Action và ActionResult trong ASP.NET Core MVC

Action trong ASP.NET Core MVC là những phương thức public của controller có thể được sử dụng để xử lý truy vấn. Action result là cách gọi chung của kết quả thực hiện của action. Kết quả trả về của mỗi action có thể được biểu diễn ở dạng giao diện HTML, ở dạng dữ liệu (XML, JSON) hoặc các dạng thức đặc biệt khác.

Theo dõi bài học để hiểu chi tiết các vấn đề trên.

NỘI DUNG CỦA BÀI [Ẩn]

Action trong ASP.NET Core MVC

Trong mô tả kiến trúc MVC tổng quát, controller được xem là điểm khởi đầu (entry point) của quá trình sinh giao diện. Tuy nhiên, trong ASP.NET Core MVC, phương thức có khả năng xử lý truy vấn trong lớp controller mới thực sự là điểm khởi đầu. Người ta gọi loại phương thức như vậy là **action**.

Bạn có thể hình dung như thế này: trong ASP.NET Core MVC, mỗi URL tương ứng với một phương thức.

Khi một truy vấn đến server, Mvc Middleware căn cứ vào URL để xác định phương thức cần thực thi. Ví dụ, truy vấn chứa Url `/home/index` tương ứng với thực thi phương thức Index trong lớp HomeController.

Khi này, Mvc Middleware sẽ tìm kiếm trong assembly class HomeController, hoặc class con Home của Controller. Nếu tìm thấy, nó sẽ khởi tạo object của class này và tìm kiếm phương thức public Index trong đó. Nếu tìm thấy, nó sẽ thực thi phương thức Index.

Index là một phương thức **action** của Home controller.

> Nếu nhớ lại cơ chế routing trong Razor Pages bạn sẽ thấy sự khác biệt: mỗi URL tương ứng với một trang nội dung trong Razor Pages.

Bất kỳ phương thức public nào của lớp controller đều có thể trở thành phương thức action. Lớp controller có thể chứa không giới hạn các action.

Nhìn chung, mỗi action chịu các trách nhiệm sau:

- Xác nhận tính chính xác của truy vấn;
- Thực thi các lệnh tương ứng với truy vấn;
- Lựa chọn loại phản hồi phù hợp.

Trong 3 nhiệm vụ trên, nhiệm vụ "lựa chọn loại phản hồi" là bắt buộc đối với một action. Ở dạng đơn giản nhất, loại phản hồi có thể chỉ là một chuỗi ký tự như bạn đã thấy trong phần thực hành 1 của bài học về dự án ASP.NET Core MVC. Đối với ứng dụng web thông thường (trả HTML lại cho trình duyệt), action thường sẽ trả về một object thuộc kiểu ActionResult (hoặc IActionResult). Mvc Middleware sẽ sử dụng object này để sinh ra HTML.

Ở đây cần lưu ý rằng, action không trực tiếp sinh ra view. Thay vào đó, action lựa chọn loại view phù hợp và chuẩn bị dữ liệu cho view. Mvc Middleware sử dụng view engine (Razor) để biến view (vốn là một template + dữ liệu) thành kết quả cuối cùng (HTML).

Một điểm lưu ý nữa là không nên thực hiện các loại tính toán nghiệp vụ trong action. Đây là một lỗi rất thường gặp ở người mới học lập trình. Thay vào đó, bạn nên gọi dịch vụ phù hợp từ application model. Ví dụ, nếu cần tải dữ liệu từ cơ sở dữ liệu, bạn không nên viết code xử lý cơ sở dữ liệu trực tiếp trong thân action. Thay vào đó, bạn nên xây dựng một class riêng chuyên cho tương tác với cơ sở dữ liệu và gọi đến class này trong action.

Tham số cho action

Một truy vấn có thể chứa tham số. Ví dụ, một truy vấn để lấy một sản phẩm xác định sẽ thường phải gửi kèm Id của sản phẩm tương ứng.

Tham số của truy vấn có thể nằm trong URL (route data), nằm trog chuỗi truy vấn (query string), nằm trong đầu (header) hoặc thân truy vấn (request body).

Mvc Middleware có khả năng trích giá trị tham số từ truy vấn và biến đổi (convert) sang kiểu tương ứng của .NET. Nếu action yêu cầu tham số, giá trị lấy từ truy vấn (sau khi convert) sẽ truyền sang cho action. Nếu action không cần tham số, giá trị trích ra sẽ không được sử dụng.

Tham số của action có thể là các kiểu cơ sở của .NET (string, int, bool, v.v.), cũng có thể là kiểu phức tạp do người dùng định nghĩa (struct, class).

Quá trình trích – biến đổi kiểu này được gọi là **model binding**. Object tạo ra trong quá trình model binding và đóng vai trò tham số đầu vào cho action được gọi là **binding model**.

Cơ chế model binding giúp đơn giản hóa việc tiếp nhận tham số từ truy vấn. Nói chung, nhờ model binding, bạn có thể khai báo phương thức action như một phương thức C# thông thường mà không cần quan tâm đến việc trích và biến đổi tham số từ truy vấn.

> Model binding trong ASP.NET Core MVC có chút khác biệt với model binding trong Razor Pages mà bạn đã học. Bạn sẽ học chi tiết về model binding trong ASP.NET Core MVC trong bài học về routing và sử dụng form.

Các loại kết quả hoạt động của action

Kết quả hoạt động của một action là đưa ra quyết định xem loại phản hồi nào cần trả lại và chuẩn bị dữ liệu tương ứng.

Phản hồi cho một truy vấn HTTP có thể chứa HTML (sinh động từ khuôn mẫu), chứa dữ liệu (như JSON hay XML), là một hành động (điều hướng sang một URL khác), là thông báo lỗi (ví dụ 404 not found), chứa một file tĩnh, v.v..

Tự bản thân action không thực hiện việc sinh phản hồi. Việc sinh phản hồi là nhiệm vụ của Mvc Middleware.

Ví dụ, nếu action quyết định rằng cần sinh ra phản hồi chứa HTML, nó sẽ trả về object ViewResult. Trong object ViewResult chứa thông tin về file mẫu (Razor view) và dữ liệu (view model) cần thiết. Mvc Middleware sử dụng object này và kích hoạt Razor view engine để sinh ra HTML và đóng gói HTML đó vào gói tin phản hồi HTTP.

Kết quả hoạt động của action được thể hiện qua kiểu dữ liệu trả về của phương thức.

Trong trường hợp đơn giản nhất, kiểu dữ liệu trả về của action có thể chỉ là một chuỗi ký tự.

Tuy nhiên, để thể hiện những kết quả hoạt động phức tạp hơn của action, ASP.NET Core tạo ra những kiểu trả về đặc biệt cho từng trường hợp.

Ví dụ, kiểu trả về ViewResult thể hiện cần sinh ra HTML từ file Razor, RedirectResult thể hiện gói tin phản hồi 302 redirect, NotFoundResult thể hiện gói tin 404.

ViewResult, RedirectResult, hay NotFoundResult đều là các class thực thi một giao diện chung: IActionResult.

IActionResult và ActionResult

Bạn đã làm quen với ActionResult và IActionResult khi học về phương thức xử lý sự kiện (handler) trong Razor Pages. Mọi class thể hiện kết quả hoạt động của action đều kế thừa từ lớp trừu tượng ActionResult hoặc thực thi giao diện IActionResult.

ActionResult và IActionResult là chung của ASP.NET Core chứ không phải tính năng riêng của Razor Pages hay MVC.

Trong Razor Pages ActionResult (IActionResult) không quá quan trọng. Bạn thường không cần sử dụng đến nó. Tuy nhiên trong MVC, gần như mọi action đều trả về một trong các

kiểu dẫn xuất ActionResult hoặc kiểu thực thi IActionResult.

ActionResult hay IActionResult? Khi mới làm quen với MVC bạn có thể phân vân xem nên sử dụng ActionResult hay IActionResult làm kiểu trả về của action. Thực tế là không có gì khác biệt nếu bạn sử dụng các class action result do ASP.NET Core xây dựng sẵn. Các class này là hậu duệ của ActionResult, do vậy cũng đồng thời thực thi IActionResult.

Dưới đây là một số class thường gặp nhất:

- ViewResult – trả lại phản hồi chứa HTML. Đây là action result thường dùng nhất khi dùng MVC để viết các ứng dụng web HTML truyền thống.
- RedirectResult – trả lại phản hồi điều hướng trang 302. Loại kết quả này dùng khi cần điều hướng người dùng tới một URL khác.
- FileResult – trả lại một file tĩnh trong phản hồi.
- ContentResult – trả lại loại phản hồi chỉ chứa một chuỗi ký tự.
- StatusCodeResult – trả lại một mã phản hồi (trong header).
- NotFoundResult – trả lại mã 404 not found.

Phương thức hỗ trợ của Controller

Để đơn giản hóa hơn nữa việc trả về kết quả của action, lớp Controller đã xây dựng sẵn nhiều phương thức hỗ trợ.

Các phương thức hỗ trợ xây dựng sẵn trong lớp Controller được đặt tên theo class mà nó hỗ trợ nhưng loại bỏ đi hậu tố Result.

Ví dụ, phương thức View hỗ trợ trả lại kết quả thuộc loại ViewResult. Sử dụng phương thức View đơn giản hơn rất nhiều so với trực tiếp khởi tạo ViewResult.

Tương tự như vậy,

- RedirectResult có phương thức hỗ trợ là Redirect()
- NotFoundResult có phương thức hỗ trợ là NotFound()
- FileResult có phương thức hỗ trợ là File(),
- v.v..

Các class hỗ trợ này giúp chúng ta được những gì? Giả sử trường hợp bạn muốn trả về một view sinh HTML có kiểu ViewResult, bạn cần viết code như sau:

```
1.  IActionResult Index(){
2.      var view = new ViewResult();
3.      view.ViewData = ViewData; // truyền ViewData của controller cho view
4.      view.ViewData.Model = ("Donald", "Trump"); // truyền view model sang cho view
5.      return view;
6.  }
```

Nếu sử dụng phương thức hỗ trợ View(), code của bạn giờ sẽ như sau:

```
IActionResult Index(){
    return View(("Donald", "Trump"));
}
```

Như vậy, View giúp chúng ta gộp lệnh khởi tạo ViewResult, truyền view model, và truyền ViewData vào một lệnh duy nhất.

Action overloading trong controller

Như chúng ta đã nói ở trên, action thực chất chỉ là một phương thức public bình thường trong controller class.

Trong class C# cho phép tình trạng có nhiều phương thức trùng tên nhưng khác danh sách tham số. Hiện tượng này trong C# được gọi là method overloading (nạp chồng phương thức). C# compiler sử dụng danh sách tham số để phân biệt các overload của cùng một phương thức.

Trong ASP.NET Core điều này vẫn đúng.

Tuy nhiên, nếu bạn xây dựng controller với hai overload của cùng một action, khi chạy ứng dụng và gọi tới action đó bạn sẽ gặp lỗi "`AmbiguousMatchException: The request matched multiple endpoints.`".

Nếu không tin, hãy xây dựng controller như sau:

```
namespace WebApplication1.Controllers {
    public class GreetingController {
        public string Hi() => $"Welcome to tuhocict.com";
        public string Hi(string name) => $"Welcome, {name}";
    }
}
```

Chạy ứng dụng và nhập url /greeting/hi, bạn sẽ gặp lỗi "`AmbiguousMatchException: The request matched multiple endpoints.`".

Tại sao lại như vậy? Bạn cần hiểu cơ chế lựa chọn action của ASP.NET Core MVC.

Khi một truy vấn tới ứng dụng, cơ chế routing của ASP.NET Core MVC sẽ trích ra tên của controller và action.

> Trong bài học sau bạn sẽ làm quen với cơ chế routing trong MVC. Bạn sẽ biết rằng trong url phải chứa thông tin để ASP.NET Core MVC xác định được tên controller và action.

Dựa vào tên controller, ASP.NET Core MVC sẽ dùng reflection để khởi tạo object của controller. Sau đó qua tên action sẽ tìm phương thức trùng tên. Tuy nhiên, trong truy vấn và url không có thông tin về tham số, cơ chế reflection chỉ có thể xác định được action qua tên gọi.

Vì vậy, nếu kết quả tìm kiếm cho ra nhiều hơn một phương thức action, ASP.NET Core MVC sẽ không biết phải thực thi action nào và sẽ báo lỗi "`AmbiguousMatchException: The`

`request matched multiple endpoints.`".

Để sử dụng method overloading trong controller class, bạn cần một loại kỹ thuật khác.

Một truy vấn, ngoài thông tin về controller và action, còn chứa thêm thông tin về loại truy vấn (verb). Các loại truy vấn thường gặp bao gồm GET, POST, PUT, DELETE.

Nếu bạn chỉ định loại truy vấn nào action sẽ xử lý, ASP.NET Core có thể phân biệt các action với nhau. Ví dụ, bạn chỉ định phương thức Hi() xử lý truy vấn GET, Hi(string) xử lý truy vấn POST, Hi(string, string) xử lý PUT, Hi(int) xử lý DELETE.

Khi một truy vấn tới, ASP.NET Core sử dụng thêm thông tin về loại truy vấn để lựa chọn action.

Vẫn ví dụ với GreeingController, giờ bạn thay đổi code như sau:

```
1.  using Microsoft.AspNetCore.Mvc;
2.
3.  namespace WebApplication1.Controllers {
4.      public class GreetingController {
5.          [HttpGet]
6.          public string Hi() => $"Welcome to tuhocict.com";
7.
8.          [HttpPost]
9.          public string Hi(string name) => $"Welcome, {name}";
10.     }
11. }
```

Để ý attribute `[HttpGet]` và `[HttpPost]` trước mỗi action. Các attribute này được gọi là HTTP verb attribute. Chúng có tác dụng giúp chỉ định loại truy vấn mà action này có thể xử lý.

Giờ nếu bạn chạy ứng dụng với url `/greeting/hi` sẽ không còn lỗi nữa. Khi này action `Hi()` sẽ được thực thi.

Trong tình huống này, `/greeting/hi` là một truy vấn GET. Do vậy, action tương ứng với truy vấn GET là Hi() sẽ được thực thi. Không có sự trùng lặp nào ở đây nữa.

Do `[HttpGet]` là mặc định cho mỗi action, bạn có thể bỏ nó khỏi Hi() mà không gây lỗi gì.

> HTTP verb attribute cho phép thực hiện mô hình GET-POST-redirect thường gặp trong ứng dụng CRUD dữ liệu.

Kết luận

Trong bài học này bạn đã học chi tiết về action trong ASP.NET Core MVC. Action thực tế chỉ là một phương thức public có trả về kết quả. Action được Mvc Middleware tự động gọi khi có truy vấn phù hợp. Sự tương quan giữa truy vấn và action được thiết lập bởi cơ chế routing. ASP.NET Core MVC xây dựng các class action result giúp đơn giản hóa việc lựa chọn kết quả trả về.

+ 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!