

Chương 2: Mô hình MVC trong ASP.NET

2.2 View trong ASP.NET MVC

- ◆ Định nghĩa và mô tả View
- ◆ Giải thích và mô tả công cụ Razor View
- ◆ Định nghĩa và mô tả phương thức hỗ trợ HTML

- ◆ Để hiển thị nội dung HTML tới người dùng, ta có thể hướng các hành động controller trong ứng dụng trả về 1 view
- ◆ Một view:
 - ◆ Cung cấp UI của ứng dụng tới người dùng
 - ◆ Được sử dụng để hiển thị nội dung của ứng dụng và tiếp nhận các input người dùng
 - ◆ Sử dụng các dữ liệu mẫu để tạo ra UI
 - ◆ Bao gồm các HTML markup và mã lệnh chạy trên Web server

- ◆ Là 1 phần của MVC Framework, có tác dụng chuyển đổi mã code của 1 view thành HTML markup để trình duyệt có thể hiểu được
- ◆ MVC Framework cung cấp 2 công cụ View sau:
 - ◆ **Công cụ Web form view:** là 1 công cụ view sử dụng các câu lệnh giống như ASP.NET web form. Công cụ view này là công cụ mặc định cho MVC 1 và MVC 2.
 - ◆ **Công cụ Razor view :** là công cụ view mặc định bắt đầu từ MVC 3. Công cụ này không giới thiệu một ngôn ngữ lập trình mới. Thay vào đó, nó giới thiệu cú pháp markup mới để chuyển đổi giữa HTML markup và mã code lập trình một cách đơn giản hơn.

Xác định view cho một hành động

- ◆ Khi tạo ứng dụng ASP.NET MVC, bạn cần xác định 1 view sẽ kết xuất output với một hành động cụ thể nào đó
- ◆ Khi tạo 1 project mới trên Visual Studio .NET, project sẽ mặc định chứa 1 danh mục **Views**
- ◆ Trong 1 ứng dụng, nếu 1 hành động controller trả về một view, ứng dụng của bạn sẽ:
 - Có một thư mục cho controller, với tên giống như tên controller mà không có hậu tố 'Controller' phía sau.
 - Có một tập tin view trong thư mục **Home**, với tên trùng với tên của hành động.

Xác định view cho một hành động

- ◆ Code Snippet dưới đây chỉ ra hành động **Index** trả về đối tượng **ActionResult** thông qua việc gọi phương thức **View()** của class **Controller**

```
public class HomeController : Controller {  
    public ActionResult Index()  
    {  
        return View();  
    }  
}
```

- ◆ Code snippet trên chỉ ra hành động **Index** của **HomeController** trả về kết quả tới phương thức **View()**. Kết quả của phương thức **View** là một đối tượng **ActionResult** kết xuất một view.

Xác định view cho một hành động

◆ Tạo tập tin view theo các bước sau:

1. Bấm chuột phải vào phương thức hành động cần tạo view
2. Chọn **Add View** từ thanh menu xuất hiện. Hộp thoại **Add View** xuất hiện sau đó.

The screenshot shows the 'Add View' dialog box with the following configuration:

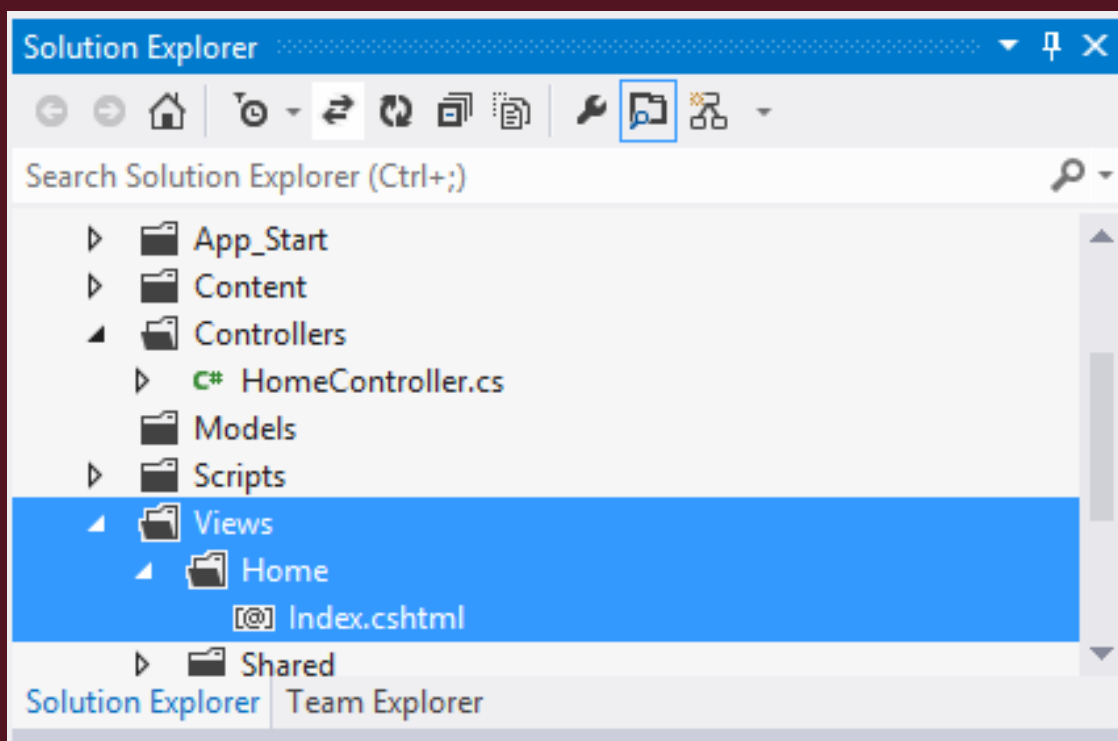
- View name:** Index
- View engine:** Razor (CSHTML)
- ☐ Create a strongly-typed view
 - Model class:** (empty)
- Scaffold template:** Empty
- ☒ Reference script libraries
- ☐ Create as a partial view
- ☒ Use a layout or master page:
 - (Leave empty if it is set in a Razor _viewstart file)
- ContentPlaceHolder ID:** MainContent

Buttons: Add, Cancel

Xác định view cho một hành động

3. Chọn **Add**. Visual Studio .NET tự động tạo ra cấu trúc thư mục phù hợp và thêm tập tin view vào.

Hình dưới đây chỉ ra tập tin view mà Visual Studio .NET tạo ra cho phương thức hành động **Index** của class **HomeController** ở cửa sổ **Solution Explorer**.



Xác định view cho một hành động

Trong tập tin ***Index.cshtml***, bạn có thể thêm nội dung mà View sẽ hiển thị. Code snippet 3 chỉ ra nội dung của View ***Index.cshtml***

```
<!DOCTYPE html>
<html>
  <head>
    <title>Test View</title>
  </head>
  <body>
    <h1> Welcome to the Website  </h1>
  </body>
</html>
```

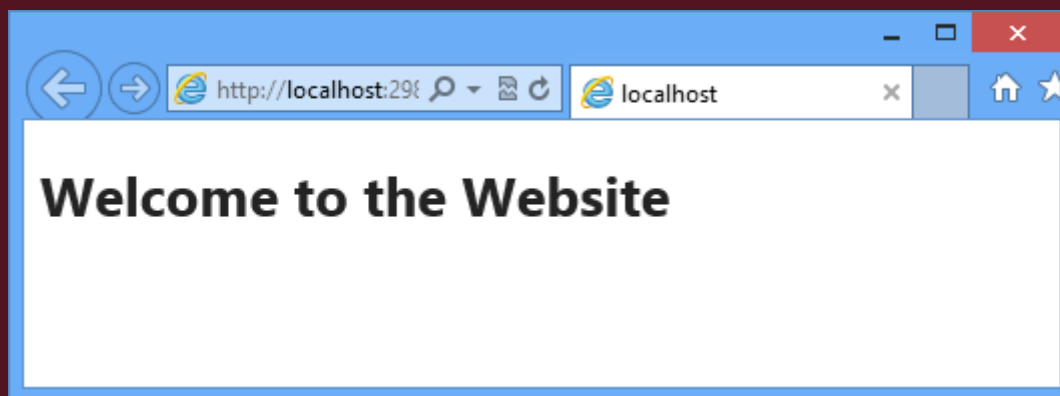
Đoạn code trên tạo ra 1 view với 1 tiêu đề và 1 thông điệp

Xác định view cho một hành động

- ◆ Khi bạn truy cập hành động **Index** của **HomeController** từ trình duyệt, View ***Index.cshtml*** sẽ được hiển thị.
- ◆ URL sau truy cập tới hành động **Index** của **HomeController**:

`http://localhost:1267/Home/Index`

Hình sau chỉ ra View ***Index.cshtml*** kết xuất trong trình duyệt



Xác định view cho một hành động

- ◆ Ta còn có thể kết xuất 1 view khác từ 1 phương thức hành động
- ◆ Để trả về 1 view khác, ta cần dùng tên của view đó dưới dạng một tham số
- ◆ Ví dụ ở đoạn code sau:

Code Snippet:

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        return View("TestIndex");
    }
}
```

Đoạn code sẽ tìm 1 view bên trong tập tin */Views/Home*, nhưng kết xuất view ***TestIndex*** thay vì kết xuất view **Index**

Xác định view cho một hành động

- ◆ Khi lập trình ứng dụng ASP.NET MVC, bạn cũng có thể cần kết xuất 1 view có trong thư mục khác thay vì thư mục mặc định
- ◆ Ta cần chỉ định đường Path chỉ tới view
- ◆ Ví dụ:

Code Snippet

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        return View("~/Views/Demo/Welcome.cshtml");
    }
}
```

Đoạn code trên hiển thị 1 view tên *Welcome.cshtml* trong thư mục */Views/Demo*.

Truyền dữ liệu từ controller tới view

- ◆ Trong ứng dụng ASP.NET MVC, 1 controller thường biểu diễn business logic của ứng dụng và cần trả về kết quả tới người dùng thông qua view.
- ◆ Bạn có thể dùng các đối tượng sau để chuyển dữ liệu giữa controller và view :
 - ◆ ViewData
 - ◆ ViewBag
 - ◆ TempData

Truyền dữ liệu từ controller tới view

◆ ViewData

- ◆ Truyền dữ liệu từ controller tới view.
- ◆ Là một dictionary của các đối tượng, được lấy từ class **ViewDataDictionary**.
- ◆ Một số tính chất:
 - ◆ 1 đối tượng ViewData chỉ tồn tại trong yêu cầu hiện tại
 - ◆ Giá trị của ViewData sẽ trở thành null nếu yêu cầu bị chuyển hướng
 - ◆ ViewData yêu cầu chuyển kiểu khi sử dụng kiểu dữ liệu phức tạp để tránh lỗi
- ◆ Cú pháp của ViewData:

Syntax:

```
ViewData[<key>] = <Value>;
```

Trong đó,

- **Key**: 1 giá trị chuỗi để xác định đối tượng hiện có trong ViewData
- **Value**: đối tượng hiện có trong ViewData. Đối tượng này có thể ở kiểu String hoặc 1 kiểu khác, ví dụ như **DateTime**.

Truyền dữ liệu từ controller tới view

Code Snippet 5 biểu thị 1 ViewData với 2 cặp key-value trong phương thức hành động **Index** của class **HomeController**

Code Snippet:

```
public class HomeController : Controller {  
    public ActionResult Index()  
    {  
        ViewData["Message"] = "Message from ViewData";  
        ViewData["CurrentTime"] = DateTime.Now; return View();  
    }  
}
```

Trong đoạn code trên, 1 ViewData được tạo ra với 2 cặp key-value. Key đầu tiên có tên **Message** chứa giá trị String *"Message from ViewData"*. Key thứ 2 có tên **CurrentTime** chứa giá trị **DateTime.Now**

Truyền dữ liệu từ controller tới view

- ◆ Đoạn code chỉ ra cách hiển thị các giá trị hiện có trong ViewData:

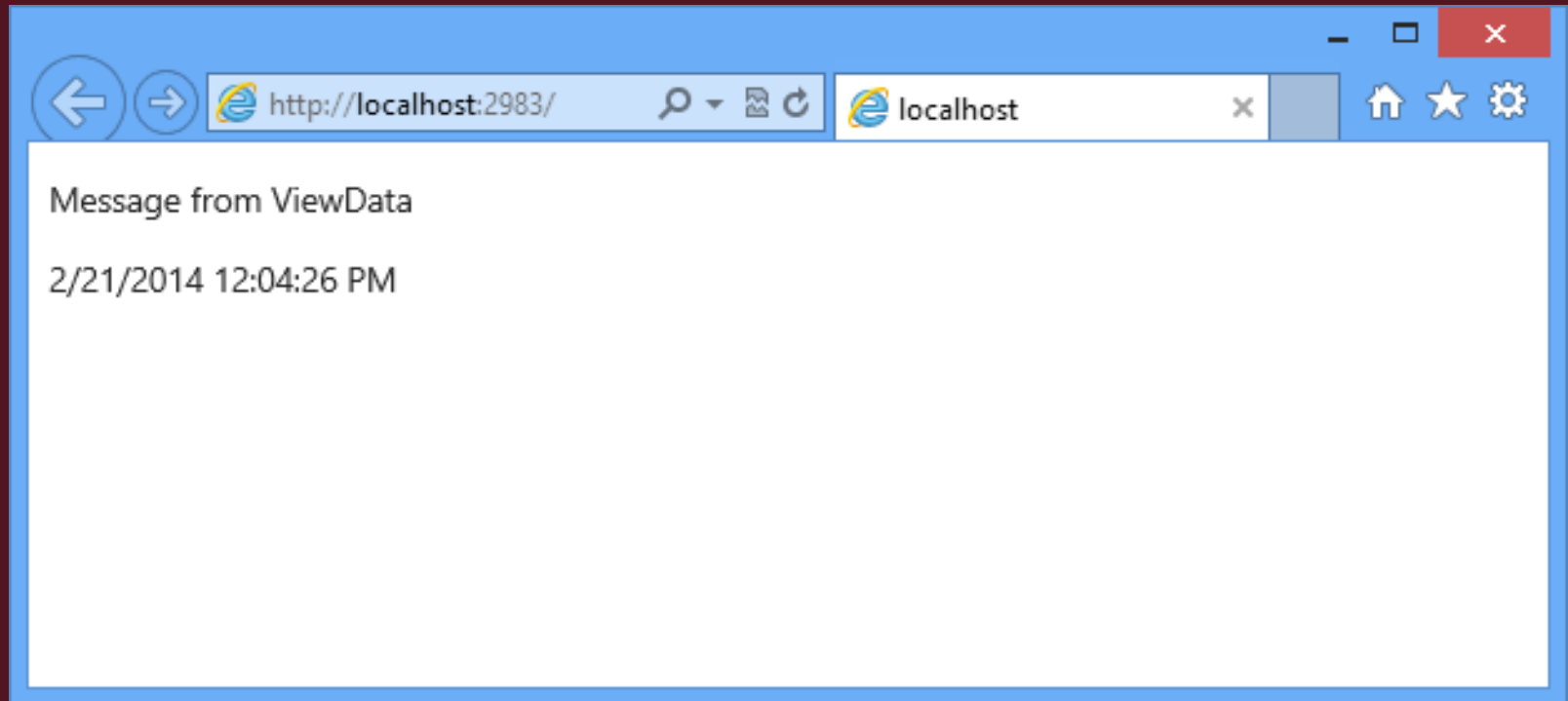
Code Snippet:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Index View</title>
  </head>
  <body>
    <p> @ViewData["Message"] </p>
    <p> @ViewData["CurrentTime"] </p>
  </body>
</html>
```

Trong đoạn code trên, ViewData được sử dụng để hiển thị các giá trị của các key **Message** và **CurrentTime**. Hình sau chỉ ra output của ViewData.

Truyền dữ liệu từ controller sang view

- ◆ Hình sau chỉ ra output của ViewData:



Truyền dữ liệu từ controller tới view

◆ ViewBag:

- ◆ Là một vỏ bọc của ViewData.
- ◆ Chỉ tồn tại trong yêu cầu hiện tại và sẽ trở thành null nếu yêu cầu bị chuyển hướng
- ◆ Là một thuộc tính động dựa trên các tính năng động được giới thiệu trong C# 4.0
- ◆ Không yêu cầu chuyển kiểu khi sử dụng các kiểu dữ liệu phức tạp

◆ Cú pháp của ViewBag:

Syntax:

```
ViewBag.<Property> = <Value>;
```

Trong đó,

- **Property**: là giá trị String biểu diễn 1 thuộc tính trong ViewBag
- **Value**: là giá trị của thuộc tính trong ViewBag

Truyền dữ liệu từ controller tới view

- ◆ Đoạn code sau chỉ ra một ViewBag với 2 thuộc tính trong phương thức hành động **Index** của class **HomeController**:

Code Snippet:

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        ViewBag.Message = "Message from ViewBag";
        ViewBag.CurrentTime = DateTime.Now;
        return View();
    }
}
```

Trong đoạn code trên, 1 ViewBag được tạo với 2 thuộc tính. Thuộc tính đầu tiên tên **Message**, có chứa giá trị String *"Message from ViewBag"*. Thuộc tính thứ hai tên **CurrentTime** chứa giá trị **DateTime.Now**

Truyền dữ liệu từ controller tới view

- ◆ Đoạn code sau chỉ ra cách hiển thị các giá trị có trong ViewBag:

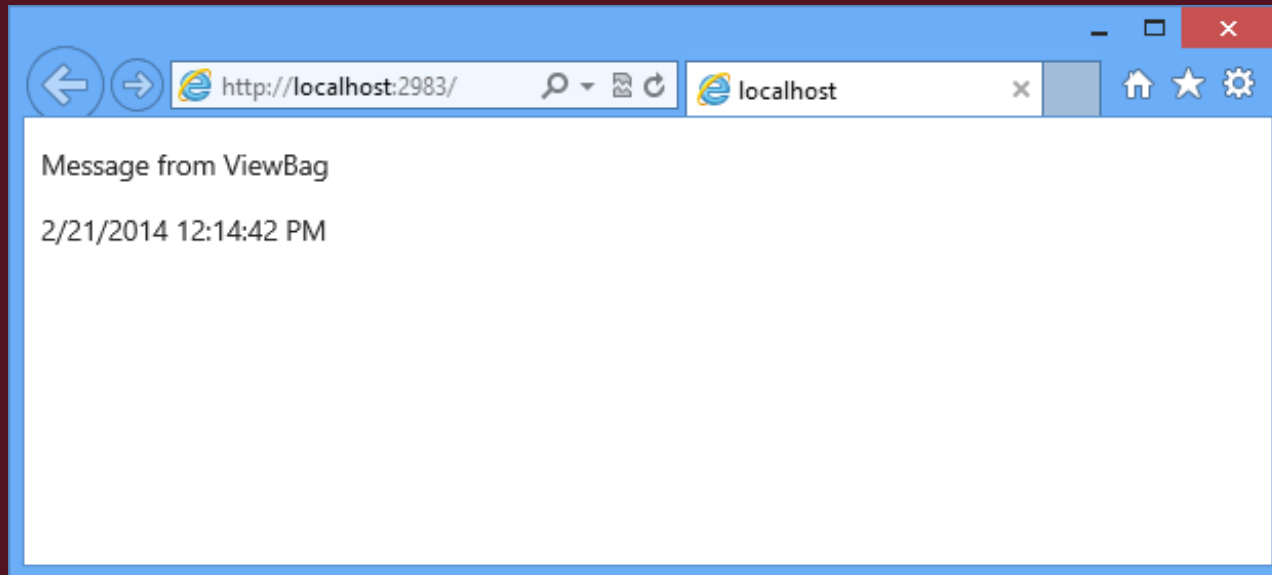
Code Snippet:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Index View</title>
  </head>
  <body>
    <p>
      @ViewBag.Message
    </p>
    <p>
      @ViewBag.CurrentTime
    </p>
  </body>
</html>
```

ViewBag được sử dụng để hiển thị các giá trị của thuộc tính **Message** và **CurrentTime**

Truyền dữ liệu từ controller sang view

- ◆ Output của ViewBag:



Khi sử dụng ViewBag để lưu trữ 1 thuộc tính và giá trị của nó trong một hành động, thuộc tính đó có thể được truy cập bởi cả ViewBag và ViewData trong 1 view.

Truyền dữ liệu từ controller tới view

♦ Ví dụ:

Code Snippet:

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        ViewBag.CommonMessage = "Common message accessible to both
ViewBag and ViewData";
        return View();
    }
}
```

1 ViewBag được tạo ra với tên **CommonMessage**

Truyền dữ liệu từ controller sang view

Code snippet sau chỉ ra 1 view sử dụng cả ViewData và ViewBag để truy cập tới thuộc tính **CommonMessage** lưu trữ trong ViewBag

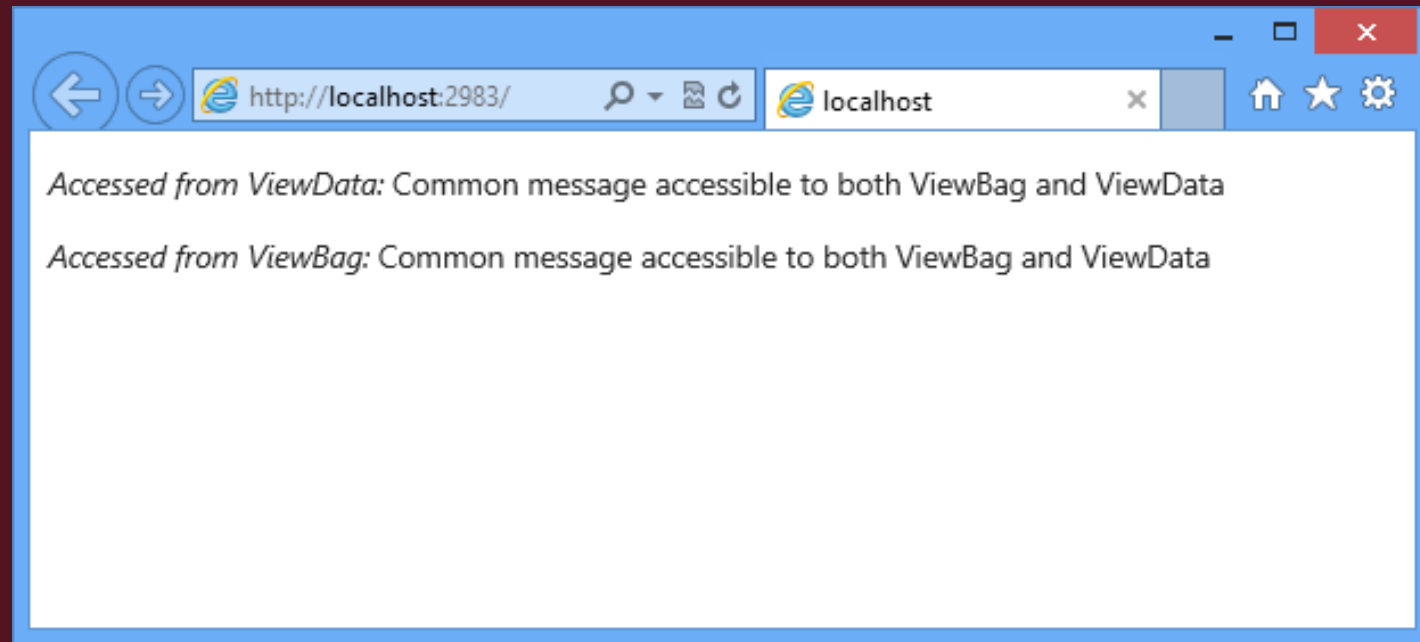
Code Snippet:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Index View</title>
  </head>
  <body>
    <p>
      <em>Accessed from ViewData:</em> @ViewData["CommonMessage"]
    </p>
    <p>
      <em>Accessed from ViewBag:</em> @ViewBag.CommonMessage
    </p>
  </body>
</html>
```

Đoạn code sử dụng cả ViewData và ViewBag để hiển thị giá trị của thuộc tính **CommonMessage** được lưu trữ trong ViewBag

Truyền dữ liệu từ controller tới view

- ◆ Kết quả:



Truyền dữ liệu từ controller tới view

◆ TempData:

- ◆ Là một đối tượng dictionary được lấy từ class **TempDataDictionary**.
- ◆ Lưu trữ dữ liệu dưới dạng key-value
- ◆ Cho phép truyền dữ liệu từ yêu cầu hiện có tới yêu cầu tiếp theo trong khi chuyển hướng yêu cầu

Cú pháp của TempData:

Syntax:

```
TempData[<Key>] = <Value>;
```

Trong đó,

- **Key**: là giá trị String xác định đối tượng hiện có trong TempData
- **Value**: là đối tượng hiện có trong TempData

Truyền dữ liệu từ controller tới view

- ◆ Code snippet sau biểu diễn cách sử dụng TempData để truyền các giá trị từ 1 view tới 1 view khác trong quá trình chuyển hướng yêu cầu:

Code Snippet

```
public class HomeController : Controller {  
    public ActionResult Index() {  
        ViewData["Message"] = "ViewData Message";  
        ViewBag.Message = "ViewBag Message";  
        TempData["Message"] = "TempData Message";  
        return Redirect("Home/About");  
    }  
    public ActionResult About() {  
        return View();  
    }  
}
```

- ◆ Đoạn code trên tạo ra 2 hành động: **Index** và **About** trong class **HomeController**. **Index** lưu trữ giá trị của các đối tượng ViewData, ViewBag và TempData. **Index** sau đó chuyển hướng yêu cầu tới hành động **About** bằng cách gọi phương thức **Redirect()**. Hành động **About** trả về view tương ứng là view **About.cshtml**

Truyền dữ liệu từ controller tới view

Hành động **About** trả về view tương ứng là view **About.cshtml** như code snippet 12:

Code Snippet 12:

```
<!DOCTYPE html>
<html>
<head>
<title>Index View</title>
</head>
<body>
<p>
<em>Accessed from ViewData:</em> @ViewData["Message"]
</p>
<p>
<em>Accessed from ViewBag:</em> @ViewBag.Message
</p>
<p>
<em>Accessed from TempData:</em> @TempData["Message"]
</p>
</body>
</html>
```

Truyền dữ liệu từ controller sang view

Hành động **About** trả về view tương ứng là view **About.cshtml** như code snippet 12:

Code Snippet:

```
<!DOCTYPE html>
<html>
<head>
<title>Index View</title>
</head>
<body>
<p>
<em>Accessed from ViewData:</em> @ViewData [ "Message"]
</p>
<p>
<em>Accessed from ViewBag:</em> @ViewBag.Message
</p>
<p>
<em>Accessed from TempData: </em> @TempData["Message"]
</p>
</body>
</html>
```

Sử dụng Partial Views

- ◆ Trong ứng dụng ASP.NET MVC, 1 partial view:
 - ◆ Hiển thị một view phụ của view chính.
 - ◆ Cho phép tái sử dụng các markup thông dụng trên khắp các view khác nhau của ứng dụng.
- ◆ Để tạo ra partial view trong Visual Studio .NET, làm theo các bước sau:
 1. Bấm chuột phải vào thư mục **Views/Shared** trong cửa sổ **Solution Explorer** và chọn **Add->View**. Hộp thoại **Add View** xuất hiện.
 2. Trong hộp thoại **Add View**, xác định tên của partial view trong trường **View Name**
 3. Chọn **Create as a partial view**.

Sử dụng Partial View

- ◆ Cửa sổ **Add View** được hiển thị:

View name:
_TestPartialView

View engine:
Razor (CSHTML)

☐ Create a strongly-typed view

Model class:

Scaffold template:
Empty

☒ Reference script libraries

☒ Create as a partial view

☒ Use a layout or master page:
...
(Leave empty if it is set in a Razor _viewstart file)

ContentPlaceHolder ID:
MainContent

Add Cancel

4. Chọn **Add** để tạo partial view

Sử dụng Partial View

- ◆ Trong partial view, thêm markup mà bạn cần hiển thị trong view chính:

```
<h3> Content of partial view. </h3>
```

- ◆ Cú pháp của partial view:

Syntax:

```
@Html.Partial(<partial_view_name>)
```

Trong đó,

- `partial_view_name`: tên của partial view không có đuôi *'`.cshtml`'*

Sử dụng Partial View

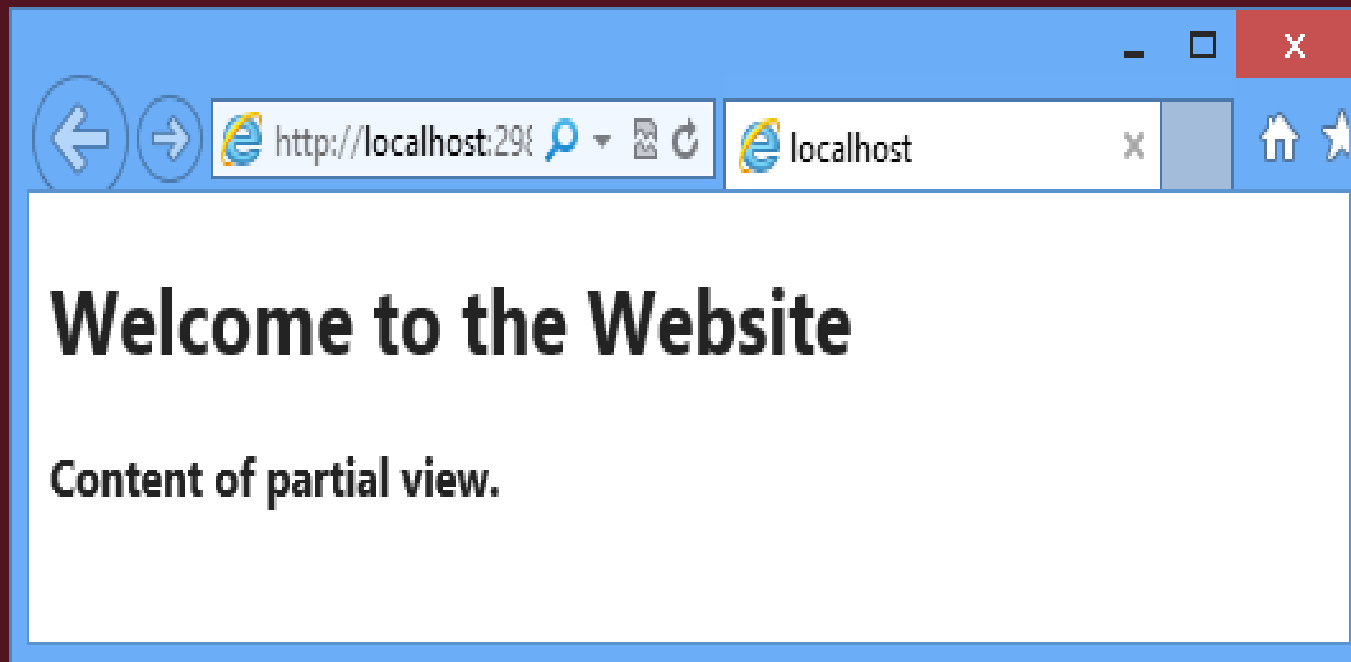
Code snippet biểu diễn 1 view chính tên *Index.cshtml* truy cập vào partial view tên *_TestPartialView.cshtml*

Code Snippet:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Index View</title>
  </head>
  <body>
    <h1>
      Welcome to the Website
    </h1>
    <div>@Html.Partial("_TestPartialView")</div>
  </body>
</html>
```

Đoạn code biểu diễn markup của view chính *Index.cshtml* hiển thị 1 thông điệp 'Welcome to the Website' và kết xuất với partial view tên *_TestPartialView.cshtml*

- ◆ Kết quả:



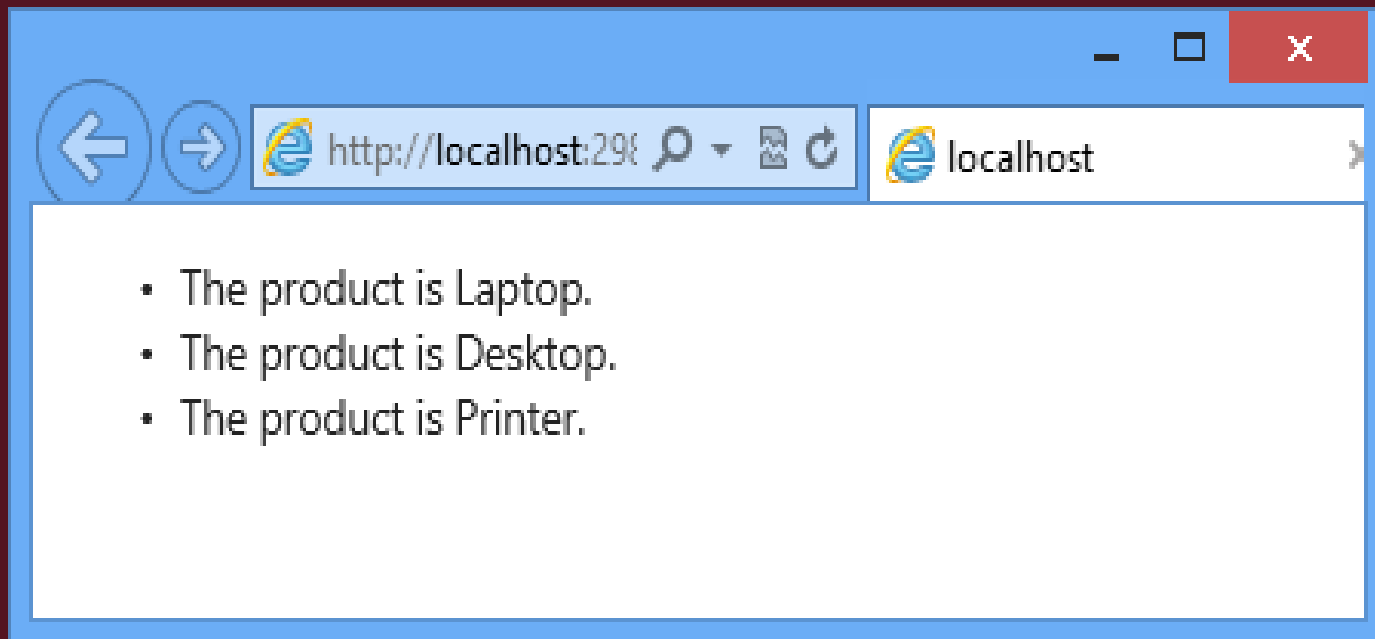
◆ Razor:

- ◆ Là cú pháp dựa trên ASP.NET Framework cho phép tạo ra các view.
- ◆ Đơn giản, dễ hiểu với các người dùng quen thuộc với các ngôn ngữ lập trình như Visual C# .NET hoặc Visual Basic (VB).
- ◆ Đoạn code snippet sau chỉ ra một Razor view đơn giản chứa các HTML markup và C# code để thực hiện logic của view:

Code Snippet:

```
@{var products = new string[] { "Laptop", "Desktop", "Printer" };}  
<html>  
<head><title>Test View</title></head>  
<body>  
<ul>  
@foreach (var product in products){  
<li>The product is @product.</li>  
</ul>  
</body>  
</html>
```

- ◆ Trong đoạn code, `string[]` được khai báo và khởi tạo sử dụng cú pháp Razor. Sau đó, cú pháp Razor được sử dụng để lặp qua các phần tử của mảng và hiển thị từng phần tử. Đoạn code còn lại trong view là HTML code hiển thị tiêu đề trang, thân trang và các thành phần mảng trong danh sách thứ tự.
- ◆ Kết quả:



- ◆ MVC Framework sử dụng công cụ view để chuyển đổi code của 1 view sang HTML markup mà trình duyệt có thể hiểu.
- ◆ Công cụ Razor:
 - ◆ Là công cụ view mặc định của MV Framework.
 - ◆ Biên dịch 1 view của ứng dụng khi view này được yêu cầu lần đầu.
 - ◆ Sau đó công cụ Razor view chuyển view mới được biên dịch tới các yêu cầu tiếp theo tới khi có thay đổi các view.
 - ◆ Không giới thiệu một tập hợp ngôn ngữ lập trình mới mà cung cấp cú pháp template markup để tách biệt HTML markup và code lập trình trong 1 view.
 - ◆ Công cụ Razor hỗ trợ Test Driven Development (TDD) cho phép kiểm thử độc lập các view trong ứng dụng.

Quy tắc cú pháp Razor

- ◆ Razor:
 - ◆ Để thông dịch mã code phía server được nhúng bên trong các tập tin view, đầu tiên Razor yêu cầu xác định mã code phía server từ markup code.
 - ◆ Razor sử dụng kí hiệu '@' để tách biệt code phía server và markup code.
- ◆ Khi tạo 1 Razor view, ta cần chú ý các quy tắc sau:
 - ◆ Các code block được chứa trong '@{' và '}'
 - ◆ Bắt đầu các biểu thức với '@'
 - ◆ Các biến được khai báo với từ khóa 'var'
 - ◆ Đặt các chuỗi string trong dấu ngoặc kép
 - ◆ Kết thúc một câu lệnh Razor bằng ';'.
 - ◆ Sử dụng *.cshtml* để lưu tập tin Razor view sử dụng ngôn ngữ lập trình C#
 - ◆ Sử dụng *.vbhtml* để lưu tập tin Razor view sử dụng ngôn ngữ lập trình VB

Quy tắc cú pháp Razor

- Razor hỗ trợ các code block bên trong 1 view. 1 code block là 1 phần của 1 view chỉ chứa mã code được viết bằng C# hoặc VB
- ◆ Cú pháp của code block:

Syntax:

```
@{ <code> }
```

Trong đó, code là phần mã C# hoặc VB được thi hành trên server.

- ◆ Code snippet sau biểu diễn 2 code block câu lệnh đơn trong Razor view

Code Snippet:

```
@{ var myMessage = "Hello World"; }  
@{ var num = 10; }
```

Mã code biểu diễn 2 code blocks là 2 câu lệnh đơn khai báo các biến **myMessage** và **num**. Kí tự '@{' đánh dấu bắt đầu mỗi code block và kí tự '}' đánh dấu kết thúc của code block.

Quy tắc cú pháp Razor

- ◆ Razor cũng hỗ trợ các code block có nhiều câu lệnh.
- ◆ Cho phép bỏ qua kí tự '@' ở các dòng code
- ◆ Ví dụ:

Code Snippet:

```
@{  
var myMessage = "Hello World";  
var num = 10;  
}
```

- ◆ Đoạn code dưới đây chỉ ra một code block có nhiều câu lệnh, khai báo 2 biến là **myMessage** và **num**

Quy tắc cú pháp Razor

- ◆ Tương tự code block, Razor sử dụng kí tự '@' cho một biểu thức.
- ◆ Ví dụ:

Code Snippet:

```
@{  
var myMessage = "Hello World";  
var num = 10;  
}  
@myMessage is numbered @num.
```

Đoạn code sử dụng 2 biểu thức với các biến **myMessage** và **num**, cùng với đó xuất ra kết quả của biểu thức. Khi công cụ Razor gặp kí tự '@', nó thông dịch tên biến ngay sau kí tự '@' thành mã code phía server và bỏ qua đoạn text tiếp theo. Do đó, bạn không cần phải chỉ định định phần cuối của Razor code cho các biểu thức.

Quy tắc cú pháp Razor

- ◆ Nhiều lúc, bạn có thể sẽ cần yêu cầu quá tải logic mà Razor sử dụng để xác định mã code phía server. Ví dụ, mỗi khi bạn yêu cầu hiển thị kí tự '@' trong 1 địa chỉ email, bạn có thể dùng @@. Code snippet sau biểu thị ví dụ:

Code Snippet:

```
<h3>The email ID is: john@@mvcexample.com</h3>
```

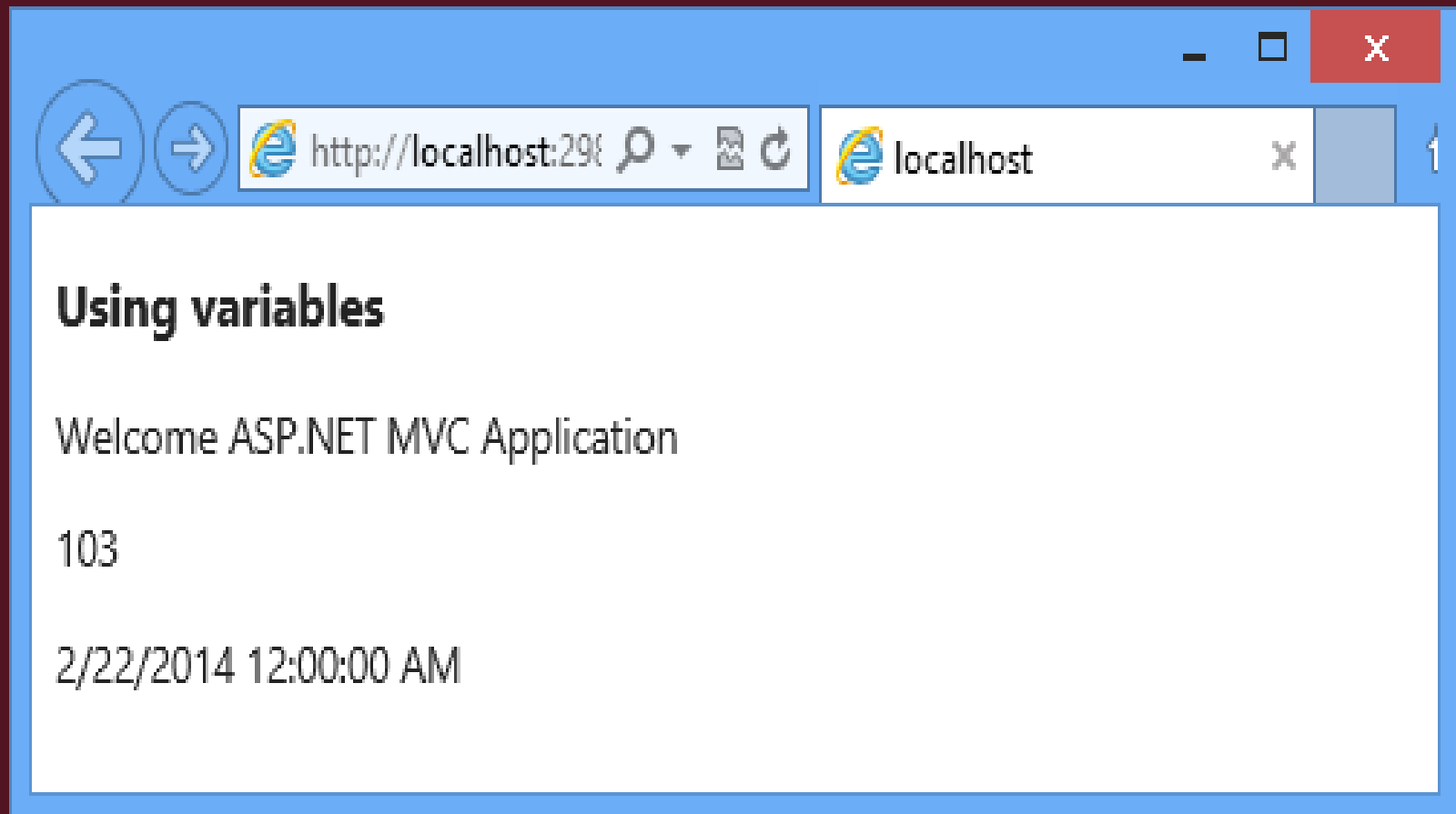
Razor code thông dịch kí tự '@' đầu tiên như một kí tự escape sequence. Để xác định phần nào là một phần của mã code phía server, Razor sử dụng một biểu thức ngầm. Đôi khi, Razor cũng sẽ thông dịch 1 biểu thức như 1 markup thay vì chạy chúng trên code phía server.

- ◆ Được sử dụng để lưu dữ liệu.
- ◆ Trong Razor, bạn khai báo và sử dụng biến giống như các chương trình C#
- ◆ Đoạn code dưới đây biểu diễn các khai báo biến sử dụng Razor

Code Snippet:

```
<!DOCTYPE html>
<html><body>
@{
    var heading = "Using variables";
    string greeting = "Welcome ASP.NET MVC Application";
    int num = 103;
    DateTime today = DateTime.Today;
    <h3>@heading</h3>
    <p>@greeting</p>
    <p>@num</p>
    <p>@today</p>
}
</body></html>
```

- ◆ Đoạn mã biểu diễn các biến được khai báo như **heading**, **greeting**, **num**, **today**
- ◆ Kết quả của đoạn code trên:



- ◆ Khi lập trình ứng dụng Web ASP.NET MVC, có thể ta sẽ cần yêu cầu thực hiện nhiều câu lệnh giống nhau một cách liên tiếp.
- ◆ Trong các trường hợp này, ta sử dụng vòng lặp
- ◆ C# hỗ trợ 4 kiểu vòng lặp:
 - ◆ `while`
 - ◆ `for`
 - ◆ `do...while`
 - ◆ `foreach`

◆ **while :**

- ◆ Sử dụng để thực hiện lặp một đoạn code khi điều kiện của vòng lặp đúng.
- ◆ Sử dụng từ khóa '*while*' ở phần đầu cùng cặp dấu ngoặc đơn
- ◆ Bên trong cặp ngoặc đơn, ta chỉ định điều kiện của vòng lặp. Cuối cùng là đoạn code thực hiện trong vòng lặp
- ◆ Đoạn code sau biểu thị Razor code bằng cách sử dụng vòng lặp để in các số từ 1 tới 7

Code Snippet:

```
<!DOCTYPE html>
<html>
<body>
@{ var b = 0;
while (b < 7) {
    b += 1;
    <p>Text @b</p> }
}
</body>
</html>
```

◆ For

- ◆ Khi ta biết được số lần mà một câu lệnh được lặp lại, bạn có thể sử dụng vòng lặp **for**
- ◆ Tương tự như **while**, vòng lặp **for** được sử dụng để lặp các phần tử
- ◆ Đoạn code sau sử dụng vòng lặp **for** để in ra các số chẵn từ 1 tới 10:

Code Snippet:

```
<!DOCTYPE html>
<html>
<body>
    <h1>Even Numbers</h1>
    @{ var num=1;
    for (num = 1; num <= 11; num++){
        if ((num % 2) == 0)
            { <p> @num</p>    }
        } }
</body>
</html>
```

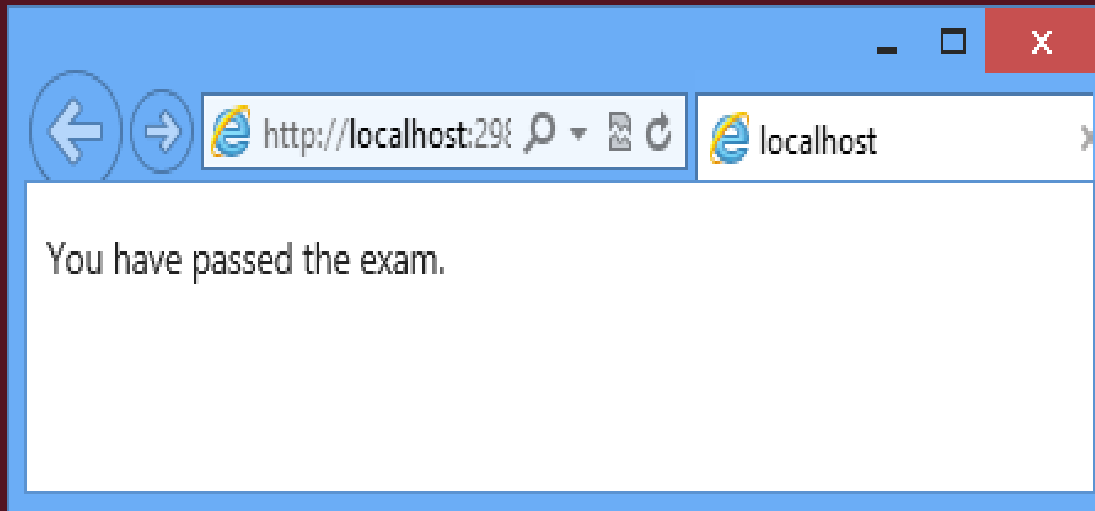
Câu lệnh điều kiện

- ◆ Cho phép hiển thị nội dung động dựa trên các điều kiện nhất định
- ◆ Câu lệnh **if** trả về true hoặc false. Sử dụng **if** và **else** để tạo ra các nội dung động cho view
- ◆ Đoạn code sau sử dụng câu lệnh **if.....else**

Code Snippet:

```
<!DOCTYPE html>
@{var mark=60;}
<html>
<body>
@if (mark>80) {
    <p>You have failed in the exam.</p> }
else {
    <p>You have passed the exam.</p>    }
```

- ◆ Kết quả của đoạn code trên:



Có thể sử dụng câu lệnh điều kiện khác, ví dụ như **switch.....case**.

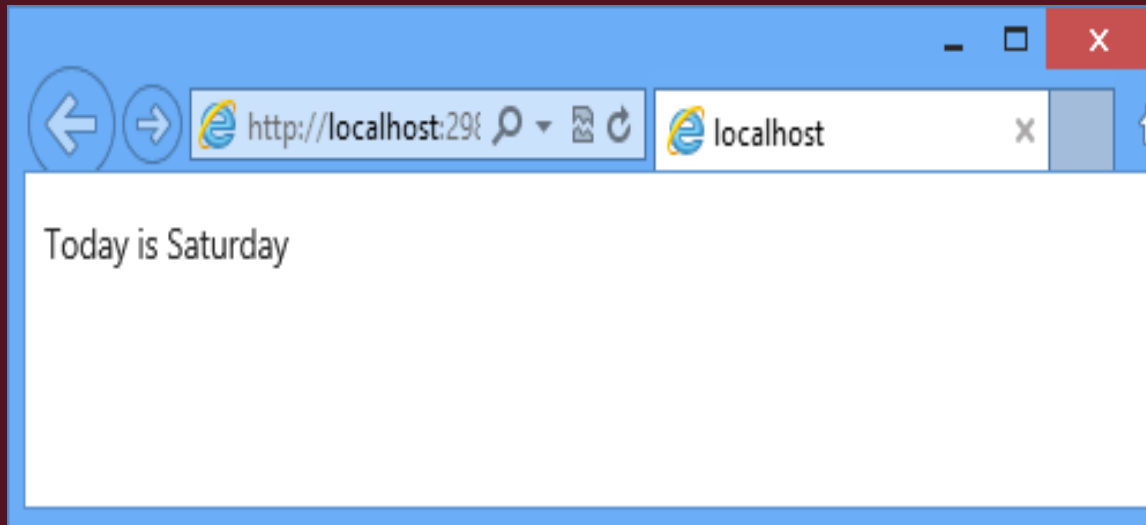
♦ Ví dụ về switch.....case:

Code Snippet:

```
<!DOCTYPE html>
@{
var day=DateTime.Now.DayOfWeek.ToString();
var msg=""; }
<html> <body>
@switch(day) {
    case "Monday":
        msg="Today is Monday, the first working day.";
        break;
    case "Friday":
        msg="Today is Friday, the last working day.";
        break;
    default:
        msg="Today is " + day;
        break; }
<p>@msg</p> </body> </html>
```

Câu lệnh điều kiện

- ◆ Kết quả của đoạn code trên:

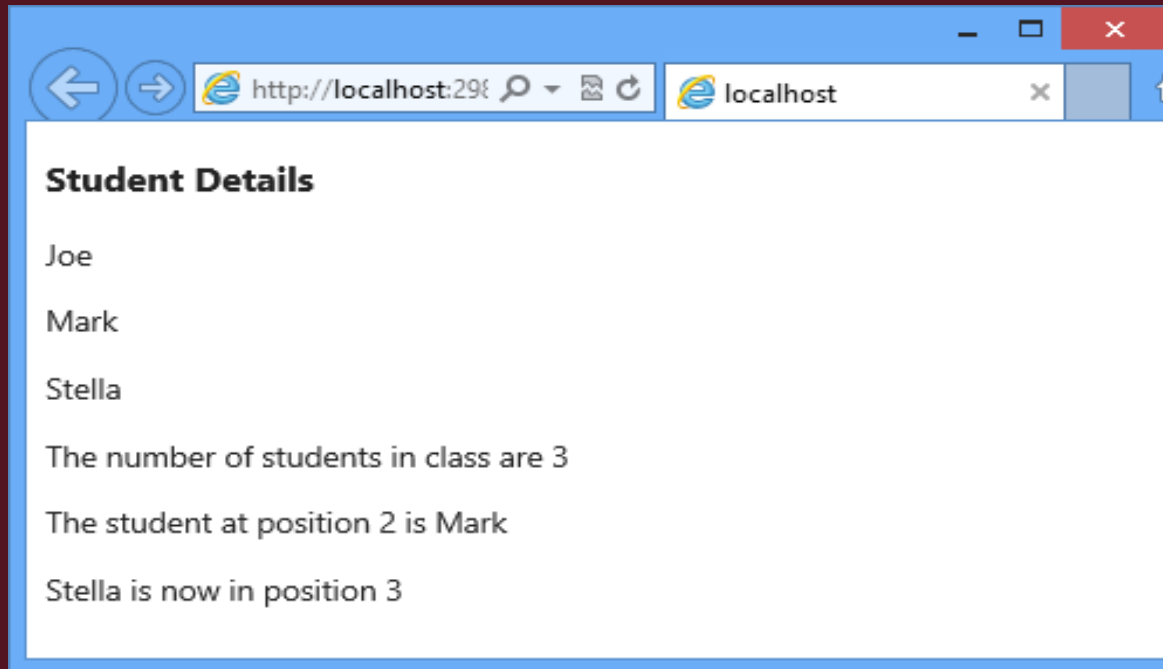


- ◆ Lưu trữ các biến có cùng kiểu
- ◆ Ví dụ về mảng:

Code Snippet:

```
<!DOCTYPE html>
@{
    string[] members = {"Joe", "Mark", "Stella"};
    int i = Array.IndexOf(members, "Stella")+1;
    int len = members.Length;
    string x = members[2-1]; }
<html> <body>
<h3>Student Details</h3>
@foreach (var person in members) {
    <p>@person</p> }
<p>The number of students in class are @len</p>
<p>The student at position 2 is @x</p>
<p>Stella is now in position @i</p>
</body>
</html>
```

- ◆ Kết quả của đoạn code trên:



Phương thức hỗ trợ HTML

- ◆ MVC Framework cung cấp các phương thức hỗ trợ HTML:
 - ◆ Là các phương thức mở rộng của class **HtmlHelper**.
 - ◆ Đơn giản hóa việc tạo view
 - ◆ Tạo các HTML markup và ta có thể tái sử dụng chúng khắp ứng dụng Web
- ◆ Một số các phương thức hỗ trợ HTML phổ biến:
 - ◆ `Html.ActionLink()`
 - ◆ `Html.BeginForm()` and `Html.EndForm()`
 - ◆ `Html.Label()`
 - ◆ `Html.TextBox()`
 - ◆ `Html.TextArea()`
 - ◆ `Html.Password()`
 - ◆ `Html.CheckBox()`

Phương thức hỗ trợ HTML

- ◆ **Html.ActionLink()** cho phép tạo ra một hyperlink chỉ tới 1 phương thức hành động của class **Controller**.
- ◆ Cú pháp cơ bản của phương thức **Html.ActionLink()** được biểu diễn dưới đây:

Syntax:

```
@Html.ActionLink(<link_text>,<action_method>,  
<optional_controller>)
```

Trong đó,

- **link_text**: đoạn văn bản hiển thị dưới dạng hyperlink
- **action_method**: tên của phương thức hành động mà hyperlink chỉ tới
- **optional_controller**: tên controller chứa phương thức hành động được gọi bởi hyperlink. Tham số này có thể bỏ qua, nếu như phương thức hành động được gọi ở trong cùng controller với phương thức hành động mà view kết xuất với hyperlink

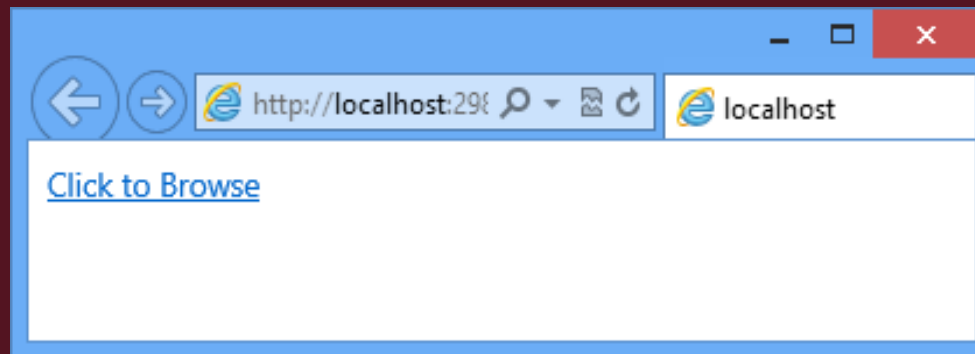
Phương thức hỗ trợ HTML

♦ Ví dụ:

Code Snippet:

```
<!DOCTYPE html>
<html>
<body>
    @Html.ActionLink("Click to Browse", "Browse", "Home")
</body>
</html>
```

Trong đoạn code, **ClicktoBrowse** là đoạn văn bản được hiển thị dưới dạng hyperlink. Phương thức hành động **Browse** của controller **Home** là phương thức được hyperlink nhắm tới.



Phương thức hỗ trợ HTML

- ◆ **Html.BeginForm()** :
 - ◆ Cho phép ta đánh dấu mở đầu của một form
 - ◆ Phối hợp cùng với công cụ định tuyến để tạo một URL
 - ◆ Có nhiệm vụ tạo ra thẻ **<form>** mở đầu
- ◆ Cú pháp cơ bản của phương thức hỗ trợ **Html.BeginForm()** được biểu thị dưới đây

Syntax:

```
@{Html.BeginForm(<action_method>, <controller_name>);}
```

Trong đó,

- **action_method**: tên phương thức hành động
- **controller_name**: tên của class controller

Khi sử dụng **Html.BeginForm()** để mở đầu form, bạn cũng cần phải kết thúc form. Sử dụng **Html.EndForm()** để kết thúc form

Phương thức hỗ trợ HTML

- ◆ Ví dụ về `Html.BeginForm()` và `Html.EndForm()` :

Code Snippet:

```
<!DOCTYPE html>
<html>
<body>
    @{Html.BeginForm("Browse", "Home");}
        <p>Inside Form</p>
    @{Html.EndForm();}
</body>
</html>
```

- ◆ Trong đoạn code, phương thức `Html.BeginForm()` xác định hành động **Browse** của controller **Home** như phương thức hành động mục tiêu mà dữ liệu form được gửi đến.
- ◆ Bạn có thể tránh việc sử dụng `Html.EndForm()` để đóng biểu mẫu bằng cách sử dụng câu lệnh `@using` ở phía trước phương thức `Html.BeginForm()`

Phương thức hỗ trợ HTML

- ◆ `Html.Label()` :
 - ◆ Hiển thị một label trong form
 - ◆ Có thể đính kèm thông tin vào các thành phần input khác, ví dụ như text input và nâng cao khả năng truy cập của ứng dụng
- ◆ Cú pháp:

Syntax:

```
@Html.Label(<label_text>)
```

Trong đó,

- **label_text**: tên của label

Phương thức hỗ trợ HTML

♦ Ví dụ về `Html.Label()` :

Code Snippet:

```
@Html.Label("name")
<!DOCTYPE html>
<html>
<body>
    @{Html.BeginForm("Browse", "Home");}
    @Html.Label("User Name:")<br>
    @{Html.EndForm();}
</body>
</html>
```

♦ `Html.TextBox()` :

- ♦ Hiển thị một thẻ nhập dữ liệu
- ♦ Sử dụng để tiếp nhận input từ người dùng.

Phương thức hỗ trợ HTML

- ◆ Cú pháp của `Html.TextBox()` :

Syntax:

```
@Html.TextBox("textbox_text")
```

Trong đó,

- **textbox_text**: tên của textbox

- ◆ Ví dụ về `Html.TextBox()` :

Code Snippet:

```
<!DOCTYPE html>
<html>
<body>
    @{Html.BeginForm("Browse", "Home");}
    @Html.Label("User Name:")</br>
    @Html.TextBox("textBox1")</br></br>
    <input type="submit" value="Submit">
    @{Html.EndForm();}
</body>
</html>
```

Phương thức hỗ trợ HTML

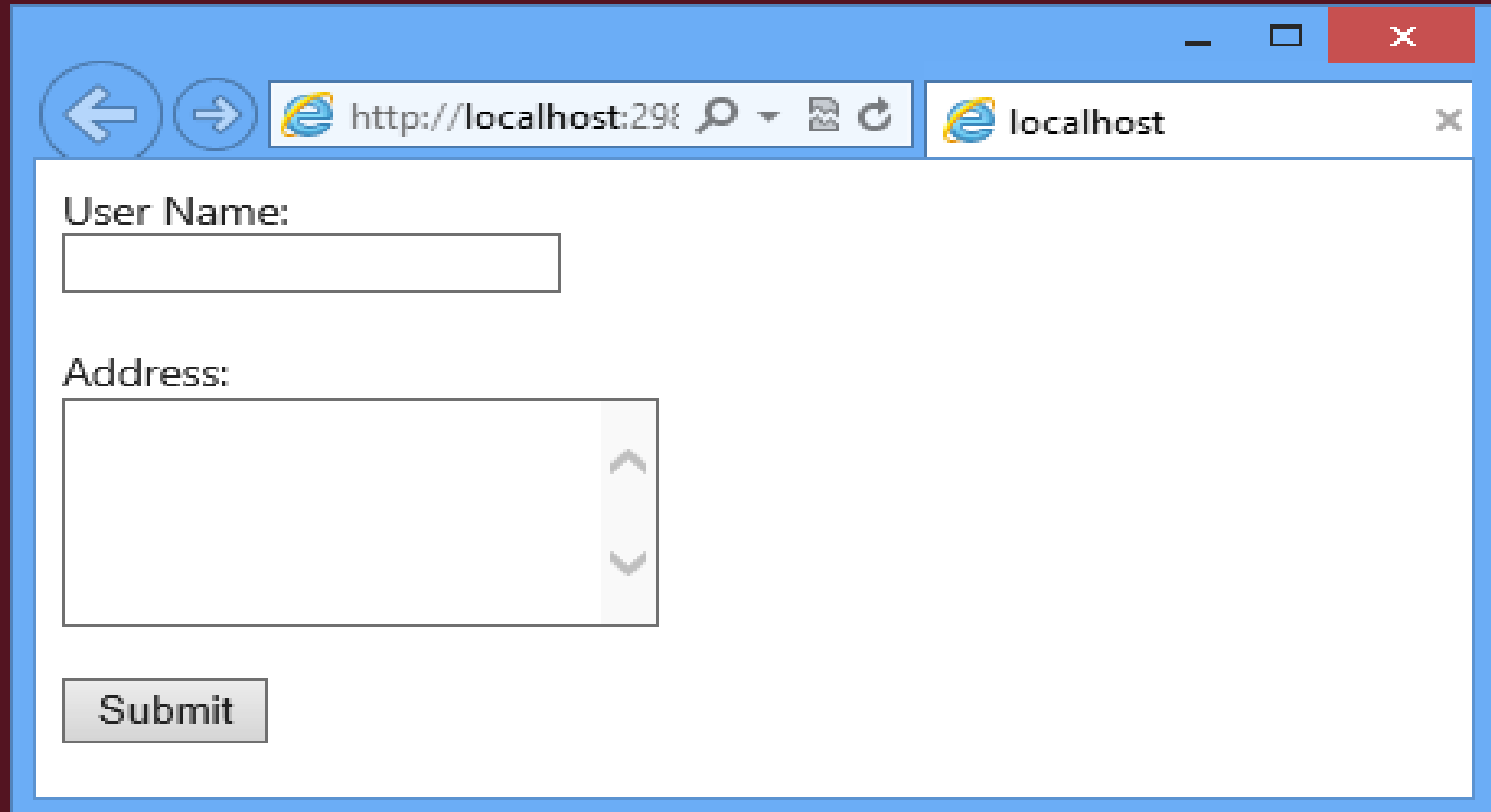
- ◆ **Html.TextArea()** :
 - ◆ Để hiển thị một thành phần **<textarea>** cho việc nhập văn bản nhiều dòng
 - ◆ Cho phép ta xác định số lượng dòng và cột xuất hiện, từ đó kiểm soát được kích thước của text area
- ◆ Ví dụ về **Html.TextArea()** :

Code Snippet:

```
<!DOCTYPE html>
<html>
<body>
    @{Html.BeginForm("Browse", "Home");}
    @Html.Label("User Name:")</br>
    @Html.TextBox("textBox1")</br></br>
    @Html.Label("Address:")</br>
    @Html.TextArea("textareal")</br></br>
    <input type="submit" value="Submit">
    @{Html.EndForm();}
</body>
</html>
```

Phương thức hỗ trợ HTML

- ◆ Kết quả của đoạn code trên:



The screenshot shows a web browser window with a blue title bar and a white address bar. The address bar contains the URL "http://localhost:2981" and a search icon. The browser window displays a simple HTML form with the following elements:

- A "User Name:" label followed by a text input field.
- An "Address:" label followed by a text area with a vertical scrollbar.
- A "Submit" button.

Phương thức hỗ trợ HTML

- ◆ Sử dụng `Html.Password()` để hiển thị trường password
- ◆ Ví dụ về `Html.Password()`:

Code Snippet:

```
<!DOCTYPE html>
<html>
<body>
  @{Html.BeginForm("Browse", "Home");}
  @Html.Label("User Name:")</br>
  @Html.TextBox("textBox1")</br></br>
  @Html.Label("Address:")</br> @Html.TextArea("textareal")</br></br>
  @Html.Label("Password:")</br>@Html.Password("password")</br></br>
  <input type="submit" value="Submit">
  @{Html.EndForm();}
</body>
</html>
```

Phương thức hỗ trợ HTML

- ◆ Sử dụng **Html.CheckBox()** để hiển thị checkbox, cho phép người dùng chọn một điều kiện đúng hoặc sai.
- ◆ Ví dụ về **Html.CheckBox()** :

Code Snippet:

```
<!DOCTYPE html>
<html><body>
    @{Html.BeginForm("Browse", "Home");}
    @Html.Label("User Name:")</br>
    @Html.TextBox("textBox1")</br></br>
    @Html.Label("Address:")</br> @Html.TextArea("textareal")</br></br>
    @Html.Label("Password:")</br>@Html.Password("password")</br></br>
    @Html.Label("I need updates on my mail:")
    @Html.CheckBox("checkbox1")</br> </br>
    <input type="submit" value="Submit"> @{Html.EndForm();}
</body> </html>
```

Trong đoạn code, **Html.CheckBox()** kết xuất một input bị ẩn với input check box. Khi người dùng tích check box, trình duyệt xác nhận giá trị của check box. Input bị ẩn này sẽ đảm bảo rằng một giá trị sẽ được xác nhận, bất kể là check box có được tích hay không

Phương thức hỗ trợ HTML

- ◆ **Html.DropDownList()** :
 - ◆ Trả về một thành phần `<select/>` hiển thị danh sách các lựa chọn và đồng thời là giá trị hiện có của trường đó
 - ◆ Cho phép chọn từng mục đơn
- ◆ Cú pháp cơ bản của **Html.DropDownList()** :

Syntax:

```
@Html.DropDownList("myList", new SelectList(new [] {<value1>, <value2>, < value3>}), "Choose")
```

Trong đó,

- **value1**, **value2**, và **value3** là các lựa chọn của danh sách
- **Choose**: giá trị ở đầu danh sách

Phương thức hỗ trợ HTML

♦ Ví dụ về `Html.DropDownList()` :

Code Snippet:

```
<!DOCTYPE html>
<html><body>    @{Html.BeginForm("Browse", "Home");}
    @Html.Label("User Name:")</br>
    @Html.TextBox("textBox1")</br></br>
    @Html.Label("Address:")</br> @Html.TextArea("textareal")</br></br>
    @Html.Label("Password:")</br>@Html.Password("password")</br></br>
    @Html.Label("I need updates on my mail:")
    @Html.CheckBox("checkbox1")</br> </br>
    @Html.Label("Select your city:")
    @Html.DropDownList("myList", new SelectList(new [] {"New York",
"Philadelphia", "California"}), "Choose")</> </br></br>
    <input type="submit" value="Submit">
    @{Html.EndForm();}
</body></html>
```

Trong đoạn code này, `Html.DropDownList()` tạo ra một danh sách drop-down trong 1 form với tên là **myList** và chứa 3 giá trị người dùng có thể chọn

Phương thức hỗ trợ HTML

- ◆ Phương thức `Html.RadioButton()` cung cấp một lượng lớn các lựa chọn giá trị đơn
- ◆ Cú pháp của phương thức `Html.RadioButton()` :

Syntax

```
@Html.RadioButton("name", "value", isChecked)
```

Trong đó,

- ◆ **name**: tên của radio button
- ◆ **value**: giá trị được gán vào các lựa chọn radio button
- ◆ **isChecked**: là một giá trị Boolean, xác định radio button nào được chọn hay không

Phương thức hỗ trợ HTML

◆ Ví dụ về `Html.RadioButton()`:

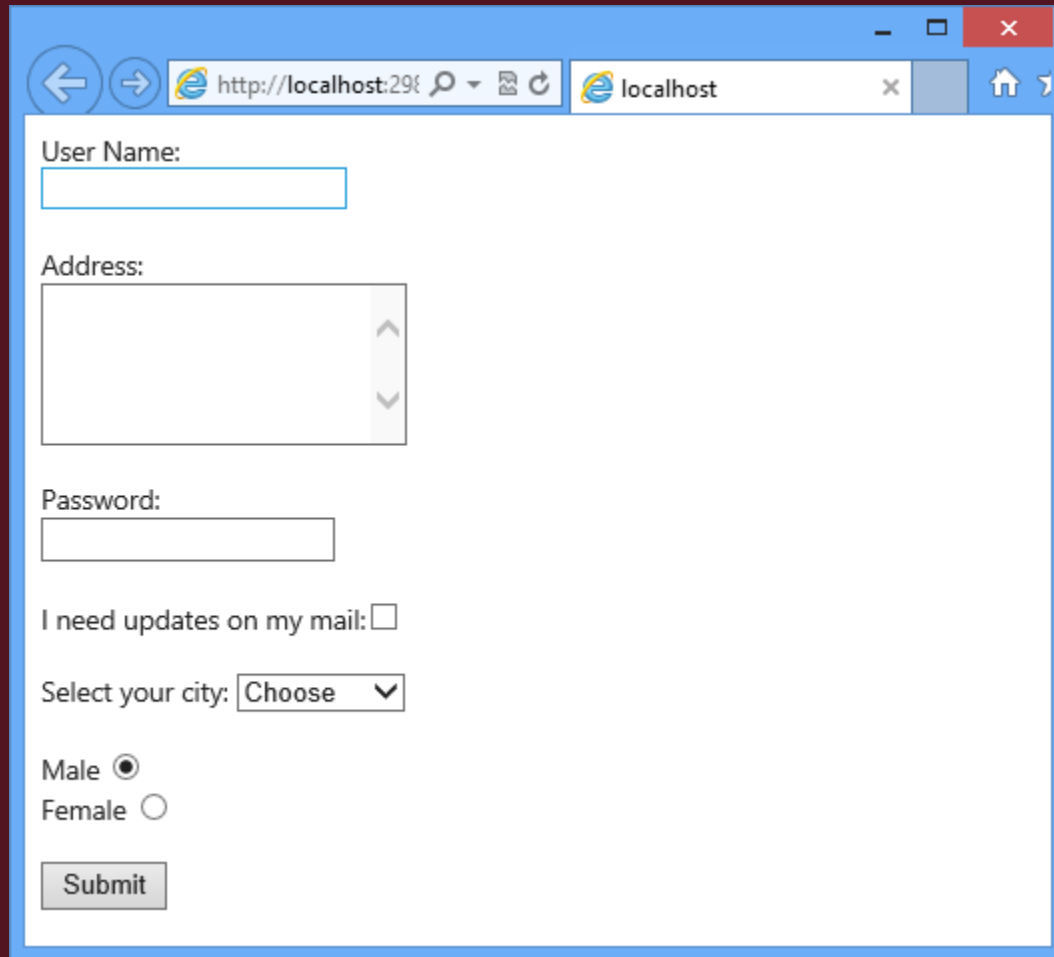
Code Snippet:

```
<!DOCTYPE html>
<html> <body>
    @{Html.BeginForm("Browse", "Home");}
    @Html.Label("User Name:")</br>
    @Html.TextBox("textBox1")</br></br>
    @Html.Label("Address:")</br> @Html.TextArea("textareal")</br></br>
    @Html.Label("Password:")</br>@Html.Password("password")</br></br>
    @Html.Label("I need updates on my
mail:")@Html.CheckBox("checkbox1")</br> </br>
    @Html.Label("Select your city:") @Html.DropDownList("myList", new
SelectList(new [] {"New York", "Philadelphia", "California"}),
"Choose")</> </br></br>
    Male @Html.RadioButton("Gender", "Male", true)</br>
    Female @Html.RadioButton("Gender", "Female")</br> </br>
<input type="submit" value="Submit">
    @{Html.EndForm();}</body> </html>
```

Html.RadioButton() trong trường hợp này tạo ra 2 lựa chọn về giới tính để người dùng có thể chọn

Phương thức hỗ trợ HTML

- ◆ Kết quả:



A screenshot of a web browser window showing a registration form. The browser's address bar displays 'http://localhost:298' and the page title is 'localhost'. The form contains the following elements:

- User Name:** A single-line text input field.
- Address:** A multi-line text area with vertical scrollbars.
- Password:** A single-line text input field.
- I need updates on my mail:** A checkbox.
- Select your city:** A dropdown menu with 'Choose' selected.
- Gender:** Two radio buttons labeled 'Male' (selected) and 'Female'.
- Submit:** A button at the bottom of the form.

Phương thức hỗ trợ HTML

- ♦ **Url.Action()** được dùng để tạo ra URL chỉ đến một phương thức hành động xác định của một controller
- ♦ Cú pháp của **Url.Action()** :

Syntax:

```
@Url.Action(<action_name>, <controller_name>)
```

Trong đó,

- **action_name**: tên của phương thức hành động.
- **controller_name**: tên của class controller
- ♦ Ví dụ về phương thức **Url.Action()**:

Code Snippet:

```
<!DOCTYPE html>
<html> <body>
    <a href='@Url.Action("Browse", "Home")'>Browse</a>
</body> </html>
```

Đoạn code tạo ra một hyperlink chỉ tới URL được tạo ra. Khi người dùng nhấn vào hyperlink này, hành hành động **Browse** của controller **Home** sẽ được gọi tới

- Trong ứng dụng ASP.NET MVC, View được sử dụng để hiển thị các nội dung tĩnh và động
- Công cụ view là 1 phần của MVC Framework chuyển đổi mã code của 1 view thành HTML markup mà một trình duyệt có thể hiểu được
- Sử dụng ViewData, ViewBag và TempData để chuyển dữ liệu từ controller sang view
- Trong ứng dụng ASP.NET MVC, partial view là một view phụ của view chính
- Razor là cú pháp, dựa trên ASP.NET Framework cho phép bạn tạo ra các view
- MVC Framework sử dụng công cụ view để chuyển đổi mã code của view thành HTML markup mà một trình duyệt có thể hiểu được
- MVC Framework cung cấp các phương thức hỗ trợ HTML có thể được sử dụng để tạo HTML markup và bạn có thể tái sử dụng nó khắp ứng dụng Web.