

# Model class trong Razor Pages

[Hướng dẫn tự học lập trình ASP.NET Core toàn tập](#) > [Model class trong Razor Pages](#)

Model class là loại class đặc biệt có nhiệm vụ cung cấp dữ liệu và khả năng xử lý cho trang trong Razor Pages. Model class cùng với cú pháp Razor tạo ra sức mạnh cho Razor Pages. Sự kết hợp này giúp biến mỗi trang Razor trở thành một ứng dụng thực sự.

Trong hai bài học trước bạn đã học cách thức viết code trên trang Razor. Lối viết code như vậy được gọi là mô hình *Single Page*. Tuy nhiên, nếu bạn tiếp tục sử dụng cách thức viết code như vậy, chương trình bạn viết ra sẽ có rất nhiều hạn chế.

Razor Pages hỗ trợ hai cách viết code để xử lý logic của một trang: *functions block*, và *model class*. Trong đó, model class là cách viết “chính thống” nhất và được khuyến khích sử dụng trong mọi trường hợp.

Việc nắm bắt và vận dụng thành thạo model class là yêu cầu bắt buộc khi học Razor Pages. Bài học sẽ giúp bạn hiểu đầy đủ về cơ chế hoạt động của model class.

## NỘI DUNG CỦA BÀI [ Ấn ]

1. Cách thức hoạt động của trang Razor, mô hình Single Page
2. Functions block
  - 2.1. Thực hành 1
  - 2.2. Phân tích
3. Model class
  - 3.1. Thực hành 2
  - 3.2. Khái niệm model class
4. Sử dụng model class
  - 4.1. Ưu điểm của model class
  - 4.2. Đặc điểm của model class
  - 4.3. Một số lưu ý khác
5. ViewData và model class
  - 5.1. ViewData object
  - 5.2. ViewData attribute
6. Kết luận
  - 6.1. Link tải solution S04\_WebApplication

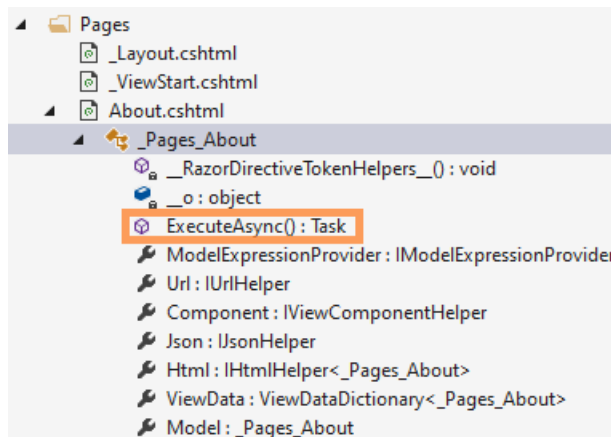
## Cách thức hoạt động của trang Razor, mô hình Single Page

Trong hai bài học về cú pháp Razor chúng ta đã nói rằng không thể khai báo kiểu dữ liệu mới (class, struct, enum, v.v.) bên trong code block.

Điều này liên quan đến cơ chế hoạt động của trang Razor.

Khi bạn tạo một page, giả sử đặt tên là About.cshtml bên trong thư mục Pages, Razor sẽ tự động tạo một class có tên là **\_Pages\_About**.

Nếu chưa tin, bạn có thể mở file About.cshtml, chờ vài giây, và để ý thấy rằng bên cạnh tên file About.cshtml trong Solution Explorer giờ xuất hiện nút tam giác nhỏ. Khi mở rộng nó ra bạn sẽ thấy cấu trúc như sau:



Nếu bạn đã từng làm việc với Visual Studio thì hẳn sẽ hiểu, đây là cách thức Visual Studio hiển thị cấu trúc class. Class này kế thừa từ class `Page` (tên đầy đủ là `Microsoft.AspNetCore.Mvc.RazorPages.Page`). Class này được gọi là **page class** của trang Index. Cũng lưu ý phân biệt **page class** với **model class** sẽ gặp ở phần tiếp theo.

Nếu chưa tin nữa, ở bất kỳ vị trí nào trong code block bạn hãy gõ **this.** (this và dấu chấm) – phép toán truy xuất thành viên class, hoặc **base.** (base và dấu chấm) – phép toán truy xuất thành viên của class cha.

Hãy chú ý đến phương thức `ExecuteAsync()` đang được khoanh.

Giờ giả sử chúng ta viết vài dòng code trên page này (đừng code theo, chúng ta chỉ lấy ví dụ thôi):

```
1. @page
2. @{
3.     var quote = "Getting old ain't for wimps! - Anonymous";
4. }
5. <div>Quote of the Day: @quote</div>
```

Ở hậu trường, Razor sẽ chuyển nó thành phương thức như thế này.

```
1. public override async Task ExecuteAsync()
2. {
3.     var output = "Getting old ain't for wimps! - Anonymous";
4.
5.     WriteLiteral("/r/n<div>Quote of the Day: ");
6.     Write(output);
7.     WriteLiteral("</div>");
8. }
```

Vậy điều này có nghĩa là gì? Tất cả những gì bạn tự viết trên page, bao gồm các biểu thức và khối code Razor, thực tế đều nằm trong một phương thức `ExecuteAsync()`!

Điều này cũng giải thích cho mọi vấn đề bạn có lẽ đã gặp phải: (1) có thể khai báo hàm cục bộ (không phải là phương thức), vì C# hỗ trợ hàm cục bộ (hàm nằm trong phương thức); (2) không thể khai báo kiểu dữ liệu (C# không cho khai báo kiểu bên trong phương thức);

(3) code block lại có thể render HTML; (4) không thể khai báo property (vì property không thể nằm trong method); v.v..

Đây cũng là cách thức viết code đơn giản nhất của page trong Razor và thường được gọi là mô hình **single page**. Mô hình này đã được chúng ta sử dụng từ đầu đến giờ.

Tuy nhiên đây lại là mô hình không khuyến khích sử dụng khi phát triển ứng dụng Razor Pages. Nó có rất nhiều nhược điểm khi xây dựng các app lớn cũng như bị hạn chế về tính năng.

Razor cung cấp thêm hai mô hình hoạt động mạnh mẽ hơn: mô hình *functions block* và mô hình *page model class*.

## Functions block

### Thực hành 1

Mô hình này được gọi theo tên của một loại code block mới trong file Razor. Hãy cùng thực hiện ví dụ sau đây:

Link tải solution ở phần Kết luận cuối bài.

**Bước 1.** Tạo project **Asp.net Core** trống đặt tên là **WebApplication** và cấu hình để project này trở thành **Razor Pages**.

**Bước 2.** Cấu hình **layout và viewstart**.

**Bước 3.** Tạo page **Index.cshtml** trong **Pages** sử dụng **layout và viewstart**

Viết code cho **Index.cshtml** như sau:

```
1. @page
2.
3. @{ // code block
4.     void SayHello(string name) {
5.         <p class="block">Hello, @name! I'm in the code block</p>
6.     }
7. }
8.
9. @functions { // functions block
10.     void SayHello(string name) {
11.         <p class="function">Hello, @name! I'm in the functions block.</p>
12.     }
13. }
14.
15. <!-- ##### -->
16.
17. <style>
18.     .block { color: dodgerblue; }
19.     .function { color: yellowgreen; }
20.     .class { color: blueviolet; }
21. </style>
22.
23. <!-- ##### -->
24.
25. @{ SayHello("Covid"); }
26. @ { this.SayHello("Covid"); }
```

Trong code Index.cshtml bạn lưu ý:

(1) Trong **code block** ở đầu file bạn tạo một hàm mẫu SayHello:

```
1. @{ // code block
2.     void SayHello(string name) {
3.         <p class="block">Hello, @name! I'm in the code block</p>
4.     }
5. }
```

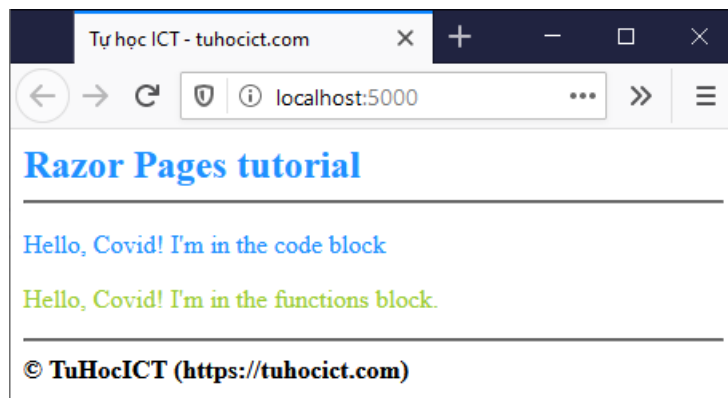
(2) bạn tạo một code block mới **@functions { ... }**, trong block này lại viết một hàm SayHello khác:

```
1. @functions { // functions block
2.     void SayHello(string name) {
3.         <p class="function">Hello, @name! I'm in the functions block.</p>
4.     }
5. }
```

(3) Bạn gọi hai "hàm" này trong page, lưu ý lệnh thứ hai có từ khóa **this**.

```
1. @{ SayHello("Covid"); }
2. @ { this.SayHello("Covid"); }
```

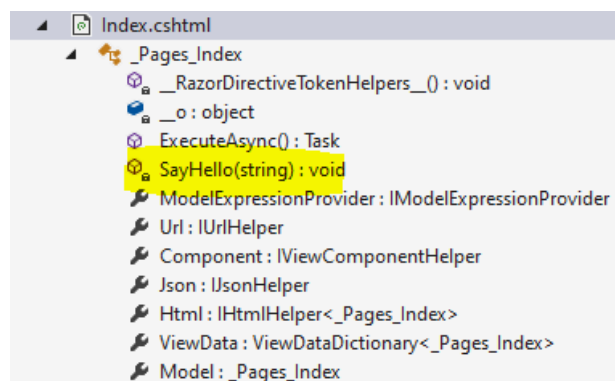
Kết quả thu được như sau:



## Phân tích

Như bạn đã thấy, mặc dù cả code và kết quả thu được là như nhau, hai "hàm" SayHello lại có bản chất hoàn toàn khác biệt.

Trước hết hãy nhìn vào class do Razor tạo ra cho page Index:



Bạn thấy xuất hiện phương thức SayHello(string):void. Vậy đó là SayHello trong code block, hay SayHello trong functions block?

Nếu bạn comment SayHello trong functions block, phương thức này sẽ biến mất khỏi class. Còn nếu bạn comment SayHello trong code block, class không thay đổi. Nếu bạn bỏ từ khóa this khỏi lệnh gọi SayHello thứ hai, bạn thu được cùng một kết quả từ hai lệnh.

Giờ nếu bạn nhớ lại mô hình Single Page ở trên thì sẽ hiểu ngay:

(1) SayHello bạn viết trong *code block* thực chất là một *hàm cục bộ* nằm trong ExecuteAsync()

(2) SayHello viết trong *functions block* sẽ trở thành *phương thức thành viên* của page class đứng sau trang Index. Cùng vì lý do này bạn có thể gọi SayHello thứ hai qua từ khóa **this**.

Như vậy, functions block cung cấp cho chúng ta khả năng xây dựng thêm các thành viên cho page class của trang tương ứng. Trong functions block bạn có thể xây dựng thêm bất kỳ thành viên nào của class, như property, method, biến, hằng, kiểu dữ liệu mới, v.v..

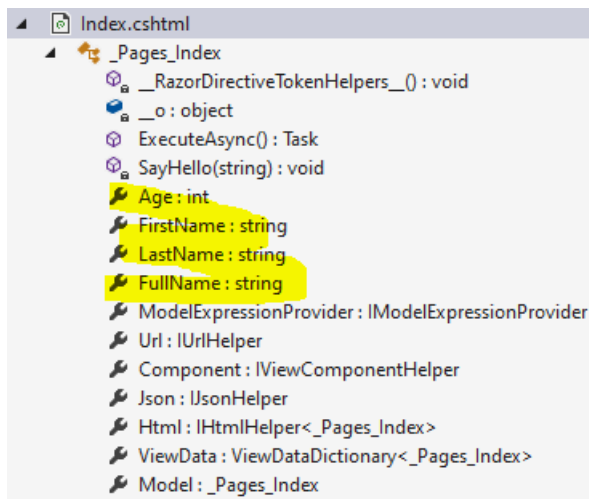
Do trong functions block bạn tạo thêm các thành viên mới của class, bạn cũng có thể tùy ý sử dụng chúng ở các vị trí khác nhau trên page.

Khả năng tạo thêm các thành viên của class cho phép một loạt tính năng mới quan trọng trên trang Razor, như [xử lý truy vấn \(handler\)](#), [Model binding](#).

Trong page class bạn để ý thêm một property quan trọng: **Model**. Property này hiện tại có kiểu chính là page class của trang. Đối với trang Index ở trên, Model sẽ có kiểu \_Pages\_Index. Đối với trang About, Model sẽ có kiểu \_Pages\_About. Property có vai trò rất quan trọng trong mô hình **model class** sẽ xem xét trong phần tiếp theo của bài học này.

Khi bạn đã hiểu bản chất của functions block, hãy tự thực hiện bài thực hành nhỏ sau: Bổ sung thêm một số property vào functions block rồi sử dụng chúng trong page.

```
1.  @functions{// functions block
2.      void SayHello(string name) {
3.          <p class="function">Hello, @name! I'm in the functions block.</p>
4.      }
5.
6.      public int Age { get; set; }
7.      public string FirstName { get; set; }
8.      public string LastName { get; set; }
9.      public string FullName => $"{FirstName} {LastName}";
10. }
11.
12. @{ FirstName = "Donald"; LastName = "Trump"; }
13. @{ this.SayHello(FullName); }
```



## Model class

Mặc dù functions block rất mạnh nhưng nó cũng có nhiều nhược điểm khi phát triển các ứng dụng lớn với nhiều page phức tạp:

- Dễ thấy nhất là khi lượng code tăng lên, kích thước của page sẽ phình ra rất khó quản lý.
- Thứ hai là mô hình này trộn lẫn logic-dữ liệu-giao diện vào cùng một khối – vốn là điều tối kỵ trong làm phần mềm.
- Thứ ba, do trộn lẫn lộn các thành phần, các page này rất khó test.

Vì vậy Razor đưa ra mô hình thứ ba – **model class**.

Để hiểu mô hình này, hãy cùng làm một bài thực hành nhỏ.

## Thực hành 2

Bạn tiếp tục sử dụng project từ phần thực hành 1-2.

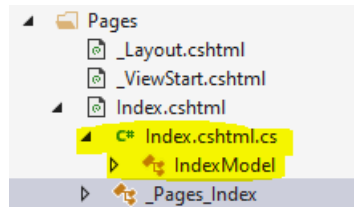
**Bước 1.** Tạo file `Index.cshtml.cs` trong thư mục Pages

**Bước 2.** Viết code cho file `Index.cshtml.cs` như sau:

```
1. using Microsoft.AspNetCore.Mvc.RazorPages;
2.
3. namespace WebApplication.Pages {
4.     public class IndexModel : PageModel {
5.         public int Age { get; set; }
6.         public string FirstName { get; set; }
7.         public string LastName { get; set; }
8.         public string FullName => $"{FirstName} {LastName}";
9.
10.        public string SayGoodbye() => $"Hello, {FullName}! I'm in the model class";
11.
12.        public void OnGet() {
13.            FirstName = "Donald";
14.            LastName = "Trump";
15.        }
16.    }
17. }
```

Bạn có thể để ý đây là một class C# tiêu chuẩn! Cũng lưu ý namespace của class này (WebApplication.Pages) vì chút nữa có thể bạn sẽ cần đến nó.

Sau bước này bạn để ý trong Solution Explorer sẽ thấy thế này



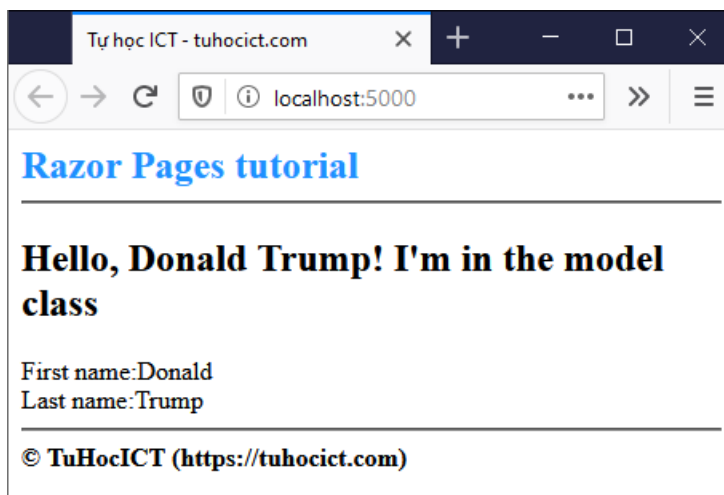
Đây là tính năng *file nesting* của Visual Studio – tự động hiển thị gộp các file có tên giống nhau. Thực chất chúng là những file riêng biệt.

**Bước 3.** Xóa hết code cũ (nếu có) của Index.cshtml và viết lại code như sau:

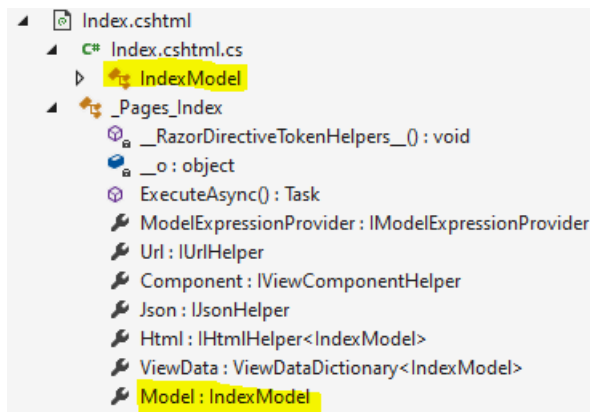
```
1. @page
2.
3. @model WebApplication.Pages.IndexModel
4.
5. <div>
6.     <h2>@Model.SayGoodbye() </h2>
7.     <div>
8.         <div><label>First name:</label>@Model.FirstName</div>
9.         <div><label>Last name:</label>@Model.LastName</div>
10.    </div>
11. </div>
```

Lưu ý dòng `@model WebApplication.Pages.IndexModel`. Ở đây sử dụng tên đầy đủ của class `IndexModel`.

Chạy thử chương trình bạn sẽ thu được kết quả như sau:



Nếu nhìn lại page class `_Pages_Index` bạn sẽ thấy property **Model** có sự thay đổi:



Giờ đây kiểu của Model không còn là page class (`_Pages_Index`) như trước kia nữa mà đã chuyển thành `IndexModel` – class mới do chúng ta xây dựng.

## Khái niệm model class

Trong phần thực hành trên chúng ta đã thực hiện viết code theo mô hình **page model class**.

Lớp `IndexModel` mà bạn đã viết trong file `Index.cshtml.cs` được gọi là **model class** của page `Index.cshtml`.

Cũng lưu ý rằng, `_Pages_Index` – class do Razor tự động tạo ra cho trang `Index` được gọi là **page class**.

Model class cho phép bạn tách rời toàn bộ data logic của page ra một class riêng. Trên page giờ chỉ còn lại UI logic. Code trở nên sáng sủa, dễ đọc, dễ quản lý hơn rất nhiều.

Nếu bạn đã từng làm việc với ASP.NET MVC thì đến đây hẳn bạn sẽ thấy rất quen thuộc. Page model class của Razor Pages có cùng nguyên tắc với MVC, chỉ khác biệt ở cách tổ chức file. Trên thực tế, Razor Pages được xây dựng bên trên ASP.NET Core MVC.

## Sử dụng model class

### Ưu điểm của model class

Mục đích chính của model class là tạo ra sự phân chia rõ ràng giữa giao diện và xử lý logic/dữ liệu của trang. Sự phân chia này đem lại một loạt ưu điểm:

(1) làm giảm sự phức tạp của giao diện và dễ bảo trì: giờ đây giao diện chỉ chịu trách nhiệm hiển thị thông tin. Các logic trên giao diện là logic phục vụ hiển thị thông tin. Những xử lý khác như dữ liệu được thực hiện ở model class. Giao diện trở nên đơn giản và dễ bảo trì.

(2) giúp thực hiện unit test: model class là một class C# bình thường được xây dựng dựa trên cơ chế Dependency Injection, rất tiện lợi cho test tự động.



(3) giúp phát triển song song trong team: do logic và UI được tách rời, team có thể dễ dàng phân chia công việc và đồng thời thực hiện.

(4) tạo ra các khối code nhỏ để dễ dàng tái sử dụng.

## Đặc điểm của model class

Khi sử dụng page model class cần lưu ý những điểm sau:

(1) Model class phải kế thừa từ lớp `PageModel` (`Microsoft.AspNetCore.Mvc.RazorPages.PagesModel`). Ngoài ra không có giới hạn gì khác.

(2) Để chỉ định một class trở thành model class cho một page, bạn phải sử dụng directive `@model <tên_class>`

```
1. @model WebApplication.Pages.IndexModel
```

(3) Một khi đã chỉ định model class, bạn có thể truy xuất object của nó thông qua property **Model** của *page class* ở bất kỳ vị trí nào trong page.

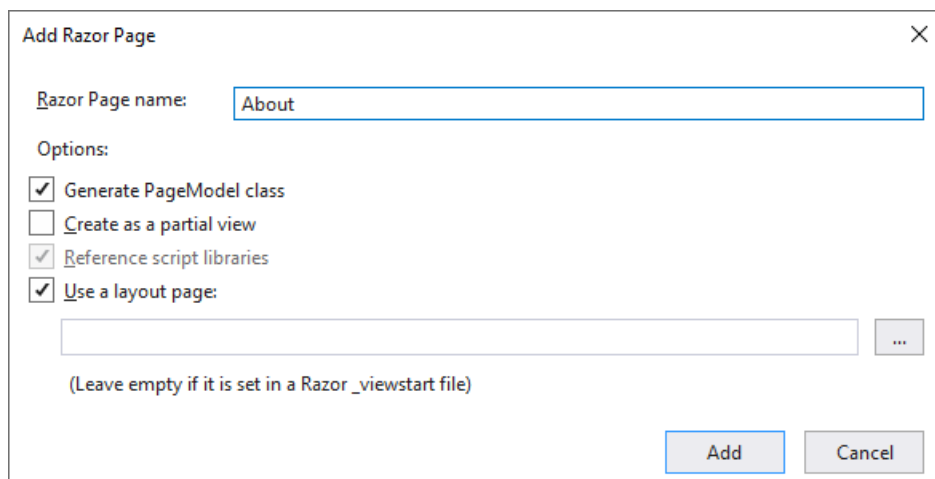
```
1. <div>
2.     <h2>@Model.SayGoodbye()</h2>
3.     <div>
4.         <div><label>First name:</label>@Model.FirstName</div>
5.         <div><label>Last name:</label>@Model.LastName</div>
6.     </div>
7. </div>
```

## Một số lưu ý khác

(1) Mặc dù không có quy định bắt buộc nhưng tên model class quy ước đặt theo tên page + Model. Ví dụ, model class của trang Index sẽ là IndexModel.

(2) Tương tự, model class thường đặt trong file cùng tên với page nhưng có đuôi cs. Ví dụ, page Index nằm trong file Index.cshtml thì model class sẽ để trong file Index.cshtml.cs. Cách đặt tên này giúp Visual Studio hiển thị ghép các file vào cùng một node (gọi là file nesting). File nesting giúp quản lý file dễ dàng hơn.

(3) Visual Studio hỗ trợ tạo page với model class trong hộp thoại Add Razor Page như sau:



Add Razor Page

Razor Page name:

Options:

- ☒ Generate PageModel class
- ☐ Create as a partial view
- ☒ Reference script libraries
- ☒ Use a layout page:

...

(Leave empty if it is set in a Razor \_viewstart file)

Add Cancel

Lưu ý tích chọn "Generate PageModel class". Bạn sẽ thu được cùng kết quả như khi làm thủ công trong phần thực hành 3.

Nói tóm lại, page model class là mô hình viết code tốt nhất trên Razor Pages và bạn nên sử dụng nó.

Nếu không có ghi chú gì đặc biệt thì page model class sẽ là mô hình được sử dụng chính trong các bài học tiếp theo.

## ViewData và model class

Trong những bài học đầu tiên về Razor Pages bạn đã gặp object ViewData. Bạn đã sử dụng ViewData để truyền tiêu đề của trang tới [layout](#). Chúng ta nhắc lại một chút về ViewData.

### ViewData object

ViewData là một object thuộc kiểu Dictionary (chính xác hơn, là kiểu từ điển đặc biệt ViewDataDictionary) được cơ chế Razor Pages tạo sẵn và có tác dụng trên toàn bộ ứng dụng. Nghĩa là ở bất kỳ đâu trong ứng dụng bạn đều có thể truy xuất được object này.

ViewData còn có "người anh em" ViewBag nhưng không được khuyến khích sử dụng.

ViewData có thể lưu trữ object thuộc bất kỳ kiểu dữ liệu nào nhưng khóa của nó phải thuộc kiểu string (để có thể truy xuất từ các file cshtml).

Dưới đây là một ví dụ về lưu trữ các cặp khóa/giá trị với ViewData:

```
1. // Index.cshtml.cs
2. public class IndexModel : PageModel {
3.     public void OnGet() {
4.         ViewData["MyNumber"] = 2020;
5.         ViewData["MyString"] = "Hello World, Razor Pages!";
6.         ViewData["MyObject"] = new Book {
7.             Title = "ASP.NET Core for Dummy",
8.             Publisher = "Tự học ICT Press",
9.             Author = "Mai Chi"
10.        };
11.     }
12. }
```

Có thể truy xuất ViewData trong page Index.cshtml như sau:

```
1. @page
2. @model IndexModel
3. @{
4.     // phải cast kiểu object sang kiểu cụ thể
5.     var book = (Book) ViewData["MyObject"];
6. }
7.
8. <h2>@ViewData["MyString"]</h2>
9. <p>The answer to everything is @ViewData["MyNumber"]</p>
10.
11. <h2>@book.Title</h2>
12. <p>@book.Author</p>
13. <p>@book.Publisher</p>
```

## ViewData attribute

Khi sử dụng mô hình model class bạn còn có một cách khác nữa để làm việc với ViewData. Hãy cùng xem ví dụ nhỏ sau:

```
1. public class IndexModel : PageModel
2. {
3.     [ViewData]
4.     public string Message { get; set; }
5.     public void OnGet()
6.     {
7.         Message = "Hello World";
8.     }
9. }
```

Chú ý attribute `[ViewData]` phía trước khai báo property `Message`.

Bạn giờ đây có thể sử dụng `ViewData["Message"]` ở bất kỳ đâu.

```
1. @page
2. @model IndexModel
3.
4. <h2>@Model.Message</h2>
5. <h2>@ViewData["Message"]</h2>
```

Cách sử dụng này có tên gọi là **ViewData attribute**.

Bất kỳ property nào được định nghĩa cùng `[ViewData]` attribute sẽ tự động xuất hiện trong danh sách dữ liệu của object ViewData.

Đây là cách thức rất tiện lợi để để tự động chia sẻ giá trị của property (trong model class) với [layout page](#) và partial page (chúng ta sẽ học sau) mà không phải gán giá trị thủ công. Điều này đặc biệt có ý nghĩa nếu cần chia sẻ giá trị của nhiều property.

## Kết luận

Trong bài học này chúng ta đã làm quen với một đề rất quan trọng trong Razor: các mô hình tổ chức code. Mỗi mô hình thực ra đều có ưu điểm riêng của mình. Tùy vào xuất phát điểm bạn có thể thấy mô hình nọ dễ hiểu hơn mô hình kia. Tuy vậy, trong đa số các trường hợp bạn nên sử dụng mô hình model class.



[Link tải solution S04\\_WebApplication](#)

1 file(s) 0.00 KB

TẢI MÃ NGUỒN SOLUTION

- + 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!

