بسم الله الرحمن الرحیم

آموزشگاه تحلیل داده

تخصصی ترین مرکز برنامه نویسی و دیتابیس در ایران

مسیریابی(Routing) در ASP.NET Web API

مدرس : مهندس افشین رفوآ

دوره آموزشی Web API

در این مقاله قصد داریم توضیح دهیم چطور ASP.NET Web API درخواست های HTTP را به controller مسیردهی می کند.

اگر شما با ASP.NET MVC آشنایی داشته باشید، متوجه خواهید شد که Routing در Web API بسیار شبیه Routing در MVC می باشد. تفاوت اصلی در این است که Web API برای انتخاب action از متد HTTP به جای مسیر URI استفاده می کند. شما همچنین می توانید از نوع مسیریابی MVC در Web API استفاده کنید. قصد داریم این مقاله را با هیچ پیش فرضی از ASP.NET MVC ادامه دهیم.

جداول مسیریابی(Routing)

در Asp.net Web API ،یک controller کلاسی است که درخواست های HTTP را مدیریت می کند. متد های عمومی در controller را متد های action می نامند. وقتی که فریم ورک Web API درخواستی دریافت می کند, این درخواست ها را به action هدایت می کند. برای فهم این موضوع که کدام action فراخوانی می شود، فریم ورک از جدول مسیریابی استفاده می کند. Visual Studio برای Web API یک مسیر به صورت پیش فرض ایجاد می کند:

```
routes.MapHttpRoute(
   name: "API Default",
   routeTemplate: "api/{controller}/{id}",
   defaults: new { id = RouteParameter.Optional }
);
```

این مسیر درون WebApiConfig.cs که در App_Start قرار دارد تعریف شده است:

اگر Web API شما خود میزبان(self-host) باشد شما باید جدول مسیریابی را مستقیما در HttpSelfHostConfiguration تنظیم کنید.

هر ورودی در جدول مسیریابی شامل یک قالب مسیر می باشد. قالب پیش فرض مسیر در Web API به صورت api/{controller}/{id} می باشد. در این مقاله api بخش تحت اللفظی مسیر و controller و id به عنوان مکان های نگه دارنده ی(placeholder) متغیر ها می باشد.

وقتی فریم ورک Web API یک درخواست HTTP دریافت می کند، سعی می کند URI را به یکی از قالب های جدول مسیریابی هدایت کند. اگر مسیری برای اتصال یافت نشود، کاربر خطای 404 دریافت می کند. برای مثال URI های زیر به مسیر پیش فرض وصل می شوند:

- /api/contacts
- /api/contacts/1
- /api/products/gizmo1

با این حال، URI زیر مسیریابی نمی شود زیرا فاقد api می باشد:

- /contacts/1

نکته: دلیل استفاده از api در مسیریابی این است که با مسیریابی Asp.net MVC تداخل نداشته باشد. در آن صورت، شما می توانید با /contacts به کنترولر MVC و با /api/contacts به کنترولر Web API بروید. در صورتی که شما این قرارداد را قبول نداشته باشید باید جدول پیش فرض مسیریابی را دستکاری کنید.

- اولین باری که یک مسیر هدایت می شود، Web API کنترولر و action را انتخاب می کند:
- برای پیدا کردن controller، Web API به مقدار متغیر، کلمه ثابت Controller اضافه می کند.
- Web API برای پیدا کردن action، به متد HTTP و سپس آن action را که نامش با نام متد HTTP آغاز می شود جستجو می کند. برای مثال، با درخواست GET، Web API یک action که با Get آغاز می شود مثل GetContact یا GetAllContacts را جستجو می کند. این

قرارداد تنها برای متد های Get، POST، PUT و DELETE برقرار است. شما می توانید متد های دیگر HTTP را با استفاده از Attribute ها در controller خود فعال کنید. مثالی در این رابطه را در ادامه خواهیم دید.

- باقی متغیر های مکان های نگه دارنده (placeholder) در قالب مسیر مثل id به پارامتر های action متصل می شوند.

فرض کنید شما controller زیر را تعریف کرده اید:

```
public class ProductsController : ApiController
{
   public void GetAllProducts() { }
   public IEnumerableGetProductById(int id) { }
   public HttpResponseMessage DeleteProduct(int id) { }

}
```

در زیر چند درخواست احتمالی HTTP را که به action فراخوانی می شود نشان داده شده است:

Parameter	Action	URI Path	HTTP Method
(none)	GetAllProducts	api/products	GET
4	GetProductById	api/products/4	GET
4	DeleteProduct	api/products/4	DELETE
	(no match)	api/products	POST

دقت داشته باشید که id در URI اگر موجود باشد به پارامتر id در action هدایت می شود. در این مثال، controller، دو متد GET که یکی با id و دیگری بدون پارامتر می باشد، تعریف کرده است. همچنین، توجه داشته باشید که درخواست POST به خاطر آن که controller متد POST را تعریف نکرده است با عدم موفقیت خواهد بود.

تغییرات Routing

در قسمت قبل مفاهیم اصلی مسیریابی در Asp.net Web API را توضیح دادیم. در این قسمت قصد داریم بعضی تغییرات را توضیح دهیم.

متد های HTTP

به جای استفاده از نام های قراردادی برای متد های HTTP شما می توانید صراحتا متد HTTP را برای یک action با تنظیمات متد action با اتربیوت HttpGet، HttpPut، HttpPost یا HttpDelete تصریح کنید.

در مثال زیر، متد FindProduct به درخواست های GET هدایت شده است:

```
public class ProductsController : ApiController
{
   [HttpGet]
   public Product FindProduct(id) {}
}
```

برای آنکه برای یک action از چند متد HTTP و یا متد هایی غیر از GET، PUT، POST و DELETE استفاده کنید می توانید AcceptVerbs Attribute را برای لیستی از متد های HTTP قرار دهید.

```
public class ProductsController : ApiController
{
   [AcceptVerbs("GET", "HEAD")]
   public Product FindProduct(id) { }
   // WebDAV method
   [AcceptVerbs("MKCOL")]
   public void MakeCollection() { }
}
```

Routing با نام Action

طبق پیش فرض قالب مسیریابی، Web API از متد HTTP برای انتخاب action استفاده می کند. با این حال، شما می توانید یک مسیر که شامل نام action در URI می باشد بسازید:

```
routes.MapHttpRoute(
   name: "ActionApi",
   routeTemplate: "api/{controller}/{action}/{id}",
   defaults: new { id = RouteParameter.Optional }
);
```

در این قالب مسیر، پارامتر action در controller نام می گیرد. با این طرح از مسیریابی برای مشخص کردن متدهای مجاز HTTP از اتربیوت ها استفاده می کنیم. برای مثال فرض کنید controller شما متد های زیر را دارد:

```
public class ProductsController : ApiController
{
   [HttpGet]
   public string Details(int id);
}
```

در این صورت، یک درخواست GET برای api/products/details/1 باید به متد Details هدایت شود. این نوع مسیریابی بسیار شبیه Asp.net MVC است و شاید برای روش های تماس راه دور(RPC) Web API مناسب تر باشد.

شما می توانید با استفاده از ActionName Attribute نام action را تغییر دهید. در مثال بعدی دو action به api/products/thumbnail/*id* *هدایت شده است که یکی با* GET *و دیگری با* POST *می باشد:*

```
public class ProductsController : ApiController
{
   [HttpGet]
   [ActionName("Thumbnail")]
   public HttpResponseMessage GetThumbnailImage(int id);
   [HttpPost]
   [ActionName("Thumbnail")]
   public void AddThumbnailImage(int id);
}
```

Non-Action ها

برای جلوگیری از فراخوانی متدی توسط action از اتربیوت NonAction استفاده می کنیم. این کار به فریم ورک می فهماند که این متد action نیست حتی اگر با جدول Routing همخوانی داشته باشد.

```
// Not an action method.
[NonAction]
string GetPrivateData() { ... } public
```