甚麼鬼的IOC & DI

Jamis Liao

- 某**EC**平台程序猿一枚
- **Asp.Net MVC**、**C#**、**DevOps**、軟體開發流程、系統架構、**Cloud**
- **Blog :** 天橋下的說書人 (**https://dotblogs.com.tw/jamis/1**)

Inversion of Control
控制反轉



http://img.anyanother.com/image/no/56880/%E4%BD%A0%E6%83%B3%E8%AA%AA%E4%BB%80%E9%BA%BC%EF%BC%9F/

調用對象的控制權由
代碼本身移到外部對象



http://www.wxrw123.com/ms/20160331/316059.html

先不要管甚麼IOC了(誤……

你有聽過DIP嗎



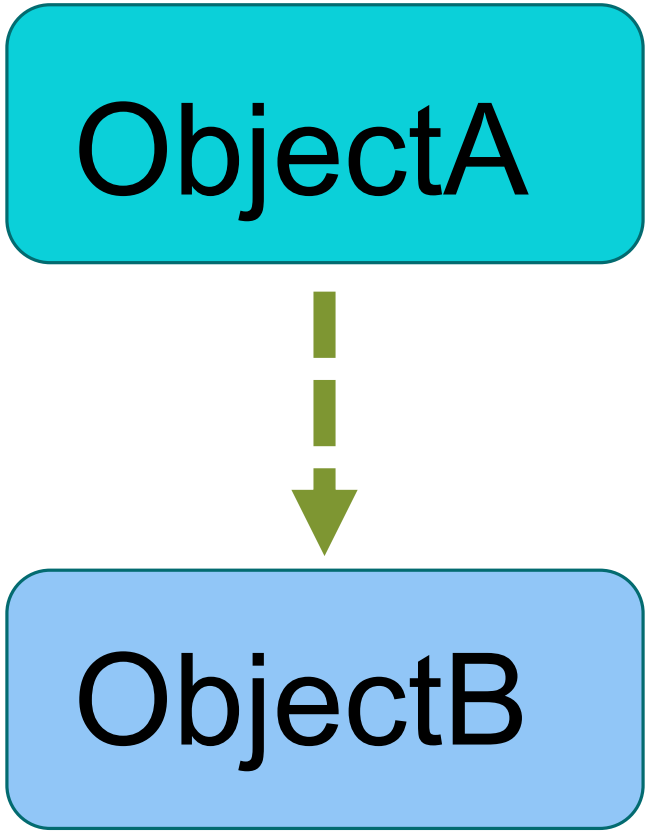
DIP

- **Dependence Inversion Principle**
- 依賴反轉原則
- 高階模組不應依賴低階模組，兩者都應該依賴抽象概念

我們最常看到的物件使用方式

```
public class ObjectA
{
    0 個參考|0 項變更|0 位作者，0 項變更
    public void DoSomething()
    {
        ObjectB objB = new ObjectB();
        objB.DoSomething();
    }
}
```

我們最常看到的物件使用方式



直接相依，高耦合

- 優點
  - ~~真他媽的~~快，又方便
- 缺點
  - 新增功能或修改功能時，缺少彈性
  - 容易改A壞B

當ObjectB換成ObjectC

```
public class ObjectA
{
    0 個參考|0 項變更|0 位作者，0 項變更
    public void DoSomething()
    {
        ObjectC objC = new ObjectC();
        objC.DoSomething();
    }
}
```

8行的程式碼改了2行，
如果你有100行的程式
碼……你會



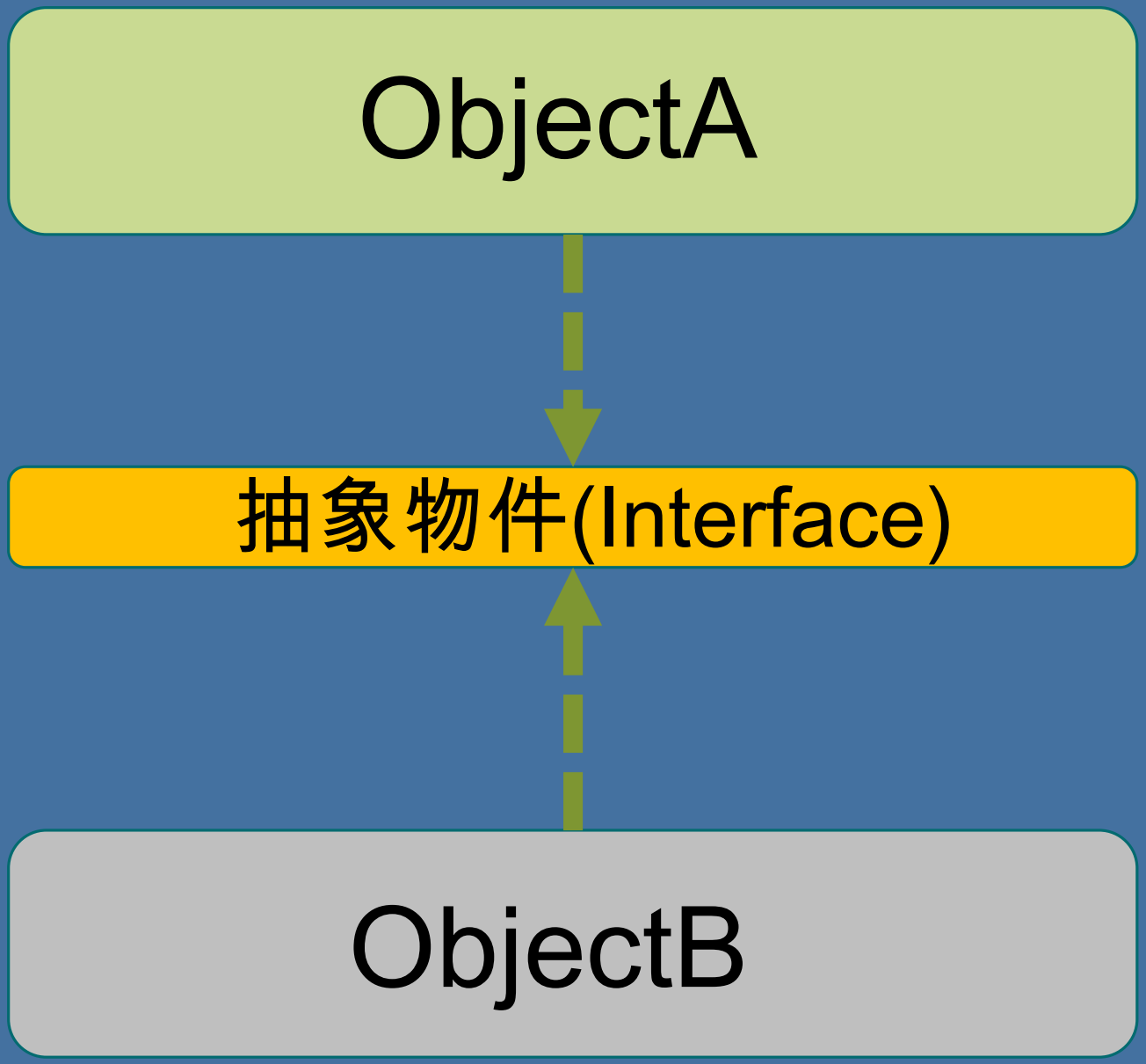
http://img.anyanother.com/image/no/54973/%E5%B9%B9%EF%BC%81%E9%A8%99%E6%88%91/

想想剛剛的**DIP**，高階模組不應依賴低階模組，兩者應依賴抽象概念

把ObjectB，ObjectC
換成Interface

```
public void DoSomething()
{
    IObject obj = new ObjectB();
    obj.DoSomething();
}
```

```
public void DoSomething()
{
    IObject obj = new ObjectC();
    obj.DoSomething();
}
```


ObjectA
抽象物件(Interface)
ObjectB

還是需要修改一行程式碼
身為一個懶惰工程師這是不允許的

如果可以有一個元件，當程式碼需要某個物件就產生某個物件那有多好

既然想要就來做一個吧

在Live coding之前
介紹一下使用到的技巧

工廠模式

- **Creational Patterns**的其中一種，專門用來生產”東西”的
- 程式碼要什麼，它就給甚麼

反射

- 反射（**Reflection**） 會提供**Type**型別的物件， 用來描述組件、模組和型別
- 反射可以
  - 動態建立型別的執行個體
  - 型別繫結至現有物件，
  - 從現有物件取得型別
  - 叫用其方法
  - 存取其欄位和屬性

反射

- 在**Live coding**中，會使用到
  - 動態載入**Assembly**
  - 取得建構式清單
  - 取得建構式所需要的傳入參數
  - 動態建立型別的執行個體



demo

Live Coding

- 專注於目前程式需要的物件
- 隱藏了物件產生的過程與需要的元素
- **少寫了好幾行程式碼**



http://img.anyanother.com/image/no/55564/%E6%93%8D%EF%BC%81%E8%AC%9B%E9%82%A3%E9%BA%BC%E5%A4%9A%E5%BB%A2%E8%A9%B1%E5%B9%B9%E5%97%8E%EF%BC%9F/

IOC

- 調用對象的控制權由代碼本身移到外部對象

IOC

- 調用對象的控制權由代碼本身移到外部對象

就是程式碼要New的對象

IOC

- 調用對象的控制權由代碼本身移到外部對象

調用對象的創建者

IOC

- 調用對象的控制權由代碼本身移到外部對象

就像是前面提到的工廠

IOC

- 調用對象的控制權由代碼本身移到外部對象

```
[TestMethod]
0 個參考|jamisliao, 1 天前|1 位作者，1 項變更
public void 工廠模式加建構式自動掃描參數傳入()
{
    IPrintService printService =
        Factory.GetInstanceAutoInjection(typeof(IPrintService)) as IPrintService;
    printService.Print("工廠模式加建構式自動掃描參數傳入");
}
```

IOC

- 實現方式
  - **Dependency Injection(依賴注入)**
  - **Service Locator**

Dependency Injection

- 外部注入需要的元件到物件中，而不必在物件中明確地建立相依的元件
  - 屬性注入
  - 建構式注入
  - 參數注入

屬性注入

```
public class PrintService : IPrintService
{
    3 個參考|0/1 通過|0 項變更|0 位作者，0 項變更
    public IPrintHelper PrintHelper { get; set; }

    2 個參考|0/2 通過|0 項變更|0 位作者，0 項變更
    public PrintService()
    {
    }

    8 個參考|0/7 通過|jamisliao, 53 天前|1 位作者，1 項變更
    public void Print(string outputStr)
    {
        this.PrintHelper.Print(outputStr);
    }
}
```

屬性注入

```
public void 工廠模式加設定屬性()
{
    IPrintService printService =
        FactoryObj.GetInstance(typeof(PrintService)) as PrintService;
    IPrintHelper printHelper =
        FactoryObj.GetInstance(typeof(ConsoleHelper)) as ConsoleHelper;
    printService.PrintHelper = printHelper;
    printService.Print("工廠模式加設定屬性");
}
```

建構式注入

```
public class PrintService : IPrintService
{
    private IPrintHelper _printHelper;

    2 個參考|0/2 通過|0 項變更|0 位作者，0 項變更
    public PrintService(IPrintHelper printHelper)
    {
        this._printHelper = printHelper;
    }

    8 個參考|0/7 通過|jamisliao, 53 天前|1 位作者，1 項變更
    public void Print(string outputStr)
    {
        this._printHelper.Print(outputStr);
    }
}
```

建構式注入

```
public void 工廠模式加加建構職參數傳入()
{
    IPrintHelper printHelper = new ConsoleHelper();
    IPrintService printService = new PrintService(printHelper);
    printService.Print("工廠模式加加建構職參數傳入");
}
```

參數注入

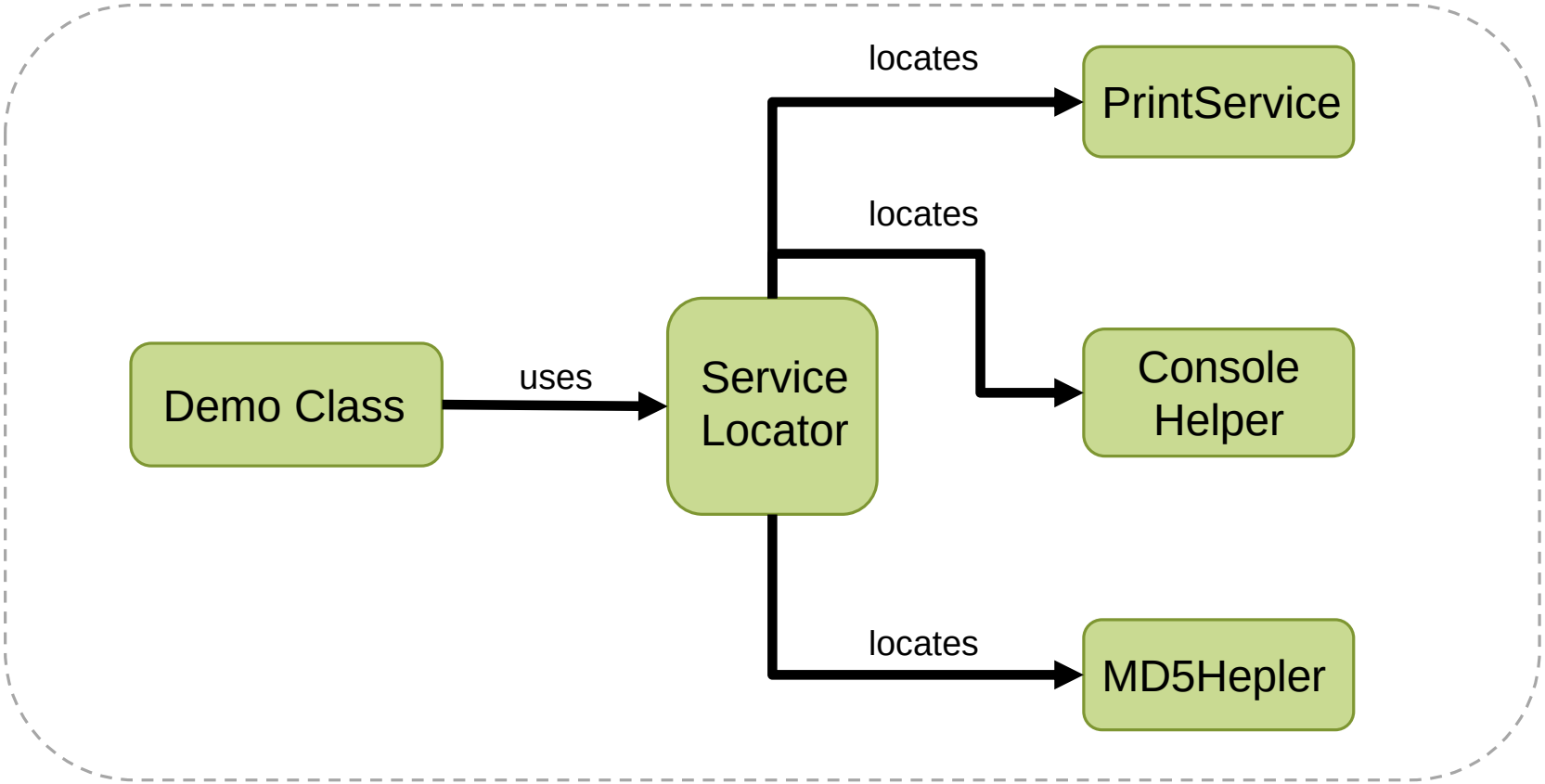
```
public class PrintService : IPrintService
{
    3 個參考|0/3 通過|0 項變更|0 位作者，0 項變更
    public PrintService()
    {
    }

    8 個參考|0/7 通過|0 項變更|0 位作者，0 項變更
    public void Print(IPrintHelper printHelper, string outputStr)
    {
        printHelper.Print(outputStr);
    }
}
```

參數注入

```
public void 工廠模式加加建構職參數傳入()
{
    IPrintHelper printHelper =
        FactoryObj.GetInstance(typeof(ConsoleHelper)) as ConsoleHelper;
    IPrintService printService =
        FactoryObj.GetInstance(typeof(PrintService)) as PrintService;
    printService.Print(printHelper, "工廠模式加加建構職參數傳入");
}
```

Service Locator



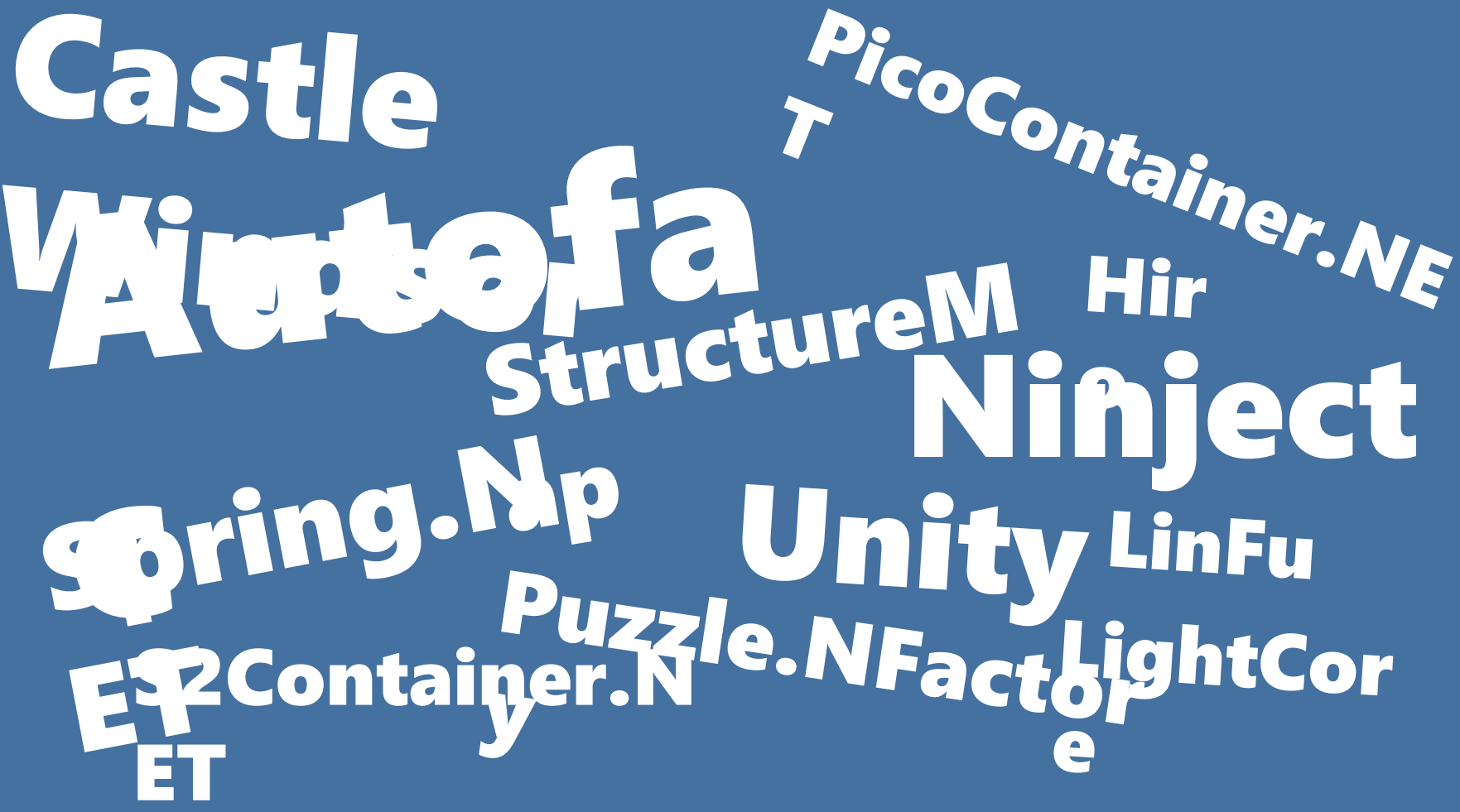
https://msdn.microsoft.com/zh-tw/library/ff648968.aspx

Service Locator

- **Abstract Factory**
  - **Generic type**
  - **Return type of the Create method is determined by the type of the factory itself**
- **Service Locator**
  - **non-generic interface with a generic method**
  - **Create method of a single Service Locator can return instances of an infinite number of types**

雖然知道IOC怎麼寫了
千萬不要自己手刻IOC

.Net Ioc Framework



.Net Ioc Framework

- IOC套件有胖有瘦，使用上有好有壞，使用前請詳閱公開說明書

史上最強的注音輸入法



千呼萬喚

MAC 平台上，最聰明的注音輸入法
現在立即前往購買 ！

樂在Coding 愛上MVC

完全支援

請放心在 Windows 8 上使用 新自然輸入法 10 系統升級免煩惱，平板模式也沒問題！

Blog 是記錄知識的最佳平台



感謝KKTIX贊助活動報名平台



感謝 KKTIX 贊助 twMVC 活動報名平台

感謝 Jetbrains 贊助贈品



ReSharper

https://www.jetbrains.com/resharper/

感謝 OzCode 贊助贈品



http://www.oz-code.com/

感謝 ALIVE 贊助贈品



https://comealive.io/

業界師資、實戰教學



http://skilltree.my

謝謝各位

http://mvc.tw

- 本投影片所包含的商標與文字皆屬原著作者所有。
- 本投影片使用的圖片皆從網路搜尋。
- 本著作係採用姓名標示-非商業性-相同方式分享 3.0 台灣授權。閱讀本授權條款，請到 http://creativecommons.org/licenses/by-nc-sa/3.0/tw/，或寫信至Creative Commons, 444 Castro Street, Suite 900, Mountain View, California, 94041, USA.