搞笑談軟工

敏捷開發，設計模式，精實開發，Scrum，軟體設計，軟體架構

2012 年 1 月 4 日 星期三

亂談軟體設計（5）：Dependency-Inversion Principle

January 04 09:18~11:15

最近為了整理「亂談軟體設計」這一系列的文章，回頭把 Agile Software Development 以及 Object-Oriented Software Construction, 2nd 又稍微翻了一下。Teddy 第一次看到 Agile Software Development 書中所提的幾個設計原則大概是在 10-12 年前，第一次接觸的時候對於某些設計原則還是處於似懂非懂的階段，這麼多年下來這些設計原則也陪了 Teddy 經歷過許多軟體開發專案，慢慢地融入 Teddy 的潛意識中。

Design Patterns 雖然好用，當鄉民們可以直接說出這邊需要一個「Observer」或是「Command」的時候，設計問題看起來立即就獲得解決。但有時候遭遇到的設計問題不是那麼明顯地可以立即套用 patterns，或是套用 patterns 之後可能會有誤用與互相牴觸的時候，亦或是當鄉民們需要幫其他人做 design review 或是 code review 的時候，這些設計原則就可以跳出來幫上忙（打架的時候人多一點總是比較好滴...XD）。

今天要談的是 The Dependency-Inversion Principle，內容請參考 Agile Software Development 這本第 127-134 頁：

DIP：The Dependency-Inversion Principle

- High-level modules should not depend on low-level modules. Both should depend on abstractions.
- Abstractions should not depend on details. Details should depend on abstractions.

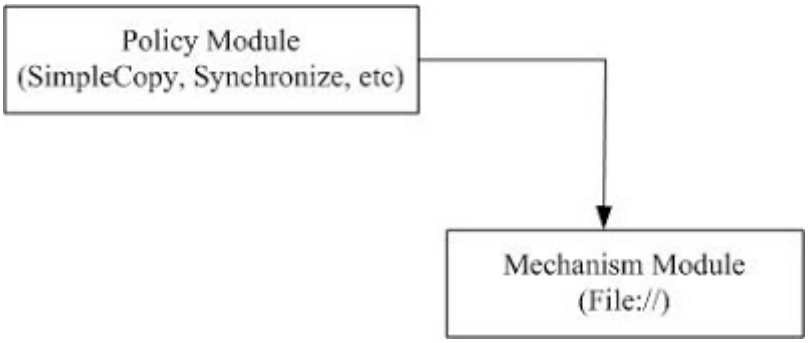
DIP 的中文有人翻成「依賴倒轉原則」、「相依性反向原則」、「依存關係反轉原則」~~、「玩具反斗原則」~~，如果一定要說中文，鄉民們就隨便自己挑一個喜歡的翻譯去記吧。這個原則聽起來很奇怪，不過了解之後就會發現怎麼又是在說同樣一件事，什麼事？就是：

program to an interface, not an implementation（有種鬼打牆的感覺...XD）

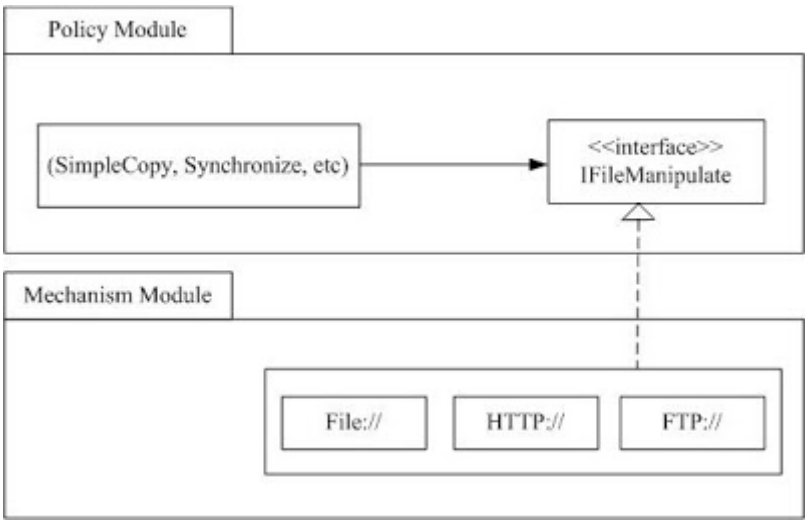
這幾年拜房價飛漲之賜，都更建案如雨後春筍般的冒出，隨便在路上逛一逛都可以看到好幾個新屋施工的建案。鄉民們不用是建築師也都知道，要蓋房子，地下室沒蓋好之前不能蓋一樓，一樓沒蓋好之前不能蓋二樓，以此類推。所以說，一個大樓的高樓層（high-level modules）相依於低樓層（low-level modules），這是一般大家所認知的相依性關係。

舉個軟體的例子，假設鄉民們要開發一個檔案同步軟體，這個檔案同步軟體可以同步來源端與目的端資料夾中的檔案。為了簡化起見，這個軟體只有兩個模組（如下圖所示），Policy module 決定了同步的策略，例如，SimpleCopy 策略只負責將來源端的資料複製到目的端（單向複製），而 Synchronize 策略則會保證來源端與目的端中的資料在同步之後會完全相同（雙向同步）。

Policy module 中的這兩種策略，都必須依靠 Mechanism Module 中的元件來實際執行檔案操作的動作。例如，使用 Java 的 IO 或是 NIO 物件來操作檔案。因此，我們說上層的 Policy Module 相依於底層的 Mechanism Module（high-level modules depend on low-level modules）。



這聽起來好像是廢話，因為上層的模組一定會需要用到下層模組提供的服務啊，所以當然是上層模組會相依於下層模組啊，不然程式要怎麼寫？這就是 DIP 這個原則會叫做 DIP 的原因，DIP 告訴我們，這樣的相依關係（上層相依於底層）要「反過來」。但是也不能說直接反過來變成底層相依於上層，而是要變成「底層與上層都相依於抽象介面（abstractions）」，請參考下圖。



把處理檔案這件事情（這個責任）抽離出來定義成 IFileManipulate 介面（interface），讓上層的 Policy Module 要處理檔案的時候只使用 IFileManipulate 介面所提供的服務，不要跟之前一樣直接去操作 Java 的 IO 或 NIO 物件。底層 Mechanism Module 可以提供多個 IFileManipulate 介面的實做（implementation），例如可以透過 File、HTTP、FTP 等通訊協定來操作檔案的實做。如此一來整個系統就變得比較有彈性且容易擴充。

看到這邊這種設計不是 program to an interface, not an implementation 那還是什麼？在更進一步往下看，這樣的設計也符合 Open-Closed Principle（open for extension, but closed for modification）。如果要擴充不同的檔案同步機制，只要在 Mechanism Module 中「外掛」新的 IFileManipulate 介面的實做就可以了（open for extension），不需要去修改原本已經可以正常運作的程式碼（closed for modification）。

在靠....近一點看，上述好處要能夠成立，整個設計與實做必須要符合 Liskov Substitution Principle（Subtypes must be substitutable for their base types），以確保底層的實做會遵守上層介面所定義的行為。

看這一集等於連續看了三集，CP 值還滿高的，如果沒看懂仔細多看幾遍，如果都還看得懂，恭喜老爺賀喜夫人你們已經免費學會了最重要的幾點物件導向設計原則了。

友藏內心獨白：原來這些原則彼此之間也有相依性的關係喔。
張貼者： Teddy Chen 於 上午 11:15