23個設計模型

* Creational Patterns 【創建型(生成)模式】：處理物件創建機制的設計模式。此類型模式試圖根據適合的情況來決定建立物件。

1. Factory Method Pattern (工廠方法模式)：

* 定義:

定義一個創建對象的接口，讓其子類自己決定實例化哪一個工廠類，工廠模式使其創建過程延遲到子類進行。

* 使用時機: 依賴的實體化對象不只一個或可能會改變的時候。
  1. 日誌記錄器：記錄可能記錄到本地硬盤、系統事件、遠程服務器等，用戶可以選擇記錄日誌到什麼地方。
  2. 數據庫訪問，當用戶不知道最後系統採用哪一類數據庫，以及數據庫可能有變化時。
  3. 設計一個連接服務器的框架，需要三個協議，"POP3"、"IMAP"、"HTTP"，可以把這三個作為產品類，共同實現一個接口。
* 目的：透過工廠的方法將物件實體化的過程與外部程式碼分離，減少對類別的耦合，符合依賴反轉原則。
* 優點:
  1. 用戶只需要知道具體工廠的名稱就可得到所要的產品，無須知道產品的具體創建過程。
  2. 在系統增加新的產品時只需要添加具體產品類和對應的具體工廠類，無須對原工廠進行任何修改，滿足開閉原則。
* 缺點: 每增加一個產品就要增加一個具體產品類和一個對應的具體工廠類，這增加了系統的複雜度。
* 網址:

<https://lightofhexagram.wordpress.com/2016/11/04/%E8%A8%AD%E8%A8%88%E6%A8%A1%E5%BC%8F-%E5%B7%A5%E5%BB%A0%E6%96%B9%E6%B3%95%E6%A8%A1%E5%BC%8F/>

1. Abstract Factory Pattern (抽象工廠模式)：

* 定義:

提供一個創建一系列相關或相互依賴對象的接口，而無需指定它們具體的類。

* 使用時機: 考慮多等級產品的生產，將同一個具體工廠所生產的位於不同等級的一組產品稱為一個產品族。
  1. 換主題，一整套一起換。
  2. 生成不同操作系統的程序。
* 目的：抽象工廠模式是工廠方法模式的升級版本，工廠方法模式只生產一個等級的產品，而抽象工廠模式可生產多個等級的產品。
* 優點:
  1. 可以在類的內部對產品族中相關聯的多等級產品共同管理，而不必專門引入多個新的類來進行管理。
  2. 當增加一個新的產品族時不需要修改原代碼，滿足開閉原則。
* 缺點: 當產品族中需要增加一個新的產品時，所有的工廠類都需要進行修改。
* 網址:

<https://rongli.gitbooks.io/design-pattern/content/chapter3.html>

1. Singleton Pattern (單例模式)：

* 定義:

保證一個類別只會產生一個物件，而且要提供存取該物件的統一方法。

* 使用時機: 當想控制實例數目，節省系統資源的時候。
  1. 要求生產唯一序列號。
  2. WEB 中的計數器，不用每次刷新都在數據庫裡加一次，用單例先緩存起來。
  3. 創建的一個對象需要消耗的資源過多，比如I/O 與數據庫的連接等。
* 目的：解決一個全局使用的類頻繁地創建與銷毀。
* 優點:
  1. 在內存裡只有一個實例，減少了內存的開銷，尤其是頻繁的創建和銷毀實例（比如網頁首頁頁面緩存）。
  2. 避免對資源的多重佔用（比如寫文件操作）。
* 缺點: 沒有接口，不能繼承，與單一職責原則衝突，一個類應該只關心內部邏輯，而不關心外面怎麼樣來實例化。
* 網址:

<https://openhome.cc/Gossip/DesignPattern/SingletonPattern.htm>

1. Builder Pattern (建造者模式)：

* 定義:

將一個複雜的構建與其表示相分離，使得同樣的構建過程可以創建不同的表示。

* 使用時機: 一些基本部件不會變，而其組合經常變化的時候。
  1. 需要生成的對象具有復雜的內部結構。
  2. 需要生成的對象內部屬性本身相互依賴。
* 目的：主要解決在軟件系統中，有時候面臨著"一個複雜對象"的創建工作，其通常由各個部分的子對像用一定的算法構成；由於需求的變化，這個複雜對象的各個部分經常面臨著劇烈的變化，但是將它們組合在一起的算法卻相對穩定。
* 優點:
  1. 各個具體的建造者相互獨立，有利於系統的擴展。
  2. 客戶端不必知道產品內部組成的細節，便於控制細節風險。
* 缺點:
  1. 產品的組成部分必須相同，這限制了其使用範圍。
  2. 如果產品的內部變化複雜，該模式會增加很多的建造者類。
* 網址:

<https://juejin.im/post/5a23bdd36fb9a045272568a6>

1. Prototype Pattern (原型模式)：

* 定義:

用原型實例指定創建對象的種類，並且通過拷貝這些原型創建新的對象

* 使用時機:
  1. 當一個系統應該獨立於它的產品創建，構成和表示時。
  2. 當要實例化的類是在運行時刻指定時，例如，通過動態裝載。
  3. 為了避免創建一個與產品類層次平行的工廠類層次時。
  4. 當一個類的實例只能有幾個不同狀態組合中的一種時。建立相應數目的原型並克隆它們可能比每次用合適的狀態手工實例化該類更方便一些。
  5. 資源優化場景。
  6. 類初始化需要消化非常多的資源，這個資源包括數據、硬件資源等。
  7. 性能和安全要求的場景。
  8. 通過new產生一個對象需要非常繁瑣的數據準備或訪問權限，則可以使用原型模式。
  9. 一個對像多個修改者的場景。
  10. 一個對象需要提供給其他對象訪問，而且各個調用者可能都需要修改其值時，可以考慮使用原型模式拷貝多個對象供調用者使用。
* 目的：解決在運行期建立和刪除原型。
* 優點:
  1. 性能提高。
  2. 逃避構造函數的約束。
* 缺點: 配備克隆方法需要對類的功能進行通盤考慮，這對於全新的類不是很難，但對於已有的類不一定很容易，特別當一個類引用不支持串行化的間接對象，或者引用含有循環結構的時候。
* 網址:

<https://www.itread01.com/content/1546617785.html>

* Structural Patterns 【結構模式】：藉由以一貫之的方式來了解元件間的關係，以簡化設計。

1. Adapter Pattern (適配器模式)：

* 定義:

將一個類的接口轉換成客戶希望的另外一個接口。適配器模式使得原本由於接口不兼容而不能一起工作的那些類可以一起工作

* 使用時機:
  1. 系統需要使用現有的類，而此類的接口不符合系統的需要。
  2. 想要建立一個可以重複使用的類，用於與一些彼此之間沒有太大關聯的一些類，包括一些可能在將來引進的類一起工作，這些源類不一定有一致的接口。
  3. 通過接口轉換，將一個類插入另一個類系中。
  4. Ex. 美國電器110V，中國220V，就要有一個適配器將110V轉化為220V。、在LINUX上運行WINDOWS程序
* 目的：主要解決在軟件系統中，常常要將一些"現存的對象"放到新的環境中，而新環境要求的接口是現對像不能滿足的。
* 優點:
  1. 客戶端通過適配器可以透明地調用目標接口。
  2. 復用了現存的類，程序員不需要修改原有代碼而重用現有的適配者類。
  3. 將目標類和適配者類解耦，解決了目標類和適配者類接口不一致的問題。
* 缺點: 對類適配器來說，更換適配器的實現過程比較複雜。。
* 網址:

<https://ithelp.ithome.com.tw/articles/10194158>

1. Bridge Pattern (橋接模式)：

* 定義:

將抽象部分與實現部分分離，使它們都可以獨立的變化。

* 使用時機: 實現系統可能有多個角度分類，每一種角度都可能變化。
  1. 如果一個系統需要在構件的抽象化角色和具體化角色之間增加更多的靈活性，避免在兩個層次之間建立靜態的繼承聯繫，通過橋接模式可以使它們在抽象層建立一個關聯關係。
  2. 於那些不希望使用繼承或因為多層次繼承導致系統類的個數急劇增加的系統，橋接模式尤為適用。
* 目的：在有多種可能會變化的情況下，用繼承會造成類爆炸問題，擴展起來不靈活。
* 優點:
  1. 由於抽象與實現分離，所以擴展能力強。
  2. 其實現細節對客戶透明。
* 缺點: 由於聚合關係建立在抽象層，要求開發者針對抽象化進行設計與編程，這增加了系統的理解與設計難度。
* 網址:

<https://medium.com/@Bear_/bridge-%E6%A9%8B%E6%8E%A5%E6%A8%A1%E5%BC%8F-37520d5f53de>

1. Composite Pattern (組合模式)：

* 定義:

將對象組合成樹形結構以表示"部分-整體"的層次結構。組合模式使得用戶對單個對象和組合對象的使用具有一致性。

* 使用時機:
  1. 想表示對象的部分-整體層次結構（樹形結構）。
  2. 希望用戶忽略組合對象與單個對象的不同，用戶將統一地使用組合結構中的所有對象。
* 目的：它在我們樹型結構的問題中，模糊了簡單元素和復雜元素的概念，客戶程序可以像處理簡單元素一樣來處理複雜元素，從而使得客戶程序與復雜元素的內部結構解耦。
* 優點:
  1. 組合模式使得客戶端代碼可以一致地處理單個對象和組合對象，無須關心自己處理的是單個對象，還是組合對象，這簡化了客戶端代碼。
  2. 更容易在組合體內加入新的對象，客戶端不會因為加入了新的對象而更改源代碼，滿足“開閉原則。
* 缺點:
  1. 設計較複雜，客戶端需要花更多時間理清類之間的層次關係。
  2. 不容易限制容器中的構件。
  3. 不容易用繼承的方法來增加構件的新功能。
* 網址:

<https://www.itread01.com/articles/1485506384.html>

1. Decorator Pattern (裝飾模式)：

* 定義:

動態地給一個對象添加一些額外的職責。就增加功能來說，裝飾器模式相比生成子類更為靈活。

* 使用時機: 在不想增加很多子類的情況下擴展類。
  1. Ex. 孫悟空有72變，當他變成"廟宇"後，他的根本還是一隻猴子，但是他又有了廟宇的功能。
  2. Ex. 不論一幅畫有沒有畫框都可以掛在牆上，但是通常都是有畫框的，並且實際上是畫框被掛在牆上。在掛在牆上之前，畫可以被蒙上玻璃，裝到框子裡；這時畫、玻璃和畫框形成了一個物體。
* 目的：通常我們為了擴展一個類經常使用繼承方式實現，由於繼承為類引入靜態特徵，並且隨著擴展功能的增多，子類會很膨脹。
* 優點:
  1. 採用裝飾模式擴展對象的功能比採用繼承方式更加靈活。
  2. 可以設計出多個不同的具體裝飾類，創造出多個不同行為的組合。
* 缺點: 裝飾模式增加了許多子類，如果過度使用會使程序變得很複雜。
* 網址:

<https://openhome.cc/Gossip/DesignPattern/DecoratorPattern.htm>

1. Facade Pattern (外觀模式)：

* 定義:

為子系統中的一組接口提供一個一致的介面，外觀模式定義了一個高層接口，這個接口使得這一子系統更加容易使用

* 使用時機:
  1. 客戶端不需要知道系統內部的複雜聯繫，整個系統只需提供一個"接待員"即可。
  2. 定義系統的入口。
  3. Ex. 去醫院看病，可能要去掛號、門診、劃價、取藥，讓患者或患者家屬覺得很複雜，如果有提供接待人員，只讓接待人員來處理，就很方便。
* 目的：降低訪問複雜系統的內部子系統時的複雜度，簡化客戶端與之的接口。
* 優點:
  1. 減少系統相互依賴。
  2. 提高靈活性。
  3. 提高了安全性
* 缺點: 不符合開閉原則，如果要改東西很麻煩，繼承重寫都不合適。
* 網址:

<https://dotblogs.com.tw/jesperlai/2018/04/15/153646>

1. Decorator Pattern (享元模式)：

* 定義:

運用共享技術有效地支持大量細粒度的對象。。

* 使用時機:
  1. 系統中有大量對象。
  2. 這些對象消耗大量內存。
  3. 這些對象的狀態大部分可以外部化。
  4. 這些對象可以按照內蘊狀態分為很多組，當把外蘊對像從對像中剔除出來時，每一組對像都可以用一個對象來代替。
  5. 系統不依賴於這些對象身份，這些對像是不可分辨的。
* 目的：在有大量對象時，有可能會造成內存溢出，我們把其中共同的部分抽出來，如果有相同的業務請求，直接返回在內存中已有的對象，避免重新創建。
* 優點: 相同對像只要保存一份，這降低了系統中對象的數量，從而降低了系統中細粒度對像給內存帶來的壓力。
* 缺點:
  1. 為了使對象可以共享，需要將一些不能共享的狀態外部化，這將增加程序的複雜性。
  2. 讀取享元模式的外部狀態會使得運行時間稍微變長。
* 網址:

<https://ithelp.ithome.com.tw/articles/10195427>

1. Proxy Pattern (代理模式)：

* 定義:

為其他對象提供一種代理以控制對這個對象的訪問。

* 使用時機: 想在訪問一個類時做一些控制。
  1. Windows裡面的快捷方式。
  2. Ex. 買火車票不一定在火車站買，也可以去代售點。
* 目的：在直接訪問對象時帶來的問題，比如說：要訪問的對像在遠程的機器上。在面向對象系統中，有些對象由於某些原因（比如對象創建開銷很大，或者某些操作需要安全控制，或者需要進程外的訪問），直接訪問會給使用者或者係統結構帶來很多麻煩，我們可以在訪問此對象時加上一個對此對象的訪問層。
* 優點:
  1. 代理模式在客戶端與目標對象之間起到一個中介作用和保護目標對象的作用。
  2. 代理對象可以擴展目標對象的功能。
  3. 代理模式能將客戶端與目標對象分離，在一定程度上降低了系統的耦合度。
* 缺點:
  1. 在客戶端和目標對象之間增加一個代理對象，會造成請求處理速度變慢。
  2. 增加了系統的複雜度。
* 網址:

<https://codertw.com/%E7%A8%8B%E5%BC%8F%E8%AA%9E%E8%A8%80/672843/>

* Behavioral Patterns 【行為模式】：用來識別對象之間的常用交流模式並加以實現。如此，可在進行這些交流活動時增強彈性。

1. Chain of Responsibility Pattern (責任鏈模式)：

* 定義:

避免請求發送者與接收者耦合在一起，讓多個對像都有可能接收請求，將這些對象連接成一條鏈，並且沿著這條鏈傳遞請求，直到有對象處理它為止。

* 使用時機: 在處理消息的時候以過濾很多道。
* 目的：職責鏈上的處理者負責處理請求，客戶只需要將請求發送到職責鏈上即可，無須關心請求的處理細節和請求的傳遞，所以職責鏈將請求的發送者和請求的處理者解耦了。
* 優點:
  1. 降低了對象之間的耦合度。該模式使得一個對象無須知道到底是哪一個對象處理其請求以及鏈的結構，發送者和接收者也無須擁有對方的明確信息。
  2. 增強了系統的可擴展性。可以根據需要增加新的請求處理類，滿足開閉原則。
  3. 增強了給對象指派職責的靈活性。當工作流程發生變化，可以動態地改變鏈內的成員或者調動它們的次序，也可動態地新增或者刪除責任。
  4. 責任鏈簡化了對象之間的連接。每個對像只需保持一個指向其後繼者的引用，不需保持其他所有處理者的引用，這避免了使用眾多的if 或者if···else 語句。
  5. 責任分擔。每個類只需要處理自己該處理的工作，不該處理的傳遞給下一個對象完成，明確各類的責任範圍，符合類的單一職責原則。
* 缺點:
  1. 不能保證每個請求一定被處理。由於一個請求沒有明確的接收者，所以不能保證它一定會被處理，該請求可能一直傳到鏈的末端都得不到處理。
  2. 對比較長的職責鏈，請求的處理可能涉及多個處理對象，系統性能將受到一定影響度。
  3. 職責鏈建立的合理性要靠客戶端來保證，增加了客戶端的複雜性，可能會由於職責鏈的錯誤設置而導致系統出錯，如可能會造成循環調用。
* 網址:

<https://xyz.cinc.biz/2013/07/chain-of-responsibility-pattern.html>

1. Command Pattern (命令模式)：

* 定義:

將一個請求封裝成一個對象，從而使您可以用不同的請求對客戶進行參數化。

* 使用時機: 在某些場合，比如要對行為進行"記錄、撤銷/重做、事務"等處理，這種無法抵禦變化的緊耦合是不合適的。在這種情況下，如何將"行為請求者"與"行為實現者"解耦？將一組行為抽象為對象，可以實現二者之間的松耦合。
* 目的：在軟件系統中，行為請求者與行為實現者通常是一種緊耦合的關係，但某些場合，比如需要對行為進行記錄、撤銷或重做、事務等處理時，這種無法抵禦變化的緊耦合的設計就不太合適。
* 優點:
  1. 降低系統的耦合度。命令模式能將調用操作的對象與實現該操作的對象解耦。
  2. 增加或刪除命令非常方便。採用命令模式增加與刪除命令不會影響其他類，它滿足“開閉原則”，對擴展比較靈活。
  3. 可以實現宏命令。命令模式可以與組合模式結合，將多個命令裝配成一個組合命令，即宏命令。
  4. 方便實現Undo和Redo操作。命令模式可以與後面介紹的備忘錄模式結合，實現命令的撤銷與恢復。
* 缺點: 可能產生大量具體命令類。因為計對每一個具體操作都需要設計一個具體命令類，這將增加系統的複雜性。
* 網址:

<https://xyz.cinc.biz/2013/07/command-pattern.html>

1. Interpreter Pattern (解譯器模式)：

* 定義:

給定一個語言，定義它的文法表示，並定義一個解釋器，這個解釋器使用該標識來解釋語言中的句子。

* 使用時機: 如果一種特定類型的問題發生的頻率足夠高，那麼可能就值得將該問題的各個實例表述為一個簡單語言中的句子。這樣就可以構建一個解釋器，該解釋器通過解釋這些句子來解決該問題。
* 目的：對於一些固定文法構建一個解釋句子的解釋器。
* 優點:
  1. 可擴展性比較好，靈活。
  2. 增加了新的解釋表達式的方式。
  3. 易於實現簡單文法。
* 缺點:。
  1. 可利用場景比較少。
  2. 對於復雜的文法比較難維護。
  3. 解釋器模式會引起類膨脹。
* 網址:

<https://xyz.cinc.biz/2013/08/interpreter-pattern.html>

1. Iterator Pattern (疊代器模式)：

* 定義:

提供一種方法順序訪問一個聚合對像中各個元素,而又無須暴露該對象的內部表示。

* 使用時機: 遍歷一個聚合對象。
  1. 訪問一個聚合對象的內容而無須暴露它的內部表示。
  2. 需要為聚合對象提供多種遍歷方式。3、為遍歷不同的聚合結構提供一個統一的接口。
* 目的：不同的方式來遍歷整個整合對象。
* 優點:
  1. 訪問一個聚合對象的內容而無須暴露它的內部表示。
  2. 遍歷任務交由迭代器完成，這簡化了聚合類。
  3. 它支持以不同方式遍歷一個聚合，甚至可以自定義迭代器的子類以支持新的遍歷。
  4. 增加新的聚合類和迭代器類都很方便，無須修改原有代碼。
  5. 封裝性良好，為遍歷不同的聚合結構提供一個統一的接口。
* 缺點: 增加了類的個數，這在一定程度上增加了系統的複雜性。
* 網址:

<https://ithelp.ithome.com.tw/articles/10196545>

1. Strategy Pattern (策略模式)：

* 定義:

定義一系列的算法,把它們一個個封裝起來,並且使它們可相互替換。

* 使用時機: 一個系統有許多許多類，而區分它們的只是他們直接的行為。
* 目的：在有多種算法相似的情況下，使用if...else所帶來的複雜和難以維護。
* 優點:
  1. 多重條件語句不易維護，而使用策略模式可以避免使用多重條件語句。
  2. 策略模式提供了一系列的可供重用的算法族，恰當使用繼承可以把算法族的公共代碼轉移到父類裡面，從而避免重複的代碼。
  3. 策略模式可以提供相同行為的不同實現，客戶可以根據不同時間或空間要求選擇不同的。
  4. 策略模式提供了對開閉原則的完美支持，可以在不修改原代碼的情況下，靈活增加新算法。
  5. 策略模式把算法的使用放到環境類中，而算法的實現移到具體策略類中，實現了二者的分離。
* 缺點:
  1. 客戶端必須理解所有策略算法的區別，以便適時選擇恰當的算法類。
  2. 策略模式造成很多的策略類。
* 網址:

<https://dotblogs.com.tw/joysdw12/2013/03/07/95769>

1. Template Pattern (模版方法模式)：

* 定義:

定義一個操作中的算法的骨架，而將一些步驟延遲到子類中。模板方法使得子類可以不改變一個算法的結構即可重定義該算法的某些特定步驟。

* 使用時機: 當有一些通用的方法的時候。
  1. Ex.在造房子的時候，地基、走線、水管都一樣，只有在建築的後期才有加壁櫥加柵欄等差異。
* 目的：改善一些方法通用，卻在每一個子類都重新寫了這一方法的情況。
* 優點:
  1. 它封裝了不變部分，擴展可變部分。它把認為是不變部分的算法封裝到父類中實現，而把可變部分算法由子類繼承實現，便於子類繼續擴展。
  2. 它在父類中提取了公共的部分代碼，便於代碼復用。
  3. 部分方法是由子類實現的，因此子類可以通過擴展方式增加相應的功能，符合開閉原則。
* 缺點:
  1. 對每個不同的實現都需要定義一個子類，這會導致類的個數增加，系統更加龐大，設計也更加抽象。
  2. 父類中的抽象方法由子類實現，子類執行的結果會影響父類的結果，這導致一種反向的控制結構，它提高了代碼閱讀的難度。
* 網址:

<https://blog.csdn.net/carson_ho/article/details/54910518>

1. Observer Pattern (觀察者模式)：

* 定義:

定義對象間的一種一對多的依賴關係，當一個對象的狀態發生改變時，所有依賴於它的對像都得到通知並被自動更新。

* 使用時機: 一個對象（目標對象）的狀態發生改變，所有的依賴對象（觀察者對象）都將得到通知，進行廣播通知。
  1. Ex. 拍賣的時候，拍賣師觀察最高標價，然後通知給其他競價者競價。
* 目的：一個對象狀態改變給其他對象通知的問題，而且要考慮到易用和低耦合，保證高度的協作。
* 優點:
  1. 降低了目標與觀察者之間的耦合關係，兩者之間是抽象耦合關係。
  2. 目標與觀察者之間建立了一套觸發機制。
* 缺點:
  1. 目標與觀察者之間的依賴關係並沒有完全解除，而且有可能出現循環引用。
  2. 當觀察者對像很多時，通知的發布會花費很多時間，影響程序的效率。
* 網址:

<https://medium.com/enjoy-life-enjoy-coding/design-pattern-%E5%8F%AA%E8%A6%81%E4%BD%A0%E6%83%B3%E7%9F%A5%E9%81%93-%E6%88%91%E5%B0%B1%E5%91%8A%E8%A8%B4%E4%BD%A0-%E8%A7%80%E5%AF%9F%E8%80%85%E6%A8%A1%E5%BC%8F-observer-pattern-feat-typescript-8c15dcb21622>

1. Mediator Pattern (中介者模式)：

* 定義:

用一個中介對象來封裝一系列的對象交互，中介者使各對像不需要顯式地相互引用，從而使其耦合鬆散，而且可以獨立地改變它們之間的交互。

* 使用時機: 多個類相互耦合，形成了網狀結構。
  1. MVC框架，其中C（控制器）就是M（模型）和V（視圖）的中介者。
* 目的：當對象與對象之間存在大量的關聯關係，這樣勢必會導致系統的結構變得很複雜，同時若一個對象發生改變，我們也需要跟踪與之相關聯的對象，同時做出相應的處理。
* 優點:
  1. 降低了對象之間的耦合性，使得對象易於獨立地被復用。
  2. 將對象間的一對多關聯轉變為一對一的關聯，提高系統的靈活性，使得系統易於維護和擴展。
* 缺點: 當同事類太多時，中介者的職責將很大，它會變得複雜而龐大，以至於系統難以維護。
* 網址:

<https://xyz.cinc.biz/2013/07/mediator-pattern.html>

1. State Pattern (狀態模式)：

* 定義:

允許對像在內部狀態發生改變時改變它的行為，對像看起來好像修改了它的類。

* 使用時機: 代碼中包含大量與對象狀態有關的條件語句。
* 目的：象的行為依賴於它的狀態（屬性），並且可以根據它的狀態改變而改變它的相關行為。
* 優點:
  1. 狀態模式將與特定狀態相關的行為局部化到一個狀態中，並且將不同狀態的行為分割開來，滿足“單一職責原則”。
  2. 減少對象間的相互依賴。將不同的狀態引入獨立的對像中會使得狀態轉換變得更加明確，且減少對象間的相互依賴。
  3. 有利於程序的擴展。通過定義新的子類很容易地增加新的狀態和轉換。
* 缺點:
  1. 狀態模式的使用必然會增加系統的類與對象的個數。
  2. 狀態模式的結構與實現都較為複雜，如果使用不當會導致程序結構和代碼的混亂。
* 網址:

<https://dotblogs.com.tw/pin0513/2011/01/16/20842>

1. Memento Pattern (備忘錄模式)：

* 定義:

在不破壞封裝性的前提下，捕獲一個對象的內部狀態，並在該對象之外保存這個狀態。

* 使用時機: 很多時候我們總是需要記錄一個對象的內部狀態，這樣做的目的就是為了允許用戶取消不確定或者錯誤的操作，能夠恢復到他原先的狀態，使得他有"後悔藥"可吃。
* 目的：所謂備忘錄模式就是在不破壞封裝的前提下，捕獲一個對象的內部狀態，並在該對象之外保存這個狀態，這樣可以在以後將對象恢復到原先保存的狀態。
* 優點:
  1. 提供了一種可以恢復狀態的機制。當用戶需要時能夠比較方便地將數據恢復到某個歷史的狀態。
  2. 實現了內部狀態的封裝。除了創建它的發起人之外，其他對像都不能夠訪問這些狀態信息。
  3. 簡化了發起人類。發起人不需要管理和保存其內部狀態的各個備份，所有狀態信息都保存在備忘錄中，並由管理者進行管理，這符合單一職責原則。
* 缺點: 資源消耗大。如果要保存的內部狀態信息過多或者特別頻繁，將會佔用比較大的內存資源。
* 網址:

<https://xyz.cinc.biz/2013/07/memento-pattern.html>

1. Visitor Pattern (訪問者模式)：

* 定義:

主要將數據結構與數據操作分離。

* 使用時機: 需要對一個對象結構中的對象進行很多不同的並且不相關的操作，而需要避免讓這些操作"污染"這些對象的類，使用訪問者模式將這些封裝到類中。
  1. 需要對一個對象結構中的對象進行很多不同的並且不相關的操作，而需要避免讓這些操作"污染"這些對象的類，也不希望在增加新操作時修改這些類。
* 目的：穩定的數據結構和易變的操作耦合問題。
* 優點:
  1. 擴展性好。能夠在不修改對象結構中的元素的情況下，為對象結構中的元素添加新的功能。
  2. 復用性好。可以通過訪問者來定義整個對象結構通用的功能，從而提高系統的複用程度。
  3. 靈活性好。訪問者模式將數據結構與作用於結構上的操作解耦，使得操作集合可相對自由地演化而不影響系統的數據結構。
  4. 符合單一職責原則。訪問者模式把相關的行為封裝在一起，構成一個訪問者，使每一個訪問者的功能都比較單一。
* 缺點:
  1. 增加新的元素類很困難。在訪問者模式中，每增加一個新的元素類，都要在每一個具體訪問者類中增加相應的具體操作，這違背了“開閉原則”。
  2. 破壞封裝。訪問者模式中具體元素對訪問者公佈細節，這破壞了對象的封裝性。
  3. 違反了依賴倒置原則。訪問者模式依賴了具體類，而沒有依賴抽象類。
* 網址:

<https://xyz.cinc.biz/2013/08/visitor-pattern.html>

補充:

Filter Pattern (過濾器模式)：

* 定義:

這種模式允許開發人員使用不同的標準來過濾一組對象，通過邏輯運算以解耦的方式把它們連接起來

* 使用時機: 當我們想要對應用程序的請求或響應進行一些預處理/後處理時，使用截取過濾器設計模式。 在將請求傳遞到實際目標應用程序之前，在請求上定義和應用過濾器。 過濾器可以進行請求的認證/授權/日誌記錄或跟蹤，然後將請求傳遞給相應的處理程序。。
* 目的：允許開發人員使用不同的標準來過濾一組物件，通過邏輯運算以解耦的方式把它們連線起來。
* 優點: 簡單，解耦，使用方便。
* 網址: <https://codertw.com/%E7%A8%8B%E5%BC%8F%E8%AA%9E%E8%A8%80/314955/>

趣味介紹: <https://jimmy0222.pixnet.net/blog/post/37216962>

參考網址:

<https://www.runoob.com/design-pattern/design-pattern-tutorial.html>

<https://jimmy0222.pixnet.net/blog/post/37095632>

<https://kknews.cc/zh-tw/tech/omy5j6.html>