Phân tích thiết kế phần mềm

MVC, Plugins

Nội dung

- Mô hình MVC
- Plugins

Nội dung

- **Mô hình MVC**
- Plugins

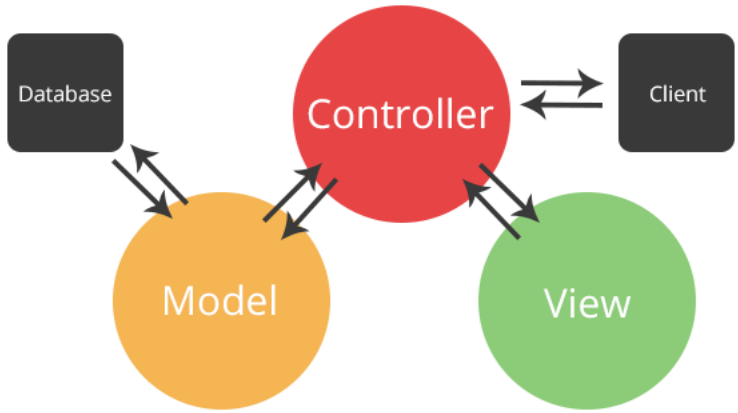
Khái niệm

- **Mô hình MVC** là một khái niệm quen thuộc trong thiết kế web và có thể được xem như là một tiêu chuẩn quan trọng không thể thiếu đối với các nhà lập trình web khi thiết lập nên những sản phẩm chất lượng, hoàn hảo cho khách hàng của mình.

ASP.NET MVC 5

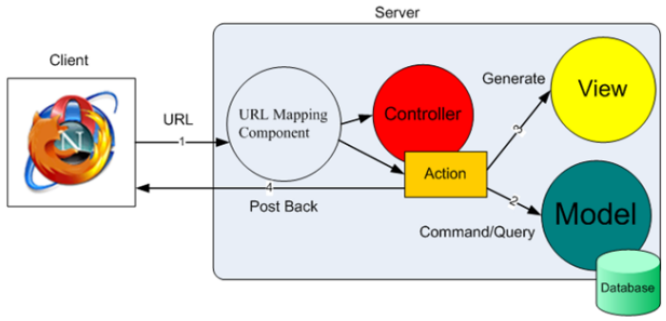
- **ASP.NET MVC** là một Framework sử dụng .NET Framework cho việc phát triển ứng dụng web động.
- ASP.NET MVC phát triển trên mẫu thiết kế chuẩn MVC, cho phép người sử dụng phát triển các ứng dụng phần mềm. **MVC** là tên một mẫu phát triển ứng dụng, phương pháp này chia nhỏ một ứng dụng thành ba thành phần để cài đặt, mỗi thành phần đóng một vai trò khác nhau và ảnh hưởng lẫn nhau, đó là ***Models***, ***Views***, và ***Controllers***.

Mô hình



Controllers

- Controller trong kiến trúc MVC xử lý bất kỳ các yêu cầu nào từ URL



Class Controller

- Các ***controller*** là ***class*** được dẫn xuất từ ***class*** cơ sở **System.Web.Mvc.Controller**.
- Class **Controller** chứa các phương thức được gọi là các ***Action***. **Controller** và các phương thức ***Action*** tiếp nhận các ***request*** từ ***client***, xử lý các thông tin cần thiết từ ***Model*** sau đó ***response*** kết quả cho ***client*** và Dữ liệu hiển thị sẽ được trình bày bởi ***View*** trong kiến trúc **MVC**.
- Trong **ASP.NET MVC** tất cả các class **controller** đều có postfix là “Controller”, ví dụ controller cho trang chủ có tên là “HomeController”, controller cho sản phẩm (Product) có tên là ProductController, ngoài ra tất cả các controller bắt buộc phải nằm trong thư mục Controller trong ứng dụng Asp.Net MVC.

Views

- **Views** là các thành phần dùng để hiển thị giao diện người dùng (UI). Views có thể hiển thị cả các thông tin tĩnh và động, View cũng có thể hiển thị các form cho phép người sử dụng có thể cập nhật thông tin trực tiếp vào các form đó. Các Controller trong ứng dụng có thể truyền dữ liệu cho View thông qua các đối tượng như *ViewData, ViewBag, TempData*.
- ASP.NET MVC lưu trữ các View trong thư mục *Views* của ứng dụng, các phương thức Action khác nhau của một lớp controller sẽ sinh ra các View khác nhau, thư mục chứa View trong ứng dụng có thể chứa nhiều các thư mục con khác, mỗi controller sẽ tạo tương ứng với 1 thư mục *View* có cùng tên với *Controller*.

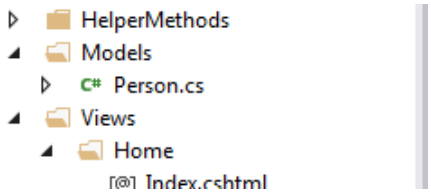
View engine

- View Engine là một phần của MVC Framework. View engine cho phép tạo ra các trang HTML kết hợp với các đoạn code mà các View engine hỗ trợ nhằm cho trình duyệt có thể hiểu và hiển thị thông tin.
- ASP.NET MVC Framework cung cấp 2 view engine cho phép sinh ra ra các trang HTML trong View đó là: Web Form View Engine và Razor View Engine

- ***Web Form View Engine*** : View Engine cho phép tạo ra các trang Views có cú pháp giống như ASP.NET Web Form. Đây là View engine mặc định của ASP.NET MVC với phiên bản 1.0 và 2.0.
- ***Razor View Engine*** : Từ phiên bản ASP.NET MVC 3, Microsoft lựa chọn View Engine mặc định là Razor View Engine, để sử dụng Razor lập trình viên cần phải nắm được cú pháp của ngôn ngữ đánh dấu của Razor nhằm kết hợp với cú pháp HTML ở phía client của ứng dụng.

Khái niệm Models

- Trong mô hình MVC thì **Model** là những thành phần có nhiệm vụ lưu trữ thông tin, trạng thái của các đối tượng, thông thường nó là một lớp được ánh xạ từ một bảng trong ***database***.
- Các model nên được đặt trong folder Models.



Truyền Model giữa Controller và View

- Trong controller

```
public ActionResult Index()
{
    var p = new Person() {
        FirstName = "Van A",
        LastName = "Tran"
    };
    return View(p);
}
```

- Trong View

```
<div>
    <h1>Xin chào @Model.FirstName @Model.LastName !</h1>
</div>
```

Nhận request là Model từ View

- Trên View

```
@model Person
@{
    Layout = null;
}
<!DOCTYPE html>
```

```
<div class="form-group">
    @Html.Label("First name:", new { @class = "col-md-2", @for = "FirstName"})
    @Html.TextBox("FirstName", @Model.FirstName, new { @class = "col-md-10" })
</div>
```

- Trong Controller

```
[HttpPost]
0 references
public ViewResult Index(Person p)
{
    return View(p);
}
```

Routing System

- Phần lớn các ***web framework*** như **ASP**, **JSP**, **PHP**, **ASP.NET** thì ***URL*** ánh xạ 1:1 với 1 file vật lý.
- Với **ASP.NET MVC** có hướng tiếp cận khác là ánh xạ ***URL*** đến ***method*** của ***class*** (***action*** trong ***controller***).
- Để xử lý các ***MVC URL***, hệ thống **ASP.NET** sử dụng ***hệ thống định tuyến*** (***routing system***).

Chức năng của Routing system

- Phân tích ***incoming URL***, sau đó gửi nhiệm vụ cho ***controller*** và ***action*** thích hợp.
- Tạo ra ***outgoing URL***. Đây là các URL xuất hiện trong HTML đã render từ các view để thực hiện action khi người dùng nhấn liên kết (lúc này nó lại trở thành incoming URL).

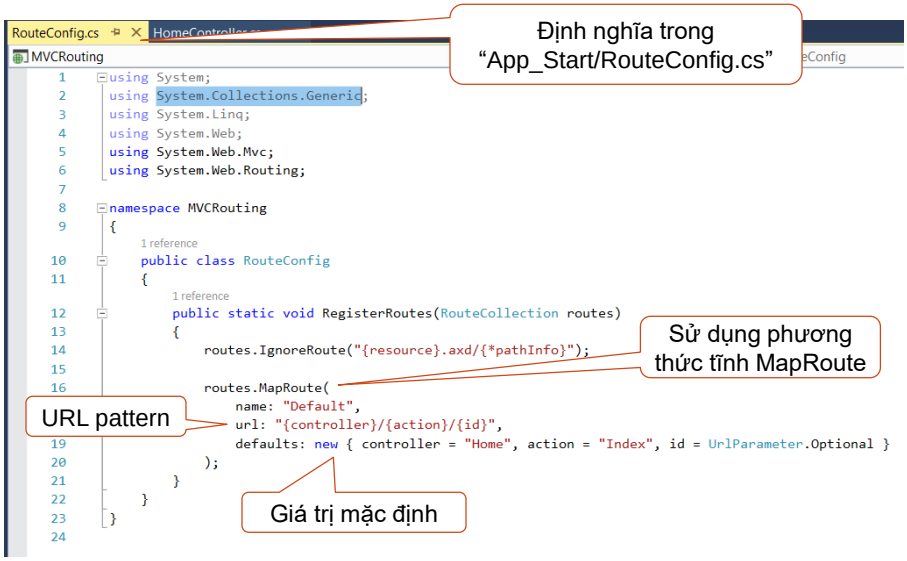
URL Pattern

- **Routing system** làm việc dựa trên 1 tập **routes**. Mỗi **route** lại chứa 1 **URL pattern**. Nếu ***pattern*** trùng với ***URL***, ***routing system*** sẽ xử lý ***URL*** đó.
- Ví dụ:
  - pattern: {controller}/{action}
  - incoming URL: http://localhost:55352/Admin/Index
  - Kết quả thực hiện định tuyến: Action Index của controller Admin được gọi.

segment

segment

Tạo route đơn giản



Các loại segment

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

    routes.MapRoute(
        name: "Default",
        url: "{controller}/{action}/{id}/{*catchall}",
        defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
    );

    routes.MapRoute(
        name: "pub",
        url: "Public/{controller}/{action}",
        defaults: new { action = "Index" }
    );

    routes.MapRoute(
        "wX",
        "my{controller}/{action}"
    );

    routes.MapRoute(
        "Play",
        "Play/{action}",
        new { controller = "Home" }
    );
}
```

sử dụng segment truyền tham số trong action

chấp nhận số segment tùy ý

segment tĩnh

segment tĩnh kết hợp

sử dụng controller mặc định mà không có segment controller

Route Constraints

```
routes.MapRoute(
    name: "Default",
    url: "{controller}/{action}/{id}/{*catchall}",
    defaults: new
    {
        controller = "Home",
        action = "Index",
        id = UrlParameter.Optional
    },
    constraints: new
    {
        controller = "^H.*",
        action = "^Index$|^About$"
    }
);
```

ràng buộc controller phải bắt đầu bởi H

ràng buộc action phải là Index hoặc About

Nội dung

- *Mô hình MVC*
- **Plugins**

Add-in Model

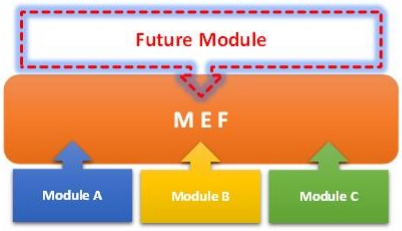
- **Add-ins** (hay **Plug-ins**) là biên dịch tách biệt các thành phần mà ứng dụng có thể tìm, tải và sử dụng động.
- Một ứng dụng được thiết kế sử dụng add-in để có thể được nâng cấp trong tương lai mà không cần phải sửa đổi, biên dịch và kiểm thử lại.
- **Add-ins** cũng cung cấp sự linh hoạt để tùy chỉnh chức năng của một ứng dụng cho mỗi thị trường hoặc khách hàng cụ thể.
- Nhưng lý do phổ biến nhất để sử dụng các tiện ích trong mô hình là cho phép các nhà phát triển bên thứ ba để mở rộng các chức năng của ứng dụng. Ví dụ, add-in trong Adobe Photoshop cung cấp một loạt các bộ xử lý hiệu ứng hình ảnh. Add-in trong Firefox cung cấp các tính năng lướt web nâng cao và chức năng hoàn toàn mới. Trong cả hai trường hợp, các add-in được tạo ra bởi nhà phát triển bên thứ ba.

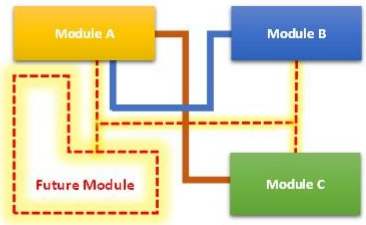
Kỹ thuật cơ bản

- Kể từ .NET 1.0, đã có tất cả các kỹ thuật cần thiết để tạo hệ thống add-in.
- Hai thành phần cơ bản là ***interfaces*** (cho phép xác định các cam kết thông qua đó ứng dụng tương tác với add-in và add-in tương tác với ứng dụng) và ***reflection*** (cho phép ứng dụng tự động phát hiện và tải add-in các loại từ một ***assembly*** riêng rẽ).
- Tuy nhiên, việc xây dựng một hệ thống add-in từ khởi đầu đòi hỏi nhiều vấn đề cơ bản như cần phải đưa ra một cách để xác định vị trí các add-in và cần phải đảm bảo rằng chúng đang được quản lý một cách chính xác (nói cách khác, chúng được thực thi trong ngữ cảnh bị giới hạn về bảo mật và có thể được gỡ bỏ khi cần thiết).

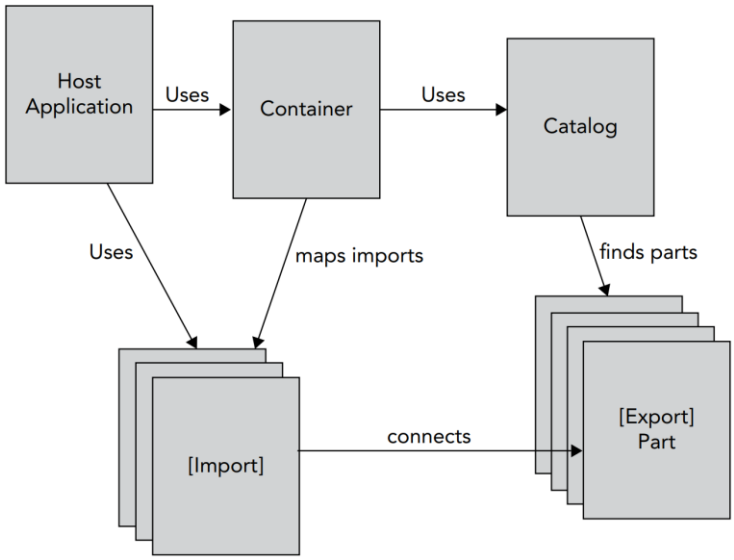
Managed Extensibility Framework (MEF)

- Có từ .NET 4, MEF là thư viện cho phép tạo các ứng dụng đơn giản có khả năng mở rộng (extensible). MEF cho phép chúng ta làm việc theo dạng module một cách đơn giản, giảm thiểu sự ràng buộc giữa các lớp đối tượng với nhau.





Kiến trúc MEF



Composition Container and Catalogs

- Thay vì đăng ký chính xác các ***component*** có thể sử dụng, MEF cung cấp một cách để phát hiện chúng ngầm định, thông qua **composition**. Một ***MEF component*** được gọi là **part**, được sử dụng thông qua ***imports*** và ***exports***.
- Trái tim của **MEF composition model** là **Composition Container** kiểm soát việc ***import*** và ***export*** các **part**.
- Để kiểm soát những **part** có thể sử dụng, **Composition Containers** sử dụng một **Catalog**.

Import và Export

- Để sử dụng MEF cần ***reference*** **System.ComponentModel.Composition**
- **[Import]** có thể coi như là điểm chờ item ***plug***
- **[Export]** để chỉ ra một item để ***plug***

```
[Import]
string message;
```

Khai báo nơi sử dụng (ví dụ class Program)

Khai báo nơi tạo plug-in (ví dụ class ExportMessage)

```
[Export()]
public string MyMessage
{
    get { return "This is my example message."; }
}
```

Compose Catalog

- Cần có **Catalog** để giữ các ***part*** (***import***, ***export***).

```
AssemblyCatalog catalog = new AssemblyCatalog(typeof(Program).Assembly);
```

- Tạo **Composition Container** để ***compose*** **catalog**

```
CompositionContainer container = new CompositionContainer(catalog);
```

- Thực hiện ***compose*** với phương thức hỗ trợ
  - ***ComposeParts***
  - ***SatisfyImportOnce***

```
container.SatisfyImportsOnce(this);
```

```
container.ComposeParts(this);
```

Các property

- Cho phép ***load import*** mặc định

```
[Import(AllowDefault =true)]
string message;
```

- **CreationPolicy** (***Shared***, ***NonShared*** và ***Any***) xác định cách tạo các ***instance*** của **Import**. Mặc định của **Export** là ***Any*** và **Import** là ***Shared***.

```
[Export, PartCreationPolicy(CreationPolicy.Any)]
public class MyClass
{
    public int info;
}
```

```
[Import(RequiredCreationPolicy = CreationPolicy.NonShared)]
MyClass _port1;
[Import]
MyClass _port2;
```

Specifying Export Types

- **Export** với chỉ định ***Type*** cụ thể

```
[Export("MyMsg", typeof(Func<string, string>))]
public string MyFunc(string msg)
{
    return string.Format("Parameter is {0}", msg);
}
[Export("Msg", typeof(Func<string, string>))]
public string FuncMsg(string msg)
{
    return string.Format("Message is {0}", msg);
}
```

```
[Import("MyMsg", typeof(Func<string, string>))]
Func<string, string> ImportFunc;
[Import("Msg", typeof(Func<string, string>))]
Func<string, string> ImportFunc2;
```

Inheriting an Export

- Cho phép ***export*** tất cả các ***class*** dẫn xuất, ***implement*** từ **InheritedExport**.

```
[InheritedExport(typeof(IExport))]
public interface IExport
{
    string GetMsg();
}
public class Export1 : IExport
{
    public string GetMsg()
    {
        return "Export1";
    }
}
public class Export2 : IExport
{
    public string GetMsg()
    {
        return "Export2";
    }
}
```

Import Many

- **ImportMany** cho phép ***import*** nhiều ***item*** đã được ***export***.

```
[ImportMany(typeof(IExport))]
List<IExport> _list;
```

```
foreach (var item in _list)
{
    Console.WriteLine(item.GetMsg());
}
```

Export metadata

```
public interface IMeta
{
    string Msg(string name);
}

[Export(typeof(IMeta))]
[ExportMetadata("Region", "HCM")]
[ExportMetadata("City", true)]
public class HCM : IMeta
{
    public string Msg(string name)
    {
        return string.Format("{0} w 08", name);
    }
}

[Export(typeof(IMeta))]
[ExportMetadata("Region", "LongAn")]
[ExportMetadata("City", false)]
public class LongAn : IMeta
{
    public string Msg(string name)
    {
        return string.Format("{0} w 11", name);
    }
}
```

```
[ImportMany(typeof(IMeta))]
IEnumerable<Lazy<IMeta, IDictionary<string, object>>> _metas;
```

```
foreach (var item in _metas)
{
    if (!(bool)item.Metadata["City"])
    {
        Console.Write("Not ");
    }
    Console.WriteLine("City: {0} - {1}",
item.Metadata["Region"], item.Value.Msg("Test"));
}
```

```
City: HCM - Test w 08
Not City: LongAn - Test w 11
```

Load Add-in từ bên ngoài

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="30"/>
        <RowDefinition/>
    </Grid.RowDefinitions>
    <Menu Name="menu">
        <MenuItem Name="mi" Header="Add-ins"/>
    </Menu>
    <TextBlock Name="tbl" Grid.Row="1" FontSize="32" FontWeight="Bold"
               VerticalAlignment="Center" HorizontalAlignment="Center"/>
</Grid>
```

```
public interface IMenu
{
    string Header { get; }
    string PerformClick { get; }
}
public interface IMenuData
{
    bool IsMenu { get; }
}
```

```
[Export(typeof(IMenu))]
[ExportMetadata("IsMenu", true)]
public class Menu1 : IMenu
{
    public string Header
    {
        get
        {
            return "Menu 1";
        }
    }

    public string PerformClick
    {
        get
        {
            return "Click from Menu 1";
        }
    }
}
```

```
[ImportMany(typeof(IMenu))]
IEnumerable<Lazy<IMenu, IMenuData>> _menus;
public MainWindow()
{
    InitializeComponent();
    DirectoryCatalog catalog = new DirectoryCatalog("../../AddIns");
    CompositionContainer cc = new CompositionContainer(catalog);
    cc.ComposeParts(this);
    foreach (var item in _menus)
    {
        if (item.Metadata.IsMenu)
        {
            var mitem = new MenuItem();
            mitem.Header = item.Value.Header;
            mitem.Click += Mitem_Click;
            mitem.Tag = item.Value.PerformClick;
            mi.Items.Add(mitem);
        }
    }
}
```