twMVC#18 – 專案分層架構

Kevin Tseng



簡介

- 微軟最有價值專家（MVP）2013~2015
- twMVC 核心成員及講師 （http://mvc.tw）
- SkillTree 專任講師 （http://skilltree.my）
- 部落格「mrkt 的程式學習筆記」
  http://kevintsengtw.blogspot.tw
- Google+ 專頁
  https://plus.google.com/+KevintsengtwBlogspot
- Github
  https://github.com/kevintsengtw

twMVC 相關架構的主題

twMVC #05

ASP.NET MVC之實戰架構探討

MVC真的不是只有簡單的Model、View、Controller三層喔！



Bibby Chung
http://bibby.be

http://goo.gl/k0PNWe

twMVC #10


ASP.NET MVC Model 的設計與使用
twMVC
Kevin Tseng
http://kevintsengtw.blogspot.tw
MVP
Microsoft
Most Valuable
Professional

twMVC #11


架構設計入門
Clark
2013/10/19
MVC
MVP
Microsoft
Most Valuable
Professional


動手打造 Application Framework
kevin@kuicker.org
tw MVC

ASP.NET MVC 專案分層架構系列

ASP.NET MVC 專案分層架構系列

- Part.1 初學者的起手式
- Part.2 抽出 Repository 裡相同的部分
- Part.3 個別 Repository 的資料存取操作
- Part.4 抽出 Model 層並建立為類別庫專案
- Part.5 建立 Service 層
- Part.6 DI/IoC 使用 Unity.MVC
- 建議與補充說明

ASP.NET MVC 專案分層架構系列

- 很多人看，但是很多人不懂
- 很多人跟著做，但是做了之後還是不懂
- 有些公司、團隊也照著做，但是只做了一半
- 重要的部分，我還沒有寫出來
- 不建議程式開發初學者閱讀
- 不瞭解物件導向的開發人員，不建議閱讀

分層與架構

架構、分層

- 架構：

  系統架構

  軟體架構

- 分層：

  系統架構的分層 – Tier (實體層)

  軟體架構的分層 – Layer (邏輯層)

架構 Architecture

- 合適的架構會讓系統有更好的靈活性和伸縮性，
  並且會使開發人員進入良性循環
- 不合適的架構則很容易導致軟體在變化的需求中崩潰

軟體架構 – Software Architecture

- Martin Fowler

  最高層次的系統分解，系統中不易改變的決定。

- Ralph Johnson

  高階開發人員對系統設計的一些可共享的理解，這些可共享的理解表現為系統中主要的組成部分以及這些組成之間的交互關係。

- 架構是必須在開發初期所做出的一組設計與決策

  （這些內容也是在開發後期難以去做改變）

系統架構的分層 – Tier（實體層）

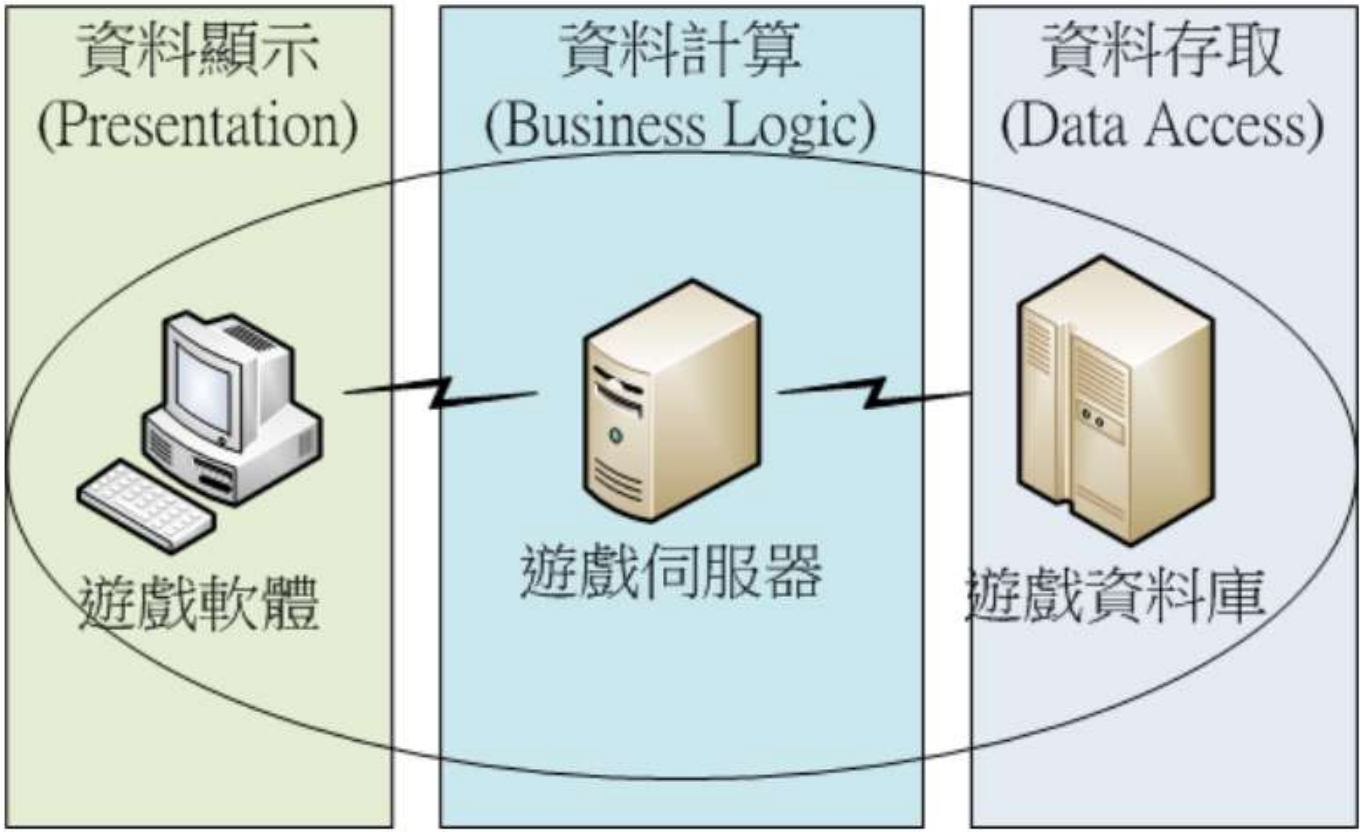
- Physical Tier
- 每一層多半代表著一個獨立的實體
- 三層式的系統：

  展現層：客戶端的電腦

  邏輯運算層：伺服器

  資料存取層：資料庫伺服器
- 每一層都會有對應的硬體與系統軟體
- 不同的實體層有時也會存在於同一台伺服器上

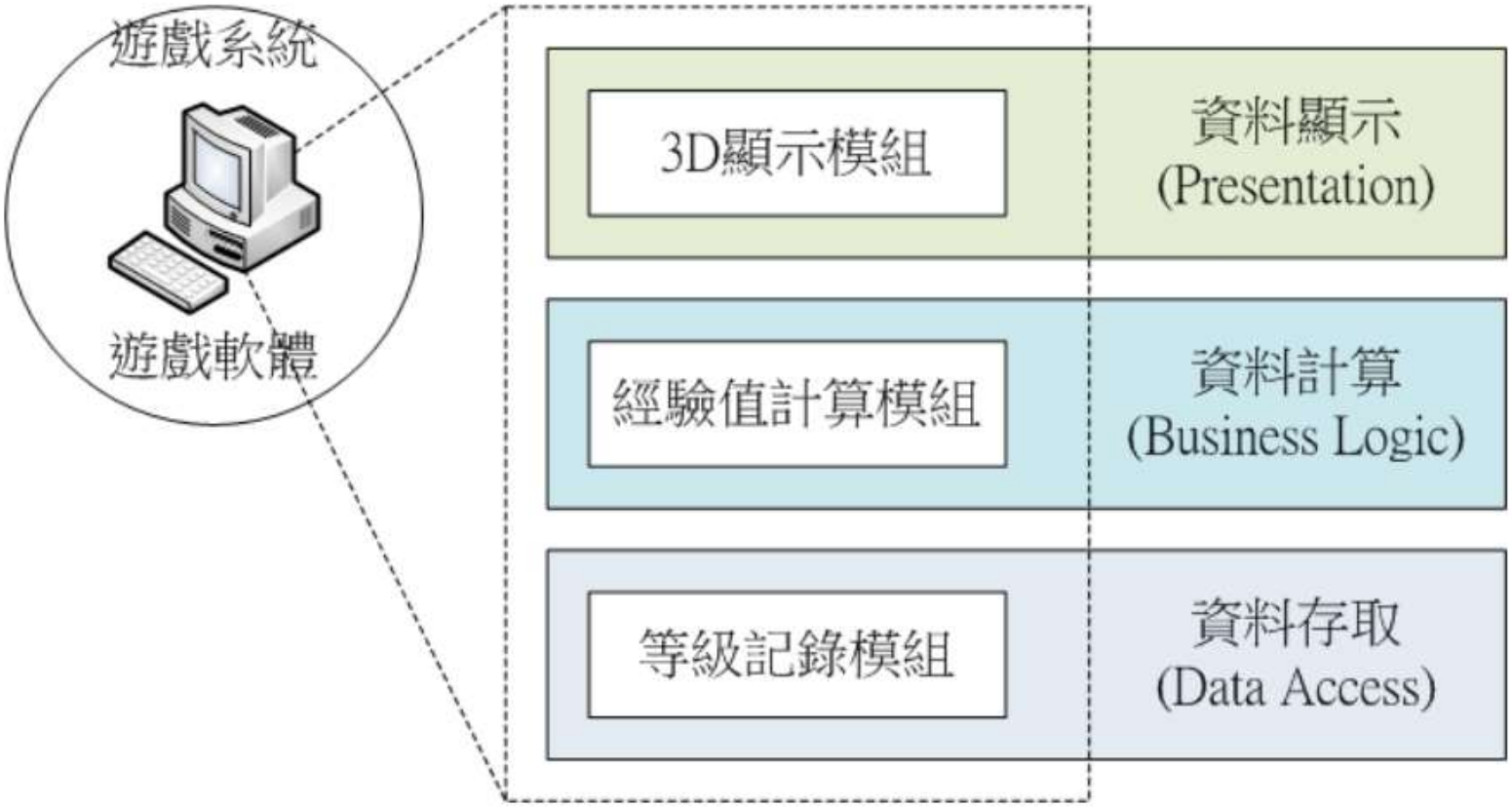
系統架構的分層 – Tier（實體層）



軟體架構的分層 – Layer（邏輯層）

- Logical Layer
- 不牽涉到硬體，偏向軟體的設計，或根據不同任務來區分
- 實體層會包含一到多個邏輯層
- 根據不同的關注點定義程式碼所在的位置
- 明確的分離與相鄰邏輯層之間的互動關係，透過介面

軟體架構的分層 – Layer（邏輯層）



分層 - Layer

- 用以分解複雜的軟體系統設計
- 每一層都依拖在其下層之上，上層使用了下層定義的各種服務，而下層對上層如何使用並不需要瞭解。
- 每一層對自己的上層隱藏其下層的細節。



分層 - Layer

- 分層是企業應用系統中最常見的一種架構模式
- 將系統以橫向切分幾個部分，每個部分負責一個相對比較單一的職責，通過上層對下層的依賴和調用來組成一個完整的系統



分層 – 好處

- 可以替換某一層的具體實現，只要前後提供的服務相同即可
- 可以將各層次間的相依性減到最低
- 分層有利於標準化
- 合理的分層有助於分工開發與維護
- 各層之間具有一定的獨立性，維持之間溝通的介面不變，各層可以根據具體問題各自實作，而不需要其他層也必須做出相應調整

分層 – 壞處

- 過多的分層會影響效能
- 不合理的分層將會導致軟體開發品質下降
- 不合理的分層使得不同層次的程式碼耦合在一起，相依性增加，使得程式越來越難以隨著需求變更而進行相應調整
- 分層無法封裝所有的東西，例如因應需求要在前端介面增加輸入欄位，必須要在表現層、商業邏輯層、資料存取層都做相對應的修改

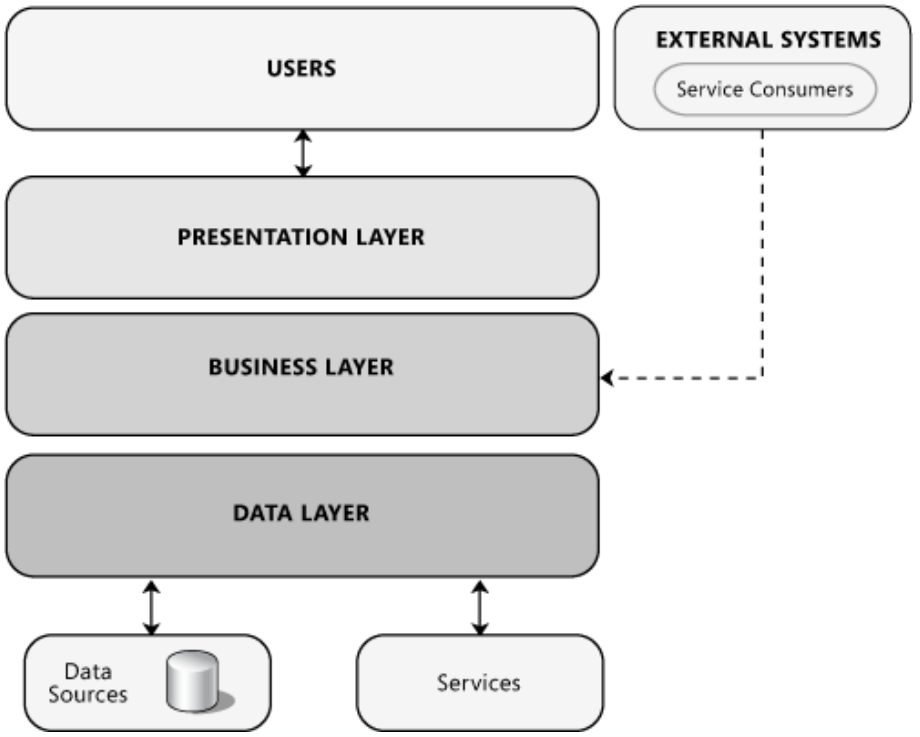
分層

- 合理的分層，什麼分層的需求變更就去修改那一個分層的程式碼，而與其他分層無關

- Client/Server
- 3 Tiers / 3 Layers

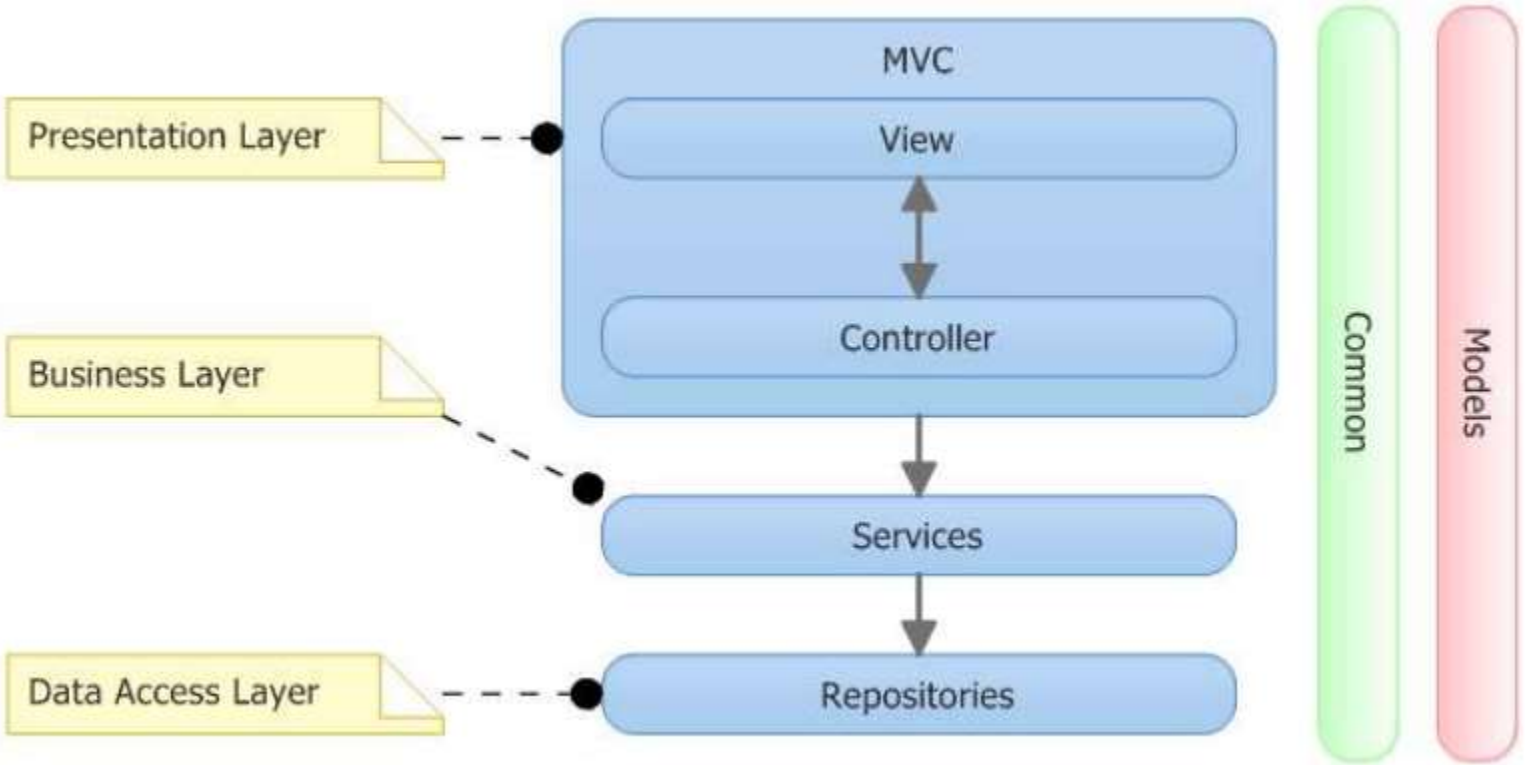
三層式架構 3 Layers

- 展現層 Presentation Layer
- 商業邏輯層 Business Layer
- 資料存取層 Data Layer



https://msdn.microsoft.com/en-us/library/ff650706.aspx

ASP.NET MVC with N-Layer Architecture



ASP.NET MVC with N-Layer Architecture

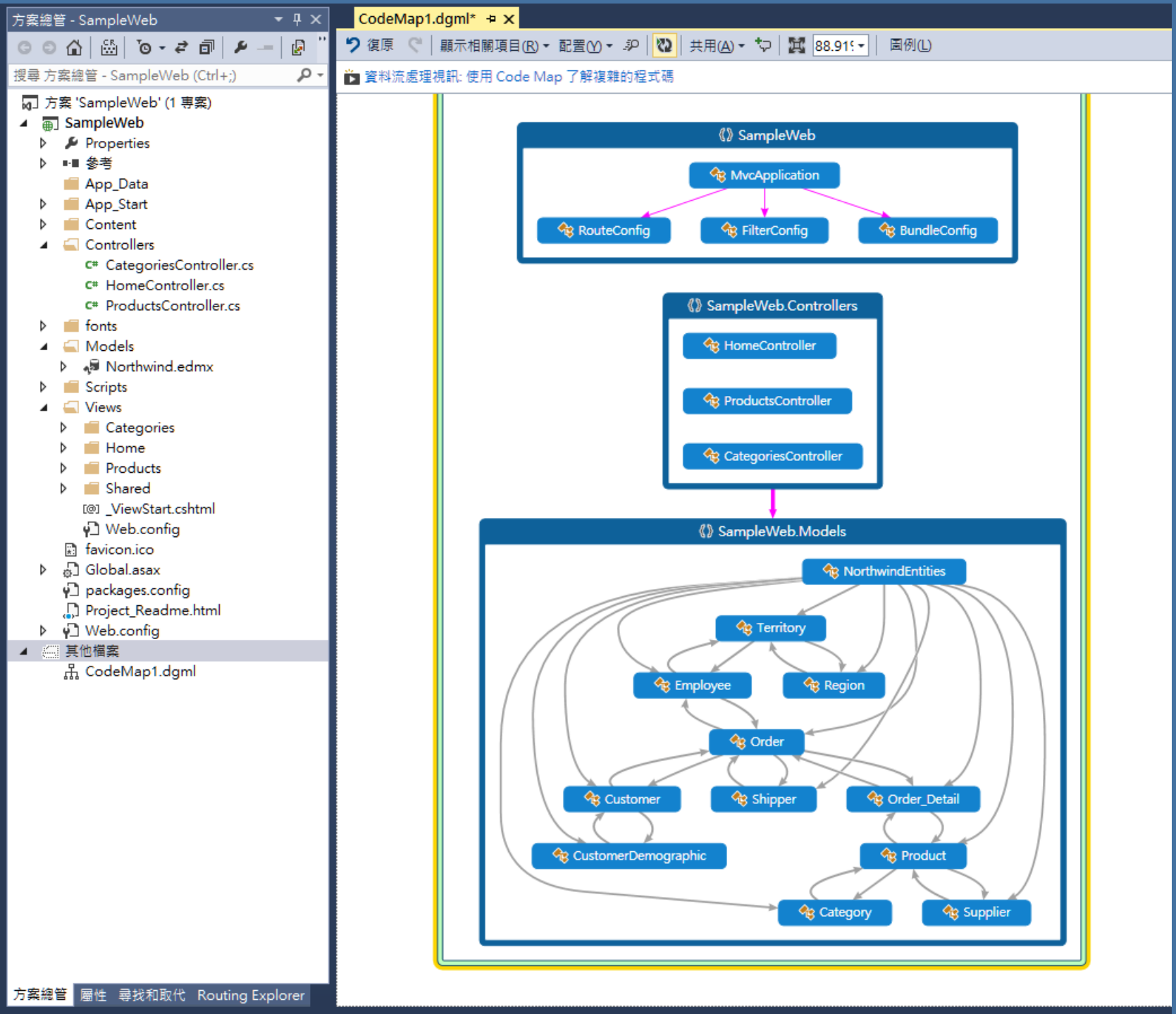
- M-V-C 三個組成部分不等於三層式架構
- ASP.NET MVC 專案可視為表現層
- 分層的問題：要決定建立哪些層以及每一層的職責為何？

  應該要如何分層?

  要分那幾層？

  什麼部分應該放在那一層?

專案分層


方案總管 - SampleWeb
搜尋 方案總管 - SampleWeb (Ctrl+;)
方案 'SampleWeb' (1 專案)
SampleWeb
Properties
參考
App_Data
App_Start
Content
Controllers
CategoriesController.cs
HomeController.cs
ProductsController.cs
fonts
Models
Northwind.edmx
Scripts
Views
Categories
Home
Products
Shared
_ViewStart.cshtml
Web.config
favicon.ico
Global.asax
packages.config
Project_Readme.html
Web.config
其他檔案
CodeMap1.dgml
方案總管
屬性
尋找和取代
Routing Explorer
CodeMap1.dgml*
復原
顯示相關項目(R)
配置(Y)
共用(A)
圖例(L)
資料流處理視訊: 使用 Code Map 了解複雜的程式碼
SampleWeb
MvcApplication
RouteConfig
FilterConfig
BundleConfig
SampleWeb.Controllers
HomeController
ProductsController
CategoriesController
SampleWeb.Models
NorthwindEntities
Territory
Employee
Region
Order
Customer
Shipper
Order_Detail
CustomerDemographic
Product
Category
Supplier



demo

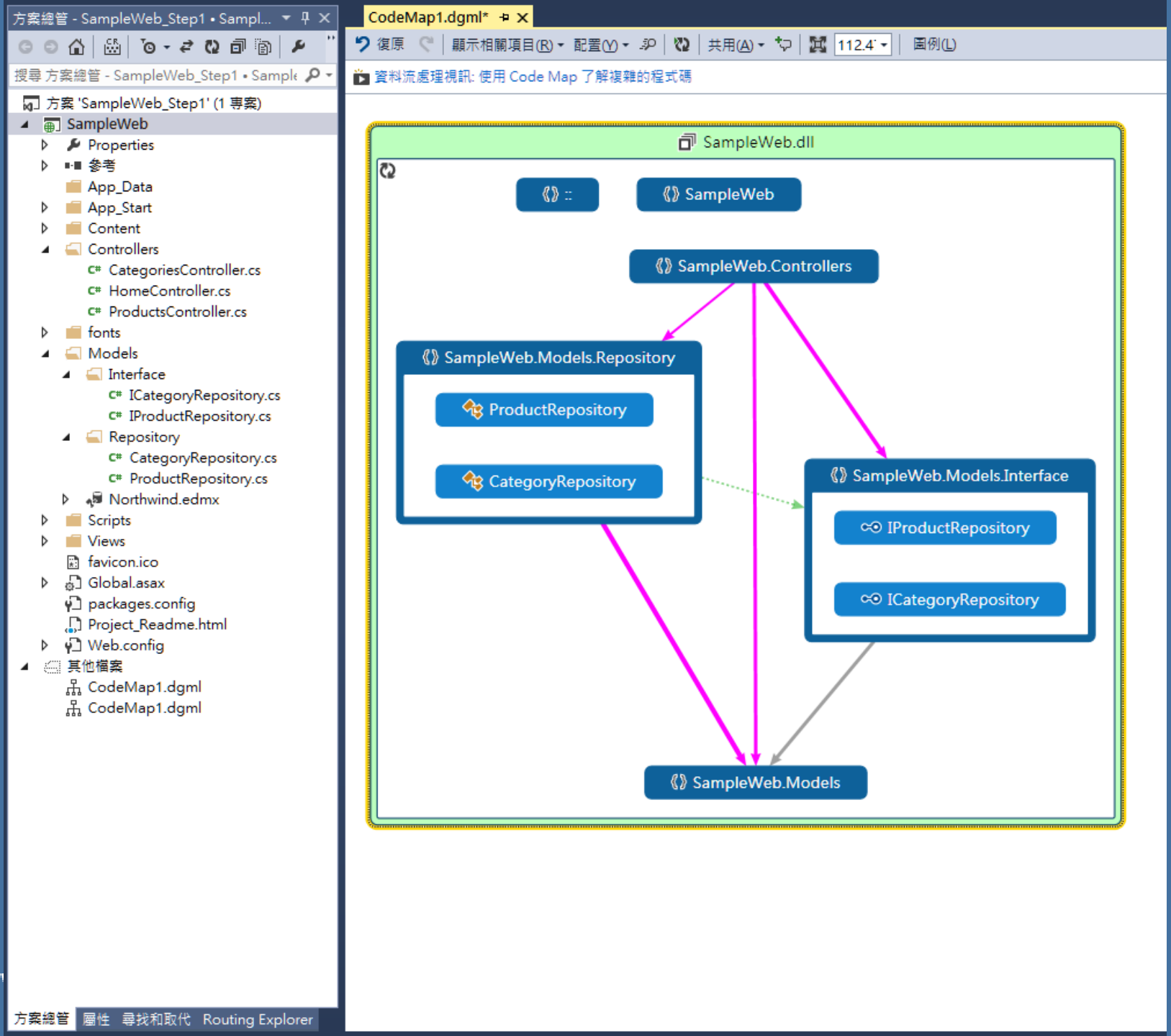
專案分層 Part.1
初學者的起手式

Part.1 初學者的起手式

- Visual Studio 2013 with Update 4
- Microsoft SQL Server 2014 Express
- ASP.NET MVC 5, Entity Framework 6.1.3
- Database: Northwind

Part.1 初學者的起手式

- 建立 Repository
  1. 建立 ICategoryRepository, IProductRepository
  2. 實作 ICategoryRepository, IProductRepository
  3. 在 Controller 裡使用 Repository


方案總管 - SampleWeb_Step1 • Sampl...
搜尋 方案總管 - SampleWeb_Step1 • Sample
方案 'SampleWeb_Step1' (1 專案)
SampleWeb
Properties
參考
App_Data
App_Start
Content
Controllers
CategoriesController.cs
HomeController.cs
ProductsController.cs
fonts
Models
Interface
ICategoryRepository.cs
IProductRepository.cs
Repository
CategoryRepository.cs
ProductRepository.cs
Northwind.edmx
Scripts
Views
favicon.ico
Global.asax
packages.config
Project_Readme.html
Web.config
其他檔案
CodeMap1.dgml
CodeMap1.dgml
方案總管
屬性
尋找和取代
Routing Explorer
CodeMap1.dgml*
復原
顯示相關項目(R)
配置(Y)
共用(A)
112.4
圖例(L)
資料流處理視訊: 使用 Code Map 了解複雜的程式碼
SampleWeb.dll
::
SampleWeb
SampleWeb.Controllers
SampleWeb.Models.Repository
ProductRepository
CategoryRepository
SampleWeb.Models.Interface
IProductRepository
ICategoryRepository
SampleWeb.Models

Part.1 初學者的起手式

- 關注點分離 SoC, Separation of Concers
- 資料倉儲模式 Repository Pattern
- 單一職責原則 SRP

  Single Responsibility Principle
- Do not Repeat Yourself (DRY)
- 分層架構不一定要分好幾個專案來操作，在同一個網站專案中也是可以做出的分層架構（目錄）

Repository 資料倉儲（資源庫）模式

- **by Edward Hieatt and Rob Mee**
- *Mediates between the domain and data mapping layers using a collection-like interface for accessing domain objects.*
- 協調領域和資料對映層，利用類似於集合的接口來訪問領域物件。

Repository 資料倉儲（資源庫）模式

- Repository 是領域層與資料對應層之間的媒介，如同存在記憶體中的物件集合。

  用戶端會建構查詢規格（query specification），並傳遞給 Repository 以執行特定的資料操作.....

  概念上，Repository 將資料與其相關操作封裝成物件集合，以便用貼近物件導向的方式來存取資料

單一職責原則 SRP

- Single Responsibility Principle
- 應該有，且只有一個原因，會引起class的變更
- 一個class只負責一個職責的內容

  一個class如果要負責多個職責，其內容應該只是組合多個class，而每個被組合的class只負責單一職責，且通常這樣的組合都是透過接口(interface/abstract)

單一職責原則 SRP

- Single Responsibility Principle

(1)class複雜性降低，因為實現什麼職責都有很清楚的定義

(2)可讀性提高

(3)可維護性提高

(4)變更引起的風險降低，變更範圍降低，擴展性提高

單一職責，也就是高內聚力的實作方式。

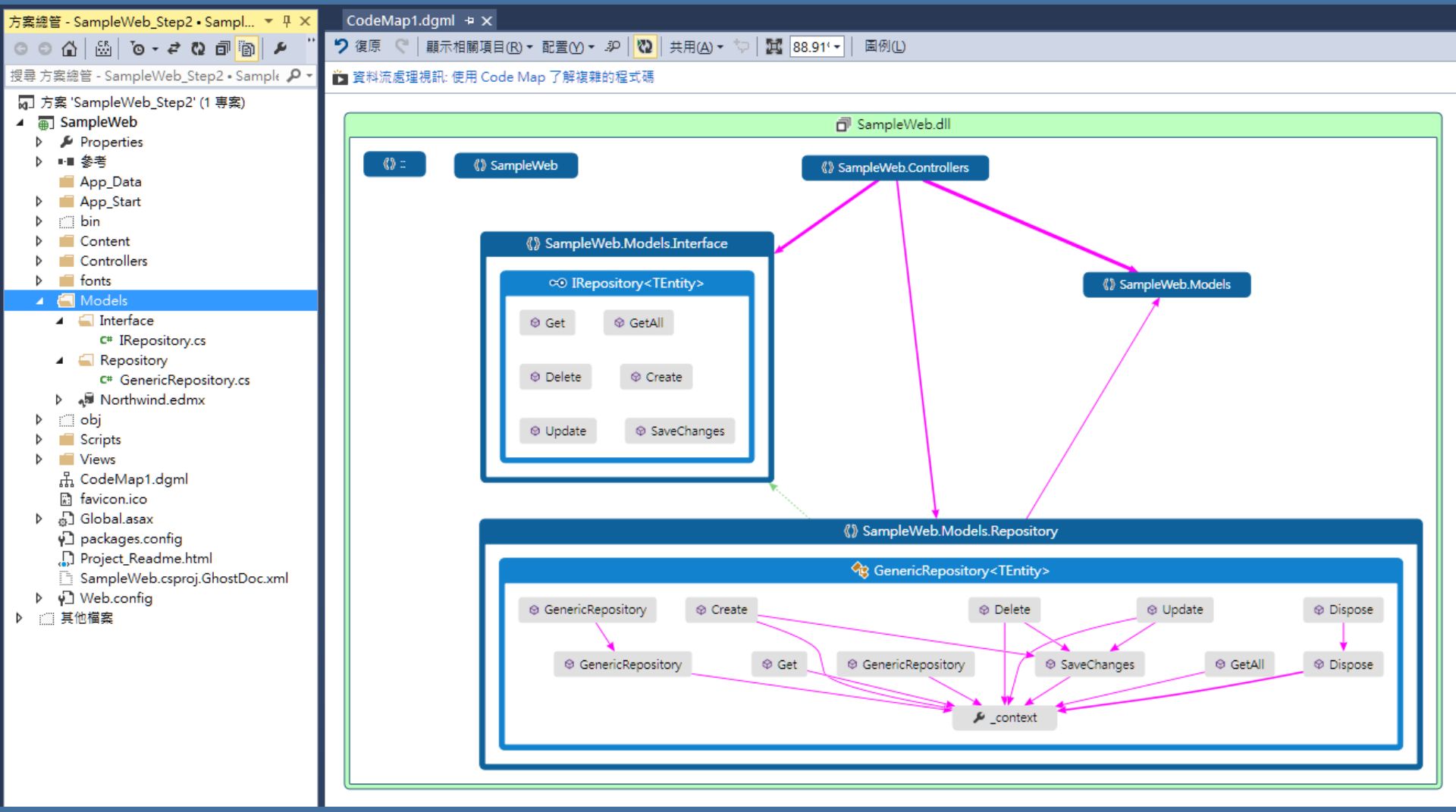


demo

專案分層 Part.2
抽出 Repository 裡相同的部份

Part.2 抽出 Repository 裡相同的部份

- 資料操作方法（CRUD）除了操作的類別不同之外，
  其餘的幾乎是一樣的
- **將共同的部份給抽離出來另外建立 IRepository 介面**
- 使用「**泛型**」的特性
- 建立通用的 Repository 類別（GenericRepository），
  並且繼承實作 IRepository


方案總管 - SampleWeb_Step2 • Sampl...
搜尋 方案總管 - SampleWeb_Step2 • Sample
方案 'SampleWeb_Step2' (1 專案)
SampleWeb
Properties
參考
App_Data
App_Start
bin
Content
Controllers
fonts
Models
Interface
IRepository.cs
Repository
GenericRepository.cs
Northwind.edmx
obj
Scripts
Views
CodeMap1.dgml
favicon.ico
Global.asax
packages.config
Project_Readme.html
SampleWeb.csproj.GhostDoc.xml
Web.config
其他檔案
CodeMap1.dgml
復原
顯示相關項目(R)
配置(Y)
共用(A)
88.91
圖例(L)
資料流處理視訊: 使用 Code Map 了解複雜的程式碼
SampleWeb.dll
SampleWeb
SampleWeb.Controllers
SampleWeb.Models.Interface
IRepository<TEntity>
Get
GetAll
Delete
Create
Update
SaveChanges
SampleWeb.Models
SampleWeb.Models.Repository
GenericRepository<TEntity>
GenericRepository
Create
Delete
Update
Dispose
GenericRepository
Get
GenericRepository
SaveChanges
GetAll
Dispose
_context

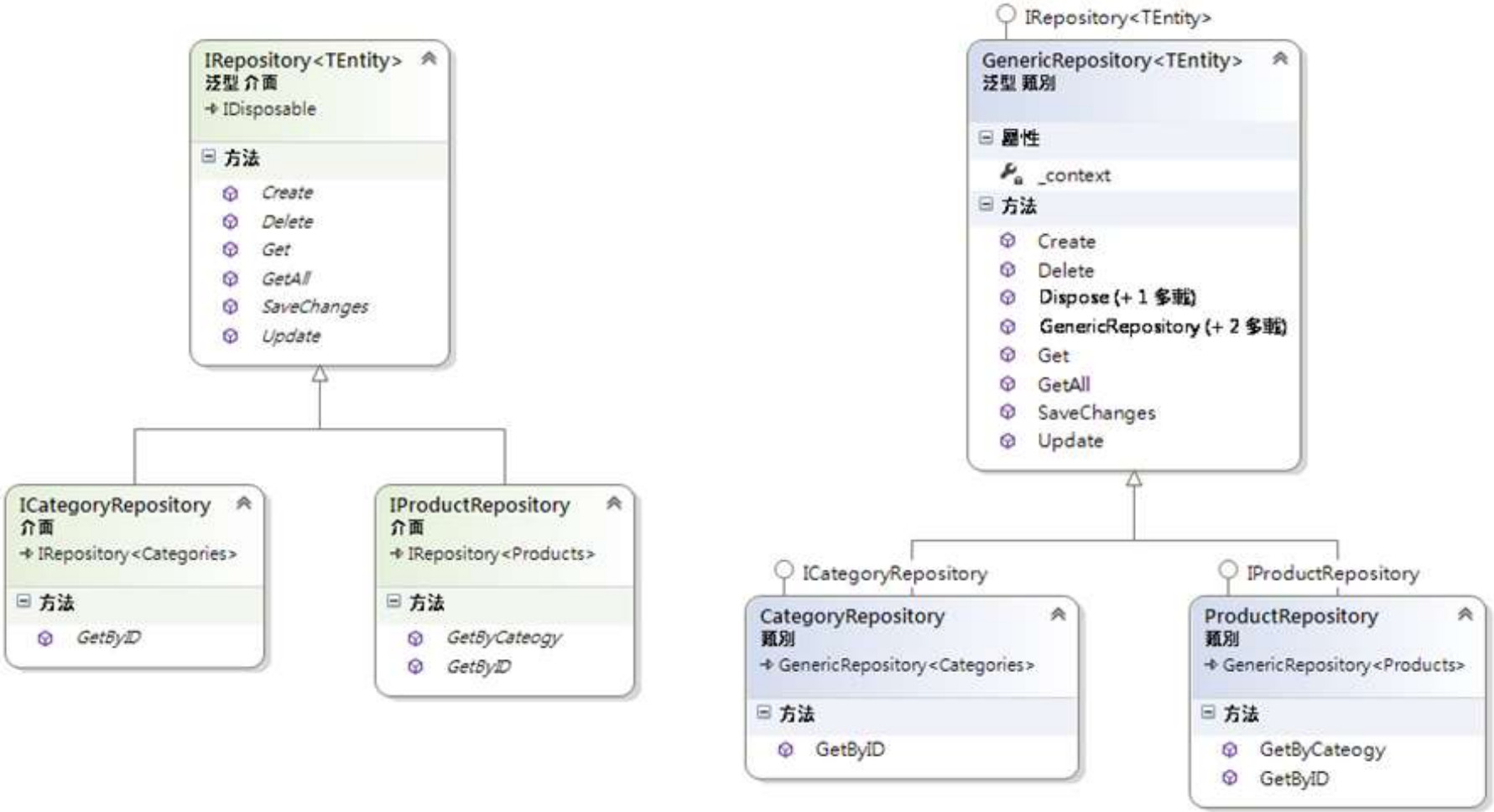


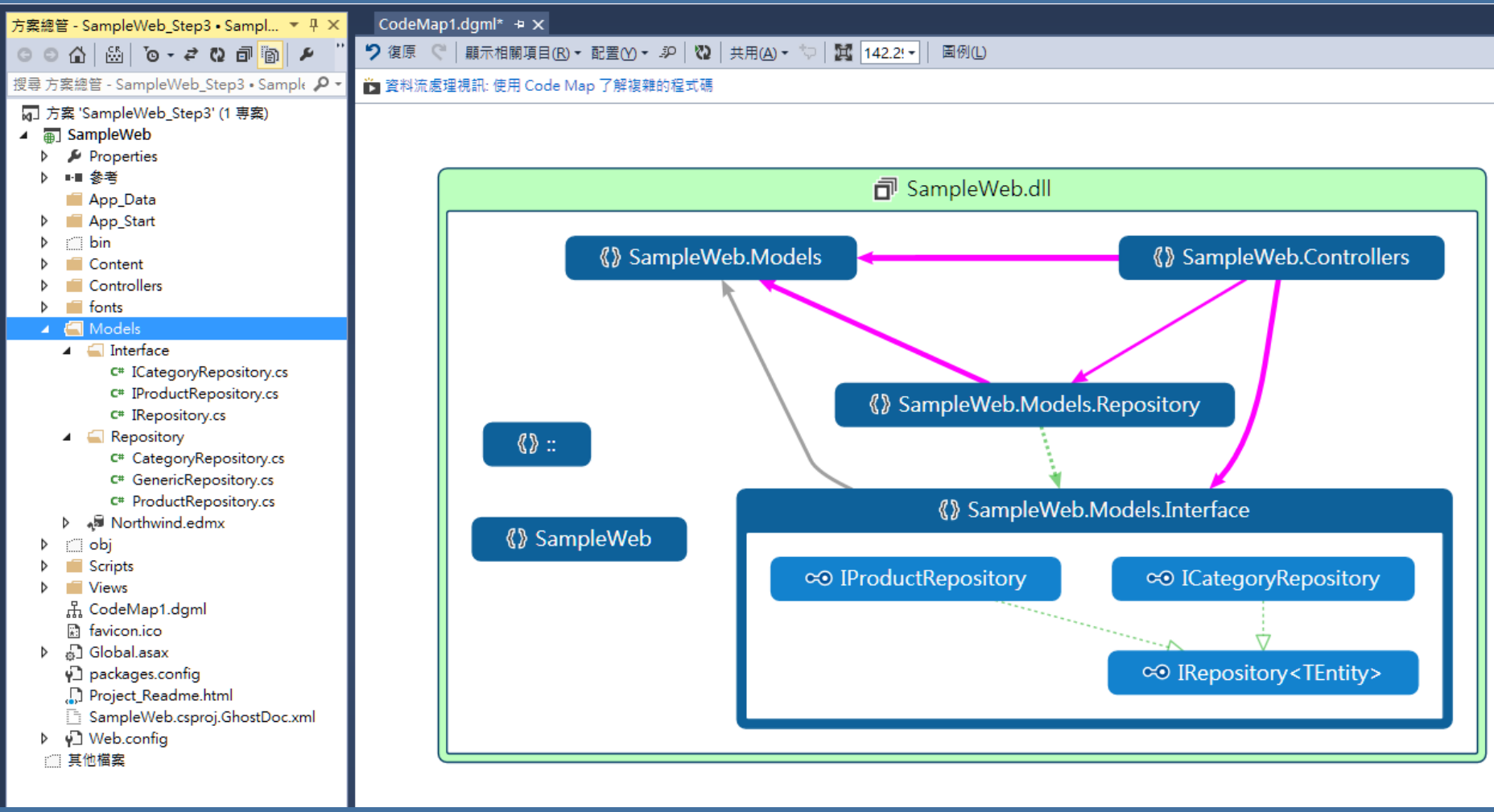
demo

專案分層 Part.3
個別 Repository 的資料存取操作

Part.3 - 個別 Repository 的資料存取操作

- 當資料的操作有別於 GenericRepository 時，
  在各自的 Repository 去加入特定的方法
- 重新建立
  ICategoryRepository, IProductRepository
- 重新建立繼承實作介面
  CategoryRepository, ProductRepository


IRepository<TEntity>
泛型 介面
IDisposable
方法
Create
Delete
Get
GetAll
SaveChanges
Update
ICategoryRepository
介面
IRepository<Categories>
方法
GetByID
IProductRepository
介面
IRepository<Products>
方法
GetByCateogy
GetByID
IRepository<TEntity>
GenericRepository<TEntity>
泛型 類別
屬性
_context
方法
Create
Delete
Dispose (+ 1 多載)
GenericRepository (+ 2 多載)
Get
GetAll
SaveChanges
Update
ICategoryRepository
CategoryRepository
類別
GenericRepository<Categories>
方法
GetByID
IProductRepository
ProductRepository
類別
GenericRepository<Products>
方法
GetByCateogy
GetByID


方案總管 - SampleWeb_Step3 • Sampl...
搜尋 方案總管 - SampleWeb_Step3 • Sample
方案 'SampleWeb_Step3' (1 專案)
SampleWeb
Properties
參考
App_Data
App_Start
bin
Content
Controllers
fonts
Models
Interface
ICategoryRepository.cs
IProductRepository.cs
IRepository.cs
Repository
CategoryRepository.cs
GenericRepository.cs
ProductRepository.cs
Northwind.edmx
obj
Scripts
Views
CodeMap1.dgml
favicon.ico
Global.asax
packages.config
Project_Readme.html
SampleWeb.csproj.GhostDoc.xml
Web.config
其他檔案
CodeMap1.dgml*
復原
顯示相關項目(R)
配置(Y)
共用(A)
142.2!
圖例(L)
資料流處理視訊: 使用 Code Map 了解複雜的程式碼
SampleWeb.dll
SampleWeb.Models
SampleWeb.Controllers
SampleWeb.Models.Repository
::
SampleWeb
SampleWeb.Models.Interface
IProductRepository
ICategoryRepository
IRepository<TEntity>

Part.3 - 個別 Repository 的資料存取操作

- 為何又要重新建立在 Part.2 已經移除的介面和實作呢？



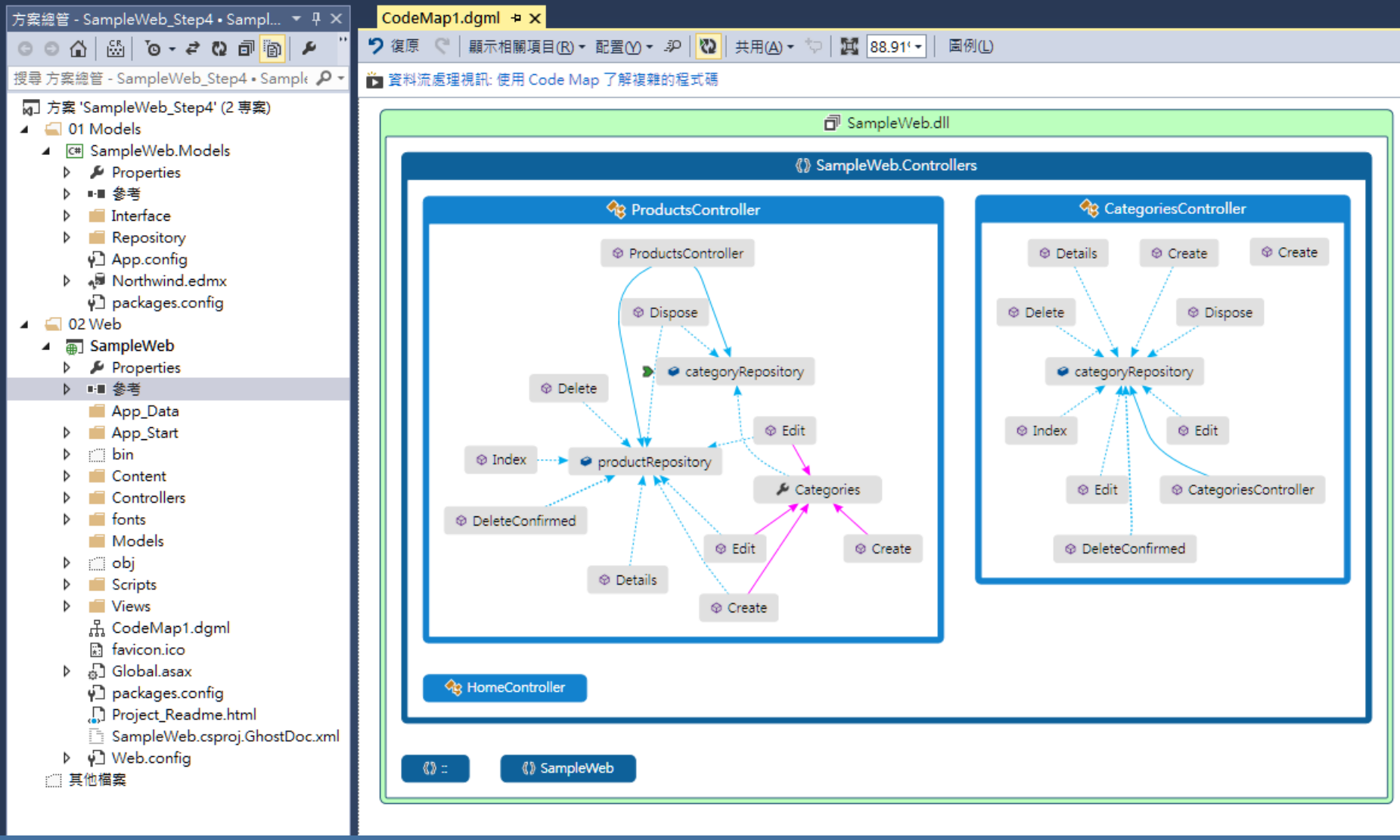


demo

專案分層 Part.4
抽出 Model 層並建立為類別庫專案

Part.4 - 抽出 Model 層並建立為類別庫專案

- 如果專案不大或並不是很複雜，其實可以不必抽出 Model


方案總管 - SampleWeb_Step4 • Sampl...
搜尋 方案總管 - SampleWeb_Step4 • Sample
方案 'SampleWeb_Step4' (2 專案)
01 Models
SampleWeb.Models
Properties
參考
Interface
Repository
App.config
Northwind.edmx
packages.config
02 Web
SampleWeb
Properties
參考
App_Data
App_Start
bin
Content
Controllers
fonts
Models
obj
Scripts
Views
CodeMap1.dgml
favicon.ico
Global.asax
packages.config
Project_Readme.html
SampleWeb.csproj.GhostDoc.xml
Web.config
其他檔案
CodeMap1.dgml
復原
顯示相關項目(R)
配置(Y)
共用(A)
88.91
圖例(L)
資料流處理視訊: 使用 Code Map 了解複雜的程式碼
SampleWeb.dll
SampleWeb.Controllers
ProductsController
ProductsController
Dispose
categoryRepository
Delete
Edit
Index
productRepository
Categories
DeleteConfirmed
Edit
Create
Details
Create
CategoriesController
Details
Create
Create
Delete
Dispose
categoryRepository
Index
Edit
Edit
CategoriesController
DeleteConfirmed
HomeController
SampleWeb



demo

專案分層 Part.5
建立 Service 層

Part.5 - 建立 Service 層

- Repository 層，主要是用來操作資料的存取
  不牽涉商業邏輯的操作
- Controller 其主要的工作為系統流程的控制，
  驗證輸入的資料，依據傳入的需求來決定要存取哪些資料，
  並且決定要選擇 View 以回應到需求端
- 商業邏輯的處理不適合放在 Controller 當中，
  過多的商業邏輯處理反而會讓 Controller 的流程控制與
  商業邏輯混在一起

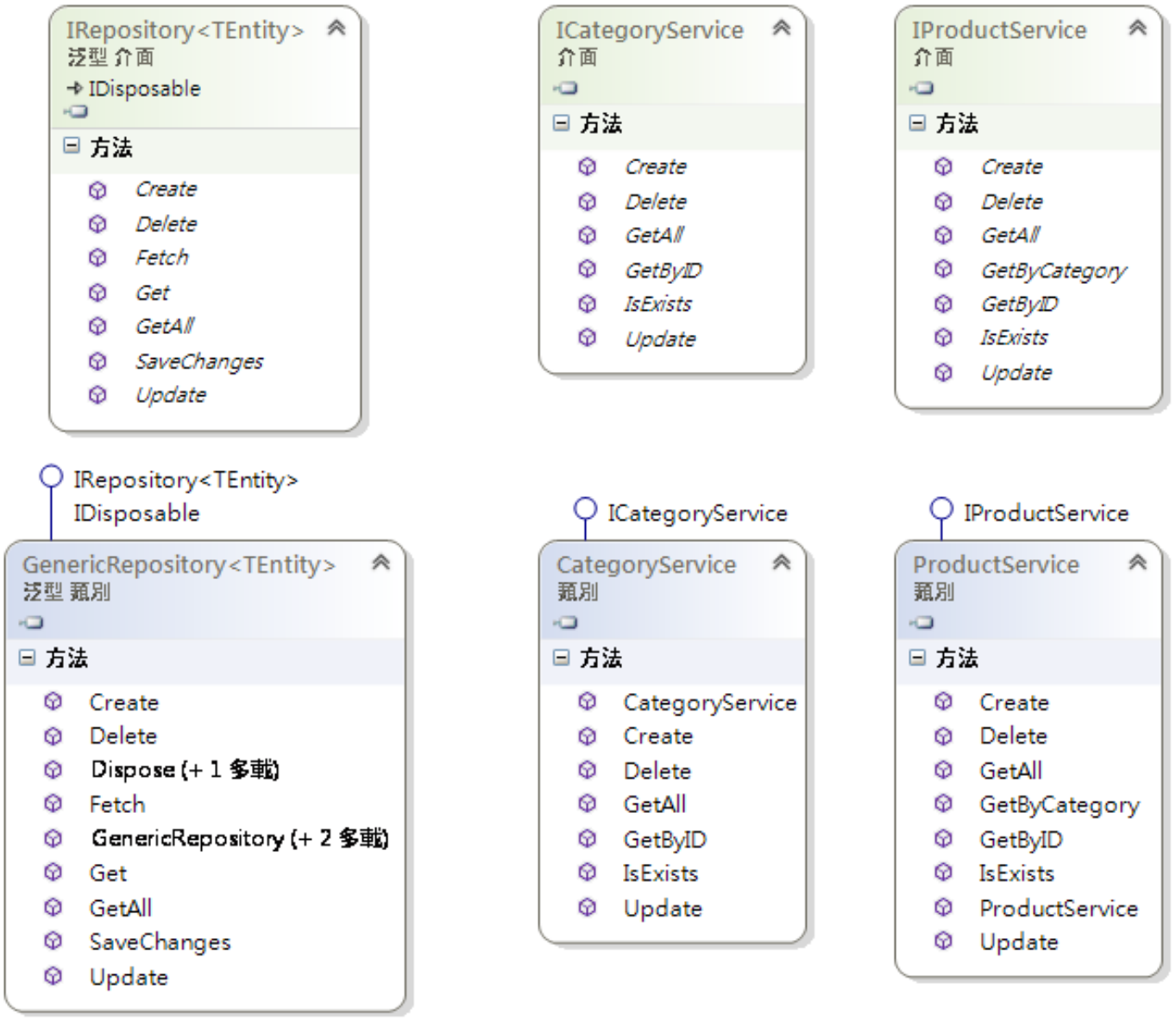
Part.5 - 建立 Service 層

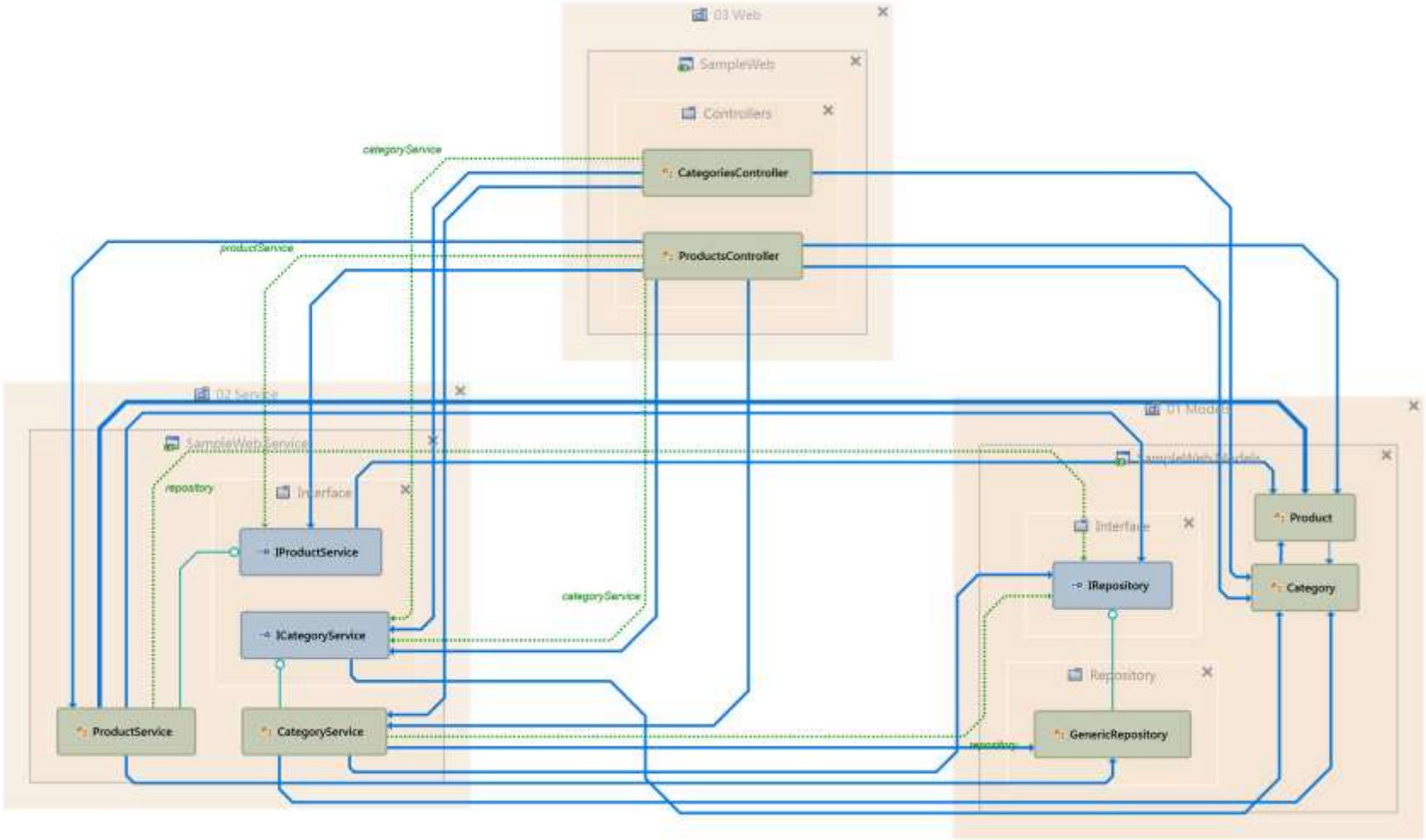
- 服務層，主要是把系統的商業邏輯給封裝起來

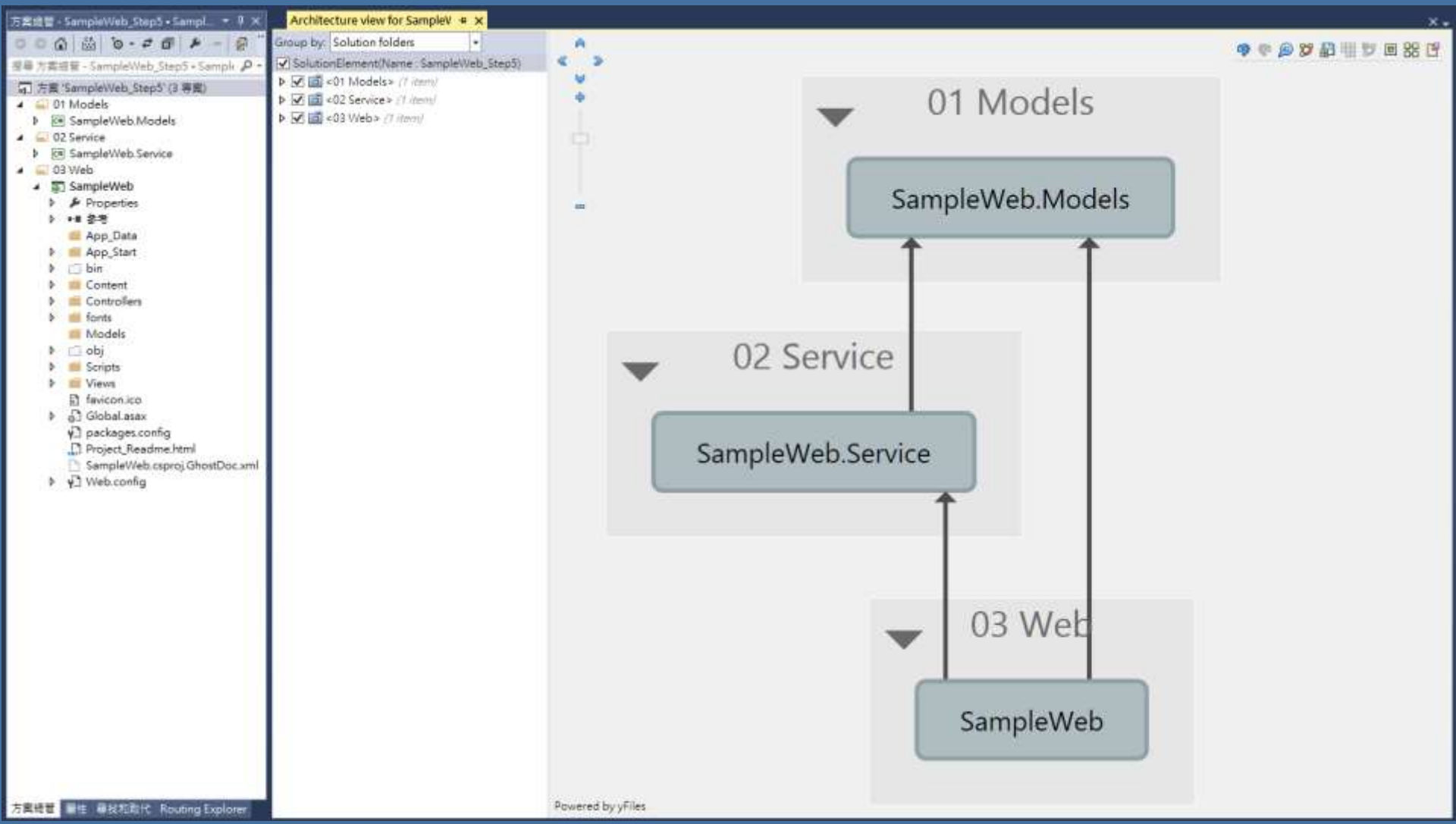
  Service 層使用 Repository 層所提供的服務來存取資料

  Controller 透過 Service 層來做資料的處理，不直接使用 Repository。

- Repositiry 主要負責資料的 CRUD，而 Service 則是處理商業邏輯的處理


IRepository<TEntity>
泛型 介面
IDisposable
方法
Create
Delete
Fetch
Get
GetAll
SaveChanges
Update
ICategoryService
介面
方法
Create
Delete
GetAll
GetByID
IsExists
Update
IProductService
介面
方法
Create
Delete
GetAll
GetByCategory
GetByID
IsExists
Update
IRepository<TEntity>
IDisposable
IRepository<TEntity>
泛型 類別
方法
Create
Delete
Dispose (+ 1 多載)
Fetch
GenericRepository (+ 2 多載)
Get
GetAll
SaveChanges
Update
ICategoryService
CategoryService
類別
方法
CategoryService
Create
Delete
GetAll
GetByID
IsExists
Update
IProductService
ProductService
類別
方法
Create
Delete
GetAll
GetByCategory
GetByID
IsExists
ProductService
Update


03 Web
SamplesWeb
Controllers
categoryService
CategoriesController
productService
ProductsController
02 Service
SampleWebService
repository
Interface
IProductService
ICategoryService
categoryService
ProductService
CategoryService
01 Model
SampleWebModels
Interface
IRepository
Repository
GenericRepository
repository
Product
Category


Architecture view for SampleV
Group by: Solution folders
SolutionElement(Name : SampleWeb_Step5)
<01 Models>
<02 Service>
<03 Web>
01 Models
SampleWeb.Models
02 Service
SampleWeb.Service
03 Web
SampleWeb
Powered by yFiles

Part.5 - 建立 Service 層

- Repository 與 Service 絕對不會依賴於展現層
- Service 層不應該也不會使用到展現層的物件

  展現層的 ViewModel 不可能也不應該出現在 Service

- Service 層與展現層可以使用 DTO (Data Transfer Object 資料傳輸物件) 來傳遞資料
- DTO 與展現層物件 (ex: ViewModel) 的對映轉換，可以使用 AutoMapper

Part.5 - 建立 Service 層

- 是否一個 Model 類別就要建立一個對應的 Service 類別?
- 如果遇到了資料需要 JOIN 處理時，要放在哪裡？
- 一定要有 Repository 嗎？
- 好多介面？
- Service 有必要做一個 Iservice 介面嗎？

關於 Repository

- MSDN – The Repository Pattern
- Huan-Lin 學習筆記: Repository，我可能不會用你

  Huan-Lin 學習筆記: Repository，我可能不會用你(2)-範例

  Huan-Lin 學習筆記: Repository，我可能不會用你(3)-正反意見



demo

專案分層 Part.6
DI/IoC 使用 Unity.MVC

```
namespace Mvc_Repository.Web.Controllers
{
    public class CategoryController : Controller
    {
        private ICategoryService categoryService;

        public CategoryController()
        {
            this.categoryService = new CategoryService();
        }
```

```
namespace Mvc_Repository.Service
{
    public class CategoryService : ICategoryService
    {
        private IRepository<Categories> repository =
            new GenericRepository<Categories>();
```

Part.6 - DI/IoC 使用 Unity.MVC

- 少了彈性就代表系統缺乏可擴充與變化性
- **程式的內容是針對介面而寫，而不是針對實作而寫**

  實作應該依賴抽象，而不是依賴細節
- 模組之間的依賴關係，要透過抽象介面

  Class 之間不發生直接的依賴關係

  而相互之間的則是透過 Interface 或 abstract class

  產生依賴關係

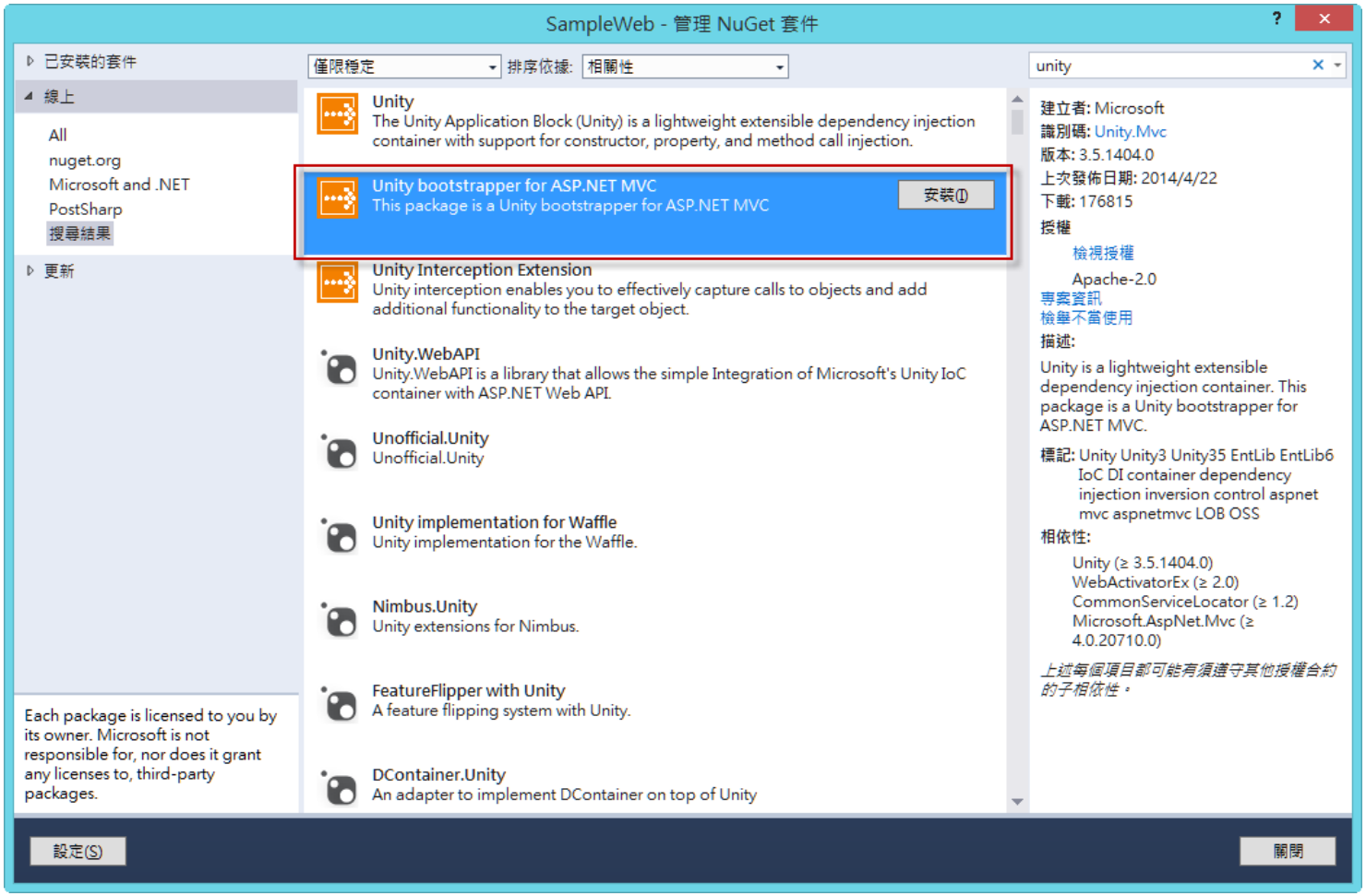
Part.6 - DI/IoC Container

除了 Microsoft’s Unity IoC container 之外，

還可以選用其他的 DI/Ioc Container

- Autofac
- Ninject
- Castle Windsor
- Spring.NET
- SimpleInjector … etc

Unity bootstrapper for ASP.NET MVC



```
using System;
using System.Data.Entity;

namespace SampleWeb.Models.DbContextFactory
{
    1 個參考
    public class DbContextFactory : IDbContextFactory
    {
        private readonly string _ConnectionString;

        0 個參考
        public DbContextFactory(string connectionString)
        {
            this._ConnectionString = connectionString;
        }

        private DbContext _dbContext;

        1 個參考
        private DbContext DbContext
        {
            get
            {
                if (this._dbContext == null)
                {
                    Type t = typeof(DbContext);
                    this._dbContext =
                        (DbContext)Activator.CreateInstance(t, this._ConnectionString);
                }
                return _dbContext;
            }
        }

        2 個參考
        public DbContext GetDbContext()
        {
            return this.DbContext;
        }
    }
}
```

```
using System;
using System.Data.Entity;
using System.Linq;
using System.Linq.Expressions;
using SampleWeb.Models.DbContextFactory;
using SampleWeb.Models.Interface;

namespace SampleWeb.Models.Repository
{
    1 個參考
    public class GenericRepository<TEntity> : IRepository<TEntity>
        where TEntity : class
    {
        10 個參考
        private DbContext Context { get; set; }

        0 個參考
        public GenericRepository(IDbContextFactory factory)
        {
            if (factory == null)
            {
                throw new ArgumentNullException("factory");
            }
            this.Context = factory.GetDbContext();
        }
```

MVC

```
using System;
using System.Collections.Generic;
using System.Linq;
using SampleWeb.Models;
using SampleWeb.Models.Interface;
using SampleWeb.Service.Interface;

namespace SampleWeb.Service
{
    1 個參考
    public class CategoryService : ICategoryService
    {
        private readonly IRepository<Category> _repository;

        0 個參考
        public CategoryService(IRepository<Category> repository)
        {
            _repository = repository;
        }


        2 個參考
        public IResult Create(Category instance)
        {
            if (instance == null)
            {
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using SampleWeb.Models;
using SampleWeb.Models.Interface;
using SampleWeb.Service.Interface;

namespace SampleWeb.Service
{
    1 個參考
    public class ProductService : IProductService
    {
        private readonly IRepository<Product> _repository;

        0 個參考
        public ProductService(IRepository<Product> repository)
        {
            _repository = repository;
        }

        2 個參考
        public IResult Create(Product instance)
        {
            if (instance == null)
            {
```

```
1  using System.Data;
2  using System.Linq;
3  using System.Web.Mvc;
4  using SampleWeb.Models;
5  using SampleWeb.Service.Interface;
6
7  namespace SampleWeb.Controllers
8  {
       1 個參考
9      public class CategoriesController : Controller
10     {
11         private readonly ICategoryService _categoryService;
12
           0 個參考
13         public CategoriesController(ICategoryService categoryService)
14         {
15             this._categoryService = categoryService;
16         }
17
18         // GET: Categories
           0 個參考
19         public ActionResult Index()
20         {
21             var categories = this._categoryService.GetAll().ToList();
22             return View(categories);
```

```
using System.Linq;
using System.Web.Mvc;
using SampleWeb.Models;
using SampleWeb.Service.Interface;

namespace SampleWeb.Controllers
{
    1 個參考
    public class ProductsController : Controller
    {
        private readonly IProductService _productService;

        private readonly ICategoryService _categoryService;

        4 個參考
        public IEnumerable<Category> Categories
        {
            get { return _categoryService.GetAll(); }
        }

        0 個參考
        public ProductsController(
            ICategoryService categoryService,
            IProductService productService)
        {
            this._productService = productService;
            this._categoryService = categoryService;
        }

        0 個參考
```

```
public class UnityConfig
{
    Unity Container

    /// <summary>Registers the type mappings with the Unity container.</summary>
    /// <param name="container">The unity container to configure.</param>
    /// <remarks>There is no need to register concrete types such as controllers or API controlle
    /// change the defaults), as Unity allows resolving a concrete type even if it was not previo
    0 個參考
    public static void RegisterTypes(IUnityContainer container)
    {
        //資料庫連接字串由執行端環境來給予，而不寫死在 DbContextFactory 當中.
        var connectionString =
            WebConfigurationManager.ConnectionStrings["NorthwindEntities"].ConnectionString;

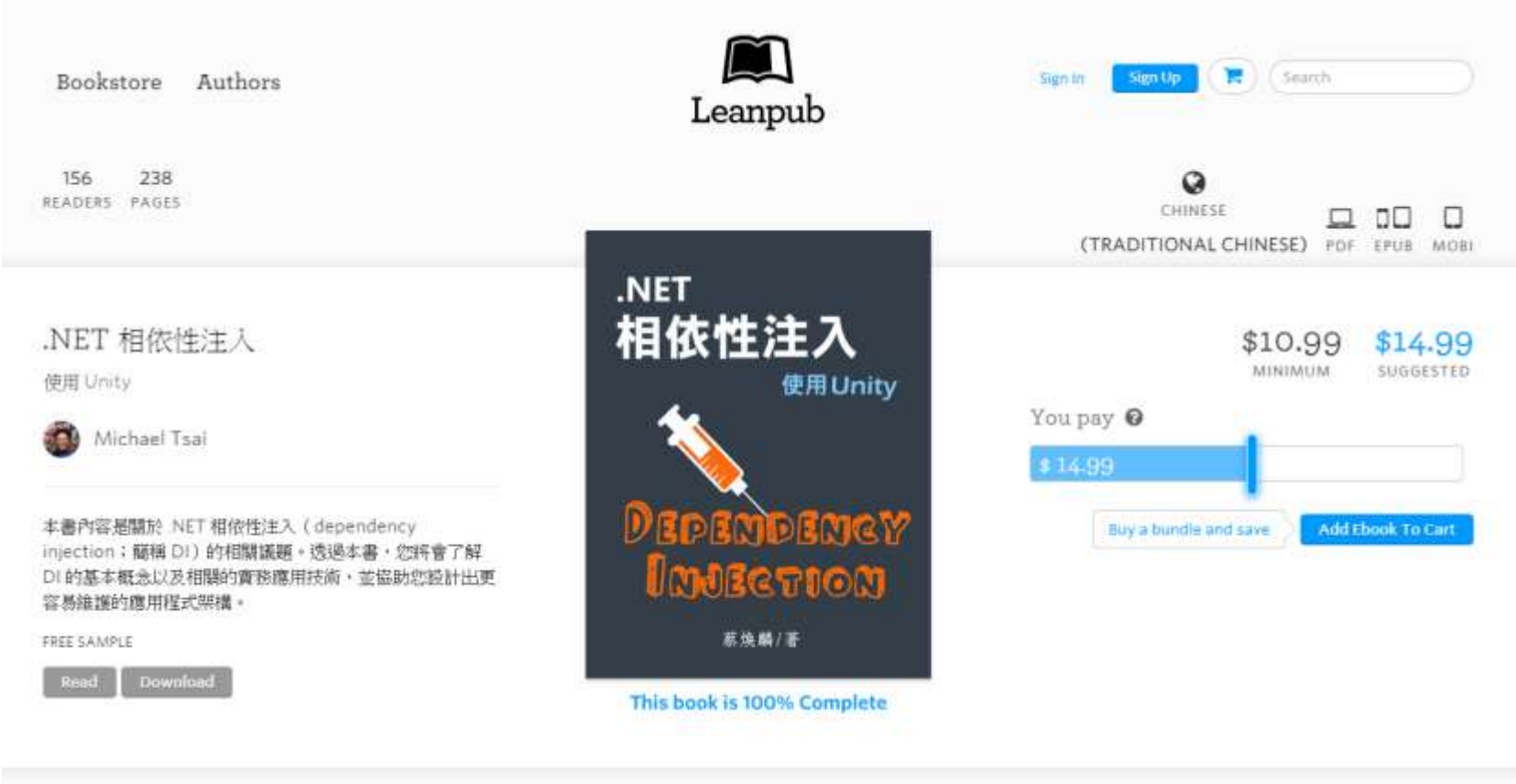
        container
            .RegisterType<IDbContextFactory, DbContextFactory>(
                new PerRequestLifetimeManager(),
                new InjectionConstructor(connectionString))
            .RegisterType(
                typeof(IRepository<>),
                typeof(GenericRepository<>),
                new PerRequestLifetimeManager())
            .RegisterTypes(
                AllClasses.FromAssemblies(true, Assembly.Load("SampleWeb.Service")),
                WithMappings.FromMatchingInterface,
                WithName.Default,
                WithLifetime.Custom<PerRequestLifetimeManager>);
    }
}
```

Huan-Lin 學習筆記: Dependency Injection 筆記

- Huan-Lin 學習筆記: Dependency Injection 筆記 (1)
- Huan-Lin 學習筆記: Dependency Injection 筆記 (2)
- Huan-Lin 學習筆記: Dependency Injection 筆記 (3)
- Huan-Lin 學習筆記: Dependency Injection 筆記 (4)
- Huan-Lin 學習筆記: Dependency Injection 筆記 (5)
- Huan-Lin 學習筆記: Dependency Injection 筆記 (6)

Dependency Injection 參考資料

- .NET 相依性注入 使用 Unity - 蔡煥麟 著



https://leanpub.com/dinet

目前問題

- 原本還沒有 Service 的時候，在 Repository 還有處理 Dispose，但是在建立 Service 之後就沒有去處理。難道是不需要嗎？
- 建立一次 Repository instance 就建立一個 DbContext instance，這成本似乎有點高？



demo

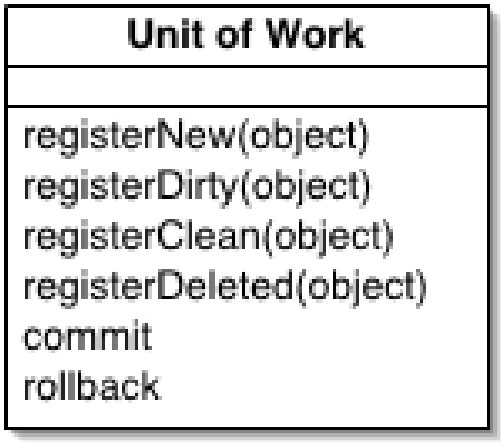
專案分層 Part.7
Unit of Work 工作單元

Part.7 - Unit of Work 工作單元

- 維護受業務事物影響的一系列物件，並協調變化寫入和併發問題的解決

Maintains a list of objects affected by a business transaction and coordinates the writing out of changes and the resolution of concurrency problems.

- Martin Fowler

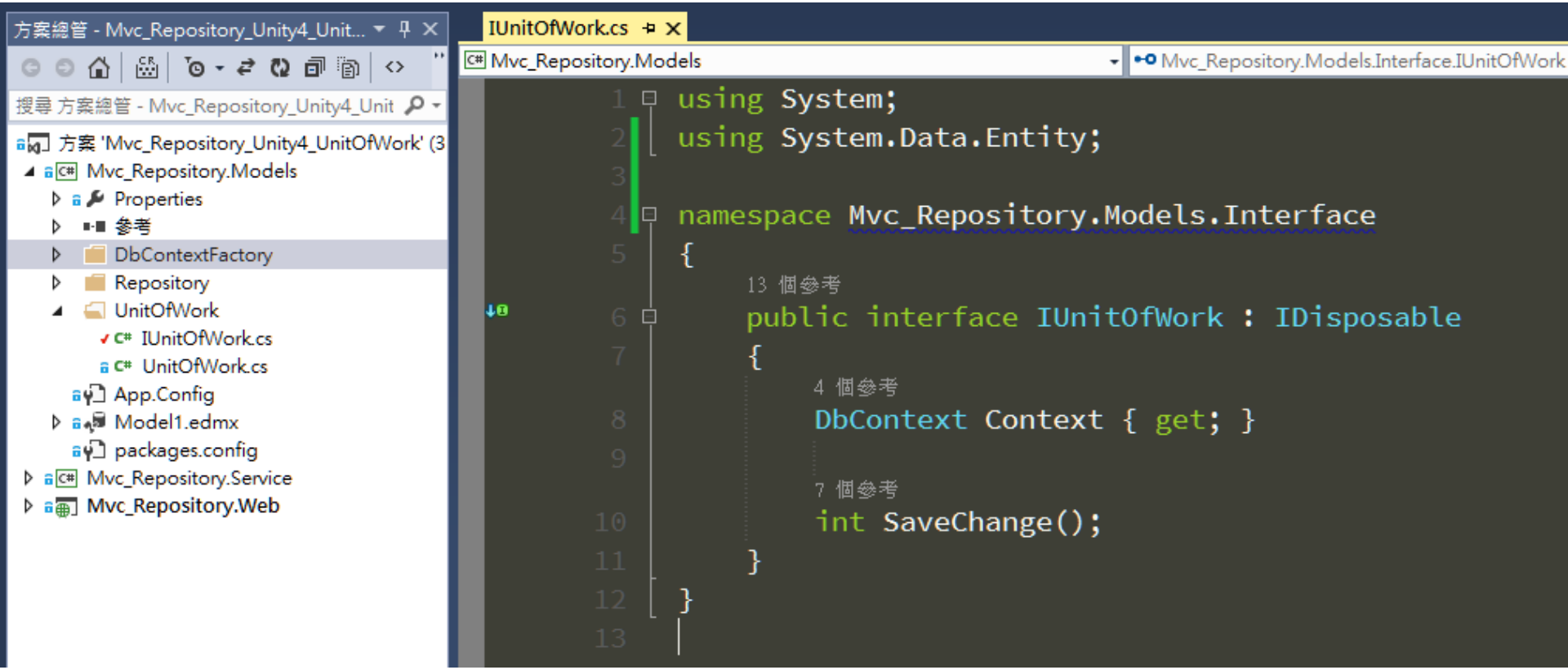


Part.7 - Unit of Work 工作單元

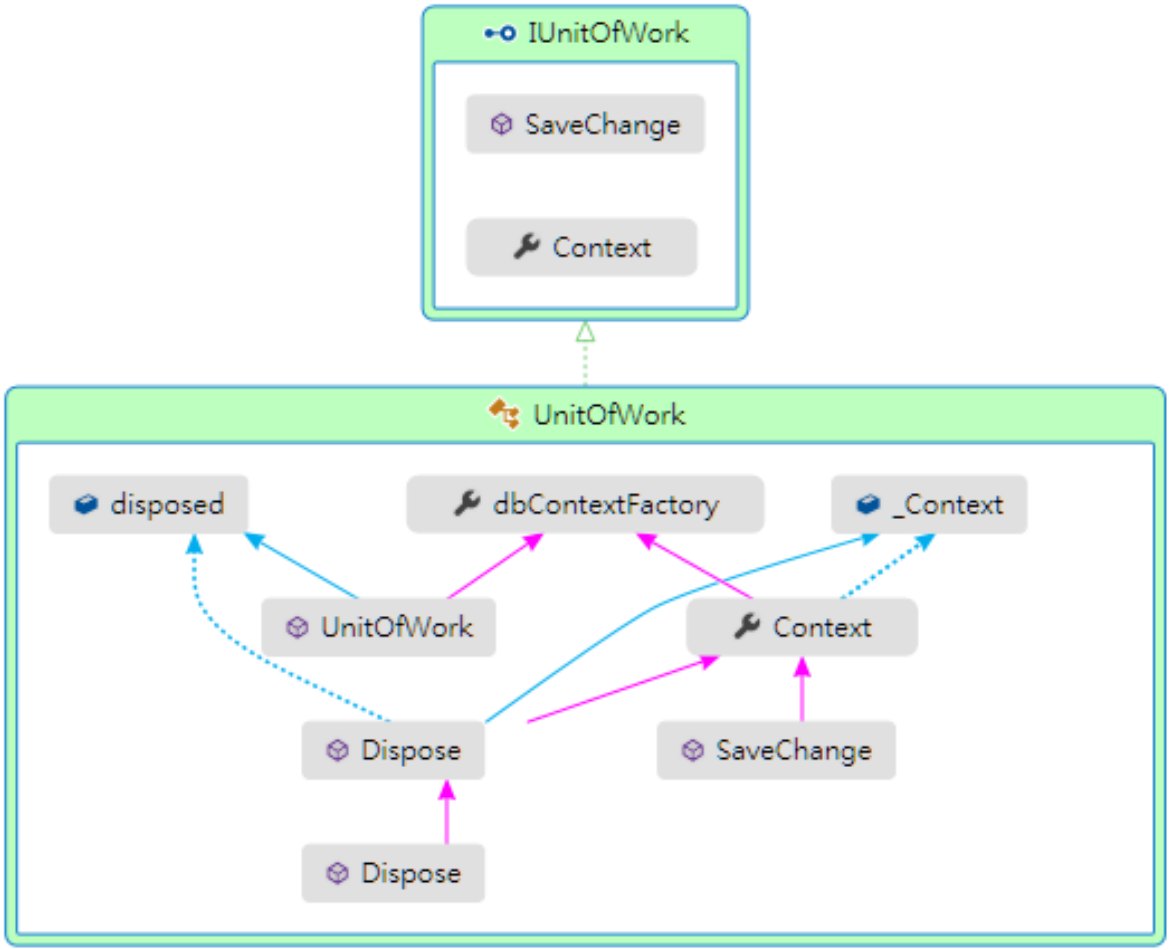
- 管理物件的 CRUD 操作，工作單元負協調這些變化的持久化工作以及所有標記的併發問題
- 確保數據在持久化過程的資料完整性
- 如果在同一個工作單元中持久化一系列物件資料時出現問題，就應該會恢復所有的變化，以確保資料始終處於有效狀態
- Unit of Work 的使用會結合 Repository

Unit of Work

- Unit of Work 的使用會結合 Repository



Unit of Work


IUnitOfWork
SaveChange
Context
UnitOfWork
disposed
dbContextFactory
_Context
UnitOfWork
Context
Dispose
SaveChange
Dispose

```
2 個參考
public class UnitOfWork : IUnitOfWork
{
    2 個參考
    private IDbContextFactory DbContextFactory { get; set; }

    private DbContext _context;

    4 個參考
    public DbContext Context
    {
        get
        {
            if (this._context != null)
            {
                return this._context;
            }
            this._context = DbContextFactory.GetDbContext();
            return this._context;
        }
    }

    0 個參考
    public UnitOfWork(IDbContextFactory factory)
    {
        this.DbContextFactory = factory;
    }
```

```
        7 個參考
        public int SaveChange()
        {
            return this.Context.SaveChanges();
        }


        private bool disposed = false;

        1 個參考
        protected virtual void Dispose(bool disposing)
        {
            if (!this.disposed)
            {
                if (disposing)
                {
                    this.Context.Dispose();
                    this._context = null;
                }
            }
            this.disposed = true;
        }

        5 個參考
        public void Dispose()
        {
            Dispose(true);
            GC.SuppressFinalize(this);
        }
    }
```

UnitOfWork

修改 GenericRepository

```
2 個參考
public class GenericRepository<TEntity> : IRepository<TEntity>
    where TEntity : class
{
    2 個參考
    public IUnitOfWork _UnitOfWork { get; set; }

    4 個參考
    private DbContext Context { get; set; }

    4 個參考
    private DbSet<TEntity> DbSet { get; set; }

    0 個參考
    public GenericRepository(IUnitOfWork unitofwork)
    {
        this._UnitOfWork = unitofwork;
        this.Context = unitofwork.Context;
        this.DbSet = this.Context.Set<TEntity>();
    }
```

修改 Service

```
1 個參考
public class CategoryService : ICategoryService
{
    private readonly IUnitOfWork _unitOfWork;

    private readonly IRepository<Categories> _repository;

    0 個參考
    public CategoryService(IUnitOfWork unitofwork, IRepository<Categories> repository)
    {
        this._unitOfWork = unitofwork;
        this._repository = repository;
    }
```

```
1 個參考
public class ProductService : IProductService
{
    private readonly IUnitOfWork _unitOfWork;

    private readonly IRepository<Products> _repository;

    0 個參考
    public ProductService(IUnitOfWork unitofwork, IRepository<Products> repository)
    {
        this._unitOfWork = unitofwork;
        this._repository = repository;
    }
```

Service 內部的使用

```
2 個參考
public IResult Create(Categories instance)
{
    if (instance == null)
    {
        throw new ArgumentNullException();
    }

    IResult result = new Result(false);
    try
    {
        this._repository.Create(instance);
        this._unitOfWork.SaveChange();
        result.Success = true;
    }
    catch (Exception ex)
    {
        result.Exception = ex;
    }
    return result;
}
```

增加 UnityConfig.cs 的類別註冊

```
container
    .RegisterType<IDbContextFactory, DbContextFactory>(
        new PerRequestLifetimeManager(),
        new InjectionConstructor(connectionString))
    .RegisterType<IUnitOfWork, UnitOfWork>(new PerRequestLifetimeManager())
    .RegisterType(typeof (IRepository<>), typeof (GenericRepository<>))
    .ConfigureAutoRegistration()
    .ExcludeSystemAssemblies()
    .Include(
        t => t.Assembly == Assembly.Load("Mvc_Repository.Service"),
        Then.Register().AsAllInterfacesOfType())
    .ApplyAutoRegistration();
```

問題

- 如果 controller 的 action 方法需要使用多個 service 的操作時，現行方法還是讓一次的 request 發動多次的 SaveChange 處理（多個工作單元）

- 修改為一次 request 的處理為一個工作單元

修改 Service

```
1 個參考
public class CategoryService : ICategoryService
{
    private readonly IRepository<Categories> _repository;

    0 個參考
    public CategoryService(IRepository<Categories> repository)
    {
        this._repository = repository;
    }
```

修改 Service

```
2 個參考
public IResult Create(Categories instance)
{
    if (instance == null)
    {
        throw new ArgumentNullException();
    }

    IResult result = new Result(false);
    try
    {
        this._repository.Create(instance);
        result.Success = true;
    }
    catch (Exception ex)
    {
        result.Exception = ex;
    }
    return result;
}
```

UnitOfWorkFactory

```
3 個參考
public interface IUnitOfWorkFactory
{
    5 個參考
    IUnitOfWork GetUnitOfWork();
}

2 個參考
public class UnitOfWorkFactory : IUnitOfWorkFactory
{
    private IUnitOfWork _unitOfWork;

    5 個參考
    public IUnitOfWork GetUnitOfWork()
    {
        if (this._unitOfWork != null)
        {
            return this._unitOfWork;
        }
        this._unitOfWork = DependencyResolver.Current.GetService<IUnitOfWork>();
        return this._unitOfWork;
    }
}
```

UnitOfWorkAttribute

```
2 個參考
public class UnitOfWorkAttribute : ActionFilterAttribute
{
    private readonly IUnitOfWorkFactory _unitOfWorkFactory = new UnitOfWorkFactory();

    private IUnitOfWork _unitOfWork;

    1 個參考
    public override void OnActionExecuting(ActionExecutingContext filterContext)
    {
        this._unitOfWork = _unitOfWorkFactory.GetUnitOfWork();

        base.OnActionExecuting(filterContext);
    }

    1 個參考
    public override void OnActionExecuted(ActionExecutedContext filterContext)
    {
        if (filterContext.Exception == null)
        {
            this._unitOfWork.SaveChange();
        }
        base.OnActionExecuted(filterContext);
    }
}
```

使用 UnitOfWorkAttribute ActionFilter

```
0 個參考
public ActionResult Create()
{
    return View();
}

[HttpPost]
[UnitOfWork]
0 個參考
public ActionResult Create(Categories category)
{
    if (category != null &&
        ModelState.IsValid)
    {
        this._categoryService.Create(category);
        return RedirectToAction("index");
    }
    else
    {
        return View(category);
    }
}
```

使用 UnitOfWorkFactory

```
0 個參考
public ActionResult Create(string category)
{
    ViewBag.CategoryID = new SelectList(this.Categories, "CategoryID", "C
    ViewBag.Category = string.IsNullOrWhiteSpace(category)
        ? "all"
        : category;

    return View();
}

[HttpPost]
0 個參考
public ActionResult Create(Products products, string category)
{
    if (ModelState.IsValid)
    {
        using (var unitOfWork = _unitOfWorkFactory.GetUnitOfWork())
        {
            this._productService.Create(products);
            unitOfWork.SaveChange();
        }
        return RedirectToAction("Index", new { category = category });
    }

    ViewBag.CategoryID = new SelectList(this.Categories, "CategoryID", "C

    return View(products);
}
```

參考資料

- 掀起你的盖头来：Unit Of Work-工作单元 - 田园里的蟋蟀 - 博客园
- Thinking In Design Pattern—Unit Of Work（工作单元）模式探索 - 木宛城主 - 博客园
- MVC3+EF4.1学习系列(八)-----利用Repository and Unit of Work重构项目 - wlf - 博客园
- Entity Framework之深入分析 - 小城岁月 - 博客园

Principle

SOLID – Robert C. Martin

- Single Responsibility Principle 單一職責
- Open-Closed Principle 開放與關閉
- Liskov's Substitution Principle 子類別取代
- Interface Segregation Principle 介面隔離
- Dependency Inversion Principle 相依性倒轉

http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod

SOLD - 91之ASP.NET由淺入深 不負責講座

- Single Responsibility Principle 單一職責
- Open-Closed Principle 開放與關閉
- Liskov's Substitution Principle 子類別取代
- Interface Segregation Principle 介面隔離
- Dependency Inversion Principle 相依性倒轉

相關書籍

相關書籍

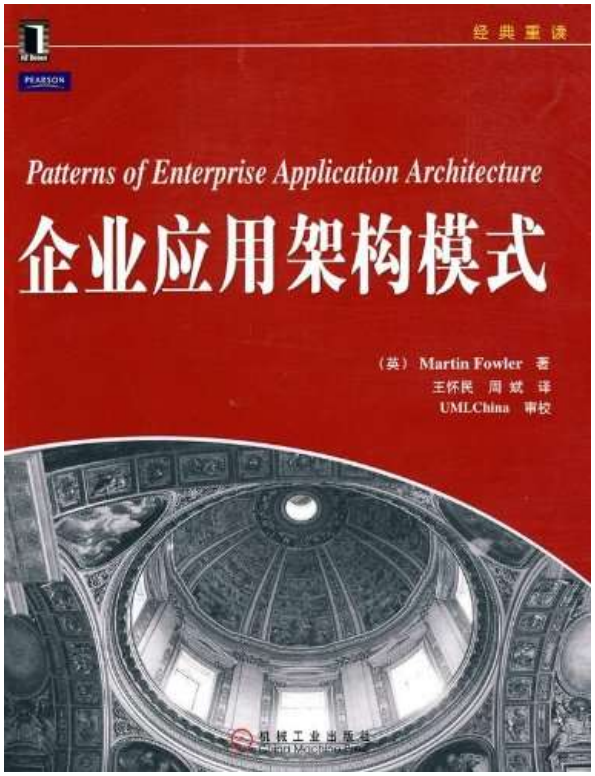
- **Patterns of Enterprise Application Architecture**

  企業應用架構模式

  Martin Fowler 著

  王懷民 周斌 譯

  P.S. 2015年一月重印新版



相關書籍

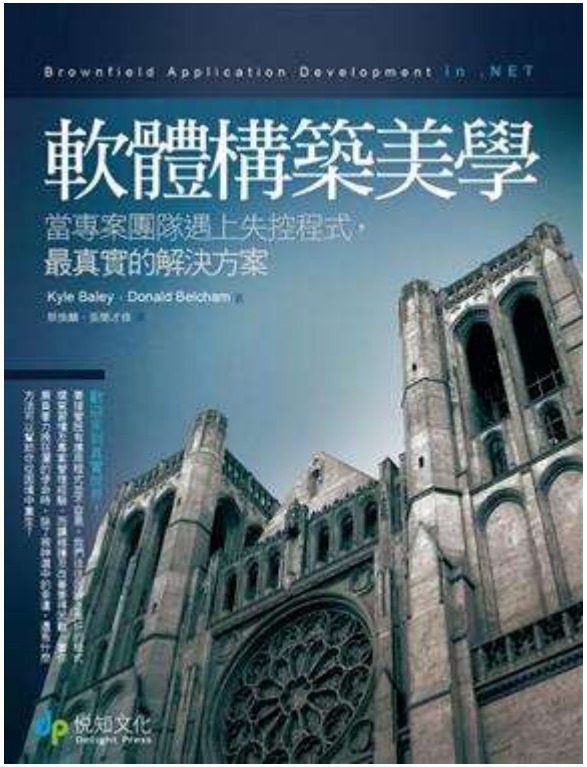
- **Brownfield Application Development in .NET**

  軟體構築美學：當專案團隊遇上失控程式，最真實的解決方案

  Kyle Baley, Donald Belcham 著

  蔡煥麟 張簡才祿 譯

  P.S. 已絕版（請找二手書）

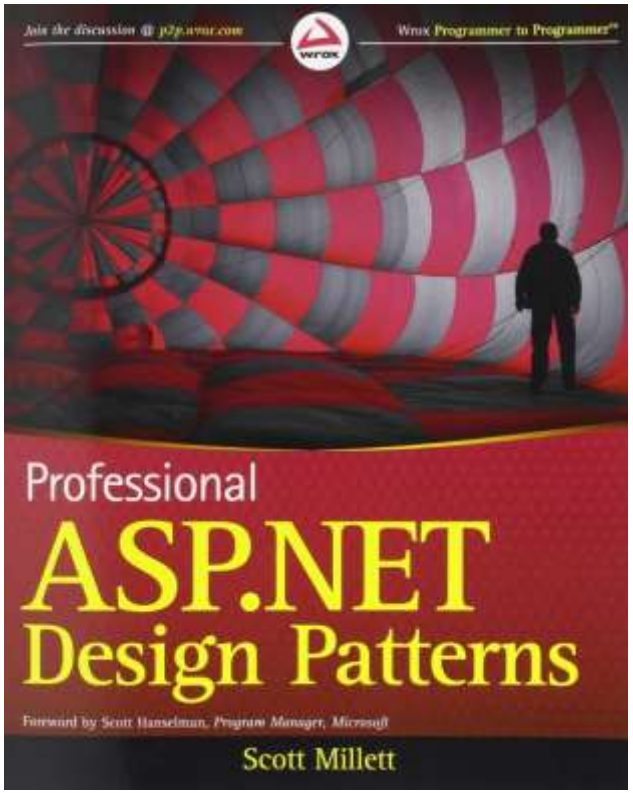


相關書籍

- **Professional ASP.NET Design Patterns**

  Scott Millett

  簡體版「ASP.NET 設計模式」



相關書籍

- **Software Architecture for Developers**

  程序員必讀之軟體架構

  Simon Brown 著

  劉鋼 譯





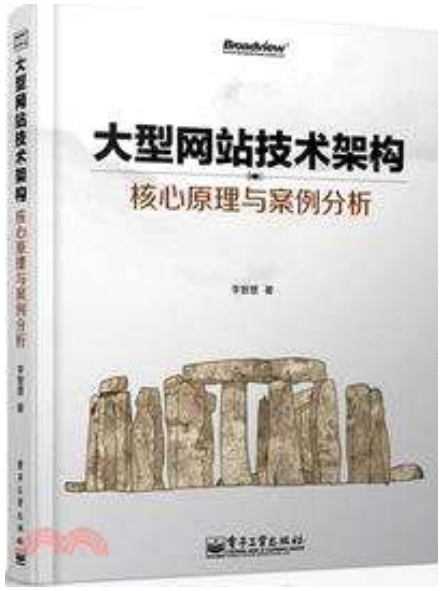
相關書籍

- **大型網站技術架構 核心原理與案例分析**

  正體中文版：從車庫的舊PC到百萬台伺服器：巨型網站成長從無到無限大，技術架構大揭祕

  李智慧 著





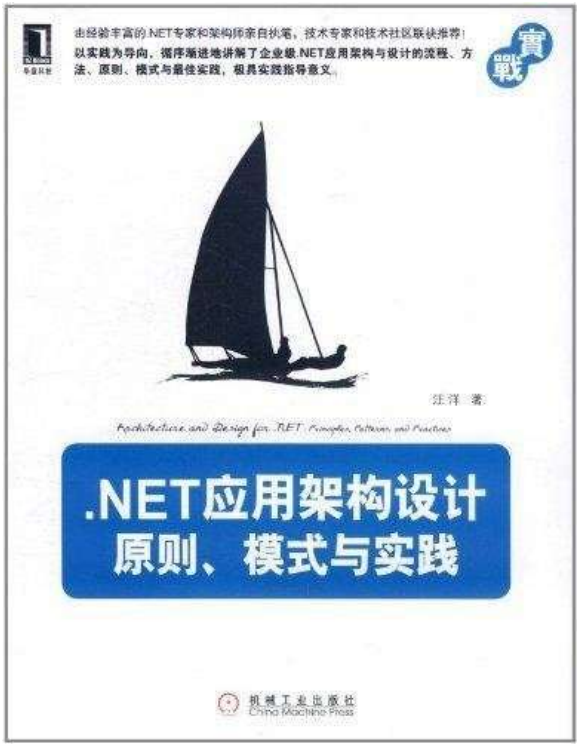
相關書籍

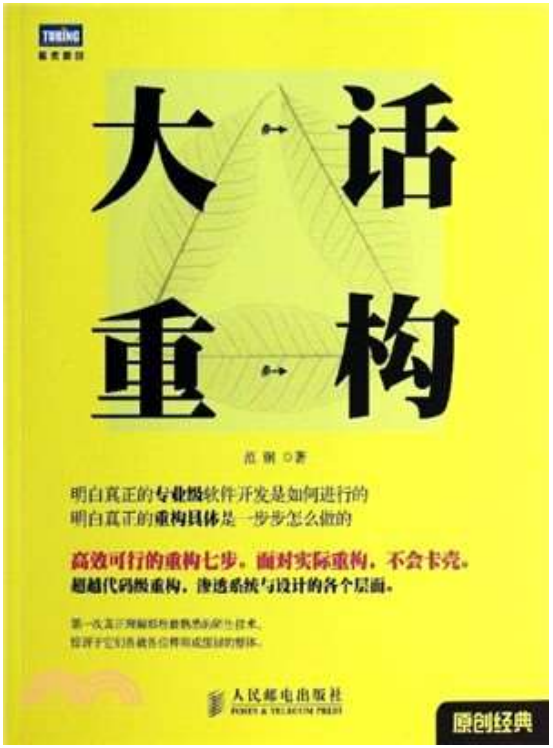
- **.NET應用架構設計 原則、模式與實踐**

  汪洋 著

- **大話重構**

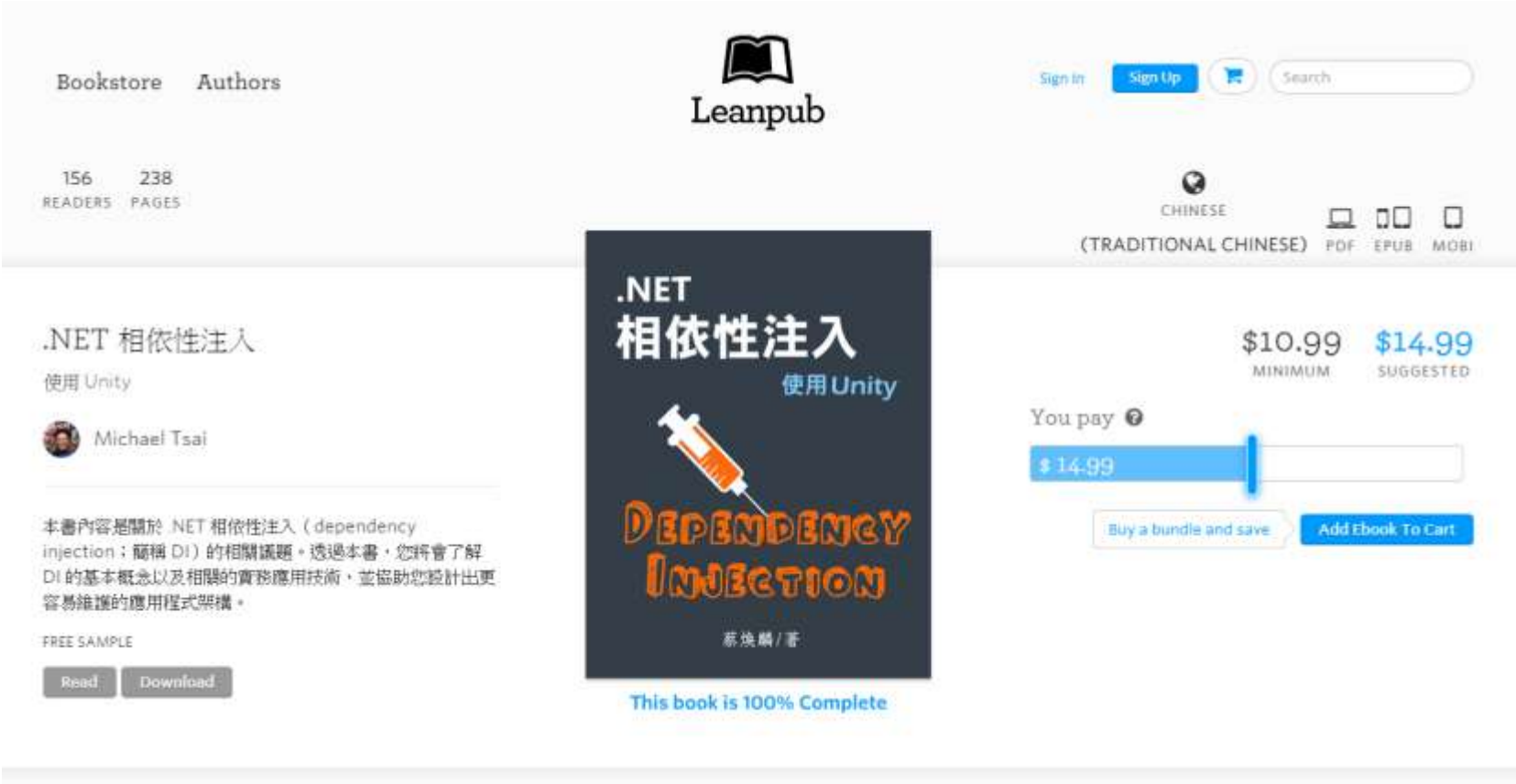
  范錮 著





Dependency Injection 參考書籍

- .NET 相依性注入 使用 Unity - 蔡煥麟 著



https://leanpub.com/dinet

好活動需要支持



感謝 KKTIX 贊助 twMVC 活動報名平台

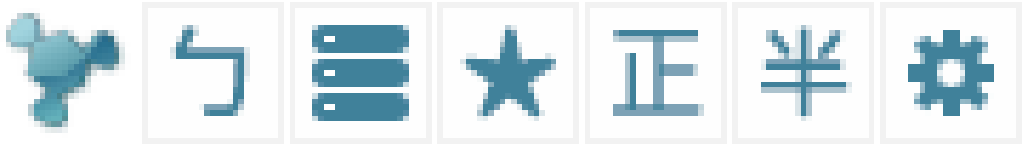
好輸入法需要露出

自然輸入法

全球最聰明的注音輸入！



http://www.iq-t.com/DOWNLOAD/software.asp



好課程不要錯過


SKILLTREE
Login
關於我們
講師介紹
課程簡介
企業包班
聯絡我們
活動花絮
課程簡介
課程簡介 » 自動測試與 TDD 實務開發(使用C#) 第三梯
自動測試與 TDD 實務開發(使用C#) 第三梯
自動測試與 TDD 實務開發(使用C#) 第三梯
2015-05-16 開班 21H
SKILLTREE
課程時間 (21H)
上課地點
Google 日曆
or
下載 ics 檔案
2015-05-16 (Sat) 09:00 – 17:00
2015-05-23 (Sat) 09:00 – 17:00
2015-05-30 (Sat) 09:00 – 17:00
本課程包含午餐
請預必於報名時勾選您的選擇。
課程開始後恕無法更改選擇。
本課程需自備筆電
實戰課程請攜帶筆電，以利課程實做。
必要軟體將會於課程通知信中說明。

好課程不要錯過


SKILLTREE
Login
關於我們
講師介紹
課程簡介
企業包班
聯絡我們
活動花絮
課程簡介
課程簡介 » 物件導向實作課程（使用C#）第四梯
物件導向實作課程（使用C#）第四梯
物件導向實作課程（使用C#）第四梯
2015-06-20 開課 21H
SKILLTREE
課程時間 (21H)
上課地點
Google 日曆
or
下載 ics 檔案
2015-06-20 (Sat) 09:00 – 17:00
2015-06-27 (Sat) 09:00 – 17:00
2015-07-04 (Sat) 09:00 – 17:00
本課程包含午餐
本課程需自備筆電

好課程需要支持


SKILLTREE
Login
關於我們
講師介紹
課程簡介
企業包班
聯絡我們
活動花絮
課程簡介
課程簡介 » ASP.NET MVC5 實戰訓練營暑修班
ASP.NET MVC5 實戰訓練營暑修班
ASP.NET MVC5 實戰訓練營暑修班
2015-07-06 開課 35H
課程時間 (35H)
上課地點
2015-07-06 (Mon) 09:00 ~ 17:00
2015-07-08 (Wed) 09:00 ~ 17:00
2015-07-10 (Fri) 09:00 ~ 17:00
2015-07-13 (Mon) 09:00 ~ 17:00
2015-07-15 (Wed) 09:00 ~ 17:00
本課程包含午餐
ASP.NET MVC5 實戰訓練營週末班 - 2015-07-25 (Sat)

好課程需要支持



http://skilltree.my/events/gcf

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.