Přehled ASP.NET Core MVC

Článek • 28. 01. 2023 • 8 min ke čtení

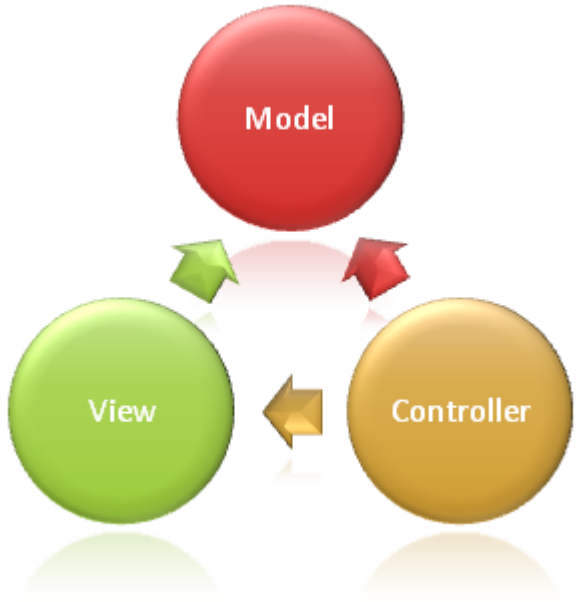
Autor : Steve Smith

ASP.NET Core MVC je bohatá architektura pro vytváření webových aplikací a rozhraní API pomocí vzoru návrhu model-View-Controller.

Model MVC

Vzor architektury MVC (Model-View-Controller) odděluje aplikaci do tří hlavních skupin komponent: Modely, zobrazení a kontrolery. Tento model pomáhá dosáhnout oddělení obav. Pomocí tohoto vzoru se požadavky uživatelů směrují na kontroler, který zodpovídá za práci s modelem za provádění uživatelských akcí nebo načítání výsledků dotazů. Kontroler zvolí zobrazení, které se má uživateli zobrazit, a poskytne mu všechna data modelu, která vyžaduje.

Následující diagram znázorňuje tři hlavní komponenty a na které odkazují ostatní:



Toto delineování zodpovědností pomáhá škálovat aplikaci z hlediska složitosti, protože je jednodušší kódovat, ladit a testovat něco (model, zobrazení nebo kontroler), který má jednu úlohu. Je obtížnější aktualizovat, testovat a ladit kód, který má závislosti rozložené do dvou nebo více těchto tří oblastí. Logika uživatelského rozhraní se například často mění než obchodní logika. Pokud se kód prezentace a obchodní logika zkombinují do jednoho

objektu, musí být objekt obsahující obchodní logiku změněn při každé změně uživatelského rozhraní. To často představuje chyby a vyžaduje opětovné testování obchodní logiky po každé minimální změně uživatelského rozhraní.

> **Poznámka**
>
> Zobrazení i kontroler závisí na modelu. Model ale závisí na zobrazení ani kontroleru. Jedná se o jednu z klíčových výhod oddělení. Toto oddělení umožňuje, aby byl model sestaven a testován nezávisle na vizuální prezentaci.

Odpovědnosti modelu

Model v aplikaci MVC představuje stav aplikace a všechny obchodní logiky nebo operace, které by měly provádět. Obchodní logika by měla být zapouzdřena v modelu spolu s jakoukoli logikou implementace pro zachování stavu aplikace. Zobrazení se silnými typy obvykle používají typy ViewModel navržené tak, aby obsahovaly data k zobrazení v daném zobrazení. Kontroler vytvoří a naplní tyto instance ViewModel z modelu.

Zobrazit zodpovědnosti

Zobrazení zodpovídají za prezentaci obsahu prostřednictvím uživatelského rozhraní. Pomocí Razor modulu zobrazení vloží kód .NET do kódu HTML. V zobrazeních by měla existovat minimální logika a veškerá logika by měla souviset s prezentováním obsahu. Pokud zjistíte, že při zobrazení souborů potřebujete provádět velkou logiku, abyste mohli zobrazit data ze složitého modelu, zvažte použití šablony View Component, ViewModel nebo view, abyste zobrazení zjednodušili.

Odpovědnost kontroleru

Kontrolery jsou komponenty, které zpracovávají interakci uživatelů, pracují s modelem a nakonec vyberou zobrazení, které se má vykreslit. V aplikaci MVC zobrazí zobrazení pouze informace; kontroler zpracovává a reaguje na uživatelský vstup a interakci. V modelu MVC je kontroler počáteční vstupní bod a zodpovídá za výběr typů modelů, se kterými se mají pracovat, a za zobrazení, se kterým zobrazením se má vykreslit (proto jeho název – řídí, jak aplikace reaguje na danou žádost).

> **Poznámka**

Kontrolery by neměly být příliš složité příliš mnoha zodpovědnostmi. Pokud chcete zachovat logiku kontroleru před příliš složitým, nasdílejte obchodní logiku z kontroleru a do doménového modelu.

Tip

Pokud zjistíte, že akce kontroleru často provádějí stejné druhy akcí, přesuňte tyto běžné akce do **filtrů**.

ASP.NET Core MVC

Architektura ASP.NET Core MVC je jednoduchá open source vysoce testovatelná prezentační architektura optimalizovaná pro použití s ASP.NET Core.

ASP.NET Core MVC poskytuje způsob, jak vytvářet dynamické weby, které umožňují čisté oddělení obav. Poskytuje úplnou kontrolu nad revizemi, podporuje vývoj vhodný pro TDD a používá nejnovější webové standardy.

Směrování

ASP.NET Core MVC je založená na směrování ASP.NET Core, což je výkonná komponenta mapování adres URL, která umožňuje vytvářet aplikace, které mají srozumitelné a prohledávatelné adresy URL. Díky tomu můžete definovat vzory pojmenování adres URL vaší aplikace, které dobře fungují pro optimalizaci vyhledávacích webů (SEO) a pro generování odkazů, bez ohledu na to, jak jsou soubory na webovém serveru uspořádané. Trasy můžete definovat pomocí vhodné syntaxe šablony tras, která podporuje omezení hodnot tras, výchozí hodnoty a volitelné hodnoty.

Směrování založené na konvencích umožňuje globálně definovat formáty adres URL, které vaše aplikace přijímá, a způsob, jakým se každý z těchto formátů mapuje na konkrétní metodu akce daného kontroleru. Když se obdrží příchozí požadavek, směrovací modul analyzuje adresu URL a shoduje se s ní s jedním z definovaných formátů adresy URL a zavolá metodu akce přidruženého kontroleru.

C#

```
routes.MapRoute(name: "Default", template: "
```

```
{controller=Home}/{action=Index}/{id?}");
```

Směrování atributů umožňuje určit informace o směrování tak, že dekódujete kontrolery a akce pomocí atributů, které definují trasy vaší aplikace. To znamená, že definice tras se umístí vedle kontroleru a akce, se kterou jsou přidružené.

C#

```
[Route("api/[controller]")]
public class ProductsController : Controller
{
    [HttpGet("{id}")]
    public IActionResult GetProduct(int id)
    {
      ...
    }
}
```

Vazby modelu

ASP.NET Core vazba modelu MVC převede data požadavku klienta (hodnoty formuláře, směrovací data, parametry řetězce dotazu, hlavičky HTTP) na objekty, které může kontroler zpracovat. V důsledku toho logika kontroleru nemusí provádět práci zjišťování příchozích dat požadavků; jednoduše má data jako parametry pro své metody akcí.

C#

```
public async Task<IActionResult> Login(LoginViewModel model, string returnUrl
= null) { ... }
```

Ověření modelu

ASP.NET Core MVC podporuje ověřování tím, že objekt modelu dekóduje pomocí atributů ověření poznámky k datům. Atributy ověření se kontrolují na straně klienta před odesláním hodnot na server a na serveru před zavolání akce kontroleru.

C#

```
using System.ComponentModel.DataAnnotations;
public class LoginViewModel
{
```

```
    [Required]
    [EmailAddress]
    public string Email { get; set; }

    [Required]
    [DataType(DataType.Password)]
    public string Password { get; set; }

    [Display(Name = "Remember me?")]
    public bool RememberMe { get; set; }
}
```

Akce kontroleru:

C#

```
public async Task<IActionResult> Login(LoginViewModel model, string returnUrl
= null)
{
    if (ModelState.IsValid)
    {
      // work with the model
    }
    // At this point, something failed, redisplay form
    return View(model);
}
```

Architektura zpracovává ověřování dat požadavků jak na klientovi, tak na serveru. Logika ověřování zadaná u typů modelů se přidá do vykreslených zobrazení jako nevěrné poznámky a v prohlížeči se vynucuje s ověřováním jQuery .

Injektáž závislostí

ASP.NET Core má integrovanou podporu injektáže závislostí (DI). V ASP.NET Core MVC můžou kontrolery vyžadovat potřebné služby prostřednictvím svých konstruktorů, což jim umožňuje dodržovat zásadu explicitních závislostí.

Aplikace může také použít injektáž závislostí v zobrazení souborů pomocí direktivy `@inject`:

CSHTML

```
@inject SomeService ServiceName
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>@ServiceName.GetTitle</title>
</head>
<body>
    <h1>@ServiceName.GetTitle</h1>
</body>
</html>
```

Filtry

Filtry pomáhají vývojářům zapouzdřit křížové otázky, jako je zpracování výjimek nebo autorizace. Filtry umožňují spouštění vlastní logiky předběžného zpracování a následného zpracování pro metody akcí a lze je nakonfigurovat tak, aby běžely v určitých bodech v rámci kanálu spouštění pro daný požadavek. Filtry lze použít na kontrolery nebo akce jako atributy (nebo je možné je spustit globálně). Součástí architektury je několik filtrů (například `Authorize`). `[Authorize]` je atribut, který se používá k vytváření autorizačních filtrů MVC.

C#

```
[Authorize]
public class AccountController : Controller
```

Oblasti

Oblasti poskytují způsob, jak rozdělit velkou ASP.NET Core webovou aplikaci MVC do menších funkčních seskupení. Oblast je struktura MVC uvnitř aplikace. V projektu MVC se logické komponenty, jako je model, kontroler a zobrazení, uchovávají v různých složkách a MVC používá konvence pojmenování k vytvoření vztahu mezi těmito komponentami. U velké aplikace může být výhodné rozdělit aplikaci do samostatných oblastí funkcí vysoké úrovně. Například aplikace elektronického obchodování s několika obchodními jednotkami, jako je pokladna, fakturace a vyhledávání atd. Každá z těchto jednotek má vlastní zobrazení logických komponent, kontrolery a modely.

Webová rozhraní API

Kromě toho, že je skvělou platformou pro vytváření webů, ASP.NET Core MVC má skvělou podporu pro vytváření webových rozhraní API. Můžete vytvářet služby, které se dostanou do široké škály klientů, včetně prohlížečů a mobilních zařízení.

Tato architektura zahrnuje podporu vyjednávání obsahu HTTP s integrovanou podporou formátování dat jako JSON nebo XML. Napište vlastní formátovací moduly , abyste přidali podporu vlastních formátů.

K povolení podpory hypermedia použijte generování odkazů. Snadno povolte podporu sdílení prostředků mezi zdroji (CORS), aby bylo možné webová rozhraní API sdílet napříč více webovými aplikacemi.

Testovatelnosti

Použití rozhraní a injektáž závislostí umožňuje dobře vyhovovat testování jednotek a architektura obsahuje funkce (jako je zprostředkovatel TestHost a InMemory pro Entity Framework), které umožňují rychlé a snadné integrační testy . Přečtěte si další informace o tom, jak otestovat logiku kontroleru.

Razor view engine

ASP.NET Core zobrazení MVC používá Razor modul zobrazení k vykreslení zobrazení. Razor je kompaktní, expressivní a fluidní jazyk značek šablon pro definování zobrazení pomocí vloženého kódu jazyka C#. Razor slouží k dynamickému generování webového obsahu na serveru. Kód serveru můžete čistě kombinovat s obsahem a kódem na straně klienta.

CSHTML

```
<ul>
    @for (int i = 0; i < 5; i++) {
        <li>List item @i</li>
    }
</ul>
```

Razor Pomocí modulu zobrazení můžete definovat rozložení, částečná zobrazení a nahraditelné oddíly.

Zobrazení se silnými typy

Razor zobrazení v MVC se dají na základě modelu silně napsat. Kontrolery můžou předat model se silným typem, aby zobrazení umožňovala kontrolu typů a podporu IntelliSense.

Například následující zobrazení vykreslí model typu `IEnumerable<Product>`:

CSHTML

```
@model IEnumerable<Product>
<ul>
    @foreach (Product p in Model)
    {
        <li>@p.Name</li>
    }
</ul>
```

Pomocné rutiny značek

Pomocné rutiny značek umožňují, aby se zúčastnili vytváření a vykreslování elementů HTML v Razor souborech. Pomocné rutiny značek můžete použít k definování vlastních značek (například `<environment>`) nebo k úpravě chování existujících značek (například `<label>`). Pomocné rutiny značek svázají s konkrétními prvky na základě názvu elementu a jeho atributů. Poskytují výhody vykreslování na straně serveru při zachování prostředí pro úpravy HTML.

Pro běžné úlohy , jako jsou vytváření formulářů, odkazů, načítání prostředků a další, a ještě více k dispozici ve veřejných úložištích GitHubu a jako balíčky NuGet existuje mnoho předdefinovaných pomocných rutin značek. Pomocné rutiny značek jsou vytvořené v jazyce C# a cílí na prvky HTML na základě názvu elementu, názvu atributu nebo nadřazené značky. Například předdefinovaný LinkTagHelper lze použít k vytvoření odkazu na `Login` akci: `AccountsController`

CSHTML

```
<p>
    Thank you for confirming your email.
    Please <a asp-controller="Account" asp-action="Login">Click here to Log
in</a>.
</p>
```

Dá `EnvironmentTagHelper` se použít k zahrnutí různých skriptů do zobrazení (například nezpracovaných nebo minifikovaných) v závislosti na prostředí modulu runtime, jako je

vývoj, příprava nebo produkční prostředí:

CSHTML

```
<environment names="Development">
    <script src="~/lib/jquery/dist/jquery.js"></script>
</environment>
<environment names="Staging,Production">
    <script src="https://ajax.aspnetcdn.com/ajax/jquery/jquery-2.1.4.js"
            asp-fallback-src="~/lib/jquery/dist/jquery.js"
            asp-fallback-test="window.jQuery">
    </script>
</environment>
```

Pomocné rutiny značek poskytují prostředí pro vývoj s podporou HTML a bohaté prostředí IntelliSense pro vytváření html a Razor značek. Většina předdefinovaných pomocných rutin značek cílí na existující prvky HTML a poskytuje atributy na straně serveru pro prvek.

Zobrazit komponenty

Zobrazit komponenty umožňují zabalit logiku vykreslování a opakovaně ji používat v celé aplikaci. Podobají se částečným zobrazením, ale s přidruženou logikou.

Kompatibilita – verze

Tato SetCompatibilityVersion metoda umožňuje aplikaci vyjádřit výslovný souhlas nebo vyjádřit výslovný nesouhlas s potenciálně zásadními změnami chování zavedeným v ASP.NET Core MVC 2.1 nebo novějším.

Další informace najdete v tématu Verze kompatibility pro ASP.NET Core MVC.

Další materiály

- MyTested.AspNetCore.Mvc – Fluent Testing Library for ASP.NET Core MVC : Knihovna testování jednotek se silnými typy, která poskytuje rozhraní pro testování aplikací MVC a webových rozhraní API. (*Microsoft neudržuje ani nepodporuje.*)
- Prerender a integrace komponent ASP.NET Core Razor
- Injektáž závislostí v ASP.NET Core