Projektowanie aplikacji ASP.NET
Wykład 09/15
ASP.NET MVC

Wiktor Zychla 2019/2020

Spis treści

1 Kontrolery, widoki

ASP.NET MVC używa własnego routera, klasy MvcRouteHandler który w tablicy routingu pozostawia po sobie dwie wartości:

- Pod kluczem „controller” – nazwę kontrolera
- Pod kluczem „action” – nazwę metody którą trzeba wywołać

Konfiguracja routera wymaga opisana formatu ścieżki, domyślna propozycja jest następująca

```
public class RouteConfig
{
    public static void RegisterRoutes(RouteCollection routes)
    {
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

        routes.MapRoute(
            name: "Default",
            url: "{controller}/{action}/{id}",
            defaults: new { controller = "Home", action = "Index", id = UrlParam
eter.Optional }
        );
    }
}
```

Wywołanie konfiguracji z **Application_Start**

```
RouteConfig.RegisterRoutes(RouteTable.Routes);
```

1.1 Tworzenie instancji kontrolera

Pierwszym krokiem potoku przetwarzania jest utworzenie instancji obiektu kontrolera. Zajmuje się tym **fabryka kontrolerów** (**DefaultControllerFactory**). Konwencja jest taka że klasa kontrolera dla ścieżki **/Foo/**… musi nazywać się **FooController** i dziedziczyć z klasy bazowej **Controller** (aby uzyskać dostęp do elementów infrastruktury takich jak Request czy Response):

```
public class HomeController : Controller
{
```

1.2 Wywołanie akcji w kontrolerze

Z kolejnego elementu ścieżki odczytana jest nazwa akcji, która musi być publiczną metodą kontrolera i zwracać wartość – obiekt dziedziczący z **ActionResult**.

```
public ActionResult Index()
```

Istnieje wiele gotowych implementacji **ActionResult,** w tym

- EmptyResult – pusta odpowiedź
- ContentResult – napis jako odpowiedź
- FileResult – strumień bajtów (plik) jako odpowiedź
- JsonResult – JSON jako odpowiedź
- ViewResult – plik jako odpowiedź

Istnieje możliwość obsłużenia akcji o nieznanej nazwie (czyli dowolnego „drugiego" segmentu ścieżki, np. /Foo/**Oiuoieruigoueriogu9595w0** gdzie w kontrolerze nie ma metody o takiej nazwie) za pomocą metody **HandleUnknownAction**, w której można np. przekierować żądanie na inną, istniejącą akcję, czy zwrócić istniejący widok:

```
protected override void HandleUnknownAction(string actionName)
{
    // redirect
    this.RedirectToAction("Bar");
    // istniejąca akcja
    this.ActionInvoker.InvokeAction(this.ControllerContext, actionName);
    // lub widok
    this.View("Foo").ExecuteResult(this.ControllerContext);
}
```

1.3 Widoki

ASP.NET MVC wspiera wymienialny mechanizm silników renderowania, referencyjnym silnikiem jest silnik Razor.

Tabela 1 https://haacked.com/archive/2011/01/06/razor-syntax-quick-reference.aspx/

Syntax/Sample	Razor	Web Forms Equivalent (or remarks)

Code Block	`@{`
	`  int x = 123;`
	`  string y = "because.";`
	`}`
		`<%`
		`  int x = 123;`
		`  string y = "because.";`
		`%>`

Expression (Html Encoded)	`<span>@model.Message</span>`	`<span><%: model.Message %></span>`

Expression (Unencoded)	`<span>`
	`@Html.Raw(model.Message)`
	`</span>`
		`<span><%= model.Message %></span>`

Combining Text and markup	`@foreach(var item in items) {`
	`  <span>@item.Prop</span>`
	`}`
		`<% foreach(var item in items) { %>`
		`  <span><%: item.Prop %></span>`
		`<% } %>`

Mixing code and Plain text	`@if (foo) {`
	`  <text>Plain Text</text>`
	`}`
		`<% if (foo) { %>`
		`  Plain Text`
		`<% } %>`

Using block	`@using (Html.BeginForm()) {`
	`  <input type="text" value="input here">`
	`}`
		`<% using (Html.BeginForm()) { %>`
		`  <input type="text" value="input here">`
		`<% } %>`

Mixing code and plain text (alternate)	`@if (foo) {`
	`  @:Plain Text is @bar`
	`}`
		Same as above

Email Addresses	`Hi philha@example.com`	Razor recognizes basic email format and is smart enough not to treat the @ as a code delimiter
Explicit Expression	`<span>ISBN@(isbnNumber)</span>`	In this case, we need to be explicit about the expression by using parentheses.
Escaping the @ sign	`<span>In Razor, you use the @@foo to display the value of foo</span>`	@@ renders a single @ in the response.

Server side Comment	`@*`
	`This is a server side multiline comment`
	`*@`
		`<%--`
		`This is a server side multiline comment`
		`--%>`

Calling generic method	`@(MyClass.MyMethod<AType>())`	Use parentheses to be explicit about what the expression is.

Creating a Razor Delegate	`@{`
	`  Func<dynamic, object> b =`
	`  @<strong>@item</strong>;`
	`}`
	`@b("Bold this")`
		Generates a `Func<T, HelperResult>` that you can call from within Razor. See this blog post for more details.

Mixing expressions and text	`Hello @title. @name.`	`Hello <%: title %>. <%: name %>.`
NEW IN RAZOR v2.0/ASP.NET MVC 4		
Conditional attributes	`<div class="@className"></div>`	When `className = null` `<div></div>`

		When `className = ""`
		
		`<div class=""></div>`
		When `className = "my-class"`
		
		`<div class="my-class"></div>`

Conditional attributes with other literal values	`<div class="@className foo bar"></div>`	When `className = null`
		
		`<div class="foo bar"></div>`
		Notice the leading space in front of `foo` is removed.
		When `className = "my-class"`
		
		`<div class="my-class foo bar"></div>`

Conditional data-* attributes.
 	
data- attributes are always rendered.*
	`<div data-x="@xpos"></div>`	When `xpos = null or ""`
		
		`<div data-x=""></div>`
		When `xpos = "42"`
		
		`<div data-x="42"></div>`

Boolean attributes	`<input type="checkbox" checked="@isChecked" />`	`When isChecked = true`
		
		`<input type="checkbox" checked="checked" />`
		`When isChecked = false`
		
		`<input type="checkbox" />`

URL Resolution with tilde	`<script src="~/myscript.js"></script>`	When the app is at `/`
		
		`<script src="/myscript.js"></script>`
		When running in a virtual application named `MyApp`
		
		`<script src="/MyApp/myscript.js"></script>`

2 Model-View-Controller

We wzorcu MVC, cykl żądanie-odpowiedź buduje zależności między elementami:

- Kontroler tworzy model i przekazuje go widokowi
- Widok renderuje zawartość odpowiedzi dla użytkownika
- Przeglądarka odsyła formularz na serwer
- Kontroler wiąże odesłane dane do instancji modelu

2.1 Kontroler -> model

Zadaniem kontrolera jest zebranie danych z różnych źródeł (m.in. baza danych) i przygotowanie **modelu** dla widoku. W ASP.NET MVC model jest silnie typowany, czyli wymaga napisania klasy której instancję akcja kontrolera przekazuje do widoku:

```
public class IndexModel
{
    public string Foo { get; set; }
}
```

```
[HttpGet]
public ActionResult Index()
{
    IndexModel model = new IndexModel();
    return View(model);
}
```

2.2 Model -> widok

Widok ma dostęp do instancji modelu, oraz do dodatkowych nieustrukturalizowanych kanałów **ViewData** i **ViewBag.**

```
@model WebApplication2.Models.IndexModel
@{
    ViewBag.Title = "Index";
}

@using (var form = Html.BeginForm())
{
    @Html.TextBoxFor( m => m.Foo )
    <button>Zapisz</button>
}
```

2.3 Widok -> kontroler

Po odesłaniu formularza na serwer, instancja modelu może być odtworzona z parametrów POST/GET na kilka sposobów:

1. Ręcznie:

```
[HttpPost]
public ActionResult Index( object o )
{
    IndexModel model = new IndexModel();
    model.Foo = this.Request.Form["Foo"];
    return View(model);
}
```

Proszę zwrócić uwagę, że wymóg innej sygnatury metody dla POST i GET wymusza dodatkowy (tu: niepotrzebny) argument metody.

2. Za pomocą wiązania (binding) do ogólnego obiektu typu **FormCollection**

```
[HttpPost]
public ActionResult Index( FormCollection form )
{
    IndexModel model = new IndexModel();
    model.Foo = form["Foo"];
    return View(model);
}
```

3. Za pomocą wiązania do instancji modelu, gdzie dodatkowo obsługiwane są atrybuty walidacyjne

```
[HttpPost]
public ActionResult Index( IndexModel model )
{
    return View(model);
}
```

2.4 Walidacja

W tym ostatnim przykładzie model udekorowany atrybutami z **System.ComponentModel.DataAnnotations** może służyć kontrolerowi informacjami o poprawnym wypełnieniu formularza:

```
public class IndexModel
{
    [Required]
    public string Foo { get; set; }
}
```

Kontroler ma dostęp do stanu walidacji modelu:

```
[HttpPost]
public ActionResult Index( IndexModel model )
{
```

```
    if (this.ModelState.IsValid)
    {
        return View(model);
    }
    else
    {
        return View(model);
    }
}
```

3 Przykład - uwierzytelnianie

3.1 Widok logowania (LogOn.cshtml)

```
@model LogOnModel

<h2>LogOn</h2>

@using (var form = Html.BeginForm())
{
    <table cellspacing="5">
        <tr>
            <td>Username</td>
            <td>@Html.TextBoxFor(m => m.UserName)</td>
        </tr>
        <tr>
            <td>Password</td>
            <td>@Html.PasswordFor(m => m.Password)</td>
        </tr>
        <tr>
            <td colspan="2">
                @Html.ValidationSummary()
            </td>
        </tr>
        <tr>
            <td colspan="2" align="right">
                <button>Zaloguj</button>
            </td>
        </tr>
    </table>
}
```

3.2 Model logowania

```
public class LogOnModel
{
    [Required]
    public string UserName { get; set; }
    [Required]
    public string Password { get; set; }

}
```

3.3 Kontroler logowania

```
public class AccountController : Controller
{
    [CustomActionFilter]
    [HttpGet]
    public ActionResult LogOn()
    {
```

```
        LogOnModel model = new LogOnModel();

        return View(model);
    }

    [HttpPost]
    public ActionResult LogOn(LogOnModel model, string ReturnUrl)
    {
        if (this.ModelState.IsValid)
        {
            if (Membership.ValidateUser(model.UserName, model.Password))
            {
                FormsAuthenticationTicket ticket = new FormsAuthenticationTicket
(model.UserName, false, 20);

                HttpCookie cookie = new HttpCookie(FormsAuthentication.FormsCook
ieName);
                cookie.Value = FormsAuthentication.Encrypt(ticket);
                this.Response.AppendCookie(cookie);

                return Redirect(ReturnUrl);
            }
            else
                this.ModelState.AddModelError("logon", "Zła nazwa użytkownika lu
b hasło");
        }

        return View(model);
    }

}
```