بسم الله الرحمن الرحیم

آموزشگاه تحلیل داده

تخصصی ترین مرکز برنامه نویسی و دیتابیس در ایران

اتربیوت مسیریابی در ASP.NET Web API 2

مدرس : مهندس افشین رفوآ

دوره آموزشی Web API

مسیریابی یعنی چگونه Web API یک URI را به یک action تطبیق می دهد. Web API 2 از یک نوع جدید مسیریابی به نام مسیریابی اتربیوت(attribute routing) پیروی می کند. همانطور که از نام آن مشخص است، مسیریابی اتربیوت از اتربیوت ها برای تعریف مسیر ها استفاده می کند. مسیریابی اتربیوت به شما کنترل بیشتری در قبال URI ها در Web API می دهد. برای مثال شما می توانید به راحتی URI هایی بسازید که سلسله مراتب منابع را توضیح می دهد.

مدل قدیمی مسیریابی که مبتنی بر قرارداد مسیریابی(convention-based routing) است هنوز هم با قدرت استفاده می شود. در واقع، شما می توانید هر دو تکنیک را در یه پروژه مشابه به کار ببرید.

این مقاله چگونگی فعال سازی مسیریابی اتربیوت را نشان می دهد و انواع مختلف مسیریابی اتربیوت را توضیح می دهد.

پیش نیازها

Visual studio 2013 و یا visual studio express 2013

در مرحله بعد هم می توانید از NuGet Package Manager برابی نصب بسته های ضروری استفاده کنید. در منو Tools در visual studio گزینه Library Package Manager را انتخاب کنید و سپس Package Manager Console را بزنید. قطعه کد زیر را درون پنجره بزنید.

```
Install-Package Microsoft.AspNet.WebApi.WebHost
```

چرا باید مسیریابی اتربیوت کرد؟

اولین انتشار Web API بر اساس قرارداد اصلی مسیریابی بود. در آن مسیریابی, شما یک یا چند قالب مسیر تعریف می کنید که اساسا پارامتر های رشته ای هستند. وقتی فریم ورک یک درخواست دریافت می کند آن URI را با توجه به قالب مسیر تطبیق می دهد.

یکی از مزایای قرارداد مسیریابی این است که قالب آن در یک مکان تعریف می شود و قوانین مسیریابی همواره برای تمام controller ها اجرا می شود. متاسفانه، قرارداد مسیریابی، پشتیبانی از الگو های معمول URI را که در API های REST شایع هستند سخت می کند. برای مثال, منابع گاهی اوقات دارای منابع فرزند می باشند: مشتری دارای سفارش است، فیلم ها بازیگر دارند، کتاب ها نویسنده دارند و غیره. طبیعی است که یک URI که این روابط را نشان می دهد به صورت زیر باشد:

```
/customers/1/orders
```

ساخت این نوع از URI بر اساس قرارداد پایه ی مسیریابی سخت می باشد. اگرچه می توان این کار را انجام داد ولی قابل اندازه گیری در زمانی که شما چند کنترولر و منبع در اختیار دارید نمی باشد. با استفاده از مسیریابی اتربیوت, یک مسیر برای این URI تعریف می کند. شما می توانید به سادگی یک اتربیوت به اکشن کنترولر خود اضافه کنید:

```
[Route("customers/{customerId}/orders")]
public IEnumerable<Order> GetOrdersByCustomer(int customerId) { ... }
```

در زیر چند الگو که مسیریابی اتربیوت را ساده تر می کند بررسی می کنیم.

نسخه API

در این مثال /api/v1/products نسبت به /api/v2/products باید به کنترولر دیگری مسیردهی شود.

```
/api/v1/products
/api/v2/products
```

قطعه های URI سربار شده (Overloaded)

در مثال زیر، 1 شماره درخواست می باشد اما pending به یک مجموعه اشاره می کند.

```
/orders/1
/orders/pending
```

انواع پارامتر چند گانه

در مثال زیر، 1 شماره درخواست می باشد ولی 2013/06/16 یک تاریخ را مشخص می سازد.

```
/orders/1
/orders/2013/06/16
```

فعال سازی مسیریابی اتربیوت

برای فعال سازی مسیریابی اتربیوت MapHttpAttributeRoutes فراخوانی می شود. این متد در کلاس System.Web.Http.HttpConfigurationExtensions تعریف شده است.

```
using System.Web.Http;
namespace WebApplication
{
    public static class WebApiConfig
    {
        public static void Register(HttpConfiguration config)
        {
            // Web API routes
            config.MapHttpAttributeRoutes();
            // Other Web API configuration not shown.
        }
    }
}
```

مسیریابی اتربیوت می تواند با قانون مسیریابی قراردادی ادغام شود. برای تعریف مسیرها با قانون مسیریابی قراردادی متد MapHttpRoute فراخوانی می شود.

```
public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        // Attribute routing.
        config.MapHttpAttributeRoutes();
```

```
        // Convention-based routing.
        config.Routes.MapHttpRoute(
            name: "DefaultApi",
            routeTemplate: "api/{controller}/{id}",
            defaults: new { id = RouteParameter.Optional }
        );
    }
}
```

نکته: کوچ کردن (migration) از Web API 1

قبل از Web API 2 قالب های پروژه های Web API کد ها را به این صورت تولید می کردند:

```
protected void Application_Start()
{
    // WARNING - Not compatible with attribute routing.
    WebApiConfig.Register(GlobalConfiguration.Configuration);
}
```

اگر مسیریابی اتربیوت فعال شده باشد، این کد با یک استثنا همراه خواهد بود. اگر می خواهید پروژه Web API خود را برای مسیریابی اتربیوت به روز رسانی کنید، مطمئن شوید کد های تنظیمات به صورت زیر به روز رسانی شوند:

```
protected void Application_Start()
{
    // Pass a delegate to the Configure method.
    GlobalConfiguration.Configure(WebApiConfig.Register);
}
```

اضافه کردن اتربیوت های مسیر

در زیر یک مسیر با استفاده از اتربیوت تعریف شده است.

```
public class OrdersController : ApiController
{
    [Route("customers/{customerId}/orders")]
    [HttpGet]
    public IEnumerable<Order> FindOrdersByCustomer(int customerId) { ... }
}
```

رشته customers/{customerId}/orders قالب URI برای مسیر می باشد. Web API درخواست URI را با قالب تطابق می دهد. در این مثال customers و orders بخش های واقعی و customerId یک پارامتر متغیر می باشد. URI ها باید به صورت زیر باشد:

- http://localhost/customers/1/orders
- http://localhost/customers/bob/orders
- http://localhost/customers/1234-5678/orders

شما همچنین می توانید دسترسی ها را محدود کنید که در ادامه به آن خواهیم پرداخت.

توجه داشته باشید که پارامتر customerId در قالب مسیر، نام پارامتر *customerId* *در این متد را تطبیق می دهد. وقتی Web API اکشن کنترولر را فراخوانی می کند، سعی می کند تا قالب های مسیر را اتصال دهد. برای مثال، اگر URI به صورت* `http://example.com/customers/1/orders` باشد، Web API تلاش می کند در اکشن، 1 را به پارامتر *customerId* بسط دهد.

یک قالب URI می تواند چند پارامتر داشته باشد:

```
[Route("customers/{customerId}/orders/{orderId}")]
public Order GetOrderByCustomer(int customerId, int orderId) { ... }
```

هر متد کنترولری که اتربیوت مسیر نداشته باشد از قانون قراردادی مسیریابی اتربیوت پیروی خواهد کرد. در آن صورت، شما می توانید در پروژه مشابه، هر دو نوع مسیریابی را ادغام کنید.

متد های HTTP

Web API در متد HTTP بر اساس نوع درخواست (GET،POST و غیره) اکشن ها را انتخاب می کند. به صورت پیش فرض، Web API بر اساس نام کنترولر جست و جو می کند. برای مثال یک متد کنترولر به نام PutCustomers یک در خواست HTTP PUT را فراخوانی می کند.

شما می توانید این قرارداد را با اعلان دادن متد هر کدام از اتربیوت های زیر، باطل کنید:

- [HttpDelete]
- [HttpGet]
- [HttpHead]
- [HttpOptions]

- [HttpPatch]
- [HttpPost]
- [HttpPut]

مثال زیر متد CreateBook را به درخواست HTTP POST بسط می دهد.

```
[Route("api/books")]
[HttpPost]
public HttpResponseMessage CreateBook(Book book) { ... }
```

برای متد های HTTP دیگر که شامل متدهای غیر استاندارد می شود از اتریبیوت AcceptVerbs استفاده می شود که یک لیست از متد های HTTP می گیرد:

```
// WebDAV method
[Route("api/books")]
[AcceptVerbs("MKCOL")]
public void MakeCollection() { }
```

پیشوند های مسیر

گاهی اوقات، همه ی مسیرها در یک کنترولر با پیشوند مشابه آغاز می شوند. برای مثال

```
public class BooksController : ApiController
{
    [Route("api/books")]
    public IEnumerable<Book> GetBooks() { ... }
    [Route("api/books/{id:int}")]
    public Book GetBook(int id) { ... }
    [Route("api/books")]
    [HttpPost]
    public HttpResponseMessage CreateBook(Book book) { ... }
}
```

شما می توانید از یک پیشوند معمول برای کل کنترولر با استفاده از اتریبیوت [RoutePrefix] استفاده کنید.

```
[RoutePrefix("api/books")]
public class BooksController : ApiController
{
    // GET api/books
    [Route("")]
    public IEnumerable<Book> Get() { ... }
    // GET api/books/5
    [Route("{id:int}")]
    public Book Get(int id) { ... }
```

```
    // POST api/books
    [Route("")]
    public HttpResponseMessage Post(Book book) { ... }
}
```

در اتربیوت متد از (~) برای باطل کردن پیشوند مسیر استفاده می شود.

```
[RoutePrefix("api/books")]
public class BooksController : ApiController
{
    // GET /api/authors/1/books
    [Route("~/api/authors/{authorId:int}/books")]
    public IEnumerable<Book> GetByAuthor(int authorId) { ... }
    // ...
}
```

پیشوند مسیر می تواند شامل پارامتر هایی باشد:

```
[RoutePrefix("customers/{customerId}")]
public class OrdersController : ApiController
{
    // GET customers/1/orders
    [Route("orders")]
    public IEnumerable<Order> Get(int customerId) { ... }
}
```

محدودیت مسیر

محدودیت مسیر به شما اجازه می دهد تا شما چگونگی تطابق در قالب مسیر را با پارامتر ها محدود سازید. نحوه ی اصلی شکل آن {parameter:constraint} می باشد. برای مثال:

```
[Route("users/{id:int}"]
public User GetUserById(int id) { ... }
[Route("users/{name}"]
public User GetUserByName(string name) { ... }
```

در اینجا، اگر id مربوط به URI از نوع اعداد صحیح باشد تنها اولین مسیر انتخاب خواهد شد و در غیر این صورت دومین مسیر انتخاب خواهد شد. در جدول زیر محدودیت هایی که پشتیبانی می شوند را می بینید.

محدودیت	توضیح	مثال
alpha	تطبیق الفبای حروف بزرگ یا کوچک	{x:alpha}
bool	تطبیق یه مقدار با نوع bool	{x:bool}
datetime	تطبیق یک مقدار **DateTime**	{x:datetime}
decimal	تطبیق یک مقدار اعشاری	{x:decimal}
double	تطبیق یک مقدار اعشاری 64 بیتی	{x:double}
float	تطبیق یک مقدار اعشاری 32 بیتی	{x:float}
guid	تطبیق یک مقدار GUID	{x:guid}
int	تطبیق یک مقدار صحیح 32 بیتی	{x:int}
length	تطبیق یک اندازه و یا یک بازه مشخص	{x:length(6)} {x:length(1,20)}
long	تطبیق یک مقدار عددی صحیح 64 بیتی	{x:long}
max	تطبیق یک مقدار عددی صحیح با بیشترین مقدار	{x:max(10)}
maxlength	تطبیق یک مقدار رشته ای با بیشترین طول	{x:maxlength(10)}
min	تطبیق یک مقدار عددی صحیح با کمترین مقدار	{x:min(10)}
minlength	تطبیق یک مقدار رشته ای با کمترین طول	{x:minlength(10)}
range	تطبیق یک مقدر عددی صحیح در یک بازه عددی	{x:range(10,50)}

regex	تطبیق regular expression	{x:regex(^\d{3}-\d{3}-\d{4}$)}

توجه داشته باشید بعضی محدودیت ها مثل min مقادیری داخل پرانتز می گیرند. شما می توانید چند محدودیت را به یک پارامتر اعمال کنید و آن ها را با : از هم جدا کنید.

```
[Route("users/{id:int:min(1)}")]
public User GetUserById(int id) { ... }
```

محدودت مسیر سفارشی

شما می توانید با پیاده سازی رابط IHttpRouteConstraint محدودیت مسیر سفارشی بسازید. برای مثال ، قطعه کد زیر، گرفتن یک مقدار عددی صحیح غیر صفر را محدود می کند.

```
public class NonZeroConstraint : IHttpRouteConstraint
{
    public bool Match(HttpRequestMessage request, IHttpRoute route, string parameterName,
        IDictionary<string, object> values, HttpRouteDirection routeDirection)
    {
        object value;
        if (values.TryGetValue(parameterName, out value) && value != null)
        {
            long longValue;
            if (value is long)
            {
                longValue = (long)value;
                return longValue != 0;
            }
            string valueString = Convert.ToString(value, CultureInfo.InvariantCulture);
            if (Int64.TryParse(valueString, NumberStyles.Integer,
                CultureInfo.InvariantCulture, out longValue))
            {
                return longValue != 0;
            }
        }
        return false;
    }
}
```

قطعه کد زیر نشان می دهد چطور یک محدودیت را ثبت نام کنید:

```
public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        var constraintResolver = new DefaultInlineConstraintResolver();
        constraintResolver.ConstraintMap.Add("nonzero", typeof(NonZeroConstraint));
        config.MapHttpAttributeRoutes(constraintResolver);
    }
}
```

حالا شما می توانید محدودیت خود را در مسیرها اعمال کنید:

```
[Route("{id:nonzero}")]
public HttpResponseMessage GetNonZero(int id) { ... }
```

شما همچنین می توانید با پیاده سازی رابط IInlineConstraintResolver کل کلاسDefaultInlineConstraintResolver را جا به جا کنید. انجام این کار تمام محدودیت های در کار(built-in) را جا به جا می کند مگر آنکه در پیاده سازی IInlineConstraintResolver به طور مشخص آنها را اضافه کند.

پارامترهای اختیاری URI و مقادیر پیش فرض

شما می توانید با اضافه کردن یک پرسش به پارامتر مسیر یک پارامتر اختیاری URI بسازید. اگر یک پارامتر مسیر اختیاری باشد، شما باید یک مقدار پیش فرض برای پارامتر متد تعریف کنید.

```
public class BooksController : ApiController
{
    [Route("api/books/locale/{lcid:int?}")]
    public IEnumerable<Book> GetBooksByLocale(int lcid = 1033) { ... }
}
```

در ای مثال، /api/books/locale/1033 و /api/books/locale منبع مشترکی را بر می گردانند.

متعاقبا شما می توانید به صورت زیر درون یک قالب مسیر یک مقدار پیش فرض مشخص سازید:

```
public class BooksController : ApiController
{
    [Route("api/books/locale/{lcid:int=1033}")]
    public IEnumerable<Book> GetBooksByLocale(int lcid) { ... }
}
```

این مثال نیز شبیه مثال قبل است ولی وقتی مقدار پیش فرض اعمال می شود کمی با هم تفاوت دارند.

در مثال اول مقدار 1033 مستقیما به {lcid?} اختصاص داده شده و این پارامتر دقیقا این مقدار را دریافت می کند.

در مثال دوم مقدار 1033 در فرآیند model-binding مقدار {lcid=1033} می گیرد. Model-binder پیش فرض مقدار "1033" را به نوع عددی 1033 تبدیل می کند.

نام های مسیر

در Web API، هر مسیر نامی دارد. نام های مسیر برای ساختن لینک ها کاربرد دارند و شما می توانید یک لینک در پاسخ HTTP وارد کنید.

```
public class BooksController : ApiController
{
    [Route("api/books/{id}", Name = "GetBookById")]
    public BookDto GetBook(int id)
    {
        // Implementation not shown...
    }
    [Route("api/books")]
    public HttpResponseMessage Post(Book book)
    {
        // Validate and add book to database (not shown)
        var response = Request.CreateResponse(HttpStatusCode.Created);
        // Generate a link to the new book and set the Location header in the response.
        string uri = Url.Link("GetBookById", new { id = book.BookId });
        response.Headers.Location = new Uri(uri);
        return response;
    }
}
```

سفارش (Order) مسیر

وقتی فریم ورک می خواهد یک URI را با یک مسیر تطبیق دهید، مسیر ها را در یک order خاص ارزیابی می کند. برای مشخص سازی order خاصیت RouteOrder را در اتربیوت مسیر اضافه کنید. ابتدا مقادیر کوچکتر ارزیابی می شود. مقدار پیش فرض order صفر می باشد.

در زیر چگونگی ordering کلی را نشان می دهد:

1– خاصیت RouteOrder اتربیوت مسیر را مقایسه می کند.

2– هر قطعه URI در قالب مسیر را در نظر می گیرد. برای هر قطعه، order به صورت های زیر است:

قطعه های حقیقی

- پارامتر های مسیر با محدودیت
- پارامتر های مسیر بدون محدودیت

- قطعه های کلمات پارامتر با محدودیت
- قطعه های کلمات پارامتر بدون محدودیت

3– در صورتی که با هم مساوی باشند، مسیرها با یک مقایسه رشته ای ترتیبی غیر حساس به حروف کوچک از قالب مسیر order داده می شوند.در زیر قطعه کدی که شما باید در کنترولر خود تعریف کنید نشان می دهد:

```
[RoutePrefix("orders")]
public class OrdersController : ApiController
{
    [Route("{id:int}")] // constrained parameter
    public HttpResponseMessage Get(int id) { ... }
    [Route("details")]  // literal
    public HttpResponseMessage GetDetails() { ... }
    [Route("pending", RouteOrder = 1)]
    public HttpResponseMessage GetPending() { ... }
    [Route("{customerName}")]  // unconstrained parameter
    public HttpResponseMessage GetByCustomer(string customerName) { ... }
    [Route("{*date:datetime}")]  // wildcard
    public HttpResponseMessage Get(DateTime date) { ... }
}
```

مسیر های order شده به صورت زیر می باشد:

1. orders/details
2. orders/{id}
3. orders/{customerName}
4. orders/{*date}
5. orders/pending

توجه داشته باشید "details" یک قطعه حقیقی است و قبل از "{id}" قرار می گیرد ولی "pending" به دلیل خاصیت RouteOrder که 1 است در انتها قرار می گیرد. در این مثال فرض بر این است که مشتری با نام "details" و "pending" وجود ندارد. در واقع سعی شده از مسیر های مبهم خودداری شود. در این مثال, قالب مسیر مناسب برای GetByCustomer مسیر "customers/{customerName}" می باشد.