Ch15 : Developing and implementing software

能夠編寫應用程式的人稱為程序員 (Programmers)

電腦能夠讀取的語言稱作機器語言 (Machine language)

組合語言 (assembly language) 是任何一種用於[電腦](https://zh.wikipedia.org/wiki/%E7%94%B5%E5%AD%90%E8%AE%A1%E7%AE%97%E6%9C%BA)、[微處理器](https://zh.wikipedia.org/wiki/%E5%BE%AE%E5%A4%84%E7%90%86%E5%99%A8)、[微控制器](https://zh.wikipedia.org/wiki/%E5%BE%AE%E6%8E%A7%E5%88%B6%E5%99%A8)，或其他可程式化器件的[低階語言](https://zh.wikipedia.org/wiki/%E4%BD%8E%E7%BA%A7%E8%AF%AD%E8%A8%80)。在不同的裝置中，組合語言對應著不同的[機器語言](https://zh.wikipedia.org/wiki/%E6%9C%BA%E5%99%A8%E8%AF%AD%E8%A8%80)[指令集](https://zh.wikipedia.org/wiki/%E6%8C%87%E4%BB%A4%E9%9B%86%E6%9E%B6%E6%A7%8B)。一種組合語言專用於某種[電腦系統結構](https://zh.wikipedia.org/wiki/%E8%AE%A1%E7%AE%97%E6%9C%BA%E7%B3%BB%E7%BB%9F%E7%BB%93%E6%9E%84)，而不像許多[高階語言](https://zh.wikipedia.org/wiki/%E9%AB%98%E7%BA%A7%E8%AF%AD%E8%A8%80)，可以在不同系統平台之間移植。

使用組合語言編寫的原始碼，然後通過相應的組譯程式將它們轉換成可執行的機器碼。這一過程被稱為組譯過程。

組合語言使用輔助記憶碼 (Mnemonics) 來代替和表示特定低階機器語言的操作。特定的組譯目標指令集可能會包括特定的運算元。許多組譯程式可以辨識代表位址和常數的標籤 (Label) 和符號 (Symbols)，這樣就可以用字元來代表運算元而無需採取[寫死](https://zh.wikipedia.org/wiki/%E5%AF%AB%E6%AD%BB)的方式。普遍地說，每一種特定的組合語言和其特定的機器語言指令集是一一對應的。

編譯器 (compiler) 是一種[電腦程式](https://zh.wikipedia.org/wiki/%E9%9B%BB%E8%85%A6%E7%A8%8B%E5%BC%8F)，它會將某種程式語言寫成的[原始碼](https://zh.wikipedia.org/wiki/%E5%8E%9F%E5%A7%8B%E7%A2%BC) (原始語言) 轉換成另一種程式語言 (目標語言)。

它主要的目的是將便於人編寫、閱讀、維護的進階電腦語言所寫作的[原始碼](https://zh.wikipedia.org/wiki/%E5%8E%9F%E5%A7%8B%E7%A2%BC)程式，翻譯為電腦能解讀、執行的低階機器語言的程式，也就是[執行檔](https://zh.wikipedia.org/wiki/%E5%9F%B7%E8%A1%8C%E6%AA%94)。

編譯語言 (Compiled languages) ：透過用接近人類語言的程式語言，將code寫好之後，再交給編譯器去幫我們「翻譯」成電腦看得懂的機器語言(也就是翻譯成滿篇的0與1)，再請電腦執行翻譯過後的程式。這類型的語言多半都是強型別的語言(strongly-typed)，有非常嚴格的檢查機制。不過它們的執行速度非常快，畢竟已經翻譯成電腦看得懂的語言了。著名的編譯語言有：C、C++、Java、C#、Rust、Objective-C、Swift等等。

直譯式語言 (Interpreted language)：既然電腦只看得懂0與1的機器語言，直譯式語言一樣會把程式語言翻譯成機器語言，差別在於它是一行一行翻譯，而不是一口氣將整篇翻好。直譯式語言會從上到下，由左至右，翻譯完一行之後，執行一行。有點像專業的口譯翻譯員，聽完一段翻譯一段，而編譯式則比較像筆譯翻譯員，將整篇文章讀完，再將整篇翻譯。

直譯式的語言多半為弱類型的語言(weakly-typed)，語法非常簡單易懂，甚至快跟平常說的英文一樣了，非常易學。著名的語言有：JavaScript、Python、Ruby、PHP

不過直譯式語言這麼做的代價當然就是，速度很慢。畢竟你想想，整篇翻完執行，跟一行一行翻譯然後一邊執行，哪邊比較快呢？當然是整篇翻完得比較快。需要追求速度的程式，基本上都是用C++去寫的。但直譯式語言雖然執行速度慢，但開發速度就很快了，除錯也比編譯式快，畢竟你一寫完可以馬上執行馬上看結果，不用等編譯過程總結

* 編譯式語言執行速度上比直譯式語言快，因為是直接執行翻譯過的機器語言，不過要執行總是得經過一段編譯過程，而且編譯出來的機器語言也有平台依賴性
* 直譯式由於一邊翻譯一邊執行，所以較慢，不過靈活性非常大，適合用於編寫較小型的專案，或一些工具程式等等

查詢語言或資料查詢語言 (Data Query Language, DQL)是用於從[資料庫](https://zh.wikipedia.org/wiki/%E6%95%B0%E6%8D%AE%E5%BA%93)或[資訊系統](https://zh.wikipedia.org/wiki/%E4%BF%A1%E6%81%AF%E7%B3%BB%E7%BB%9F)中查詢[資料](https://zh.wikipedia.org/wiki/%E6%95%B0%E6%8D%AE)的[電腦語言](https://zh.wikipedia.org/wiki/%E8%AE%A1%E7%AE%97%E6%9C%BA%E8%AF%AD%E8%A8%80)。例如[SQL](https://zh.wikipedia.org/wiki/SQL)語言是查詢語言裡比較知名的一種。

數據查詢語言泛指向[資料庫](https://zh.wikipedia.org/wiki/%E8%B3%87%E6%96%99%E5%BA%AB)或[資訊系統](https://zh.wikipedia.org/wiki/%E4%BF%A1%E6%81%AF%E7%B3%BB%E7%BB%9F)查詢的各種[程式語言](https://zh.wikipedia.org/wiki/%E7%BC%96%E7%A8%8B%E8%AF%AD%E8%A8%80)。資料查詢語言必須要能表達所有[關係代數](https://zh.wikipedia.org/wiki/%E5%85%B3%E7%B3%BB%E4%BB%A3%E6%95%B0_(%E6%95%B0%E6%8D%AE%E5%BA%93))所能表達的查詢，這樣才被稱為關係完整的 (Relational complete)。[[1]](https://zh.wikipedia.org/zh-tw/%E6%9F%A5%E8%A9%A2%E8%AA%9E%E8%A8%80#cite_note-1)

DQL的主要功能是查詢資料，本身核心指令為SELECT，為了進行精細的查詢，加入了各類輔助指令。

虛擬碼 (pseudocode)，又稱為偽代碼，是一種高層次描述[演算法](https://zh.wikipedia.org/wiki/%E7%AE%97%E6%B3%95)的方法。它不是現實存在的[程式語言](https://zh.wikipedia.org/wiki/%E7%BC%96%E7%A8%8B%E8%AF%AD%E8%A8%80)；它可能綜合使用多種[程式語言](https://zh.wikipedia.org/wiki/%E7%BC%96%E7%A8%8B%E8%AF%AD%E8%A8%80)的語法、保留字，甚至會用到[自然語言](https://zh.wikipedia.org/wiki/%E8%87%AA%E7%84%B6%E8%AF%AD%E8%A8%80)。

它以程式語言的書寫形式指明演算法的職能。相比於程式語言 (例如Java、C++、C、Delphi 等等) 它更類似自然語言。它是半形式化、不標準的語言。我們可以將整個演算法執行過程的結構用接近自然語言的形式 (這裡可以使用任何一種作者熟悉的文字，例如中文、英文，重點是將程式的意思表達出來) 描述出來。使用虛擬碼，可以幫助我們更好地表述演算法，不用拘泥於具體的[實現](https://zh.wikipedia.org/wiki/%E5%AE%9E%E7%8E%B0)。