

簡單介紹一可操作的方法，濃縮到倆規則上去，「以執行流為線索」和「以數據流為線索」

一套代碼拿到手，

0、對總體功能有個瞭解（看文檔，或搜資料）

1、猜測其「初始化流程」，並驗證

2、猜測其「接口」（如linux kernel，所謂接口就是syscall、各種devfs、tmpfs、procfs等的read/write操作等），並驗證

3、猜測細節「子模塊的分解」方式，驗證，並「不斷細分子模塊」，瞭解熟悉其實現方式和運行原理

4、選擇感興趣的部分，追蹤主數據的改變路徑，瞭解涉及到的原理

其中0、1、2是基礎，即，探究細節之前，必須弄清整體、初始化、邊界接口。3、4可循環進行，不斷加深對代碼的理解

不難看到，其中1、2、3、4都需要結合「猜測」步驟進行，所謂猜測，即面對「若由我實現這個功能，我應如何作」的問題，給出簡要回答。顯然這個步驟極其依賴於經驗和積累。在經驗積累基礎知識等不足夠時，過程會很艱辛。解決方法也簡單，選擇簡單一點的project來閱讀，或者選擇某個具備嚮導性學習資料的project來閱讀。滿滿積累經驗

若不具備step 0的前提，即既無文檔，又無資料，那麼第一步就只能依賴於經驗和積累了。且無可適用於任何人的方法

=====

## 源碼閱讀

### 閱讀方法

一直以來零星閱讀源碼，沒有很好的方法。有一天終於找個一個不錯的方法。

把源碼版本庫放到github

master分支用於跟官方源碼同步gnuhub\_master用於註解源碼，源碼註解不要破壞裡面的內容，僅在裡面單獨加入註釋，這樣可以把master分支rebase到gnuhub\_master而不會有衝突，而且可以把所有的修改歷史記錄保存進來本地搭建jenkins 同步源碼自動rebase 使用java版的lrx工具opengrok快速搜索閱讀源碼，記錄閱讀歷史與步驟快速

註解源碼在vagrant虛擬機使用emacs+gdb不斷編譯調試源碼

從優秀開源軟件源碼中學習，在所閱讀的源碼中，找出可以把自己引向深入的東西，把其他的一切統統拋棄

讀懂是第一步，跨越語言障礙

領會大師的意境，跟隨大師學習,每天進步一點...

大師之路：

關注大師的言行，  
跟隨大師的舉動，  
和大師一併修行，  
領會大師的意境，  
成為真正的大師。

### 關注項目

php

ruby

m4

binutils

flex

bison

gcc

nginx

prototype

sysstat

git

pkg-config

strace

glib

=====

## 1. 工具篇

雖然有專門用來追蹤原始碼的工具，例如 [Source Insight](#)、[Source Navigator](#)，不過作者還是偏好數個小工具的組合。

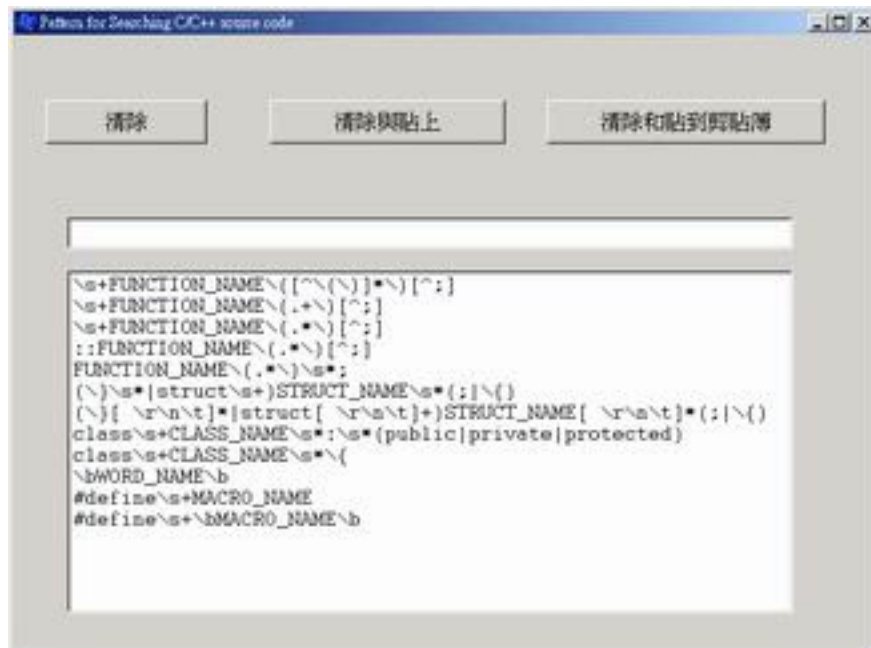
### SourceStyler:

這是用來排版程式碼的工具，研究別人的程式碼最怕碰到排版的亂七八糟的程式，同類型的軟體還有免費的 [GNU indent](#)。

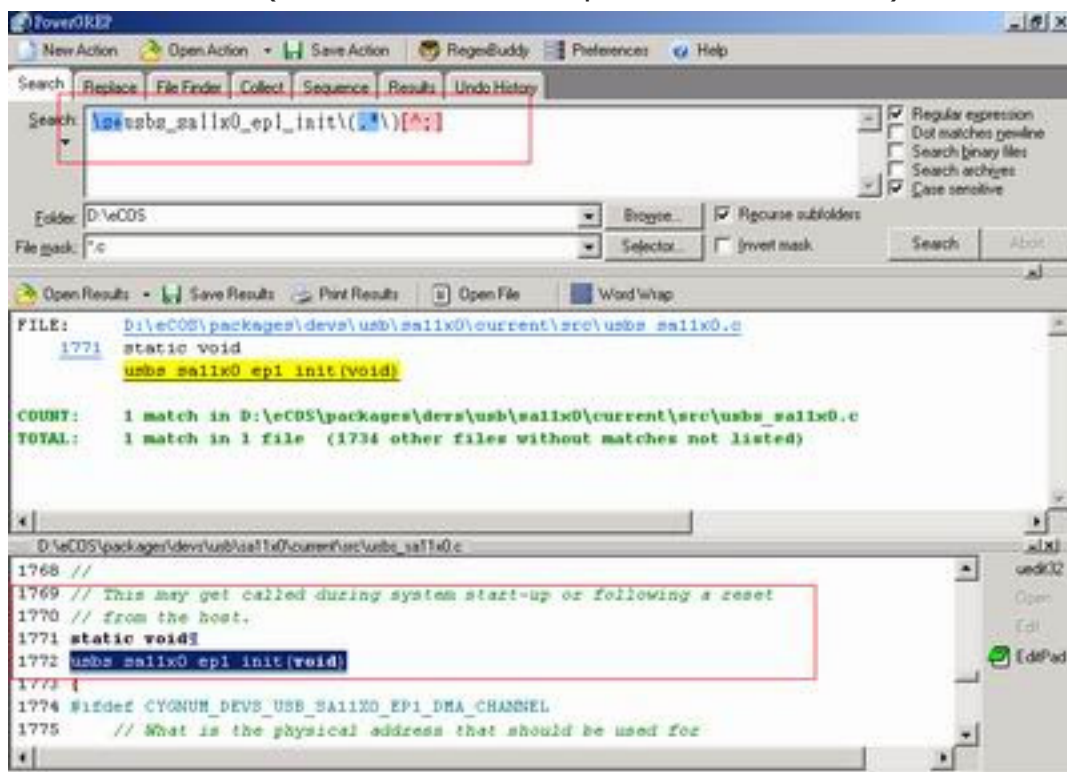
### PowerGREP:

研究別人的程式，不能不會 *Regular Expression*，regex 可以幫你快速正確的找到目標，但是 console mode 版的 grep 太難用了，PowerGREP 可以對一整個目錄(含子目錄)搜尋，並且列出符合對象的上下文，方便你快速判斷是否是你要找的東西，此外可以結合外部編輯器，例如筆者就結合 UltraEdit，可以打開編輯器直接跳到符合的那一行。

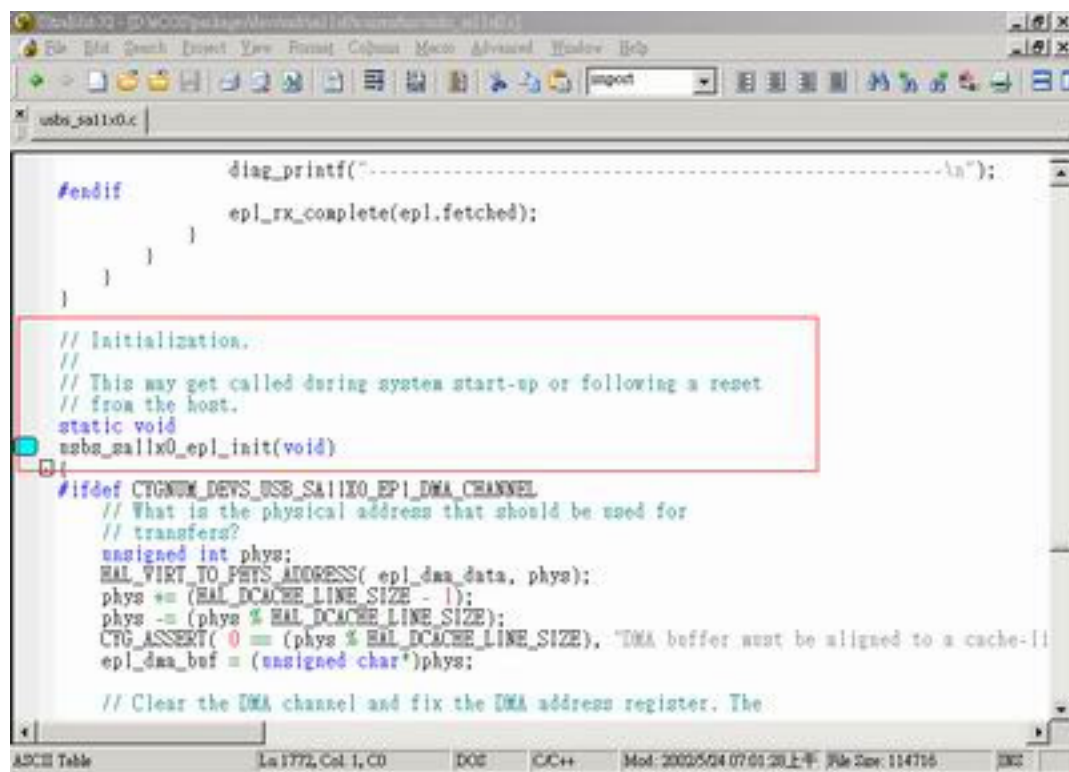
作者自製的小工具，可以用來快速產生 expression



貼到 PowerGREP(上面那隻程式會把 expression 貼到剪貼簿)



跳到指定的那一列，接著可以用 Ctrl + F2 標記起來，  
以後就可以用 F2 在這些標記間跳躍。



```
diag_printf("-----\n");
#endif
    epl_rx_complete(epl.fetched);
}
}
}

// Initialization.
//
// This may get called during system start-up or following a reset
// from the host.
static void
msbs_salix0_epl_init(void)
{
#ifdef CTGNUM_DEVS_USB_SALIX0_EPI_DMA_CHANNEL
    // What is the physical address that should be used for
    // transfers?
    unsigned int phys;
    HAL_VIRT_TO_PHYS_ADDRESS( epl_dma_data, phys);
    phys += (HAL_DCACHE_LINE_SIZE - 1);
    phys -= (phys % HAL_DCACHE_LINE_SIZE);
    CTG_ASSERT( 0 == (phys % HAL_DCACHE_LINE_SIZE), "DMA buffer must be aligned to a cache-lin
    epl_dma_buf = (unsigned char*)phys;

    // Clear the DMA channel and fix the DMA address register. The
```

## 2. 戰術篇

### 檔案格式:

假如該程式自訂檔案格式，分析檔案格式可以得到許多寶貴的資訊，例如程式中哪些部分負責讀取/寫入/產生檔案的內容，檔案中哪些內容對於程式是必須的，經由這個過程，可以很快瞭解部分程式的流程，這招對於繪圖類軟體尤其有效。

### Data Flow Diagram:

DFD 也是用來瞭解程式一個強而有力的工具，DFD 個人認為最大的用處就是用來畫出資料加工的流程，相信大家知道程式 = 演算法 + 資料結構，畫出 DFD 可以讓你知道資料結構是被程式的哪些部分新增/刪除/修改，很快就可以抓住程式的流程與精神！(感謝陳正哲先生提供此方法)

### UML:

UML 的圖形我只用到兩種，分別是：

### Sequence Diagram:

因為作者研究的是 multitasking system 的原始

碼，sequence diagram 可以畫出 task 是如何互動的，一旦瞭解 task 間的互動過程，很快就可以抓住整個系統的精神。

## **State Diagram:**

瞭解系統是因為哪些事件(event)而改變狀態(state)也是一件很重要的事，比方說按鍵就可以分成按下/放開/連續，系統會對此做出什麼改變？試著用 state diagram 塑模，可以瞭解系統是如何變化的。

## **Call Graph:**

這是針對細節部分的研究手法，建立模組的函式呼叫關係圖，可以知道哪隻函式是模組最常用到的(也可能是最重要的)，當然最大的收穫是瞭解模組的執行流程與其他模組的關連。

## **流程圖:**

這是大家都知道的圖示法，不過作者只在碰到組合語言時才應用此法，作者採取土法煉鋼的方式，第一次邊看組合語言邊旁邊寫上註解(C Pseudo Code)，然後依註解畫出流程圖，假如您有更好的作法，不妨分享一下，謝謝！

=====

覺得這文，非常不錯，應合了自己在讀一些別人的代碼和開源代碼的一些做法

1：從文件結構上得到一個大體的認識。

其實編程語言和IDE工具功能，框架都考慮了代碼的組織問題，基本有經驗的程序員在組織代碼和文件時，給的文件名都有功能的針對性，別人一看都知道的大概。

2：如果開始有條件，最好能瞭解一些大體的代碼流程，對於面向對像的C++/JAVA這樣的如幾個主要的類的關係，畫一畫關係圖，像C寫的，從main入口，走一個典型的業務流程。

3：深入代碼細節，自己的理解寫在代碼邊。

對代碼的框架大體瞭解後，拷貝一份代碼，深入代碼的底層，搞清楚基礎的類，過程的功能，我都在看過的代碼邊上加上自己理解的註釋，有時不對，當返回來再看就能明白了，所以加自己的註釋是很重要的，主要是代碼量大的時候，我再返回同一個地方，我已經不記得它是做什麼的了。

4：賴心。

有時代碼大的時候，真不是一天兩天能讀完的，所以加筆記，在代碼邊上加理解是很重要的。

我發現當你讀別人的代碼讀得越多的時候，就能更快速的理解別人的代碼，也能更好的分辨出為什麼哪些代碼寫得好，哪些寫得不好了。

=====

### 【譯】如何閱讀大型代碼庫

2013-11-24 13:20 181人閱讀 評論 (0) 收藏 舉報

程序員開發者文檔閱讀職業生涯

原始鏈接：[Reading Large Codebases](#)

我在「伯樂在線」上的譯文鏈接：<http://blog.jobbole.com/51973/>

譯文：

Casey問我：

*對於新手，有什麼有針對性的訣竅來閱讀大型代碼庫嗎？*

碰巧，我認為這是一個非常好的問題。我覺得想要成為一個好的開發者，閱讀代碼庫並弄清楚內部是怎麼回事的能力非常重要。在你的職業生涯中你會中途加入一個現有的項目並被要求迅速融入進去。或者，甚至更難，會有一個項目丟給你讓你自己一個人搞清楚。

最壞的情景就是你被帶入一個項目，要你替換掉讓工程運行失敗的「那些肆無忌憚的\*雜種」，並且讓工程運行起來。不過更常見的情景是你被要求維護一個已經離職的員工寫的代碼庫。最後，當然，如果你用了任何開源的項目，很大的可能是被要求「你可以擴展它讓它也能做這個功能嗎？」亦或者你只是好奇。

尤其是新手程序員，我強烈建議閱讀代碼庫，看看以下我是怎麼做的，然後你需要實際的去閱讀代碼。

當我接觸到新的代碼庫時，我常常忽略文檔和表面的細節。目的是摒棄先入為主的關於它怎麼運行的想法。我試圖從文件結構上找出項目的結構。僅僅這個就能告訴你很多，我常常試圖找出它的結構。這是整個系統的核心嗎？它是怎麼分割的？等等。

之後我會找到最底層的代碼然後開始閱讀。我常常用字典序來讀。找到一個文件，讀完它，然後讀下一個文件。我儘量記錄下來關於這些東西是如何連接在一起的（你可以在博客裡找到關於記筆記的例子），但我做的最多的找到對這個代碼的感覺。有很多代碼常常是項目風格的一部分，比如預處理檢查，日誌記錄，抓取錯誤等等。你可以先瞭解這部分內容，之後就可以忽略它們閱讀有趣的部分。

我通常不在某一點上閱讀太深，我會試圖宏觀的找到感覺。比如：這個文件通過調用Y和Z返回了X，但在這個點上閱讀每一個細節對我來講並不真的重要。哦對了，我還記錄筆



記，很多筆記。往往它們不是真的筆記而更像是問題清單，在這裡我理解的越多，加入的問題和寫入的回答也就更多。在閱讀完我能找到的最底層代碼之後，我會做一個縱向的比較。這是最讓我能弄清楚事情是如何佈局和工作的。這就意味著下一次我來看這部分的時候，對於代碼結構我會有更好的想法。

接下來，我會找有意思的部分。系統當中對我有意思的部分而不是被我束之高閣的部分。

這部分內容很多，但其實要做的並不多。我僅僅是通讀一遍代碼首先找到結構，之後我會認真研讀獨一無二的部分並找出他們是如何寫的。

在這期間，尤其是遇到難點的時候，我會試圖尋找任何文檔（只要有的話）。對於這一點，我應當首先知道代碼是如何組織的，這樣我才能更快的閱讀文檔。

\*原作者註：我一開始只寫了肆無忌憚，不過這樣更有趣。

=====

## 學會讀代碼

分類：[學習指導](#) 2013-09-30 10:21 377人閱讀 [評論 \(3\)](#) [收藏](#) [舉報](#)

作為研發人員，閱讀別人的代碼是一件經常要做的事情。在學習編程語言的時候，也需要通過閱讀代碼理解語法和語言機制。通過閱讀別人的代碼，學到自己編寫程序的本領，這也是一種極佳的學習方法。有很多公司給實習生的工作安排，就是閱讀代碼。通過讀代碼，可以掌握公司的業務，以及生產組織的方法。大學生常說缺乏實踐經驗，其實閱讀程序也是一種積累編程經驗的方法。

寫程序和寫文章可以放在一起類比。寫文章需要先從宏觀入手，構思文章的結構；程序的結構設計好了，程序也差不多能寫出來了。一篇好的文章需要各種句式和措辭的合理

組合；寫程序就是要熟練使用控制結構和語句。但凡文章寫得好的人，一定看過很多別人寫的文章；同樣的道理，多看別人的程序，用心地去看，也可以提高自己寫代碼的能力。

從20世紀90年代開始，開源運動為IT業界開創了一個嶄新的世界。開源運動奉獻出了一大批的優秀項目，為學習者在共享協議框架內免費使用。這為學習者提供了難得的直接觀摩成熟項目的機會。甚至於，大學生在校期間就有機會自由加入到開源社區，參與到項目的開發中去。每年的暑假，也有一些組織提供讓大學生短期參加開源項目的活動，使在校大學生獲得產業一線的指導，獲得在大企業實習的機會，當然，還有豐厚的報酬。

閱讀數萬行開源軟件項目的代碼需要不少高級技巧，對初學者有些為時過早，這個可以作為翅膀長硬後的選項。不積跬步，無以致千里，在學習程序設計之初，要意識到讀源代碼對技術成長的重要意義，嘗試從一些小段的代碼入手，在閱讀中找到有效的閱讀程序的方法。

初學者閱讀代碼的來源，可以是教科書上的例題，也可以是一些案例集中的程序庫，當然也可以是一些已經開發完成的小型項目。

閱讀代碼要採用先整體，後局部的方法。在閱讀時一頭就扎到細節中的做法不可取，容易只見樹木不見森林。先對整個程序有個總體的認識，將有助於細節的把握。很多的程序在開始時都會提供註釋，對整個程序進行說明，這也是編碼規範的要求。如果有註釋的話，可以先讀到程序的功能是什麼，以及程序中主要算法的思想，這些將有助於從整體上把握程序。要先從整體上了解其中各個子程序（函數）的大致功能及調用關係，然後再細讀其中的每一個函數。為找出各函數的調用關係，可以從程序執行的入口（對C、C++、Java而言，都是main()函數）開始讀，以此找出程序間相互調用、返回的路徑。每個函數是程序中相對獨立的模塊，要先看整體的控制結構，再到具體的語句；先看出解決問題的整體方案，再發現其中的技巧。這種閱讀程序的習慣，體現的也是自頂向下的思想，這和設計程序的方法是一致的，體現的就是一種專業的思維模式，在閱讀中逐步深化這種思

維方式。

程序的三種控制結構分別是順序、分支和循環。問初學者哪種結構最複雜時，往往回答是循環。的確，在學習程序設計中，用循環結構解決問題將不少人繞了個夠嗆。如果需要循環嵌套，那更是不得了的事。設計程序中要自頂向下地分析問題，閱讀程序時也是優先成塊的整體閱讀。從這種角度，先能看到由順序結構形成的大塊，然後再關注用邏輯更為複雜的分支和循環描述的細節，體現出來的是對順序結構的重視。從這個角度，簡單的順序結構竟包含著守拙中的極巧，體現的是“大道至簡”。

在編程時，在需要的地方寫上註釋是良好編碼風格的要求。在閱讀不熟悉的程序時，尤其是代碼量比較大的程序，及時、準確地加上你自己的註釋也是很重要的事。添加註釋不限方式，可以在打印稿或者書上寫註釋；如果閱讀的是電子版的代碼，直接在編輯器中寫上註釋。通過寫註釋，將別人的編程思想及時用自己的話寫下來，用“寫下來”的這個動作，與代碼產生交互，在大腦中留下較為深刻的印象。寫註釋時也體現先整體再細節的原則，優先對大段功能的註釋，然後追求對細節的註釋。閱讀代碼時難免會出現理解上的錯誤，及時地修改註釋，記錄下正確理解的歷程。寫註釋還能避免讀明白了卻後邊忘了前邊的現象，也可以避免不必要的重複閱讀，有助於保證學習效率。

同一段代碼可以有意識地對其反復閱讀，這有助於對代碼的理解。在第一次閱讀代碼的時候可以跳過非常多的一時不明白的代碼段，只寫一些簡單的註釋或提出問題。在以後的重複閱讀過程中，對代碼的理解會一次比一次更深刻，及時修改那些註釋錯誤的地方，並且補足上一次沒有理解的地方。對一段認為已經理解的代碼再重讀一次，往往還會發現以前沒有注意到的細節。

閱讀程序時。不管程序是否能夠運行，要學會“用人腦執行程序”，按照計算機執行程序的步驟去“執行”程序。大腦是CPU，拿張空白紙過來，當作內存，在程序運行過程中產生的各種數據的變化都寫到“內存”上去，手、眼、腦並用。這樣讀程序，是掌握程序中的

細節的方法，尤其是當程序中包含複雜的算法時，這更是一種必要且有效的方法。有些複雜的程序，涉及到函數參數傳遞、數組、指針，寫出內存中數據變化的過程，會讓程序在大腦中的執行可視化，“紙上談兵”中，將程序玩弄於股掌之中。

可以將要閱讀的代碼先運行起來，先對程序有感性認識，再設法對其進行理性分析。如果閱讀的程序本來就是可以執行的，這樣做很容易。而如果閱讀的程序只是一些獨立的函數，可以圍繞這些函數編些測試程序，調用要閱讀的函數。可以用單步跟踪的方法執行程序，可以理解函數的調用關係和執行流程，開啟觀察窗口可以獲得更多的細節，設置斷點提高跟踪的效率，讓計算機充分參與、輔理解程序。

=====

Robert C.Martin在《代碼整潔之道》中，強調「代碼被閱讀的時間，遠遠多於被修改的時間」。經過幾個項目的實踐之後，更加能夠體會其中五味。本文整理了一些目前常用的代碼閱讀方法，以供分享和討論。

## 項目文件樹視圖

一個實用項目，其項目文件成百上千，分佈在多層次的目錄中，包含編譯配置文件、圖片資源文件、運行參數文件、源碼文件、第三方支持文件等等。就源碼而言，又包含了公用頭文件、模塊頭文件、模塊實現文件等等。只靠缺省的資源管理器查看，不能很好地滿足要求。

通過以下方法，可以從整體有效閱讀理解項目文件。

1. 資源管理器。能夠按層次查看指定目錄的文件。

2. 目錄展開模式。所有文件平行列出，通過字母排序、模糊查找，能夠查看指定名字的文件。

3. 類展開模式。所有類平行列出，通過字符排序、模糊搜索，能夠查看指定類文件，以及類包含的接口。

4. 結合過濾器，能夠減少關注的文件數量。

## 文件結構視圖

瞭解整個項目，及模塊的關係後，隨之開始文件級別的理解。一般文件包含包含頭文件包含，類接口定義，類內部函數定義，類變量定義等幾部分。實踐時，根據功能，可能幾百行或幾千行。

通過以下方法，可以有效閱讀單個文件。

1. 文件結構樹。頭文件、變量、函數，按文件中的順序顯示，如果複雜代碼分區存放，有助於理解文件結構。

2. 排序顯示。頭文件、變量、函數全部顯示，按字符排序、按類型排序、模糊查找，能夠查看指定名字的符號。

## 函數視圖

接下來，進入了長期工作的具體地點。我們的編碼工作，一半以上是和函數在親密

接觸。對函數的閱讀方法，很大程度上影響著工作效率。

通過以下方法，可以有效閱讀函數。

1. 函數定義查看。閱讀函數時，遇到調用的函數，迅速查看其定義，及其上下文。
2. 函數調用關係。閱讀到一個函數，迅速找出有那些調用者，或調用了哪些函數；且可以向上、或向下調用層次查看。
3. 函數查找。本文件、指定目錄、整個項目的函數查找。調用位置統計視圖和詳細上下文的方便切換。下一個函數位置的快捷操作。

## 代碼分類視圖

終於，我們走入了實際的瑣碎代碼片段：關鍵字、全局變量、成員變量、參數變量、局部變量、宏、常量、註釋等等。我們透過上下文主窗口，窺視著代碼海洋的一丁點區域。

通過以下方法，可以有效觀察上下文主窗口內容。

1. 類型區分。通過顏色、字體、字號等，直觀區分不同類型的代碼。
2. 標籤。多個關聯的感興趣區域，通過標籤記住，然後切換到窗口。
3. 多窗口。多個關聯的感興趣區域，通過多個窗口保留上下文，能夠幫助閱讀。

4. 大屏幕顯示器。Mr. Bean說過：「大，就是好」。

瑣碎羅列了上述閱讀代碼的方法。目前，只使用notepad，vi來閱讀實用項目文件，已經不多了（臨時快捷查看單個文件除外）。大多時候，實用專業的代碼編輯工具，如SourceInsight，Eclipse，VS2010等等。這些工具支持了上述所有閱讀方法，還包含更多的閱讀方法：行數顯示，項目統計等等。善用這些方法，必須的。

工具和閱讀方法，只能在你痴心研究代碼時，感覺到她的實用；而不是能幫助你理解代碼。再多再好的工具和方法，也不能掩蓋代碼的臭味；通過多角度、深層次、立體化接觸臭味，可能麻木到習慣，可能噁心致死亡，可能堅持去重構。

生活沒有捷徑。

## 參考資料

1. 《代碼整潔之道》。

=====

linux內核源代碼目前的版本已經到了3.8，其源代碼行數已經過千萬了，規模之大，任何一個人想要全部看完，幾乎是不可能的事情。

linux內核源代碼大部分使用c語言編寫的，少部分是彙編語言。知道c語言是面向過程的，其重要的特點是完成一個功能需要調用一系列函數是實現。在linux內核這樣龐大的源碼中，初學者往往會沉沒在很多層次的函數調用海洋裡，A函數的實現需要B函數，B函

數的實現需要C函數。。。一步一步跟下去，當達到一定深度的時候，就會出現之間樹木不見森林的情況，看代碼的時候就會被複雜的函數調用層所嚇到。

可以發現，很多層次的調用，其實是形成了一顆函數調用樹，如果我們在閱讀源代碼的過程中，把這可函數調用樹能夠繪製出來，但看到某個具體函數的時候，我們可以迅速的在調用樹上看到它是那個分支，屬於那個分支，有了這樣一個函數調用樹，我們就能把握整個森林，而不會被細節淹沒。

如下圖所示，這是在閱讀linux0.11內核中關於加載根文件系統setup系統調用時繪製的函數調用樹。



圖1.安裝根文件系統函數調用樹

查看這個調用樹，可以清楚地看到調用層次，每一層的函數都調用了哪些主要的下一層函



數。要是在XMind中，還可以點擊話題(topic，圖中每個方框都是一個topic)下方的加號或減號，折疊或展開函數的下一層調用情況，這樣就可以以較粗的粒度查看整個函數的功能。通過XMind繪製函數調用樹，可以讓我們很容易地掌握函數調用情況，一邊閱讀代碼，一邊繪製，能夠幫助我們一層一層地看清整個實現思路。希望大家能喜歡這個方法。

=====

=====

## 代碼閱讀的必要性

閱讀別人的代碼作為研發人員是一件經常要做的事情。一個是學習新的編程語言的時候通過閱讀別人的代碼是個最佳的學習方法，另外是積累編程經驗。如果你有機會閱讀一些操作系統的代碼會幫助你理解一些基本的原理。更有就是在你作為一個質量確保人員或一個小領導的時候如果你要做白盒測試的時候沒有閱讀代碼的能力是不能完成相應的任務。最後一個就是如果你中途接手一個項目的時候或給一個項目做售後服務的時候是要有閱讀代碼的能力的。

## 收集所有可能收集的材料

閱讀代碼要做的第一件事情是收集所有和項目相關的資料。比如你要做一個項目的售後服務，那麼你首先要搞明白項目做什麼用的，那麼調研文件、概要設計文件、周詳設計文件、測試文件、使用手冊都是你要最先搞到手的。如果你是為了學習那麼儘量收集和你的學習有關的資料，比如你想學習linux的文件系統的代碼，那最佳要找到linux的使用手冊、及文件系統設計的方法、數據結構的說明。(這些資料在書店裡都能找到)。

### 材料的種類分為幾種類型

#### 1.基礎資料。

比如你閱讀turbo c2的原始碼你要有turbo c2的函數手冊，使用手冊等專業書籍，msc

6.0或java 的話不僅要有函數手冊，還要有類庫函數手冊。這些資料都是你的基礎資料。另外你要有一些關於uml的資料能作為查詢手冊也是個不錯的選擇

## 2.和程式相關的專業資料。

每一個程式都是和相關行業相關的。比如我閱讀過一個關於氣象分析方面的代碼，因為裡邊用到了一個複雜的數據轉換公式，所以不得不把自己的大學時候課本 找出來來複習一下高等數學的內容。如果你想閱讀linux的文件管理的代碼，那麼找一本講解linux文件系統的書對你的幫助會非常大。

## 3.相關項目的文件資料

這一部分的資料分為兩種，一個相關行業的資料，比如你要閱讀一個稅務系統的代碼那麼有一些財務/稅務系統的專業資料和國家的相關的法律、法規的資料是 必不可少的。此外就是關於這個項目的需求分析報告、概要設計報告、周詳設計報告，使用手冊、測試報告等，儘量多收集對你以後的代碼閱讀是非常重要的

## 知識準備

瞭解基礎知識，不要上來就閱讀代碼，打好基礎能做到事半功倍的效果

## 留備份,構造可運行的環境

代碼拿到手之後的第一件事情是先做備份，最佳是刻在一個光盤上，在代碼閱讀的時候一點不動代碼是非常困難的一件事情，特別是你要做一些修改性或增強性 維護的時候。而一旦做修改就可能發生問題，到時候要恢復是經常發生的事情，如果你不能非常好的使用

版本控制軟件那麼先留一個備份是個最起碼的需求了。

在做完備份之後最佳給自己構造一個可運行的環境，當然可能會非常麻煩，但可運行代碼和不可運行的代碼閱讀起來難度會差非常多的。所以多用一點時間搭建一個環境是非常值得的，而且我們閱讀代碼主要是為了修改其中的問題或做移植操作。不能運行的代碼除了能學到一些技術以外，用處有限。

## 找開始的地方

做什麼事情都要知道從那裡開始，讀程式也不例外。在c語言裡,首先要找到main()函數，然後逐層去閱讀，其他的程式無論是vb、delphi都要首先找到程式頭，否則你是非常難分析清晰程式的層次關係。

## 分層次閱讀

在閱讀代碼的時候不要一頭就紮下去，這樣往往容易只見樹木不見森林，閱讀代碼比較好的方法有一點像二叉樹的廣度優先的遍歷。在程式主體一般會比較簡單，調用的函數會比較少，根據函數的名字及層次關係一般能確定每一個函數的大致用途，將你的理解作為註解寫在這些函數的邊上。當然非常難一次就將全部註解都寫正確，有時候甚至可能是你猜測的結果，不過沒有關係這些註解在閱讀過程是不斷修正的，直到你全部理解了代碼為止。一般來說採用逐層閱讀的方法能是你系統的理解保持在一個正確的方向上。避免一下子扎入到細節的問題上。在分層次閱讀的時候要注意一個問題，就是將系統的函數和研發人員編寫代碼區分開。在c, c++ , java ,delphi中都有自己的系統函數，不要去閱讀這些系統函數，除非你要學習他們的編程方法，否則只會浪費你的時間。將系統函數表示出來，註明他們的作用即可，區分系統函數和自編函數有幾個方法，一個是系統函數的編程風格一般會比較好，而自編的函數的編程風格一般比較會比較差。從變量名、行之

間的縮進、注解等方面一般能分辨出來，另外一個是象ms c6++會在你編程的時候給你生成一大堆文件出來，其中有非常多文件是你用不到了，能根據文件名來區分一下時候是系統函數，最後如果你實在確定不了，那就用研發系統的幫助系統去查一下函數名，對一下參數等來確定即可。

## 寫注解

寫注解是在閱讀代碼中最重要的一個步驟，在我們閱讀的原始碼一般來說是我們不熟悉的系統，閱讀別人的代碼一般會有幾個問題，1搞明白別人的編程思想不是一件非常容易的事情，即使你知道這段程式的思路的時候也是相同。2閱讀代碼的時候代碼量一般會比較大，如果不及時寫注解往往會造成讀明白了後邊忘了前邊的現象。3閱讀代碼的時候難免會出現理解錯誤，如果沒有及時的寫注解非常難及時的發現這些錯誤。4不寫注解有時候你發生你非常難確定一個函數你時候閱讀過，他的功能是什麼，經常會發生重複閱讀、理解的現象。

好了，說一些寫注解的基本方法：1猜測的去寫，剛開始閱讀一個代碼的時候，你非常難一下子就確定所有的函數的功能，不妨採用採用猜測的方法去寫注解，根據函數的名字、位置寫一個大致的注解，當然一般會有錯誤，但你的注解實際是不僅調整的，直到最後你理解了全部代碼。2按功能去寫，別把注解寫成語法說明書，千萬別看到fopen就寫打開文件，看到fread就寫讀數據，這樣的注解一點用處都沒有，而應該寫在此處研發參數設置文件(\*\*\*\*.dat)讀出系統初始化參數。。。。，這樣才是有用的注解。3在寫注解的使用另外要注意的一個問題是分清晰系統自動生成的代碼和用戶自己研發的代碼，一般來說沒有必要寫系統自動生成的代碼。象delphi的代碼，我們往往要自己編寫一些自己的代碼段，還要對一些系統自動生成的代碼段進行修改，這些代碼在閱讀過程是要寫注解的，但有一些沒有修改過的自動生成的代碼就沒有必要寫注解了。4在主要代碼段要寫較為周詳的注解。有一些函數或類在程式中起關鍵的作用，那麼要寫比較周詳的注解。這樣對你理解代碼有非常大的幫助。5對你理解起來比較困難的地方要寫周詳的注解

，在這些地方往往會有一些編程的技巧。不理解這些編程技巧對 你以後的理解或移植會有問題。6寫中文註解。如果你的英文足夠的好，不用看這條了，但非常多的人英文實在不怎麼樣，那就寫中文註解吧，我們寫註解是為了加 快自己的理解速度。中文在大多數的時候比英文更適應中國人。和其寫一些誰也看不懂的英文註解還不如不寫。

## 重複閱讀

一次就能將所有的代碼都閱讀明白的人是沒有的。至少我還沒有遇見過。反覆的去閱讀同一段代碼有助於得代碼的理解。一般來說，在第一次閱讀代碼的時候你 能跳過非常多一時不明白的代碼段，只寫一些簡單的註解，在以後的重複閱讀過程用，你對代碼的理解會比上一次理解的更深刻，這樣你能修改那些註解錯誤的地方 和上一次沒有理解的對方。一般來說，對代碼閱讀3，4次基本能理解代碼的含義和作用。

## 運行並修改代碼

如果你的代碼是可運行的，那麼先讓他運行起來，用單步跟蹤的方法來閱讀代碼，會提高你的代碼速度。代碼通過看中間變量瞭解代碼的含義,而且對 以後的修改會提供非常大的幫助

用自己的代碼代替原有代碼，看效果，但在之前要保留原始碼

600行的一個函數，閱讀起來非常困難，編程的人不是個好的習慣。在閱讀這個代碼的時候將代碼進行修改，變成了14個函數。每一個大約是40-50 行左右。

=====

## 隨想錄 ( 怎麼閱讀代碼 )

作為程序員來說，看代碼是我們的必修課。記得以前《Linux內核修煉之道》的作者說過，學習linux最好的途徑就是看代碼。《深入淺出MFC》的作者侯捷也說過，代碼面前沒有秘密可言。但是，代碼真的是那麼容易讀的嗎？其實就我個人的經驗而言，代碼裡面有的不僅僅是知識，更多的是對業務的一種提煉和理解。換句話說，我們在代碼中看到的不僅僅是業務的處理，看得更多的可能是對異常和突發情況的處理。這在某種情況下很有可能左右我們對代碼的認識。不熟悉語言、不熟悉編譯環境、不熟悉業務、不熟悉測試場景、不熟悉別人的開發思路，這些都會成為我們學習的障礙。那麼，有什麼辦法可以提高自己看代碼的能力呢？

### ( 1 ) 熟悉編程語言

不同的項目有不同的編程語言，這一點在sourceforge、google code上面可以看得很清楚。現在開源的項目很多，小的代碼庫比如說有linux 0.11，前後大約有兩萬行左右，大的代碼就不勝枚舉了，什麼linux 3.0、open office、mplayer、gimp、android，基本上每一個項目的代碼行數都是在百萬行以上。所以，如果你想對開源項目有一個基本的了解，那麼你就需要對它的基本編程語言有一個透徹、詳盡的了解。很難想像一個對C語言、彙編語言都不是很了記得人怎麼能夠看懂linux kernel的代碼？當然了，這些還只是基礎。它是你進一步提高的必要條件，卻不是充分條件。

### ( 2 ) 學會使用開源項目的基本功能

很多朋友在接觸開源的時候都有一個很不好的習慣。那就是，不管是什麼項目的代碼，他都會不管三七二十一，下載下來立馬解壓，接著就看起來代碼來。至於說，這個代碼有那些功能，這個開源軟件是怎麼使用的，他全然不管。舉個例子來說，很多朋友都喜歡

notepad++這個工具，它也是一個很有名的開源項目。如果我們對notepad++的相關功能沒有一個清晰的認識，那麼我們就會把自己陷入到代碼當中去，沒有一個突破點。開源項目大多經過多年的發展，整個代碼量遠不是幾千行那麼簡單。所以說，如果剝開軟件的基本功能，只注重代碼本身，你學到的東西其實是很少的。

### **( 3 ) 學會編譯代碼**

編譯代碼是我們學習軟件的基本功課。在widnows上面，你要學會利用visual studio編譯軟件；在linux上面，你要學會gcc、ld、makefile等工具，同時還要學會自己搭建編譯環境，不同的編譯環境還要注意在版本上是否相互匹配。即使是Windows上面，由於版本的差別，有的軟件代碼只能使用高版本的visual studio才能編譯，有的軟件需要directx庫、ddk才能參與編譯。所以編譯代碼本身並不是那麼簡單的一件事情。就拿ucos嵌入式操作系統來說，相信很多人都看過它的代碼。但是我想問的是，真正把ucos編譯成可執行文件的朋友究竟有多少？

### **( 4 ) 學會看懂、修改代碼**

在代碼編譯之後，我們常常就可以得到執行文件了。但是，這些工作只是一些基本工作，我們還有很多其他的工作要做。比如說，你要知道這個軟件的入口點有哪些？實現了那些功能？基本機制是什麼？模塊和模塊之間究竟是怎麼銜接的？協議棧或者軟件是怎麼分層的？在真正看懂代碼之前，你需要

- a ) 分清代碼的結構，這些一般可以從readme文件可以看出來；
- b ) 弄清代碼本身有多少線程，每個線程的功能是什麼；
- c ) 分清數據和代碼的走向，理清楚代碼的執行流程；
- d ) 善於利用軟件的基本功能點來看代碼；
- e ) 編譯代碼的時候添加-g選項，這樣可以實現單步調試，查看每一步數據的變化情況；
- f ) 給原來的開源模塊添加新的功能，但是不能破壞原來的基本構架和機制，驗證和深化

自己的認識。

## **( 5 ) 學會仿真代碼、重構代碼**

如果我們看代碼的目的只是為了利用某一些lib函數的使用方法，那麼做到上面4點就已經很不錯了。但是，我們常常有更高的要求。通過學習，我們可以發現原來的開源軟件有哪些設計的閃光點？自己是不是可以對它的基本功能用相同的編程語言盡快仿真出來？甚至於，我們可以自己編寫一個類似的開源軟件，實現更多的功能，而在資源的利用和數據的訪問速度上有一個質的提高。等到我們可以仿真代碼、重構代碼的時候，那個時候我們才能說對這個代碼真正掌握了。

不同的人看代碼的目的是不一樣的。但是，既然我們看了這些代碼，那麼至少需要掌握一些東西、學習一些東西，否則那不是在浪費時間嗎？不管什麼方法，有幾個原則是我們必須牢記的，否則看代碼的效果就會大打折扣的，

- 1】代碼必須完全編譯出來，不能編譯的代碼基本是沒有什麼用的；
- 2】代碼必須單步調試，慢工才能出細活；
- 3】修改代碼必須建立在對軟件深刻理解的程度上，否則bug會越改越多的；
- 4】代碼是需要反復看、反復驗證的，通常每一次看都會有新的收穫；
- 5】看代碼的時候注意相互交流彼此的看法，這樣會受益良多。

=====



## Eric Lippert：閱讀代碼真的很難

編者按：原文作者Eric Lippert 是微軟一名資深軟件設計工程師，從1996年起一直在微軟開發部門任職，協助設計並實現VBScript、JScript、JScript .NET、Windows Script Host、Visual Studio Tools for Office 和C#。

Escalation的工程師JeremyK在他博客中問到：

你是怎麼教人們快速深入挖掘不熟悉的代碼（不是自己所寫的）？我學習如何編程的方法很傳統——自己動手編碼。但我現在很糾結：到底是集中精神閱讀源碼，還是自己編寫。對我而言，似乎唯一有效的方法就是自己寫過。

不是和Jeremy開玩笑，寫代碼的確沒有讀代碼難。

首先，我同意你的看法，幾乎很少有人能讀代碼但不會寫代碼。這不像自然書面語或口語，理解他人的意思並不需要去理解他們為什麼要那樣說。比如，如果我說：

“寫代碼有兩種方式：一種嚴格且詳細，另一種模糊且草率。前者生成簡潔分層的婚禮蛋糕，後者卻是意大利麵條。”

上面這句話產生一個平衡且幽默的效果，但即使聽眾和讀者不理解我使用“零照應”和“並列句”這樣的文字技巧，也會理解我要說的意思。但是說到代碼，既要從代碼本身中理解代碼作者的意圖，又要理解代碼產生的預計效果，這兩者都極為重要。



*Eric Lippert*

因此，我又回到那個問題了，有些人需要快速切入代碼，但不需要動手寫代碼，那我們如何編寫適合這些人的代碼？

下面是在編寫代碼時，盡力去做的事，目的就是使其他人能輕鬆閱讀：

**1.使代碼遵從工具。**Object Browsers和Intellisense雖然很好，但我告訴你，我是守舊派。如果找不到我想要的，我會不高興。什麼使得代碼成為可查詢的呢？

- 像“i”這樣的變量名不好。如果沒有明確的錯誤提示，你就無法輕易查找代碼。
- 避免使用是其他名字前綴的名字。比如，在代碼中有個“perfExecuteManifest”，如果再有一個“perfExecuteManifestInitialize”，這就會讓我抓狂，因為每次在源碼中查詢前者時，我不得不費力地過濾掉後者所有的實例。
- “臨時傳遞數據”（tramp data）應使用相同的名字。所謂“臨時傳遞數據”（tramp data），就是指那些傳遞給方法A的變量，還要傳給方法B的變量。這兩類變量實際上是相同的，所以如果它們有著相同的名字，則更好。
- 別用宏來重命名東西。如果有個方法叫get\_MousePosition，那別這樣

GETTER(MousePosition)來聲明該方法。因為我會找不到實際的方法名。

●Shadowing會引起很多問題，請不要用它。

**2.堅持使用一種命名模式。**如果你打算用匈牙利命名法，那就堅持並廣泛使用，否則將適得其反。使用匈牙利命名法來記錄數據，而不是存儲類型；記錄普遍事實，而不是臨時條件。

**3. 使用斷言來記錄先決條件 ( preconditions ) 和後置條件 ( postconditions ) 。**

**4.別縮寫英文單詞。**確切來說，別縮寫成稀奇古怪的形式。在腳本引擎中，有個變量名叫“NME”，這讓我非常抓狂！它應當叫“VariableName”。

**5. C語言標準運行時庫的設計不是很優秀。別去效仿它。**

**6.別寫“聰明”的代碼；**當代碼出現問題，維護代碼的程序員沒時間去理解你的聰慧。（編註：這條建議即為：編寫可維護的代碼，詳情可參見《[明星軟件工程師的10種特質](#)》中的第8點。）

**7.理解編程語言特性的設計初衷，使用這些特性去做它們適合完成的工作，而不是它們能做到的工作。**例如：別把異常當做一般的流控制機制來使用（即便你能做到），而應該用它們來報告錯誤。別強制把接口指針轉換成類指針，即便你知道這樣沒問題。

**8.按功能單元劃分源碼樹，而不是按組織結構。**比如：我目前所在團隊中，有2個根子目錄的名字是“Frameworks”和“Integration”，這是兩個團隊的名字。不巧的是，Frameworks團隊名下有一個叫“Adaptor”的子目錄，而“Adaptor”卻是Integration的子目錄，這就非常令人迷惑。同理，（如果）有著相同子目錄的不同的子樹，有些是客戶端的組件，有些是服務端的組件；有些是管理組件，有些是非管理組件；有些是處理型組件，有些是非處理型組件；有些是零售組件，有些內部測試工具。這就會亂七八糟的。

當然，我實際上根本沒有回答Jeremy的問題—— **如何調試不是我寫的代碼？**

這取決於我的目的。如果我只是因為一個bug，而深挖一段具體的代碼，我會在調試器中逐步跟蹤所有代碼，寫下我“走過”的調用分支，記錄下哪些方法是特定數據結構的“生產者”，哪些方法是“消費者”；我也會仔細盯著輸出窗口，查看出現的有用信息；還要打開異常捕捉器，因為異常通常是問題所在。設置斷點；我會記錄所有和我上面建議相反的地方，因為這些東西很可能

誤導了我。

如果我想在理解一段代碼後修改它，我通常是從代碼頭部開始，或者先查找公共方法。我要知道類是如何實現的，它是如何擴展的，它的作用，它是如何嵌入整個代碼中的？我會盡力理解這些東西後，才去了解這些特定部分（代碼）是如何實現的。這耗時雖更長些，但如果你準備改動複雜代碼，你應當那樣做。

=====

## 閱讀優秀代碼是提高開發人員修為的一種捷徑

編者按：原文作者Alan Skorkin是一名軟件開發人員，他在博客中分享對軟件開發相關的心得，其中有很多優秀的文章，本文是其中的另一篇。Alan認為：**閱讀優秀代碼是提高開發人員修為的一種捷徑**。以下是全文。

我突然想起來，很多程序員都討厭閱讀代碼。來吧，承認吧！每個人都喜歡編寫代碼，編代碼是件趣事。另一方面，閱讀代碼也不容易。不僅不容易（編註：參見《[微軟資深軟件工程師：閱讀代碼不容易](#)》），而且還非常枯燥，咱們要面對這一事實。任何不是你的代碼都不怎樣。（雖然我們沒有說出來，但我們都是這樣想的。）

即便是你自己幾個小時之前寫的代碼，也會看起來很爛。時間越久，看起來越爛。所以，為什麼你要浪費時間去看其他人的糟糕代碼，而你完全可以利用這段時間編寫你自己的優秀代碼。其實我們可以一試，幾個小時之後回頭再看，看看你的代碼是否還依舊優秀。如果你不能吸收前輩大師的經驗知識，那你永遠都無法成為一位大師。

成為大師的方法之一是，找到一位大師，讓其傾囊傳授其所知。有這種可能麼？當然了，有這可能，雖然機會不大，但你必須極其走運。不過你不必十分走運，因為我們幸運地處於這樣一個職業，一個充滿著大師知識和技能的職業，等待我們去汲取吸收，這些東西就在他們所編寫的代碼中。你要做的就是去閱讀代碼，當然了，這或許耗時不少，畢竟沒有人坐在那裡給你講解，但這種方法的成效還很高。打個比方，要想成為一名卓越的木匠，得觀察大量結構優良的家具。



我喜愛閱讀代碼，我的直覺告訴我，你也會從中獲益頗豐。雖然閱讀過程惱人並煩人，但其回報是非常值得你為之努力的。說到這個，如果你想成為一名卓越的作家，你會專注於寫作麼？你或許已經嘗試，但你並沒有走得很遠。大多數的偉大作家也是如飢似渴的讀者，這是一個普遍事實。在你寫出任何拿得出手的東西之前，你需要品讀其他偉大作家，吸收不同的風格，看看前輩已嘗試過的東西，從中吸取精華。你的知識會慢慢增長，你自己的作品最終會透露出些許成熟，你也會找到一種“感覺”。編寫代碼和寫作沒什麼不同，如果你都沒有閱讀過任何卓越的代碼，你為什麼期望自己能寫出像樣的代碼呢？你顯然不應該那樣。對於程序員來說，閱讀卓越代碼就如同作家閱讀優秀書籍一樣重要（這話可不是我說的，這是[Peter Norvig](#)（Google 研究院總監）說的，他非常優秀，大家也要向他學習了）。

即便所有這些都無法讓你信服，那這裡有一個不可置否的事實。對你作為一名專業開發人員的生存來說，善於閱讀代碼至關重要。如今，任何有一定規模的項目，都是團隊的成果。所以，你通常要處理、修改和擴展大量不是你寫的代碼。因此，閱讀代碼可能是你能掌握的最常用並最有用的技能。挺過這個難關，好好掌握。

### 如何閱讀代碼？像某些人一樣.....

我已經記不清有多少次看到程序員（用鼠標）滾上滾下地看著不熟悉的代碼，幾分鐘過後，他們的臉上浮現出不悅的表情。他們不久後會宣告說，那代碼不值一讀，為什麼要浪費時間呢？我們只能用其他方法解決問題。我不確定（他們）在期待什麼，是通過潛移默化來吸收代碼的含義，還是集中精神盯著代碼來得到啟發？你不能只靠長時間盯著代碼來閱讀代碼，你要理解它並化為己用。這裡有一些我喜歡用的技巧，雖然這不是一份詳盡的列表，但我發現其中有些特別有用。

**1.盡力構建並運行代碼。**這通常是一個簡單的步驟，就像你在看可運行的代碼（這和隨機代碼相反）。不過，並非總是如此。通過構建和執行代碼，你能從中學到很多上層代碼結構。說到

工作代碼，你是否非常熟悉如何構建你的當前項目？雖然構建通常非常複雜，但通過構建並生成可執行的代碼，你能學到很多。

**2.不要只注重細節。**你要做的第一件事是，在你正閱讀的代碼中，找到代碼結構和風格的。首先瀏覽一下代碼，盡力理解不同代碼段要做什麼。這會讓你熟悉整個代碼的上層結構，你也能領會到你正處理的代碼的一些構思（良好架構和意大利麵條等）。這時候，你可以找到切入點（不管它是什麼，主函數、servlet或控制器等），並查看代碼如何在那里分支。不要在這上面花過多的時間，隨著你愈加熟悉代碼，你可以隨時回來查看。

**3.確信自己理解所有結構。**除非你碰巧是所用編程語言的首席專家，否則該語言有些它能做的事你可能還不知道。當你在瀏覽代碼時，記下所有你或許不熟悉的结构。如果有很多不熟悉的結構，你要做的下一步非常明顯。如果你不知道代碼要做什麼，那你就走不了很遠。即便只有幾個你不熟悉的結構，你應當深入查看。你現在是在探索你所用編程語言中你以前不知道的東 西，為此花幾個小時來閱讀代碼，我也非常樂意。

**4.既然你對大多數結構已有很好的了解，那現在是該做些隨機深入研究了。**就像步驟2，開始瀏覽代碼，當這次要挑選一些隨機函數或類，並開始逐行詳細查看。這是硬仗開始的地方，但也是你要取得主要成功的地方。這裡的構想，會形成你正在查看的代碼庫的思維模式。也不要在這上面花過長的時間，但在繼續前行之前，你要盡力並極大吸收一些有內容的代碼塊。這個步驟，你也可以隨時反復回過頭來，每次你都會了解更多的背景，並收穫更多。

**5.毫無疑問，在前面這些步驟中，肯定有你困惑的地方，所以這是你做些測試的最佳時間。**在測試的時候，你的麻煩可能會更少，同時你也能理解代碼。我一直感到奇怪，開發人員忽略一套寫得很好很全面的測試代碼，而盡力去閱讀並理解某些代碼。當然了，有時候並沒有測試。

**6.如果你說沒有測試，那這聽起來是編寫測試的時候了。**（編寫測試）有很多益處，有助於你自己的理解，有助於你提升代碼庫，閱讀代碼時也能編寫代碼，這是該你出手做些事的時候。即便已經有了測試，通常你也可以編寫一些測試，你總能受益的。測試代碼通常需要換種方式思考問題，那些你以前不太明了的概念也會變得更清晰。

**7.提取奇特的代碼，使其成為單獨的程序。**我發現閱讀代碼是個非常有趣的練習，即便只為節奏變化。即便你不了解代碼的底層細節，你或許能知道一些代碼在上層結構上要做什麼。什麼不提取一些特定的函數，單獨列為獨立的程序。當你在執行小段程序時，調試也會更簡單。反過來說，可能還需要一些額外的步驟，才能理解你正查看的代碼。

8. **代碼不干淨？有異味？為什麼不重構它？**我並不建議你重寫整個代碼庫，但重構部分代碼，真的有助於你理解層次上升一層。把你理解的函數拿出來，改成獨立的函數。在你知道之前，原來的大函數看起來易管理，你可以在腦海中修改它。重構允許你把代碼變成自己的，無需完成重寫代碼。如果有好的測試，有助於重構，但即便你沒有好的測試，抽取你確定的函數並做測試。即便測試看起來完全不充分，但作為一個開發人員，你得學著相信你的技能，有時候你只需努力去做（重構）。（如果你必須重構，你通常都可以把代碼恢復原狀。）

9. **如果沒什麼能幫上忙，那你就找個閱讀代碼的同伴。**或許並非只有你一個人能從這代碼中獲益，所以去找一個人，一起閱讀代碼吧。但你別找專家，他們會從上層結構上，向你解釋所有東西，你會錯失那些你自己詳細查看代碼時所能學到的細微差別。然而，如果不見效的話，你也不能理解，有時候，你能做的最好的事就是去問。向你的同事請教，如果你正在閱讀開源代碼，可以在互聯網上找人問問。但是你要記住，這是最後一步，而不是第一步。

如果我時間緊迫，需要快速合理地理解某些代碼，並且我只能挑選上述步驟的其中一個，那我會選擇“重構”（即：第8個步驟）。雖然你能理解的東西不會很多，但那些你領會的東西，你會牢牢記住的。總之，有件事你需要記在心裡。如果你新接觸一個重要的代碼庫，你不可能立即能理解它。這需要數天、數周和數月的潛心努力，接受這個事實。即便有一位專家和你在一起，也不能明顯地縮短時間（。然而，當涉及到代碼庫時，如果你能耐心並有條不紊地閱讀（和編寫）代碼，你最終能熟悉項目的方方面面，你能成為大牛。你或者是逃避閱讀代碼，經常尋求某人幫你講解某事。我知道我會成為哪一種人。

## **尋找閱讀代碼的機遇– 不要錯失**

我們喜歡編寫新代碼，是因為我們這次能正確處理問題。好吧，也許不是這次，但一定是下次。事實上是，你經常改進你的技術，但你從沒有恰當地處理問題。這就是編寫新代碼的價值所在，你可以歷練並磨練你的技能，但閱讀和把玩其他人編寫的代碼，（如果沒有更多的價值，）也是有同樣多的價值。你不僅能從中獲得一些有價值的技術知識，也能收穫領域知識，領域知識通常仍具更多價值（畢竟，代碼是文檔的最終形式）。

即便代碼寫得很神秘，無任何慣例可言，但還是有價值。你知道我在說的代碼，它幾乎看起來晦澀難懂，但不是有意而為之（因某些原因，Perl語言代碼通常是這樣的）。不管什麼時候我看到那樣的代碼，我都會這樣想：把它想像成只有你破譯它後才能學到的東西。是的，這是主要的痛楚之處，但要接受它，有時候你自己也會因瑣碎的原因而寫出那種使人困惑的代碼（否

認沒有用，你知道這是真的）。好了，如果你花些時間來閱讀那樣的代碼，你更有可能最終寫出同樣的代碼。並不說你將會寫出那樣的代碼，但你有能力寫出那樣的代碼。最後，態度通常是最重要的（編註：態度決定一切）。如果你視閱讀代碼為日常繁瑣的工作，那它就是（繁瑣的工作），並且你會逃避，但如果你視其為一個機遇，那好事終將到來。

## 編者後話

你會經常去閱讀優秀的開源代碼麼？歡迎在評論中和大家分享，也歡迎大家推薦優秀的開源代碼。