg.h`；`xxxCore.cpp` 內含類別核心定義，例如 `appcore.cpp` (`CWinApp`)，`wincore.cpp` (`CWnd`)，`doccore.cpp` (`CDocument`)，`viewcore.cpp` (`CView`)。`Winxxx.cpp` 代表 `CWnd` 衍生類別的定義，例如 `winfrm.cpp` (`CFrameWnd`)，`winmdi.cpp` (`CMDIFrameWnd`, `CMDIChildWnd`) ...。`DocXxx.cpp` 表示 document 相

侯捷觀點

關類別，例如 `DocTempl.cpp` (`CDocTemplate`)，`DocMgr.cpp` (`CDocManager`)，`DocSingl.cpp` (`CSingleDocTemplate`)，`DocMulti.cpp` (`CMultiDocTemplate`)。這些檔名或許一回生二回熟，但最好你能夠做做筆記，用點心思強記下來。掌握檔案的命名哲學，對你順利追蹤源碼很有幫助。

閱讀源碼的過程中會湧現大量的變數名稱、函式名稱。它們也都有某種命名規則。這個也必須用點心思歸類整理記錄下來。例如 MFC classes 的成員函式中有 On 開頭、Do 開頭、Pre 開頭、Post 開頭、Get 開頭、Set 開頭、Open 開頭、Load 開頭、Create 開頭…等各種名稱，掌握它們的命名規則能使你閱讀時印象加深，事半功倍。

爲了熟悉檔案組態，也爲了方便觀察，請熟用任何一個令你舒服的文字檔案快速瀏覽工具。圖 9 是我慣使的 `FileCtrl.com`，一個老掉牙的 DOS 小程式（你看它還是 .COM 呢），在 MS-DOS 視窗中跑得很好。只要以光棒選擇左視窗的檔案名稱，右視窗立刻顯現內容，反應極快，操作簡單，執行檔小到不行，才 8,466 bytes。把它放在 PATH 所指目錄中，便可以在任何時刻任何地點調用它。

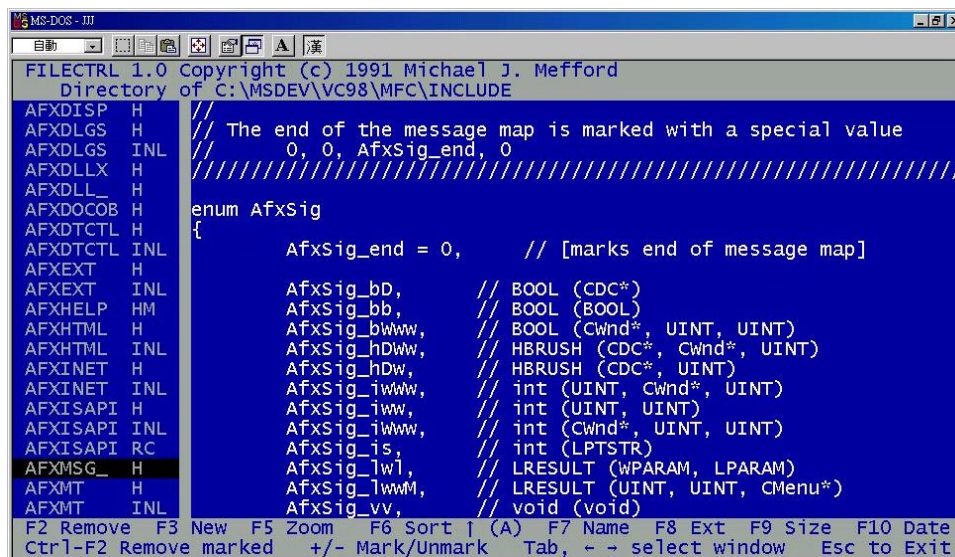


圖 9. FileCtrl，一個小工具，可方便而快速地觀察檔案內容。

線頭

剖析源碼，像玩拼圖遊戲。你一定先拼四個角落，是吧。線頭找到，抽絲剝繭就很容易。線頭在哪裡？考驗你的基礎知識。例如先前我所說，C/C++ Windows 程式必定以 `WinMain()` 為程式進入點（entry point），當我找到其源碼所在，循序追蹤，至少就可以挖掘出 application framework 的一條重要主幹（其他諸如 msg mapping, msg routing, document/view...還得另起爐灶）。再以 MFC msg mapping 為例，我從來不會在其他 C++ 程式中看過 `DECLARE_MESSAGE_MAP()` 和 `BEGIN_MESSAGE_MAP()`, `END_MESSAGE_MAP()` 這種東西，用 `grep` 工具一找，找出其源碼，發現都是 macros，於是我就老老實實把這些 macros 的定義代入隨便一個測試用的 class 內外（或利用 VC 編譯器選項 `-FI` 直接取得代入結果），老老實實觀察程式碼的變化，再把這些 macros 所構築出來的資料結構畫出，輕易就破解了法老王密碼。雖然隱微的 `AfxSig_xxx` 還有待理解（從歷史上看，羅塞達石碑也是很晚才發現◎），但我已能大略掌握整個設計精神。MFC 的三層基礎設施（Dynamic/DynCreate/Serial）也是這樣破解的。這些方法笨嗎？我做這些事情的時候（1994），世上沒有任何一本書一篇文章能夠引導我，我的辦法是唯一的辦法，不笨。

諸如 MFC 這樣的框架系統，組織龐大線頭紛歧。STL 就單純許多。STL 有六大組件，你可以任選一種開始。一般人應該會從容器開始，尤其是最簡單的 `vector`，畢竟它只是動態 array，而 array 是大家耳熟能詳的結構。但是當你進入 `vector` 的源碼一看，乖乖，記憶體動態配置是以 STL allocator 為之，與 STL algorithm 之間的橋樑則是透過 STL iterators。這時候你可以選擇先跳開研究後兩者，或是抱持「反正是那麼一種東西，有那麼一種功能」的心情，先解決 `vector` 再說。在此我可以告訴各位，破解 STL 實作奧秘的最大關鍵在 iterator traits 身上，因為它不但觀念新穎，實作手法也新穎（對大部份 C++ 程式員而言）。另一關鍵是 function adaptors，同樣因為觀念新穎，實作手法新穎。至於 containers 和 algorithms，教科書上都找得到它們的詳盡說明，狠狠給它流點汗，不可能沒有收穫。

面對作業系統，線頭又在哪裡？經驗告訴我應該在資料結構；可執行檔格式尤其關鍵。連 Windows 動態聯結的奧秘都藏在可執行檔格式中呢（見 PE 格式中

的 .edata 和 .idata 兩個 sections；'e' 代表 export，'i' 代表 import）。如果你手上有源碼，那麼系統的資料結構的呈現很具體，明明白白就寫在表頭檔內；（廣義的）演算法比較沒那麼實象，萬一缺乏良好註解，你只得一步一步追蹤推演。然而，（廣義的）演算法離不開資料結構，掌握了資料結構，你就有所依恃。稍後「瓶頸」一段我另有說明。

除了以上所說，另有一些難以言傳的東西，一問一答的方式或許更能傳承。面對陌生架構，不同的人有不同的組織手法和觀察焦點，一開始跌跌撞撞都是難免。大勢逐漸明朗後，兩岸猿聲啼不住，輕舟已過萬重山。

筆記

大系統源碼都很龐大很複雜。如果你以為你可以像看電視連續劇一樣地每天邊吸咖啡邊搖頭晃腦地輕鬆自在看看，一旁音樂零嘴侍候，時而還要應付小傢伙的搗蛋或大傢伙的嘮叨，還可以一心二用做點旁務，我告訴你，別作夢了。面對這一大坨代碼，不論時刻長短，你必須戰戰兢兢心無旁騖，像準備大學聯考一樣專心；靈光乍現、心得偶拾之際，立刻做筆記。

筆記做在電腦上最好。

請熟用一個文字輸入工具，一個繪圖工具。請加快你的打字速度。多快？不會影響你的書寫速度就行。圖 10 是我追蹤 MFC 視窗/文件關閉系統的過程中，以 PowerPoint 畫下的圖，這樣的圖我在追蹤過程中產出不下數百張。圖 11 是我追蹤 STL RB-tree（紅黑樹）的過程中，以 PowerPoint 畫下的圖，這樣的圖也不下百張。有了它們，配合少量文字，我可以自信滿滿地說，20 年內，任何時候你問我關於這個系統，我可以複習一個小時後便回答得頭頭是道。超過 20 年我滿 60 歲，萬一得阿茲海默症（老人癡呆啦）可就抱歉啦。

分析與記錄方式要規範 — OO 這東西老實說複雜得很。CASE tool（如 Rational Rose）很昂貴，我不能冀望買了「保時捷」才上路，但至少 UML（Unified Modeling Language）的各種 "diagrams" 要啃一啃，要學著用用。圖 12 是我剖析 MFC 及撰寫 MFCLite3 的過程中，手工繪製的 UML class diagram。嗯，富人有富人的辦法，

侯捷觀點

我的成果可用於書寫與出版，當然我寫字畫圖起來就格外帶勁兒。你的情況不同，不必像我一樣畫得那麼精美漂亮。我要強調的是，你得勤做筆記，份量要足，不能偷懶。曾經走過的路，再走一遍真令人不耐，曾經理解的知識，重新推演一遍真令人懊惱。

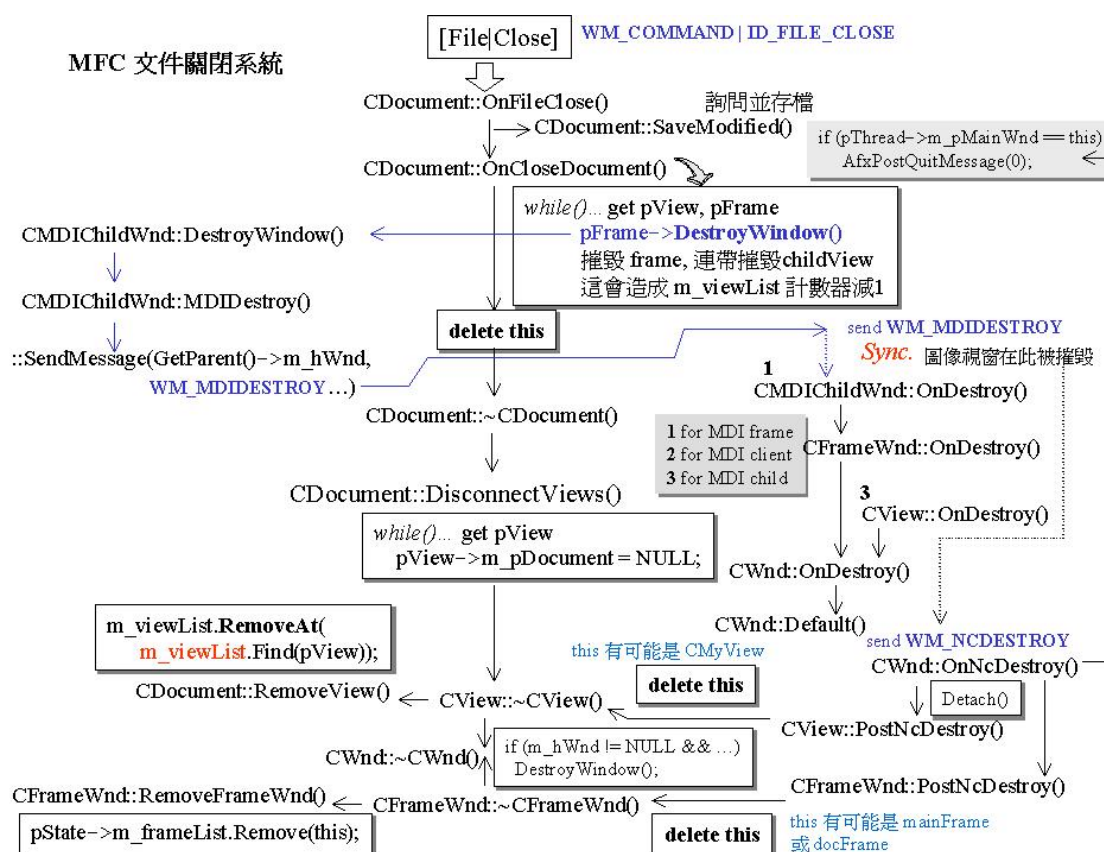
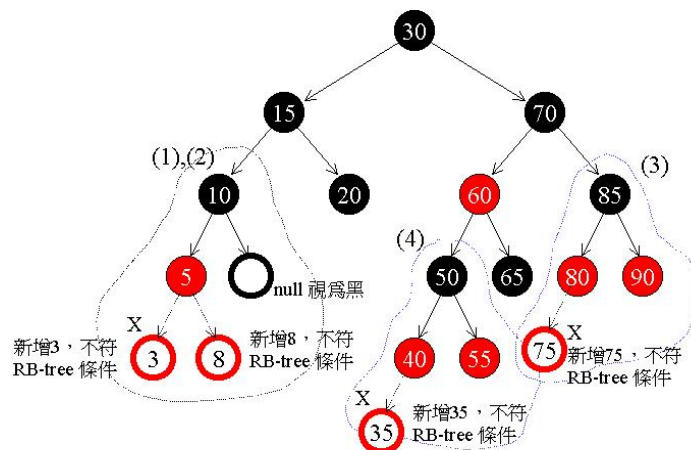


圖 10. 追蹤 MFC 的視窗-文件關閉系統時，我以 PowerPoint 畫下的圖。



新（葉）節點 X 必為紅，其父節點 P 若亦為紅（於是我們必須調整樹形），其祖父節點 G 必為黑，其伯父節點（父節點之兄弟節點）為 S。數種考量如下：

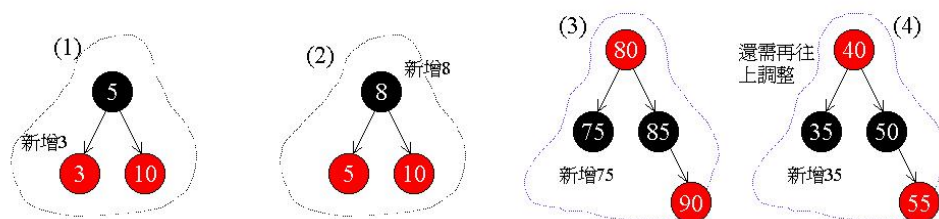


圖 11. 追蹤 STL RB-tree（紅黑樹）源碼時，我以 PowerPoint 畫下的圖。

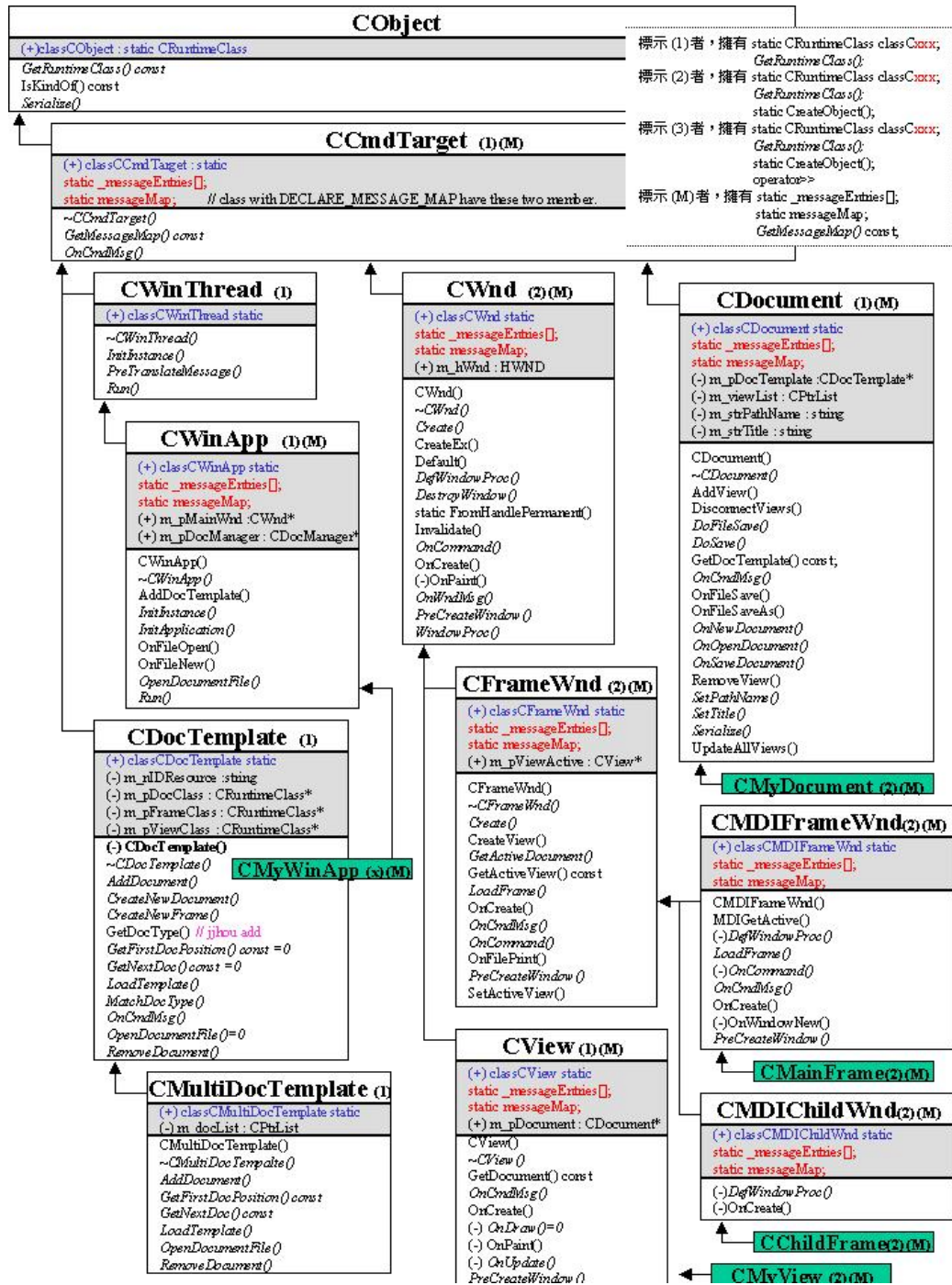


圖 12. 追蹤 MFC 及撰寫 MFCLite3 時，我以 PowerPoint 畫下的 UML class diagram。

侯捷觀點

Design Patterns

先有雞還是先有蛋？一，二，三，請回答。

答不出來是吧。

先有 design 還是先有 design patterns？

在自然演化的世界中，當然是先有 design 才有 patterns。後者是前者的淬鍊與分類。

但我們希望予程式員以訓練，讓他們在還未能完成那麼多設計之前，先獲得前人的加持灌頂。如果他們心中有了 design patterns，他們就可以在適當時機運用前人的經驗完成最理想（或足夠理想）的設計。

源碼追蹤和 design patterns 關係幾何？是否一定先要熟透那些名聞遐邇的 design patterns，追蹤與剖析才有依據？不，具備 design patterns 知識，你在分析源碼時接觸會更敏銳，文字說明或總結時可以更言簡意賅，但即使不知道 design patterns 也不會影響你的追蹤與學習。《深入淺出 MFC》2e,p82³最後一行說：『我要在這裡說明虛擬函式另一個極重要的行為模式』，p84 第二段第一行說：『這種行為模式非常頻繁地出現在 application framework 身上』。1996 年我寫下上述文字時，並不知道它就是如今大名鼎鼎的 **Template Method**⁴。但這不影響我的認識和我的體會。當然，如果當初我就讀過 GOF 的名著，可能對我的剖析和書寫更有幫助。

瓶頸

當你的知識水平和你所閱讀的對象差距太遠，你也只好暫時放下，補齊必要的基礎。舉個例子，當你追蹤 STL 的 allocator，研究它的記憶體配置策略時，如果不知道什麼是 memory pool，源碼又無法讓你參悟，你只好先去了解 memory pool 是

³ 這裡說的是繁體版頁次。簡體版出現於 p68 中央和 p69 最下。

⁴ 詳見《Design Patterns》by Gamma, etc. 1995, Addison Wesley。"Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithms without changing the algorithm's structure."

何方神聖。如果你不知道什麼是 Red Black tree，你也絕不可能剖析 STL 的 map 和 set 兩種容器，因為它們的底層機制都是 Red Black tree。不了解 Hash table？先去看看資料結構教科書；不解 QuickSort 和 Insertion Sort？先去看看演算法教科書。

沒有哪份名家源碼是易與之輩。它們都是大系統，包羅萬象。面對作業系統源碼或編譯器源碼，需要的基礎知識就更多更底層更艱澀了。中斷、迂迴、定點攻堅是你常遇到的情況和必要措施，頹喪和興奮是你情緒輪迴。每一項知識都有其基礎知識，每一項基礎知識又有其更基礎知識。一再地中斷、轉換、挫折，難以行雲流水，大概是源碼追蹤工程的最大失敗潛因。

先前讀者來信問到，如果對程式所用的演算法不熟悉，怎麼突破障礙？我必得告訴你，你只好以修復古蹟的態度，一磚一瓦重建整個脈絡。然而經驗告訴我，演算法和資料結構脫不了干係，把資料結構摸清楚，再耐心地步進追蹤，終有水落石出的一天。剖析 STL deque 時我有類似經驗。我對 deque 實作技術的唯一理解是，一個分段連續空間。Deque 的實作碼相較於其他序列式容器如 vector, list 龐大很多，但當我耐心地把 class deque 中的所有 data member 畫出來，如圖 13，再實際放些元素進去（特別注意邊界狀態），我就可以輕鬆觀察資料結構的內容變化。有了這些認識，再搭配 deque member functions 源碼，疑惑迎刃而解。

圖 14 是我的另一個經驗。我從 SGI STL allocator 的源碼變數名稱中隱約知道它實作有 memory pool，同樣地我把資料結構畫出來，塞幾個元素進去，觀察內容的變化和指標的移動，疑惑迎刃而解。

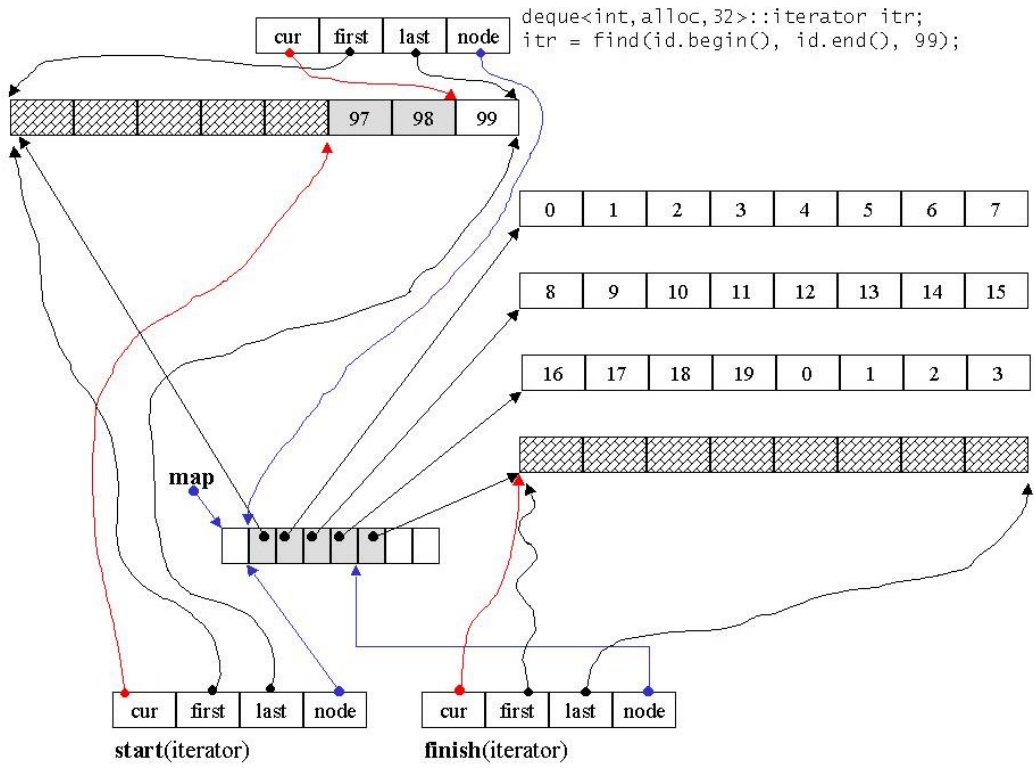


圖 13. SGI STL deque 的實作手法。

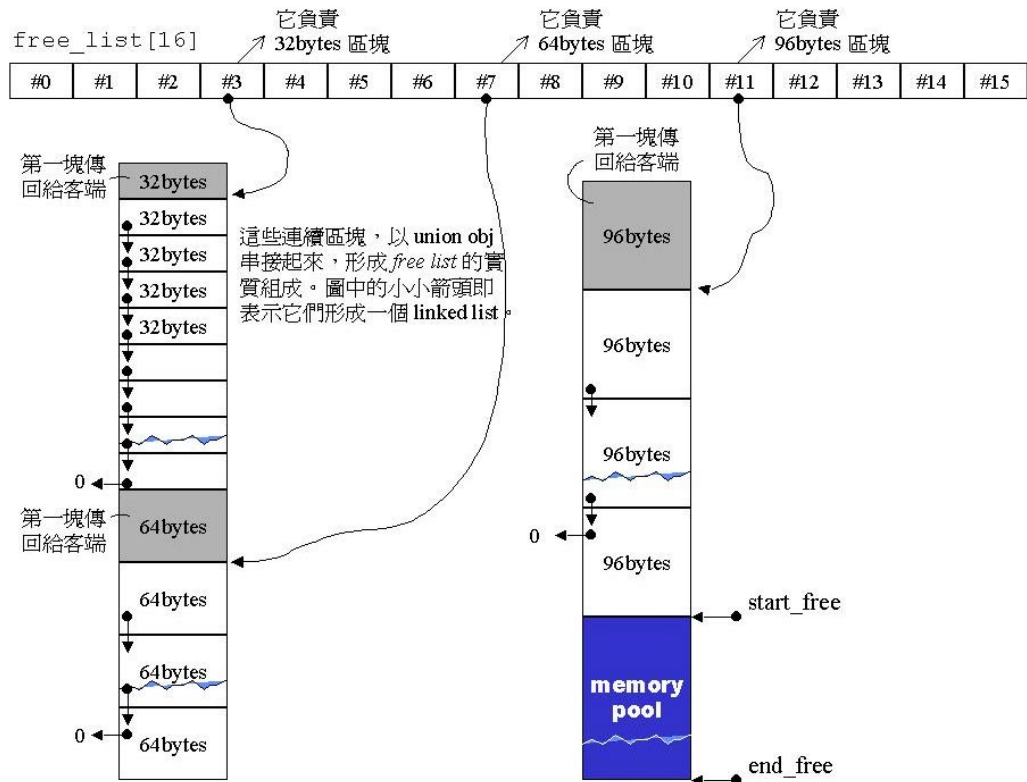


圖 14. SGI STL allocator 所實作的 16 個 memory pool，分別應付 8, 16, 24, 32, ...128 bytes 的小塊記憶體索求。

當然也有些情況非常複雜，不那麼容易對付。奉勸一句，不要硬鑽牛角尖！就算不是牛角尖，也不能硬鑽。不懂還是不懂，硬鑽也是不懂，那就放下吧（還能怎樣）。幸運的話，在偶然的時機裡，也許貴人相助，也許心有靈犀，也許觸類旁通，你就手到擒來得之不費功夫了。

我有一個切身實例。1997 年我完成《深入淺出 MFC》，其中第八章剖析 document 檔案結構，當時我已經搞清楚 **Serialization** 的來龍去脈，也可以解釋許多 document 的二進位內容，但對於為什麼有些 tag 是 8001，有些 tag 是 8003，我不了解。當時我認為我已經達到了我設定的目標，對於更進一步剖析已無興趣（沒興趣和遇上不易突破的障礙多少有點因果循環），而且我認為《深入淺出 MFC》的讀者最

侯捷觀點

終目標是要撰寫 MFC 應用程式，未能把屬於極內部機制的 tag 編碼 (encode) 方式搞清楚，無關宏旨 — 甚至連理解 document 存檔格式在我認為都已是「一窺天機了」。

一晃就是五年，直到最近我開始撰寫《多型與虛擬》2e 第六章的 MFCLite3 — 一個模擬 MFC 的輕量級文字模式 application framework。由於我對上述主題的認識只及某個層次，與真正的 MFC 還有段距離，造成 MFCLite3 在某種情況下出錯。甫自浙江大學電子系畢業的肖翔先生來信給了我一份錯誤報告（全文見侯捷網站「汗如雨下」），以下為來函摘要：

... psqr1 和 psqr2 指向同一對象，寫入文件時應該只有一份，但是在您的實現中卻寫了兩次！導致讀出時，psqr1 和 psqr2 指向了不同的對象。顯然這是不正確的。我覺得對於 C++ 對象持久性而言，最重要的問題：一個是如何保存相關的類信息，另一個就是如何解決上述問題！在您的兩本著作《多形與虛擬》、《深入淺出 MFC》中對前者都有很精闢的論述，唯獨後者一點也沒有提及，不能不說是一個很大的瑕疵。對於如何解決這個問題也不是很困難，只要先實現 CMapPtrToPtr 和 CPtrArray，在寫入時先查 map 如果已寫過，就只把輸出序號寫入文件，如果沒有就把對象的地址和輸出序號插入 map，再把數據寫入文件。讀出時，遇到第二種情況（即文件中有實際數據，而非只是序號），就先創建一個對象把數據讀出，接著再把新建對象的地址加到數組 array 尾端，遇到第一種情況，就以輸出序號為索引直接從數組中得到對象（由於寫入和讀出的順序一樣，僅用輸出序號就可以完全解決問題）。

※原函之大陸術語，對臺灣讀者十分陌生。以下修改為臺灣術語以利臺灣讀者閱讀。謹此。

... psqr1 和 psqr2 指向同一物件，寫入文件時應該只有一份，但是在您的實作中卻寫了兩次！導致讀出時，psqr1 和 psqr2 指向了不同的物件。顯然這是不正確的。我覺得對於 C++ 物件永續性而言，最重要的問題：一個是如何保存相關的類別資訊，另一個就是如何解決上述問題！在您的兩本著作《多形與虛擬》、《深入淺出 MFC》中對前者都有很精闢的論述，唯獨後者一點也沒有提及，不能不說是一個很大的瑕疵。對於如何解決這個問題也不是很困難，只要先實現 CMapPtrToPtr

侯捷觀點

和 `CPtrArray`，在寫入時先查 `map` 如果已寫過，就只把輸出序號寫入文件，如果沒有就把物件的地址和輸出序號插入 `map`，再把資料寫入文件。讀出時，遇到第二種情況（即文件中有實際資料，而非只是序號），就先產生一個物件把資料讀出，接著再把新建物件的地址加到陣列 `array` 尾端，遇到第一種情況，就以輸出序號為索引直接從陣列中得到物件（由於寫入和讀出的順序一樣，僅用輸出序號就可以完全解決問題）。

一看這幾句話，我就知道它的價值。高手過招需要真正發力嗎？比個招式就夠了。這些提示有如醍醐灌頂，我的興奮難以言傳。這些年來我對 STL 有了很多認識，所以我以 `std::map` 取代 `CMapPtrToPtr`，以 `std::vector` 取代 `CPtrArray`，快速而成功地模擬出完完整整的 MFC document 精確檔案格式。這是我寫作生涯以來和讀者互動的一個最精彩實例。

價值

源碼之前，了無秘密！

閱讀源碼，猶如私淑大師儀采，親炙大師風範。大師往前一站，淵停嶽峙，大師往後一退，瀟灑從容。誰不嚮往做大師人物？看多了大師身手，舉手投足自然也就有了樣子。

追蹤名家源碼，歷經震撼與洗禮，你將有如脫胎換骨。說白一點，個人談吐思想眼界的檔次都會高出不少，當然前提是你受教。

常有人詢問，編程需要天賦嗎？哦，任何事情走往極致，都需要天賦。任何一個軟體產品的極致成功，都需要創意天賦、編程天賦、管理天賦、行銷天賦…。然而，只需用心模仿，再加一點匠心獨具，任何人都能夠把編程路走得穩當順遂。能讀千賦則善賦，能觀千劍則曉劍，巧者不逆習者之門也。你把名家源碼融為己用，別人也會讚嘆一聲『你有編程天賦』。

我個人認為，剖析大系統源碼的最大價值不在於編程技術上的小枝小節，而在於宏觀視野與大格局的陶養。看過 MFC 源碼、STL 源碼、Windows 內核結構和 kernel

侯捷觀點

APIs 假碼(pseudo code)⁵，使我對於 Large Scale Object Oriented System、Application Framework、Generic Programming、Operating System kernel 成竹在胸，從容自在。我雖沒有開發類似產品⁶，但胸中丘壑已成，自有一番風景。

附加價值

計算機前輩大師們開放源碼，山高水長，典範長存。這些大系統源碼固然是寶，對一般人而言猶如天際明星，只能瞻仰。若有智慧言語，引領眾人認識這些寶貝，不啻亦如寶貝。

千辛萬苦窺探這些寶藏並獲得了具體成果，你會不會希望讓別人也分享你的成果和喜悅？懷寶迷世，聖人不許，相信 100% 的人都願意分享。把你的心得整理出版，立言立功，不但為後學鋪路，對自己也有省思反芻的技術效益和版稅的經濟價值。

不過，自己理解是一回事，讓別人理解又是一回事。思想是一回事，文字表達又是一回事。這正是為什麼得道者不乏其人，善書卻少得可憐的原因。要立言立功，首先，追蹤剖析的過程中筆記要記得勤、記得足。其次，繁複如斯的架構該如何起頭說明，起承轉合該如何設計，使讀者循序漸進而不至於愈來愈迷糊，有賴良好的組織能力。寫這樣一本書，規模、難度、人力時間的規劃，和做專案沒什麼兩樣，該有的準備一樣也不能少。至於以圖馭文，文圖並茂，那已是書寫功力了，不在討論之列，怕也準備不來。

無論如何，解脫之味不獨飲，開心之果不獨證，我鼓勵曾經用功並得到具體收穫的你，留下足跡，把心得寫成文字，化為圖形，以文章或書籍或其他任何型式，讓眾人分享你的成果。一人得道，雞犬升天，何樂如之。✍

⁵ Windows 源碼並未開放。這些都是 Andrew Schulman 和 Matt Pietrek 的勞動成果，載於《Undocumented Windows》、《Windows Internals》、《Windows 95 System Programming SECRETS》三本書中。我站在他們的肩膀上。

⁶ 我比較喜歡大刀闊斧修剪一番，寫些“lite”版本，如 MFCLite, STLLite，做為教育之用。