Tag Helper trong ASP.NET Core

Tag helper trong Asp.net Core là loại thẻ (tag) đặc biệt do người dùng tạo ra và được cơ chế Razor xử lý nhằm sinh ra HTML/CSS/JavaScript tương ứng.

Asp.net Core tạo sẵn rất nhiều tag helper khác nhau để sử dụng. Nếu không hài lòng với những gì có sẵn, bạn có thể tự tạo tag helper của riêng mình.

Về mặt hình thức, tag helper nhìn tương tự như những thẻ HTML thông thường nhưng do bạn tự tạo ra và quyết định đầu ra (HTML/CSS/JS) nào sẽ tương ứng với nó.

Kỹ thuật tạo và sử dụng tag helper sẽ được xem xét chi tiết trong bài học này.

> Các vấn đề trình bày trong bài học này có thể áp dụng chung cho cả Razor Pages và MVC. Để đơn giản, các ví dụ được thể hiện với Razor Pages. Bạn có thể áp dụng nguyên vẹn với MVC.

NỘI DUNG CỦA BÀI [Ẩn]

1. Tag helper là gì
2. Bổ sung chức năng cho tag HTML
3. Xây dựng tag mới
4. Đặc điểm của tag helper
5. Sử dụng tag helper trong trang Razor
6. Kiểm soát sinh HTML trong tag helper
7. Các tag helper "chuẩn" của ASP.NET Core
8. Kết luận

Tag helper là gì

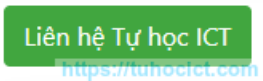
Đã bao giờ bạn cho rằng các thẻ HTML quá yếu không sao thể hiện được dữ liệu theo ý muốn của bạn, hoặc quá rắc rối khi thể hiện một vấn đề đơn giản. Bạn muốn tạo ra các thẻ riêng để thể hiện các khối giao diện theo mong muốn. Hoặc, bạn muốn "cải tạo" thẻ HTML sẵn có theo ý mình.

ASP.NET Core cho phép bạn đạt hiệu quả tương tự như vậy trên trang Razor thông qua một tính năng đặc biệt: **tag helper**.

Nói chi tiết hơn một chút.

Giả sử bạn cần tạo ra một link có dạng nút bấm, và khi click vào sẽ mở chương trình email mặc định của hệ thống. Nếu sử dụng HTML, bạn sẽ phải dùng thẻ <a> và thiết lập thuộc tính class (sử dụng Bootstrap) phù hợp:

```
<a href="mailto:contact@tuhocict.com"
class="btn btn-success">Liên hệ Tự học
ICT</a>
```



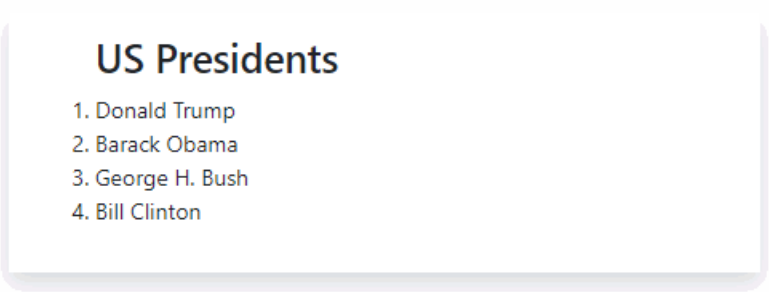
Nếu phải liên tục sử dụng thẻ a cho email, bạn muốn nó đơn giản hơn và được "chuyên môn hóa" khi thể hiện địa chỉ email. Giả sử chỉ đơn giản như thế này:

```
<a email="contact@tuhocict.com">Liên hệ Tự học ICT</a>
```

Thẻ <a> "chuyên môn hóa" như vậy rất tiện lợi khi tái sử dụng ở nhiều nơi.

Thêm một tình huống khác.

Giả sử bạn phải đưa danh sách sau vào nhiều trang khác nhau trên site:



Danh sách này được tạo từ cụm Razor đại khái như sau:

```
<div class="shadow p-3 mw-50">
    <p class="h3">US Presidents</p>
    <ul>
        @foreach (var p in Model) {
            <li>@p.firstName <b>@p.lastName</b></li>
        }
    </ul>
</div>
```

Liệu bạn có thích sử dụng phương án như thế này không?

```
<list ordered="true" data="data" title="US Presidents"></x:list>
```

Đây là hai trong số các tình huống thường gặp dẫn tới nhu cầu xử lý riêng biệt cho một thẻ HTML sẵn có, hoặc sinh ra một loại thẻ hoàn toàn mới.

Bạn có thể thực hiện được mong muốn đó trên trang Razor thông qua **tag helper**.

Tiếp theo đây chúng ta sẽ xem cách thực hiện cụ thể.

Bổ sung chức năng cho tag HTML

Chúng ta cùng xây dựng tag helper đầu tiên cho phép đơn giản hóa việc thể hiện nút bấm chứa địa chỉ email qua thẻ <a>.

Bước 1. Tạo thư mục Tags trực thuộc dự án. Bạn có thể đặt tên khác cho thư mục này. Nó không ảnh hưởng đến tag helper sẽ xây dựng.

Bước 2. Tạo class EmailTag trong thư mục Tags và viết code như sau:

```
using Microsoft.AspNetCore.Razor.TagHelpers;

namespace CustomTagHelpers.Tags {
    [HtmlTargetElement("a")]
    public class EmailTag : TagHelper {
        public string Email { get; set; }
        public override void Process(TagHelperContext context, TagHelperOutput output) {
            output.TagName = "a";
            output.Attributes.SetAttribute("href", $"mailto:{Email}");
            if (output.Attributes.ContainsName("class")) {
                var classes = output.Attributes["class"].Value.ToString();
                output.Attributes.SetAttribute("class", classes += " btn btn-success");
            }
            else
                output.Attributes.SetAttribute("class", "btn btn-success");
        }
    }
}
```

Hãy để ý rằng class này kế thừa từ TagHelper. Nếu bạn cần xây dựng tag helper, class tương ứng đều phải kế thừa từ TagHelper.

Thứ hai, class này được đánh dấu với attribute [HtmlTargetElement(“a”)]. Attribute này báo hiệu rằng tag helper đang xây dựng sẽ tác động lên các thẻ <a> trên trang Razor.

Property Email về sau sẽ tương ứng với attribute email khi sử dụng tag helper này.

Lớp tag helper phải ghi đè phương thức Process của lớp cha TagHelper. Trong đó, những gì cần xuất ra trong chuỗi HTML đích cần được đưa vào tham số output.

Bước 3. Dịch solution. Chỉ khi dịch solution thành công bạn mới có thể sử dụng tag helper trên trang Razor.

Bước 4. Bổ sung thêm directive sau vào đầu trang nơi bạn muốn sử dụng tag helper.

```
@addTagHelper *, CustomTagHelpers
```

Chú ý rằng CustomTagHelpers là tên của project (assembly). Directive này chỉ yêu cầu bạn chỉ định tên assembly nơi chứa tag helper. Ký hiệu * phía trước báo hiệu rằng trong assembly này có bao nhiêu tag helper thì lấy hết ra để sử dụng.

Nếu muốn sử dụng tag helper này trên tất cả các trang, bạn có thể đặt directive vào trang ViewImports.

Bước 5. Sử dụng tag helper trên một trang bất kỳ như sau:

```
<a email="contact@tuhocict.com" class="m-2">Liên hệ Tự học ICT</a>
```

Khi chạy chương trình, nếu mở cửa số Developer (F12) và inspect nút bấm bạn sẽ thấy đoạn HTML như sau được sinh ra:

```
<a class="m-2 btn btn-success" href="mailto:contact@tuhocict.com">Liên hệ Tự học ICT</a>
```

Như vậy, bạn có thể thấy rằng, thẻ <a> thông thường khi qua bàn tay của Razor sẽ được chuyển sang class EmailTag. Trong đó bạn tự mình can thiệp vào HTML kết quả bằng cách thêm attribute hreft theo địa chỉ email, thêm btn và btn-success vào attribute class.

Xây dựng tag mới

Bước 1. Tạo class ListTagHelper trong thư mục Tags và viết code như sau:

```
using System.Collections.Generic;

using Microsoft.AspNetCore.Razor.TagHelpers;

namespace CustomTagHelpers.Tags {

    public class ListTagHelper : TagHelper {
        public List<string> Data { get; set; }
        public string Title { get; set; }
        public bool Ordered { get; set; }
        public override void Process(TagHelperContext context, TagHelperOutput output) {
            output.TagName = Ordered ? "ol" : "ul";
            output.PreContent.SetHtmlContent($"<p class='h3'>{Title}</p>");
            foreach (var item in Data) {
                output.Content.AppendHtml($"<li>{item}</li>");
            }
        }
    }
}
```

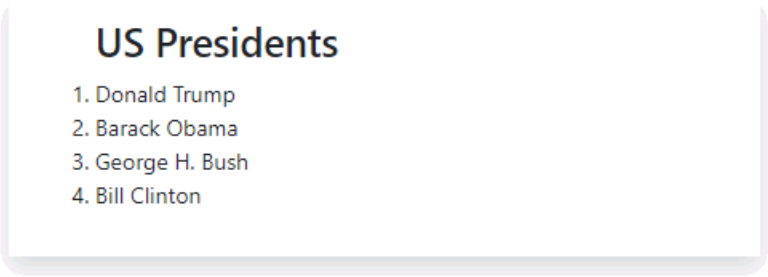
Để ý rằng tên gọi của class là ListTagHelper, cũng không có attribute nào hết. Tên class này đã tuân thủ theo quy ước của Asp.net Core nên không cần attribute hỗ trợ.

Bước 2. Ở một trang Razor bất kỳ bổ sung thêm code sau:

```
@{
    var data = new List<string> { "Donald Trump", "Barack Obama", "George H. Bush", "Bil
}

<div class="shadow p-3">
    <list ordered="true" data="data" title="US Presidents"></list>
</div>
```

Khi chạy chương trình bạn thu được kết quả như sau:



Mã HTML sinh ra như sau:

```
<div class="shadow p-3">
    <ol><p class="h3">US Presidents</p><li>Donald Trump</li><li>Barack Obama</li><li>Geo
</div>
```

Như vậy, thẻ <list> trên trang Razor được chuyển đổi thành khối HTML.

Đặc điểm của tag helper

Khi xây dựng tag helper có một số vấn đề sau cần lưu ý.

Thứ nhất, **thư mục** đặt class tag helper không quan trọng. Bạn có thể để class cho tag helper ở bất kỳ thư mục nào. Thông thường để dễ phân biệt người ta hay đặt trong thư muc Tags hoặc TagHelpers.

Thứ hai, **tên class** theo quy ước cần có hậu tố TagHelper. Như trong ví dụ trên, ListTagHelper là một class tuân thủ quy ước.

Asp.net Core nhìn nhận các class có tên kết thúc bằng TagHelper, và kế thừa từ class cha TagHelper, là các class đặc biệt. Với class ListTagHelper trên, nó sẽ tương ứng với thẻ <list> </list>.

Thứ ba, nếu không muốn tuân thủ quy ước về cách đặt tên class, bạn cần sử dụng **attribute [HtmlTargetElement]** để chỉ định thẻ tương ứng.

Ví dụ, nếu muốn class ListTagHelper trên được sử dụng cho thẻ <xtra-list></xtra-list>, bạn cần đặt attribute như sau:

```
[HtmlTargetElement("xtra-list")]
public class ListTagHelper : TagHelper { ... }
```

Nếu giá trị cung cấp trong [HtmlTargetElement] là một thẻ sẵn có, thẻ đó sẽ chịu tác dụng của class.

Đây là tình huống khi chúng ta xây dựng class EmailTag. Attribute [HtmlTargetElement(“a”)] khiến cho các thẻ <a> chịu tác dụng của class EmailTag. Do vậy chúng ta có thể tạo ra loại thẻ <a> chuyên dụng.

Thứ tư, để nhận giá trị qua attribute của thẻ, bạn cần tạo ra các **public property** tương ứng trong class.

Như trong class ListTagHelper có các property `public List Data { get; set; }` và `public string Title { get; set; }` thì thẻ chịu tác động của nó sẽ có attribute `data` và `title` để tiếp nhận dữ liệu.

Đây là cơ chế tự động binding attribute-property của tag helper.

Thứ năm, các **attribute của thẻ** có thể nhận cả biểu thức hoặc giá trị có kiểu phù hợp.

Như trong trường hợp thẻ

```
<list ordered="true" data="data" title="US Presidents"></list>
```

thì data=”data” là attribute có giá trị là cả một biến.

Sử dụng tag helper trong trang Razor

Nếu tag helper nhắm tới điều chỉnh một thẻ HTML cụ thể, bạn sử dụng nó như sử dụng thẻ HTML thông thường. Sự khác biệt là giờ đây bạn có thể cung cấp thêm attribute riêng để cơ chế tag helper xử lý.

Trên thực tế, khi bạn dùng thẻ , bạn đang yêu cầu Razor can thiệp vào việc tạo ra HTML mỗi khi nó nhìn thấy thẻ <a>.

Hoặc nói theo cách khác, thẻ không thực sự còn là thẻ <a> bình thường nữa.

Để sử dụng song song cả thẻ "chuẩn" của HTML và thẻ tự tạo tag helper, Asp.net Core cho phép chỉ định tiền tố (prefix) riêng cho các tag helper bằng directive @tagHelperPrefix như sau:

```
@tagHelperPrefix x:
```

Khi này bạn chỉ định thẻ tag helper theo cách sau:

```
<x:a email="contact@tuhocict.com" class="m-2">Liên hệ Tự học ICT</x:a>
<x:list ordered="true" data="data" title="US Presidents"></x:list>
```

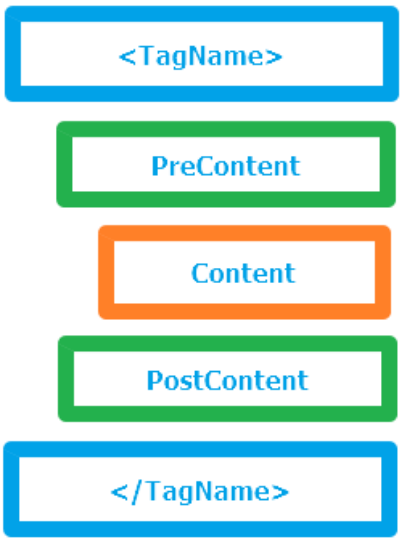
Khi đó, thẻ <a> thông thường sẽ không bị ảnh hưởng. Mã Razor của bạn nhìn cũng rành mạch hơn.

Bạn có thể chọn bất kỳ cụm ký tự nào làm tiền tố. Tuy nhiên x: là loại tiền tố thường được lựa chọn.

Kiểm soát sinh HTML trong tag helper

Qua các ví dụ trên bạn có lẽ đã hình dung ra, tag helper thực chất là loại kỹ thuật ánh xạ một tag trên trang Razor sang một cụm HTML tiêu chuẩn (chứa trong phản hồi HTTP và sẽ được trình duyệt hiển thị). Nếu không có cơ chế xử lý của Razor, những gì viết trên trang có thể sẽ được chuyển nguyên vẹn thành HTML đầu ra.

Việc sinh HTML trong tag helper tuân theo mô hình sau:



Theo đó, khối HTML sinh ra luôn nằm trong một cặp thẻ **<TagName></TagName>**. Ví dụ trong EmailHelper ở trên, TagName chính là a, nghĩa là khối HTML sinh ra sẽ nằm trong cặp <a>. TagName là bắt buộc để xác định HTML tương ứng với tag helper. Trong ListTagHelper, TagName là ol hoặc ul, phụ thuộc vào tham số Ordered.

Trong lớp tag helper, TagName được chỉ định qua thuộc tính tương ứng của output: `output.TagName`.

Content là những gì sẽ nằm trong cặp tag đích <TagName>… content …</TagName>. Nó có thể là văn bản hoặc là các khối HTML khác. Content là không bắt buộc. Bạn có thể tác động vào phần Content thông qua property cùng tên của biến output: output.Content.

Khi tác động vào Content, bạn có thể sử dụng một loạt phương thức như SetContent, SetHtmlContent, Append, AppendHtml, v.v., tương tự như khi làm việc với chuỗi ký tự. Qua tên gọi bạn cũng có thể tự hình dung ra vai trò của các phương thức này.

Ví dụ trong EmailHelper, bạn không tác động gì vào Content. Khi này, cơ chế tag helper tự lấy nội dung của tag để đặt vào Content.

Còn trong ListTagHelper, content chính là tập hợp thẻ :

```
foreach (var item in Data) {
    output.Content.AppendHtml($"<li>{item}</li>");
}
```

Ở đây chúng ta lưu ý, những phương thức mà trong tên có chứa cụm Html cho phép bạn nhập thẳng chuỗi ký tự chứa thẻ Html vào. Còn ở phương thức tương ứng không có Html, cụm ký tự bạn nhập vào sẽ được hiểu là văn bản chứ không phải Html, và do vậy sẽ bị mã hóa để hiển thị thành văn bản ở chuỗi HTML đích.

PreContent và PostContent là những gì được chèn vào trước và sau phần Content. Nội dung của PreContent và PostContent được chứa trong các property cùng tên của output.

Như trong ListTagHelper chúng ta chèn PreContent là một thẻ p chứa dòng tiêu đề:

```
output.PreContent.SetHtmlContent($"<p class='h3'>{Title}</p>");
```

Ở cuối phương thức Process, bạn nên gọi output.Attributes.Clear() để xóa hết các attribute mà người dùng nhập vào tag. Nếu không xóa, các attribute này sẽ xuất hiện trong HTML đích, mà chúng hầu hết đều không phải là các attribute tiêu chuẩn của HTML.

Các tag helper "chuẩn" của ASP.NET Core

Asp.net Core đã xây dựng sẵn rất nhiều tag helper mà bạn có thể sử dụng. Hầu như tất cả các thẻ HTML đều có tag helper tương ứng. Tuy nhiên, các tag helper mặc định không được sử dụng cho dự án. Bạn cần "bật" chúng lên trước khi sử dụng bằng directive sau:

```
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

Nếu sử dụng Web Application template, directive này được đặt sẵn trong ViewImports nên bạn không cần tự mình thêm vào nữa.

Nếu không muốn bật tag helper trên toàn bộ site, bạn có thể lựa chọn đặt directive trên ở từng page riêng biệt nơi cần dùng.

Directive trên sẽ bật tất cả tag helper trong asembly Microsoft.AspNetCore.Mvc.TagHelpers.

Bạn cũng có thể lựa chọn chỉ bật những tag helper cần thiết, thay vì bật tất cả. Ví dụ:

```
@addTagHelper Microsoft.AspNetCore.Mvc.TagHelpers.AnchorTagHelper,
Microsoft.AspNetCore.Mvc.TagHelpers
```

Tất cả các tag helper xây dựng sẵn của Asp.net Core đều chứa các attribute mới bắt đầu bằng `asp-`. Bạn đã từng gặp cách viết này khi học về named handler:

```
<a asp-page-handler="register">Register</a>
```

asp-page-handler thực chất tương ứng với property AspPageHandler trong class tag helper tương ứng (AnchorTagHelper) của Asp.net Core.

Trọn vẹn danh sách tag helper của Asp.net Core bạn có thể tìm thấy ở địa chỉ sau: https://docs.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/built-in/?view=aspnetcore-3.1

Chúng ta sẽ có một bài học riêng về cách sử dụng các tag helper xây dựng sẵn của Asp.net Core.

Kết luận

Bài học này đã giới thiệu với bạn những kiến thức cơ bản về xây dựng và làm việc với tag helper trong Asp.net Core. Đây là một công cụ rất mạnh và được sử dụng rất phổ biến trên trang Razor.

> + 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!