Routing in Razor Pages

11 July 2017 13:28

ASP.NET CORE RAZOR PAGES

One of the top level design considerations for the developers of a server-side web application framework is how to match URLs to resources on the server so that the correct one processes the request. The most straightforward approach is to map URLs to physical files on disk, and this is the approach that has been implemented by the ASP.NET team for the Razor Pages framework.

There are some rules to learn about how the Razor Pages framework matches URLs to files, and how the rules can be customised to give different results if needed. If you are comparing Razor Pages to the Web Pages framework, you also need to understand what has replaced `UrlData`, the mechanism for passing data in URLs.

Rule number one is that Razor Pages need a root folder. By default, this folder is named "Pages" and is located in the root folder of the web application project. You can configure another folder as the root folder in the application's `ConfigureServices` method in the `Startup` class. Here's how you would change the root folder to one named "Content" located in the application's root folder:

```
public void ConfigureServices(IServiceCollection services)
{
    services
        .AddMvc()
        .AddRazorPagesOptions(options => {
            options.RootDirectory = "/Content";
    });
}
```

Rule number two is that URLs should not include the file extension if they map to Razor pages.

Rule number three is that "Index.cshtml" is a default document, which means that if a file name is missing from the URL, the request will be mapped to Index.cshtml in the specified folder. Here are some examples of how URLs are mapped to file paths:

URL	Maps To
www.domain.com	/Pages/Index.cshtml
www.domain.com/index	/Pages/Index.cshtml
www.domain.com/account	/Pages/account.cshtml /Pages/account/index.cshtml

In the last example, the URL maps to two different files - account.cshtml in the root folder, and index.cshtml in a folder named "account". There is no way for the Razor Pages framework to identify which of these options to select, so if you actually have both of these files in your application, an exception will be raised if you tried to browse to `www.domain.com/account`:

> **AmbiguousActionException: Multiple actions matched. The following actions matched route data and had all constraints satisfied:**
>
> **Page: /account/Index**
> **Page: /account**

Passing Data in URLs

Data can be passed in the URL as query string values just as in most other frameworks e.g. `www.domain.com/product?id=1`. Alternatively, you can pass it as route parameters, so the preceding example become `www.domain.com/product/1`. Parts of the URL must be mapped to parameter names, which is achieved by providing a route template to the page in question as part of the @page directive:

```
@page "{id}"
```

This template tells the framework to treat the first segment of the URL after the page name as a route parameter named "id". You can access the value of the route parameter in a number of ways. The first is to use the ViewData dictionary:

```
@page "{id}"
{
    var productId = RouteData.Values["id"];
}
```

Alternatively, you can add a parameter with the same name as the route parameter to the `OnGet()` method for the page and assign its value to a public property:

```
@page "{id}"
@{
    @functions{

        public int Id { get; set; }

        public void OnGet(int id)
        {
            Id = id;
        }
    }
}
<p>The Id is @Id</p>
```

Here's how that works if you are using a PageModel:

```
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace RazorPages.Pages
{
    public class ProductModel : PageModel
    {
        public int Id { get; set; }
        public void OnGet(int id)
        {
            Id = id;
        }
    }
}
```

```
@page "{id}"
@model  ProductModel
<p>The Id is @Model.Id</p>
```

Finally, you can use the `BindProperty` attribute on the public property and dispense with the parameter in the `OnGet` method. The Razor file content stays the same but the PageModel code is slightly altered:

```
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace RazorPages.Pages
{
    public class ProductModel : PageModel
    {
        [BindProperty(SupportsGet = true)]
        public int Id { get; set; }
        public void OnGet()
        {
        }
    }
}
```

Constraints

At the moment, the only constraint applied to the parameter in this example is that it must have a value. The URL `www.domain.com/product/apple` is just as valid a match for the route as `www.domain.com/product/21`. If you expect the id value to be an integer, you can specify this as a constraint by adding the data type to the template:

```
@page "{id:int}"
```

Now if you try to pass "apple" as a parameter value, the application will return a 404 Not Found status code.

You can specify that no value is required, by making the parameter nullable:

```
@page "{id:int?}"
```

If your application permits the use of "apple" as a parameter value, you can specify that only the characters from A-Z and a-z are permitted:

```
@page "{id:alpha}"
```

You can combine this with a minimum length requirement:

```
@page "{id:alpha:minlength(4)}"
```

You can see a full range of supported constraints in this article that introduced attribute routing in ASP.NET MVC 5.

Friendly URLs

Friendly URLs enable you to map URLs to an arbitrary file on disk, breaking the one-to-one mapping based on the file name. You might use this feature to preserve URLs for SEO purposes where you cannot rename a file, or for example if you want all requests to be handled by one file. Friendly URLs are a `RazorPagesOption`, configured in the `ConfigureServices` method in `StartUp` via the AddPageRoute method. The following example maps the URL `www.domain.com/product` to a file named "products.cshtml" in a folder named "extras" in the root Razor Pages folder (default: "Pages"):

```
public void ConfigureServices(IServiceCollection services)
{
    services
        .AddMvc()
        .AddRazorPagesOptions(options =>
        {
            options.Conventions.AddPageRoute("/extras/products", "product");
        });
}
```

If you are used to working with Friendly URLs in Web Forms, you should note that the order of the parameters to the `AddPageRoute` method is the opposite to the Web Forms `MapPageRoute` method , with the path to the file as the first parameter. Also, the `AddPageRoute` takes a route template as the second parameter rather than a route definition, where any constraints are defined separately.

The final example illustrates mapping all request to a single file. You might do this if the site content is stored in a specific location (database, Markdown files) and a single file (e.g. *index.cshtml*) is responsible for locating content based on the URL and then processing it into HTML:

```
public void ConfigureServices(IServiceCollection services)
{
    services
        .AddMvc()
        .AddRazorPagesOptions(options => {
            options.AddPageRoute("/index", "{*url}");
    });
}
```

The wild-card character in the route template (*) means "all". Even with this configuration, matches between existing files on disk and URLs will still be honoured.

Summary

The routing system in Razor Pages is very intuitive, being based on file locations, but it is also extremely powerful and configurable if you need to override the default conventions.

◀◀ Razor Pages - The Elevator Pitch

Sending Email in Razor Pages ▶▶

Other Sites

Learn Razor Pages
Learn Entity Framework Core
Learn Dapper

Categories

Archive

2020

November 2020 (3)
June 2020 (2)
May 2020 (1)
January 2020 (1)

2019

December 2019 (3)
November 2019 (3)
October 2019 (2)
August 2019 (2)
July 2019 (2)
May 2019 (1)
April 2019 (2)
March 2019 (1)
February 2019 (1)

2018

October 2018 (3)
September 2018 (4)
August 2018 (2)
July 2018 (2)
May 2018 (2)
March 2018 (2)
February 2018 (1)
January 2018 (1)

2017

September 2017 (2)
July 2017 (2)
May 2017 (3)
February 2017 (2)

2016

October 2016 (2)
September 2016 (1)
July 2016 (2)
June 2016 (1)
May 2016 (1)
April 2016 (1)
March 2016 (2)

2015

2014

2013

2012

2011

2010

2009

2008

2007
