Các cấu trúc điều khiển của Razor, directive, ViewImports

Trong bài học này chúng ta sẽ học cách diễn đạt các cấu trúc điều khiển của C# (như cấu trúc điều kiện, lặp, v.v.) theo cú pháp Razor. Chúng ta cũng sẽ tiếp xúc với cấu trúc điều khiển đặc biệt của Razor gọi là Directive.

> Nội dung bài học này sẽ được áp dụng nguyên vẹn khi bạn chuyển sang học ASP.NET Core MVC và Blazor. Khi học đến ASP.NET Core MVC bạn sẽ không cần nhắc lại nội dung này nữa.

NỘI DUNG CỦA BÀI [Ẩn]

Cấu trúc điều khiển của Razor

Các **cấu trúc điều khiển của Razor** là những **code block** có nhiệm vụ điều chỉnh quá trình sinh mã HTML như phân nhánh, lặp, v.v., tương tự như vai trò của cấu trúc điều khiển trong C#. Các cấu trúc điều khiển giúp đơn giản hóa việc sinh mã HTML dựa theo các logic phức tạp.

> Nếu không nhớ khái niệm code block, hãy đọc lại bài học về code block trong Razor.

Trước hết cần lưu ý rằng *bên trong code block* (nơi ngôn ngữ mặc định là C#), bạn sử dụng các cấu trúc điều khiển của C# như bình thường.

Tuy nhiên, *ở ngoài code block* (nơi ngôn ngữ mặc định là HTML), bạn không thể trực tiếp sử dụng cách viết cấu trúc điều khiển của C# được. Khi đó bạn cần sử dụng lối viết markup của Razor dành cho cấu trúc điều khiển.

Các *cấu trúc điều khiển của Razor* là dạng mở rộng của code block. Vì vậy những đặc điểm của code block (như viết code tự do, chuyển đổi ngôn ngữ) đều áp dụng cho các cấu trúc

điều khiển. Cũng do vậy, bạn sẽ thường xuyên gặp lối viết trộn lẫn lộn C# và HTML.

Mặc dù bạn có thể sử dụng hàm mẫu trong code block để tạo ra các khối code HTML phức tạp theo logic, khả năng sử dụng các cấu trúc điều khiển sẽ giúp bạn đơn giản hóa hơn nữa việc sinh ra mã HTML theo các logic phức tạp.

Razor định nghĩa các markup dành cho:

(1) Các cấu trúc điều kiện, bao gồm cấu trúc @if-else if-else và @switch;

(2) Các cấu trúc lặp, bao gồm @for, @foreach, @while, và @do while;

(3) Cấu trúc @using;

(4) Cấu trúc bắt lỗi @try-catch-finally.

Các cấu trúc điều kiện

Cấu trúc @if-else if-else

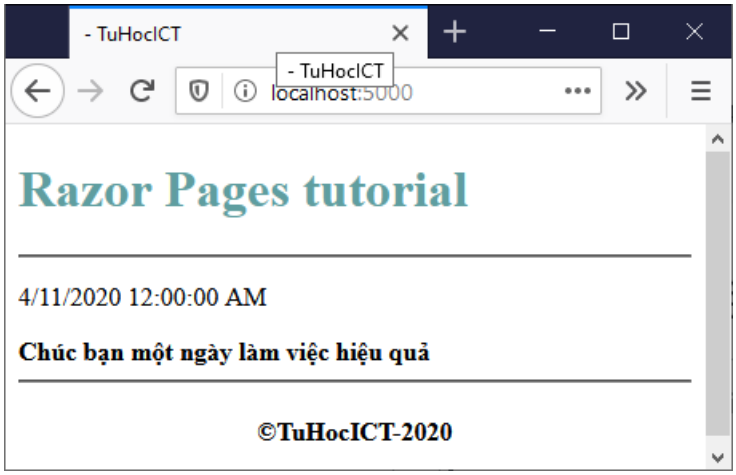
Hãy cùng xem ví dụ sau:

```
@page
<p>@DateTime.Today</p>
@if(DateTime.Today.DayOfWeek == DayOfWeek.Monday){
    var today = DateTime.Today;
    if(today.Day == 1) {
        <i>Hôm nay là ngày đầu tháng</i>
    }
    <strong>Chúc bạn một tuần làm việc mới hiệu quả</strong>
}
else if(DateTime.Now.DayOfWeek == DayOfWeek.Friday){
    <strong>Chúc bạn ngày cuối tuần vui vẻ</strong>
}
else{
    <strong>Chúc bạn một ngày làm việc hiệu quả</strong>
}
```

Trong ví dụ này, chúng ta hiển thị thông tin thời gian hiện tại trong thẻ <p>, sau đó viết một câu chúc trong thẻ . Nội dung của câu chúc phụ thuộc vào ngày trong tuần. Nếu là thứ 2 hoặc thứ 6 sẽ có lời chúc riêng. Những ngày còn lại dùng chung một câu chúc.

Nếu trúng vào ngày mồng một sẽ có thêm câu nói “Hôm nay là ngày đầu tháng”.

Ví dụ với ngày 11/4/2020 (thứ 7) thì kết quả như sau:



Chúng ta có những nhận xét như sau:

(1) Cả cấu trúc @if-else if-else là một code block, và bạn chỉ cần viết @ ở trước if

(2) Các điều kiện logic được viết theo cách bình thường của C#

(3) Bên trong mỗi nhánh của cấu trúc này bạn có thể *viết code tự do* như trong một *code block* thông thường.

Nghĩa là nếu bạn viết code C#, Razor sẽ dịch code C#. Nếu bạn chuyển sang HTML (bằng cách viết cặp thẻ HTML) thì Razor tự động chuyển sang chế độ render HTML.

> Nhắc lại bài học trước:
> (1) **trong** code block, ngôn ngữ mặc định là **C#**; **ngoài** code block, ngôn ngữ mặc định là **HTML**
> (1) trong một code block, nếu bạn viết cặp thẻ HTML, Razor tự động chuyển sang chế độ xuất (render) HTML
> (2) nếu không muốn dùng thẻ HTML để tự động chuyển đổi từ C# -> HTML, bạn có thể tự viết nội dung cần render trong cặp "thẻ" <text></text>

Như bạn đã thấy trong ví dụ trên, bên trong nhánh đầu tiên chúng ta viết code:

```
@if(DateTime.Today.DayOfWeek == DayOfWeek.Monday){
    var today = DateTime.Today;
    if(today.Day == 1) {
        <i>Hôm nay là ngày đầu tháng</i>
    }
    <strong>Chúc bạn một tuần làm việc mới hiệu quả</strong>
}
```

Ở đây bạn lại có thể trộn lẫn C# (mặc định) với HTML (tự động chuyển ngôn ngữ). Đây là cơ chế rất mạnh mẽ và tiện lợi của Razor. Hai ngôn ngữ hòa trộn tự nhiên với nhau trong code block.

Cấu trúc @switch

Chúng ta thực hiện lại ví dụ trên nhưng sử dụng cấu trúc rẽ nhiều nhánh @switch:

```
@page
```

```
@switch (DateTime.Today.DayOfWeek) {
    case DayOfWeek.Monday:
        var today = DateTime.Today;
        if (today.Day == 1) {
            <i>Hôm nay là ngày đầu tháng</i>
        }
        <strong>Chúc bạn một tuần làm việc mới hiệu quả</strong>
        break;
    case DayOfWeek.Friday:
        <strong>Chúc bạn ngày cuối tuần vui vẻ</strong>
        break;
    default:
        <strong>Chúc bạn một ngày làm việc hiệu quả</strong>
        break;
}
```

Qua ví dụ trên bạn có thể thấy, chỉ cần đặt @ vào đầu cấu trúc switch của C#. Mỗi một case của @switch đều là một code block nên bạn có thể viết code tự do.

Các cấu trúc lặp

Cấu trúc @for

Hãy xem ví dụ sau đây:

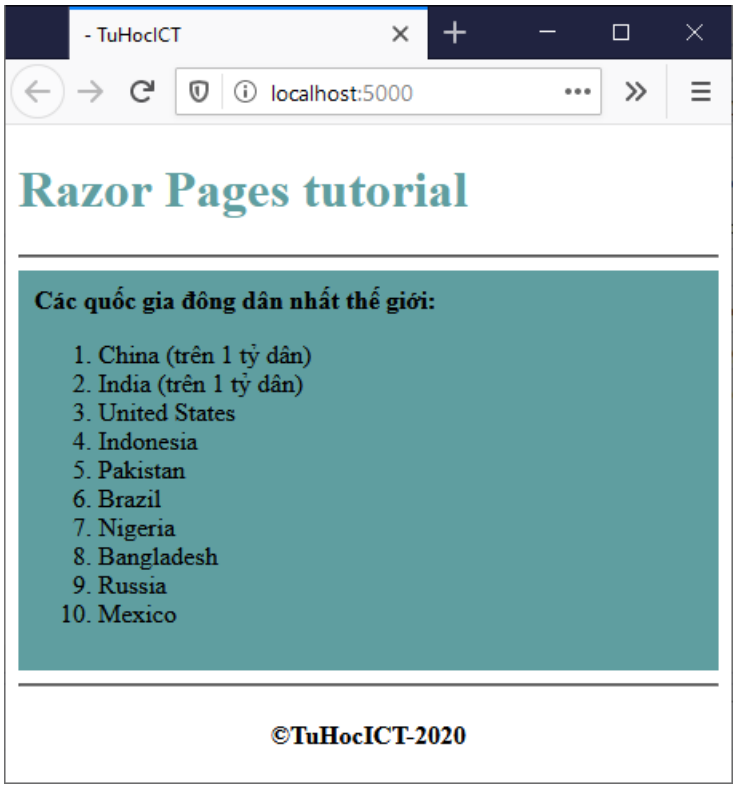
```
@page

@{
    var countries = new[] {
        "China",
        "India",
        "United States",
        "Indonesia",
        "Pakistan",
        "Brazil",
        "Nigeria",
        "Bangladesh",
        "Russia",
        "Mexico" };
}

<div style="background-color:cadetblue;padding:10px;">
    <strong>Các quốc gia đông dân nhất thế giới:</strong>
    <ol>
        @for (var i = 0; i < countries.Length; i++) {
            var country = countries[i];
            if (country == "China" || country == "India") {
                <li>@country (trên 1 tỷ dân)</li>
            }
            else {
                <li>@country</li>
            }
        }
    </ol>
</div>
```

Trong code block đầu tiên chúng ta khởi tạo một mảng string chứa danh sách các quốc gia đông dân nhất thế giới.

Chúng ta muốn xuất danh sách trên trong cặp thẻ với một số style css inline đơn giản. Để tạo danh sách trong HTML bạn dùng thẻ <ul> hoặc <ol>, bên trong là danh sách các thẻ <li>. Để lặp lại danh sách thẻ <li>, bạn sử dụng cấu trúc điều khiển lặp.

Razor cho phép chúng ta xây dựng code block dựa trên vòng lặp for bằng cách thêm ký tự @ vào trước for. Bên trong @for lại là một code block, do đó bạn có thể code tự do giữa C# và HTML.

Kết quả thu được như sau:



Cấu trúc @foreach

Chúng ta làm lại ví dụ trên sử dụng vòng lặp @foreach như sau:

```
@page

@{
    var countries = new[] {
        "China",
        "India",
        "United States",
        "Indonesia",
        "Pakistan",
        "Brazil",
        "Nigeria",
        "Bangladesh",
        "Russia",
        "Mexico" };
}

<div style="background-color:cadetblue;padding:10px;">
    <strong>Các quốc gia đông dân nhất thế giới:</strong>
    <ol>
        @foreach(var country in countries) {
            if (country == "China" || country == "India") {
                <li>@country (trên 1 tỷ dân)</li>
            }
            else {
                <li>@country</li>
            }
        }
    </ol>
</div>
```

Cấu trúc @while

Chúng ta làm lại ví dụ trên với cấu trúc @while

```
@page

@{
    var countries = new[] {
        "China",
        "India",
        "United States",
        "Indonesia",
        "Pakistan",
        "Brazil",
        "Nigeria",
        "Bangladesh",
        "Russia",
        "Mexico" };
}

<div style="background-color:cadetblue;padding:10px;">
    <strong>Các quốc gia đông dân nhất thế giới:</strong>
    <ol>
        @{ var i = 0; }
        @while (i < countries.Length) {
            var country = countries[i++];
            if (country == "China" || country == "India") {
                <li>@country (trên 1 tỷ dân)</li>
            }
            else {
                <li>@country</li>
            }
        }
    </ol>
</div>
```

Để ý rằng, khi dùng @while bạn cần khai báo một biến điều khiển i = 0 trong một code block trước đó. Trong vòng lặp bạn phải tăng giá trị i lên 1 đơn vị.

Cấu trúc @do-while

Cùng là ví dụ trên như sử dụng vòng @do-while:

```
@page

@{
    var countries = new[] {
        "China",
        "India",
        "United States",
        "Indonesia",
        "Pakistan",
        "Brazil",
        "Nigeria",
        "Bangladesh",
        "Russia",
        "Mexico" };
}

<div style="background-color:cadetblue;padding:10px;">
    <strong>Các quốc gia đông dân nhất thế giới:</strong>
    <ol>
        @{ var i = 0; }
        @do {
            var country = countries[i++];
            if (country == "China" || country == "India") {
                <li>@country (trên 1 tỷ dân)</li>
            }
            else {
                <li>@country</li>
            }
        } while (i < countries.Length);
    </ol>
</div>
```

@do-while chỉ đơn giản là đảo ngược của @while.

Bắt và xử lý ngoại lệ

Razor cũng hỗ trợ cách viết cấu trúc @try-catch-finally như sau:

```
@try {
    throw new InvalidOperationException("You did something invalid.");
}
catch (Exception ex) {
    <p>The exception message: @ex.Message</p>
}
finally {
    <p>The finally statement.</p>
}
```

Như vậy, cấu trúc này chỉ đơn giản là thêm @ vào đầu cáu trúc try-catch-finally của C#. Mỗi nhánh của cấu trúc này đều là một code block nên bạn có thể code thoải mái.

Khái niệm directive trong Razor

Để kết thúc bài học này chúng ta nhắc tới một khái niệm bạn vốn đã gặp khá nhiều: directive.

Trong tất cả các page từ trước đến giờ bạn đều gặp `@page` ở đầu trang. `@page` được gọi là directive trong Razor.

Directive là những từ đặc biệt được Razor lựa chọn để điều khiển những tính năng nhất định. Bạn cũng có thể hình dung directive như những từ khóa hoặc điều khiển riêng của Razor.

Directive thường được đặt ở đầu page và chỉ có tác dụng trên page đó.

Bạn sẽ thường xuyên gặp các directive sau đây:

@page – bắt buộc phải nằm ở dòng đầu tiên của một content page. Không có bất kỳ ký tự nào được phép đứng trước @page, nếu không Razor sẽ báo lỗi. @page báo cho Razor rằng đây là một content page và cần xử lý. Directive @page giúp phân biệt một trang Razor thông thường với view component hay partial page.

@model – chỉ định model class cho page. Sau @model có thể là tên đầy đủ của class, hoặc là tên ngắn gọn nếu như có thêm directive @using trước đó.

@using – tương tự như using của C# dùng để chỉ định namespace của các class sẽ được sử dụng trong page. Ví dụ muốn sử dụng class File (trong System.IO), bạn cần gọi @using System.IO.

@namespace – tương tự như namespace của C#, dùng để chỉ định không gian tên chứa page class nếu không muốn sử dụng không gian tên mặc định theo quy ước của Razor Pages.

@functions – chỉ định functions block.

Directive và ViewImports

Thông thường directive được đặt ở đầu của mỗi trang và chỉ có tác dụng trên trang đó. Tuy nhiên trong một số trường hợp bạn cần directive tự động có tác dụng trên tất cả các trang của site.

Lấy ví dụ, bạn cần chỉ định một namespace (sử dụng directive `@using`) mà tất cả các page đều cần truy cập, hoặc bạn cần thiết lập namespace của tất cả các page class (sử dụng directive `@namespace`).

Razor cung cấp một cơ chế để thiết lập directive có tác dụng toàn cục như vậy, gọi là import.

Hãy cùng thực hiện một ví dụ nhỏ sau:

Bước 1. Tạo một page mới trong thư mục Pages và đặt tên là **_ViewImports.cshtml**.

Bước 2. Nhập nội dung như sau cho _ViewImports.cshtml:

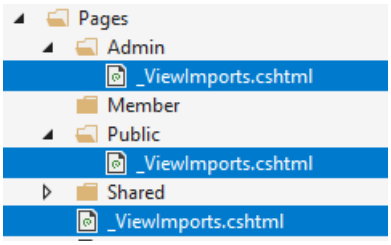
```
1.  @using WebApplication
2.  @namespace WebApplication.Pages
3.  @addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

Từ giờ trở đi, tất cả các page trong thư mục Pages đều có namespace mặc định là `WebApplication.Pages`, đồng thời có thể truy xuất tất cả các class trong namespace `WebApplication` qua tên ngắn gọn.

Lưu ý:

(1) Tương tự như _ViewStart.cshtml, _ViewImports.cshtml cũng có tác dụng trong thư mục chứa nó và tất cả các thư mục con của nó, với điều kiện trong thư mục con không có _ViewStarts của riêng mình.

Ví dụ, trong cấu trúc dưới đây, _ViewImports trong Pages sẽ có tác dụng tới tất cả các page trong thư mục Pages và thư mục con Member nhưng lại *không* có tác dụng trong thư mục Admin và Public.



(2) _ViewImports trong thư con sẽ đè lên _ViewImports của thư mục cha.

Cùng trong sơ đồ trên, các page trong thư mục Admin (và các thư mục con của Admin) sẽ chịu tác dụng của _ViewImports của Admin chứ không chịu tác dụng của _ViewImports trong Pages.

Kết luận

Qua bài học này bạn đã nắm được cách viết các cấu trúc điều khiển trong Razor.

Trong Razor, các cấu trúc điều khiển đều được xem là những code block thu được bằng cách viết thêm ký tự @ vào đầu cấu trúc điều khiển tương ứng của C#.

Vì là code block, trong mỗi cấu trúc điều khiển bạn có thể tự do viết code và chuyển đổi qua lại giữa ngôn ngữ C# và HTML qua một trong hai cơ chế đã biết (tự động/thủ công).

+ Nếu bạn thấy site hữu ích, trước khi rời đi hãy **giúp đỡ** site bằng một hành động nhỏ để site có thể phát triển và phục vụ bạn tốt hơn.
+ Nếu bạn thấy bài viết hữu ích, hãy giúp **chia sẻ** tới mọi người.
+ Nếu có thắc mắc hoặc cần trao đổi thêm, mời bạn viết trong phần **thảo luận** cuối trang.
Cảm ơn bạn!