MISA®
TIN CẬY-TIỆN ÍCH-TẬN TÌNH

ASP.NET Web API 2: Routing

Nguyễn Văn Mạnh

I.Routing

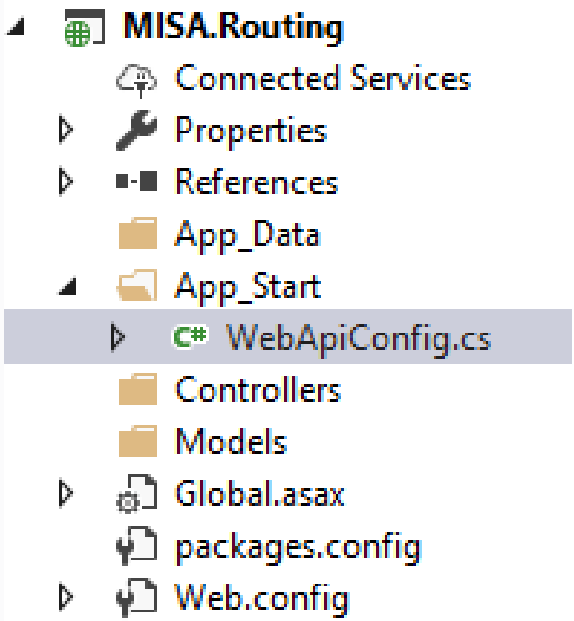
1) Convention-based routing

1) Attribute routing

1. Convention-based routing

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

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



1. Convention-based routing

Create a new route and add it into a collection manually

```
IHttpRoute defaultApi =
    config.Routes.CreateRoute(
        "api/{controller}/{id}",
        new { id = RouteParameter.Optional },
        null
    );

config.Routes.Add("DefaultApi", defaultApi);
```

1. Convention-based routing

Parameters of MapHttpRoute() method

```
config.Routes.MapHttpRoute(
    name: "defualtApi",
    routeTemplate: "api/{controller}/{id}",
    defaults: new {id = RouteParameter.Optional},
    constraints: new {id = @"\d+"}
);
```

1. Convention-based routing

Understand routeTeamplace

```
routeTemplate: "api/{controller}/{action}/{id}"
```

1. "api" is a literal path segment
2. {controller}, {action} and {id} are placeholder variables

1. Convention-based routing

Understand defaults

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

For the URI path "api/products", the route dictionary will contain:

- controller: "products"
- category: "all"

For "api/products/toys/123", however, the route dictionary will contain:

- controller: "products"
- category: "toys"
- id: "123"

1. Convention-based routing

Understand constraints

Regex expression to specify characteristic of route values

```
constraints: new {id = @"d+"}
```

1. Convention-based routing

HTTP Method

Name of the controller method starts with "Get", "Post", "Put", "Delete", "Head", "Options", or "Patch"

```
public IEnumerable<Product> GetAll() {}
public Product GetById(int id, double version = 1.0) {}
public void Post(Product value) {}
public void Put(int id, Product value) {}
```

2. Attribute routing

Enabling Attribute Routing

```
public static void Register(HttpConfiguration config)
{
    // Web API routes
    config.MapHttpAttributeRoutes();

    // Other Web API configuration not shown.
}
```

2. Attribute routing

Adding Route Attributes

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

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

2. Attribute routing

HTTP Methods

- **[HttpDelete]**
- **[HttpGet]**
- **[HttpHead]**
- **[HttpOptions]**
- **[HttpPatch]**
- **[HttpPost]**
- **[HttpPut]**

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

```
[HttpPost]
0 references
public HttpResponseMessage GetAll()
{
    return null;
}
```

2. Attribute routing

Route Prefixes

```
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) { ... }
}
```

2. Attribute routing

Route Prefixes

```
[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) { ... }
}
```

2. Attribute routing

Override the route prefix

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

    // ...
}
```

2. Attribute routing

Route Constraints

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

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

2. Attribute routing

Route Constraints

Constraint	Description	Example
alpha	Matches uppercase or lowercase Latin alphabet characters (a-z, A-Z)	{x:alpha}
bool	Matches a Boolean value.	{x:bool}
datetime	Matches a **DateTime** value.	{x:datetime}
decimal	Matches a decimal value.	{x:decimal}
double	Matches a 64-bit floating-point value.	{x:double}
float	Matches a 32-bit floating-point value.	{x:float}

2. Attribute routing

Route Constraints

length	Matches a string with the specified length or within a specified range of lengths.	{x:length(6)} {x:length(1,20)}
long	Matches a 64-bit integer value.	{x:long}
max	Matches an integer with a maximum value.	{x:max(10)}
maxlength	Matches a string with a maximum length.	{x:maxlength(10)}
min	Matches an integer with a minimum value.	{x:min(10)}
minlength	Matches a string with a minimum length.	{x:minlength(10)}
range	Matches an integer within a range of values.	{x:range(10,50)}
regex	Matches a regular expression.	{x:regex(^\d{3}-\d{3}-\d{4}$)}

2. Attribute routing

Other Attribute

[NonAction]

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

[AcceptVerbs]

```
[AcceptVerbs(HttpVerbs.Post | HttpVerbs.Get)]
public ActionResult GetAndPostAction()
{
    return RedirectToAction("Index");
}
```

Routing and Action Selection

Route Dictionary

If the framework finds a match for a URI, it creates a dictionary that contains the value for each placeholder

A default can have the special value RouteParameter.Optional. If a placeholder gets assigned this value, the value is not added to the route dictionary

Routing and Action Selection

Route Dictionary

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

For the URI path "api/products", the route dictionary will contain:

- controller: "products"
- category: "all"

For "api/products/toys/123", however, the route dictionary will contain:

- controller: "products"
- category: "toys"
- id: "123"

Routing and Action Selection

Selecting a Controller

1. Look in the route dictionary for the key "controller"
2. Take the value for this key and append the string "Controller" to get the controller type name
3. Look for a Web API controller with this type name

Routing and Action Selection

Selecting a Controller

If the route dictionary contains the key-value pair "controller" = "products"

Then the controller type is "ProductsController"

Routing and Action Selection

Action Selection

HTTP Methods: The framework only chooses actions that match the HTTP method of the request, determined as follows

1.HTTP method with an attribute: AcceptVerbs, HttpDelete, HttpGet, HttpHead, HttpOptions, HttpPatch, HttpPost, or HttpPut

2. If the name of the controller method starts with "Get", "Post", "Put", "Delete", "Head", "Options", or "Patch"

3. If none of the above, the method supports POST.

Routing and Action Selection

Action Selection

Parameter Bindings. A parameter binding is how Web API creates a value for a parameter. Here is the default rule for parameter binding

- Simple types are taken from the URI
- Complex types are taken from the request body.

Simple types include all of the .NET Framework primitive types, plus DateTime, Decimal, Guid, String, and TimeSpan

For each action, at most one parameter can read the request body.

Routing and Action Selection

Action Selection

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

```
public class ProductsController : ApiController
{
    public IEnumerable<Product> GetAll() {}
    public Product GetById(int id, double version = 1.0) {}
    [HttpGet]
    public void FindProductsByName(string name) {}
    public void Post(Product value) {}
    public void Put(int id, Product value) {}
}
```

Routing and Action Selection

Action Selection

1. Create a list of all actions on the controller that match the HTTP request method

```
GET http://localhost:34701/api/products/1?version=1.5&details=1
```

`GetAll`

`GetById`

`FindProductsByName`

Routing and Action Selection

Action Selection

2. If the route dictionary has an "action" entry, remove actions whose name does not match this value.

Routing and Action Selection

Action Selection

3. Try to match action parameters to the URI, as follows:
    a. For each action, get a list of the parameters that are a simple type, where the binding gets the parameter from the URI. Exclude optional parameters

Action	Parameters to Match
`GetAll`	none
`GetById`	"id"
`FindProductsByName`	"name"

Routing and Action Selection

Action Selection

3. Try to match action parameters to the URI

b. From this list, try to find a match for each parameter name, either in the route dictionary or in the URI query string.

Matches are case insensitive and do not depend on the parameter order.

- *id* = 1
- *version* = 1.5

Routing and Action Selection

Action Selection

3. Try to match action parameters to the URI

c. Select an action where every parameter in the list has a match in the URI

```
GetById
```

Routing and Action Selection

Action Selection

3. Try to match action parameters to the URI
    d. If more that one action meets these criteria, pick the one with the most parameter matches.

Routing and Action Selection

Action Selection

4. Ignore actions with the **[NonAction]** attribute.

Question???


THANK YOU