什 麼 是 “物 件 ”?

--- 簡單嗎? 抽象嗎? 還是大哉問? ---

丁培毅

也許你知道什麼是物件, 但是講不太出來, 或是像我一樣, 講了半天, 旁敲側擊, 支吾其辭, 繞來繞去, 就是不直接給它一個清楚的定義, … 老是說它抽象, 甚至說它太簡單了, 有一點不屑回答, … 哈哈! 是抽象還是不懂啊?!!

也許你看過 “物件” 的定義, 看過 “物件” 的實例, 但是大家才學了幾個星期的 C++, 我敢打賭一定有一半以上的同學沒有辦法回答這個問題, 聽到這個問題就不太想正面回答。 物件不就是物件嗎? 答來答去總是會有漏洞, 會答不上來, 那麼多的書本裡也好像沒有看到很簡潔清楚的定義, 可是這個課是 C++, 而且是用 C++ 來寫 “物件” 導向的程式, 課程都已經過了 1/3, 再迴避這個問題也太不像話了, … 應該是老師的責任吧!! 大家醒醒, 看我用物件砸過去!!

這樣吧, 定義太複雜就沒有人會瞭解了, 簡單一點, 行嗎?

物件是 “有自己私有的狀態、有行為能力、負責提供某些指定功能的軟體單元”

物件是一個服務者, 提供有用的服務

用我們已經談過的 OO 性質來描述:

> “物件可以保存資料, 但是 封裝 得很好,
> 只有透過適當的 介面 才能夠操作它, 或是說得到它的服務”

還抽象嗎? 嗯… 是, 大概還是很抽象, 看了這樣一個定義, 如果你心中還是沒有浮現對應的程式碼的樣子的話, 如果你分不出來什麼是物件、什麼不是物件的話, 那就是抽象, 就算記著這個定義也是沒有用的 … 好討厭的感覺!

程式片斷一:

```
Student john;
Teacher david;
Course englishCourse;
…
david.teaches(john, englishCourse);
john.takeAnExam(date, englishCourse);
…
```

程式片斷二:

```
Student john;
Teacher david;
…
david.students[david.studentCount++] = “john”;
```

```
…
john.examList[john.examNumber].date = “940415”;
john.examList[john.examNumber++].course = “English”;
…
```

在上面兩個程式片斷中, john 和 david 的定義方法看起來是一模一樣的, 假設兩個程式中的 Student 和 Teacher 兩個型態都是用 C++ 的 class 語法宣告出來的, 是不是在兩個程式中 john 和 david 都是所謂的物件呢?

顯然他們都可以暫存一些資料在裡面, 都會占用記憶體空間, 都可以協助達成程式的功能, 但是我們說前者是物件, 後者只是變數。

物件是提供服務的, 變數是暫存資料的, 物件是聰明的, 能夠讓你在不知道細節的狀況下滿足你的需求, 好的物件能夠幫你達成很多你自己做不到、或是無法一一辦妥的事, 例如在日常生活中我們說打火機是一個不錯用的物件, 沒有它, 光光給你丁烷瓦斯, 想必你要讓它順利安全地燃燒十分鐘是有困難的, 有了這個物件, 可以省卻你許多煩惱, 你只需要知道它的使用方法、它的功能、和它的限制就好了, 生活是很愉快的。

反之變數的最主要功能是暫存資料, 資料該怎麼樣使用, 什麼資料格式是對的, 什麼資料是有用的, 都要看是誰拿資料去使用而定, 正確的資料給不會運用的人是沒有幫助的。所以使用資料的人的角色非常重要, 他要整合運用所有的資料, 也是能否達成使命的關鍵, 當然如果這個人發生錯誤、掛了、或是心情不佳, 就算是資料齊全了, 也不會有什麼結果的。

只運用資料變數來完成的程式, 基本上就是 “程序化 (procedure-oriented)” 的程式, 你一定會發現程式在運作時, 好像有一個超人在一步一步仔細地處理暫存在各個變數裡的資料, 那個超人就是 CPU, 喔! 不不! 其實那個超人就是程式設計者, 就是你啦! 是你在教 CPU 該怎樣做, 怎樣處理的, 真是很了不起的工作, 實在該給自己鼓鼓掌。不過萬般事項只要錯一個小小的環節, 程式裡就有虫虫留在裡面, 當然不是故意養的囉, 人非聖賢, 這個工作又那麼繁雜, … 怎麼說都不得不原諒自己。那麼我們究竟該作什麼改變來避免自己犯錯呢?

先岔開話題一下, 我們常常看到很多失敗的企業主管, 疲於奔命, 都已經盡心盡力了, 卻還是常出紕漏, 以第三者的角度去看, 常常都可以看到一個超人, 事必躬親, 常常責備下屬什麼都做不好, 常常埋怨下屬不能幫忙, 下屬最後常常只能裝笨, 要什麼資料給什麼資料, 多做多錯, 少做少錯, 大家都成為存放資料的桶子, 讓聰明愛表現、不懂得協調鼓勵下屬的主管自己唱獨腳戲。

呵呵! 這聽起來和寫程序化程式的程式設計師還真有點像! 不過程式設計師還是挺厲害的, 至少發明了很多整理資料的方法 -- 你看過有主管把資料整整齊齊地吊在一棵樹上的嗎?! 寫程式的人還發明了有效率、很神奇的標準資料處理方式 -- 你看過有人在整理排序資料的時候, 很神奇地找出某一筆資料最後應該放在哪裡, 然後把資料分成兩半個別排序的嗎?

哎呀! 看到這種主管, 聰明的你一定知道他缺乏的是信任、分層負責、分工的方法-這些是在群體社會中非常重要的成功法則。那麼對於寫程式這件事, 你是不是也得到了一些啟發或是想法呢? 怎樣分工? 分工的精神又是什麼呢?

分工以後就是希望各人能夠對自己的工作負責, 分工時最重要的就是要訂定清楚的工作界限與目標, 什麼工作是誰的? 在什麼時機、將什麼資料處理成什麼樣子, 交給什麼人接手…

分工以後基本上每個人的工作應該變得單純化, 變得簡單不易出錯, 如何完成工作的細節也應該不需要別人插手, 如此大家專注於自己的工作, 事情才容易成功。當然要在分工的環境中成功的話需要遵守合作的基本原則, 不管別人是怎樣達成要求的, 只要滿足原先定的規格, 就需要接受, 主管需要協調工作的進度, 整合工作的成果, 統合分配運用資源, 增進團隊的效率, 降低執行工作的成本。

唉呀呀! 上面所說到的每一個分配到工作的成員, 該不會就是物件導向程式設計時的物件吧! 性質真的很像, 提供智慧型的服務, 內部工作的細節都隱藏起來, 合作時遵守明確的界面規格, 連分工環境中會有一定程度工作的重疊與效率的損失都很像哩! 那麼以物件導向方式設計程式的人基本上就是要把工作依照性質做適當的分工, 將分工過的各個部份以物件的形式實作起來, 然後再運用這些物件來組合出程式的功能, 如果有因為分工而導致效率不好的地方, 再調整各個物件的功能來加強效率, 很像一個主管該做的事情喔。

程序化的程式設計當然也講求分工, 也就是常常聽到的結構化和模組化, 通常在設計程式時以程式想要完成的功能為起點, 由上而下地拆解各部份的功能, 將個別細項功能逐步分配給各個模組, 實作上最主要看到的是整個程式分解為許多函式, 處理一些為這個程式的功能特別設計的資料結構, 性質類似的函式又放在同一個檔案模組裡, 不過這樣子的分工是 “形式上” 的分工, 實際上程式流程、資料流向、演算法、資料結構等等常常都是透明的, 由程式設計者全盤掌控的。物件化的程式設計的分工, 特別重視分層負責的 “分工精神”, 封裝和界面就是實現這種分工精神的工具, 也因為每個物件需要自行負責, 因此程式的基本設計由下而上, 整體的功能架構在物件個別的功能之上, 在軟體的需求逐漸演進更改時, 能夠由一組完善的物件中, 迅速調整組織與架構來滿足新的要求。

這樣子再回頭想想剛才物件的定義, 和那兩個程式片斷, 有比較清楚的概念、比較深入的體會了嗎? 沒有嗎? 我被你打敗了嗎?

註: 一般來說, 演算法和物件導向設計應該有些衝突, 物件導向著重的是分工合作的模型和清楚的程式, 演算法則只強調效率, 所以就算你看到坊間有使用 C++/JAVA 寫演算法的書時, 裡面常常看不到物件導向程式設計中分工與模組化的精神, 應該不會覺得很奇怪吧! 還好基本上這兩個工具是解決不同層次問題的, 演算法解決比較低階的效率問題!! 物件導向解決比較高階的模組間分工合作的問題!!

註: 使用物件導向 (object-oriented) 程式語言實作物件系統時的物件, 和使用 object-based 程式語言來實作物件系統時的物件, 和使用一般程序式程式語言來實作物件系統時的物件, 應該都沒有差別, 精神上都是相同的。

題外話: 曾經在業界當過幾年的主管, 最怕的就是那種交代一件事才做一件事、過程還需要你監督的屬下, 這樣的人有時只要心態稍微調整一下, 能夠思考每一件事該以什麼方法做到什麼程度, 他就可以真正負責一些工作, 有機會可以逐漸得到主管的信賴與重用, 老闆也才能夠整合大家的力量來完成更多的事, 他也才能夠分配到更重要的工作, 累積更多、更有用的經驗; 有的時候找工作看的不是工作內容, 那太短視了, 也太低估自己的能力了, 如果你對自己工作的態度、工作的方法有信心的話, 那麼你要看的是老闆用人的方法。