大多数MVC框架路由规则的配置都大同小异，一般都是通过模板的方式来配置路由规则。有的还支持在Controller和Action上通过Attribute（Java中叫注解）进行更细粒度的配置。

.NetCore MVC支持通过全局的路由模板配置路由规则，也支持在Controller和Action上通过Attribute进行细粒度的路由配置。下面先说一下在Startup.cs中配置全局路由规则。

## **0x01 在Startup.cs中配置路由**

所谓的路由的模板就是一串字符串，当接收到Http请求后取出其中的Path部分，和模板进行对照，如果匹配模板则路由到对应的Controller和Action进行处理。我们可以在Startup.cs文件中的Configure方法中，添加MVC功能时进行路由配置，例如：

app.UseMvc(routes =>

{

routes.MapRoute(

name: "default",

template: "{controller}/{action}/{id}");

});

其中name为路由规则的名称，template为路由模板。这也引出了我们第一个概念，路由模板中的变量。

### **1.路由模板中的变量**

在模板"{controller }/{action }/{id}"中，用花括号括起来的是路由模板中的变量。例如其中变量的作用并不是必须在Path中匹配某个固定的字符串，而是起到一个占位的作用，例如上面的模板就可以匹配由“/”隔开的共三部分的Path，例如a/b/c可以匹配成功。而各个变量的值从Path中对应部分提取出来。例如

Home/Index/12可以匹配，其中controller为Home，action为Index，id为12

Home/Index则匹配失败，因为只有2部分

Home/Index/12/34同样匹配失败，因为超过了3部分。

模板匹配成功后，会根据controller和action提取出的值路由：

Home/Index/12会路由到HomeController的Index方法，变量id为12

Test/Show/ab会路由到TestController的Show方法，变量id为ab

### **2.变量值得获取**

在Index或Show方法中，我们可以有两种方法提取变量：

一种是在方法的参数列表中加入和变量相同名称的参数，MVC会自动从变量列表中寻找并转换为对应类型：

public IActionResult Index(string id, string controller, string action)

{

ViewData["Message"] = "id is " + id + ", controller is " + controller + ", action is " + action;

return View();

}

另一种就是从RouteData中取出：

[IMG_256](http://www.cnblogs.com/durow/p/javascript:void(0);)

public IActionResult Index()

{

var controller = RouteData.Values["controller"].ToString();

var action = RouteData.Values["action"].ToString();

var id = RouteData.Values["id"].ToString();

ViewData["Message"] = "id is " + id + ", controller is " + controller + ", action is " + action;

return View();

}

[IMG_257](http://www.cnblogs.com/durow/p/javascript:void(0);)

路由模板中的变量名称是可以自己定义的，但controller和action（包括后面讲的area）都是比较特殊的变量。其中controller提取出的值作为Controller的名称，action提取出的值作为Controller中方法的名称。为了让每条路由规则都能够路由到Controller和Action，在路由模板中都应该出现controller和action变量，但我们也可以给controller和action变量指定默认值，这样在Path中省略了这部分时会用默认值代替。

### **3.变量的默认值**

由两种方法可以配置变量的默认值：

app.UseMvc(routes =>

{

routes.MapRoute(

name: "default",

template: "{controller=Home}/{action=Index}/{id=0}");

});

或者

[IMG_258](http://www.cnblogs.com/durow/p/javascript:void(0);)

routes.MapRoute(

name: "default",

template: "{controller}/{action}/{id}",

defaults: new

{

controller = ”Home”,

action = ”Index”,

id = 0,

});

[IMG_259](http://www.cnblogs.com/durow/p/javascript:void(0);)

这样配置后Path中带有默认值的部分可以省略，省略的规则和C#中带默认值的参数一样，例如：

空Path会被路由到HomeController，Index方法

Test会被路由到TestController，Index方法

Test/Show依然会被路由到TestController，Show方法

一般我会用第一种方法配置默认值，更加直观和方便。但有时候有些需求是第一种方法难以做到的。例如我想给TestController的Show方法配置路由为TestShow，使用第一种方法可以这样配置：”TestShow/{controller=Test}/{action=Show}”，这样配置当Path为TestShow时的确可以路由到TestController的Show方法，但当Path为TestShow/Home/Index时会路由到HomeController的Index方法。

使用第二种方法配置：

[IMG_260](http://www.cnblogs.com/durow/p/javascript:void(0);)

routes.MapRoute(

name: "test",

template: "TestShow",

defaults: new

{

controller = ”Test”,

action = ”Show”,

});

[IMG_261](http://www.cnblogs.com/durow/p/javascript:void(0);)

当Path为TestShow时可以路由到TestController的Show方法，但Path为Test/Home/Index则无法匹配模板。关于细粒度的路由配置更好的方法是给Test方法使用Route特性（Attribute）进行配置，后面会说到。

### **4.路由规则中的静态字符**

除了使用变量来配置路由模板，还可以使用静态字符串。静态字符串可以单独使用，也可以与变量混合使用。

例如模板为：

”Durow/{controller}/{action}”

Durow/Home/About会路由到HomeController，About方法

Durow/Test/Show会路由到TestController，Show方法

也可以把静态字符和变量混合起来，例如配置模板为：

”My{controller}/My{action}”

MyHome/MyAbout会被路由到HomeController，About方法

MyTest/MyShow会被路由到TestController，Show方法

### **5.使用?标记变量可选**

除了通过给变量提供默认值使其可选外，也可以使用?把变量标记为可选。例如模板

“{controller}/{action}/{id?}”

其中id为可选变量，这样配置后

Home/Index和Home/Index/12都会成功匹配。

### **6.使用\*提取Path中剩余的所有部分**

如果一个模板需要匹配包含任意多个部分（Segments）的Path，可以使用\*符号指定变量，使用\*制定过的变量会把Path中匹配完成后剩余部分全部提取出来，例如模板：

”{controller}/{action}/{id?}/{\*others}”

Home/Index/12/a/b/c/d，会路由到HomeController的Index方法，id为12，others为a/b/c/d

实际上仅从模板匹配的角度来说，上面的模板可以匹配所有的Path。唯一的问题就是匹配后对应的Controller和Action可能不存在。

### **7.多条路由规则的选择**

实际应用中很可能会配置多条路由规则，当接收到Path时很可能不止一条规则能够匹配。

最简单的，我们配置以下两条模板：

“{controller }/{action =About}”

“{controller }/{action =Index }”

当Path为Home时两条路由都能匹配，那要怎么选择呢？其实很简单粗暴，就是看哪条路由在前面。也就是说Path一旦成功匹配到模板后就会立即实施路由并忽略后面的模板。对于上面的配置来说Home会被路由到HomeController的About方法。所以在配置路由时一定要注意顺序。

## **0x02 使用Attribute配置路由**

除了在Startup.cs中配置全局路由规则外，也可以针对单个Controller和其中的Action配置路由。方法就是在Controller类和Action方法上使用Route特性。例如在TestController的Show方法上使用Route特性：

[Route("TestShow")]public IActionResult Show()

{

return View();

}

当Path为TestShow时，即可路由到TestController的Show方法。

上面我们在介绍默认值时提到过，通过全局模板配置：

[IMG_262](http://www.cnblogs.com/durow/p/javascript:void(0);)

routes.MapRoute(

name: "test",

template: "TestShow",

defaults: new

{

controller = ”Test”,

action = ”Show”,

});

[IMG_263](http://www.cnblogs.com/durow/p/javascript:void(0);)

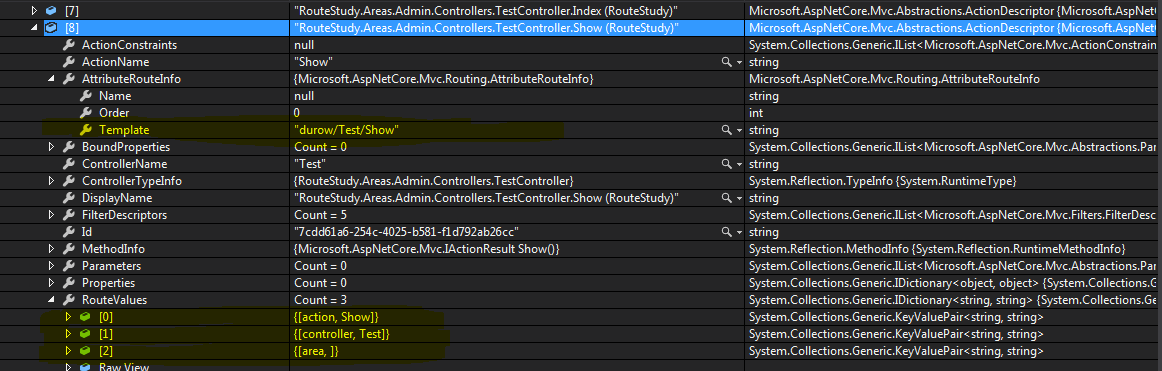
也可以达到同样的目的。不过区别在于，使用后一种方法时，如果还有”{controller}/{action}”这样的模板，除了TestShow外，当Path为Test/Show可以匹配这个模板并路由到TestController的Show方法。而通过在Show方法上配置Route特性后，只有TestShow才可以路由，即使同时存在”{controller}/{action}”这样的模板，Test/Show也无法路由。

第一次接触用Route特性配置路由时，我很疑惑路由组件是如何把Path路由到对应的Controller和Action的，后来下了个断点看了下RouteData对象，发现对于配置了路由的Action方法，其controller为方法所在的Controller的名称，action为方法的名称，而且在Route特性配置的路由模板中不能够使用{controller}变量和{action}变量。这样就保证了匹配模板的Path总能路由到这个Action。

对于在Controller类上配置的Route特性最终会分别配置到Controller中的每个Action上。例如我们在TestController上配置Route(“TestShow”)，实际上就是给每个方法配置了Route(“TestShow")，所以当Path为TestShow时会报错，提示有两个action满足匹配。那么应当如何给Controller通过Route配置路由呢，可以使用[controller]和[action]。

### **Route特性中的[controller]和[action]**

对于[controller]和[action]我也不知道该怎么叫，不能叫变量，功能上类似占位符。当我们在Controller类用Route特性配置路由时，如果使用了[controller]和[action]，这样当Route特性给Controller中每个Action配置路由时，[controller]会被替换为Controller名称，[action]会被替换为Action名称。举个例子还是给TestController配置Route特性，配置为Route(“durow/[controller]/[action]”)，这样对于其中的Index方法来说，其路由模板为”durow/Test/Index”，controller为Test，action为Index。而对于Show方法来说路由模板为”durow/Test/Show”，controller为Test，action为Show。前面说过MVC会为每个Action创建一个ActionDescriptor对象存储这个Action的路由信息。对于配置了Route特性的Action（再重复一下，给Controller类配置Route特性相当于给Controller中的每个Action配置Route特性），其ActionDescriptor中会有一个AttributeRouteInfo对象，对于未配置Route特性的Action，该对象为空。AttributeRouteInfo中包含了路由模板信息。



所以对于上面TestController的Route特性的配置，配置为Route(“durow/Test/[action]”)也能达到同样的效果。不过使用Route(“durow/[controller]/[action]”)语义更强更通用。

### **在Route特性中使用变量**

在Route特性中配置模板也是可以使用变量的，同样可以使用?标记变量可选。例如可以给TestController配置Route(“durow/[controller]/[action]/{id?}”)。但需要注意的是Route特性的模板中变量不能使用默认值（包括[controller]和[action]），也不能使用\*提取Path所有剩余部分。

## **0x03 写在最后**

啰啰嗦嗦居然写了这么多，其实实际使用中很可能用不到多么复杂的路由，一般一条通用规则，一条Area相关的规则就可以了。不过详细了解了路由规则，当以后遇到有些奇葩的特殊需求时能够有更加开阔的思路。后面讲讨论一下路由模板中的约束和自定义约束。再后面讨论一下使用Areas。

## **0x00 路由模板中的约束**

在配置路由模板时，除了以静态字符和变量的形式外，还可以通过配置对变量进行约束。实际如果不嫌麻烦的话在路由到的Action中对变量进行检查也是一种方法，不过对于变量的通用的约束当然是放在路由层面更加合适。而且这样做会简化Action的代码，使Action更加专注于自身业务，符合AOP的思路。这篇文章主要介绍路由模板中的约束。

## **0x01 在路由模板中使用约束**

在路由模板中使用约束是很简单的，如下所示：

{controller=Home}/{action=Index}/{id:int?}

这个模板就对id进行了约束，约束是通过冒号（:）实现的，冒号后为约束的内容。Int?代表id约束为int?类型，只有id能被转换为int?类型模板匹配才算成功。即使用int.TryParse()方法返回true。

Home/Index/123匹配成功，id为123

Home/Index/abc 匹配失败，因为abc无法转换为int?

Home/Index匹配成功，此时id为null

和配置默认值一样，除了直接在模板中配置外，也可以在MapRoute方法中通过传入参数进行配置：

[IMG_256](https://www.cnblogs.com/durow/p/javascript:void(0);)

routes.MapRoute(

name: "default",

template: "{controller}/{action}/{id?}",

defaults: new

{

controller = "Home",

action = "Index",

},

constraints: new

{

id = new IntRouteConstraint()

});

[IMG_257](https://www.cnblogs.com/durow/p/javascript:void(0);)

效果是一样的。

如果我们要对一个变量使用多个约束，只需要在约束后面继续用冒号加约束即可。例如

{controller=Home}/{action=Index}/{id:int:min(0)}

这个模板中id被约束为int类型且最小值为0。同样的也可以以参数的方式配置：

[IMG_258](https://www.cnblogs.com/durow/p/javascript:void(0);)

routes.MapRoute(

name: "default",

template: "{controller}/{action}/{id?}",

defaults: new

{

controller = "Home",

action = "Index",

},

constraints: new

{

id = new CompositeRouteConstraint(

new IRouteConstraint[]

{

new IntRouteConstraint(),

new MinRouteConstraint(0)

})

});

[IMG_259](https://www.cnblogs.com/durow/p/javascript:void(0);)

可以达到同样的效果，不过直接写到模板更加简洁和直观。

## **0x02 .NetCore MVC中内建的约束**

.NetCore MVC中内建了大量的约束可供使用。其中大多数基于类型的约束都是使用该类型的TryParse()来尝试转换变量提取出来的字符串，返回ture则通过约束，返回false违反约束。

1.约束为int：int，对应的类为IntRouteConstraint

例子：”{controller=Home}/{action=Index}/{id:int }”

2.约束为float：float，对应的类为FloatRouteConstraint

例子：”{controller=Home}/{action=Index}/{id:float }”

3.约束为long：long，对应的类为LongRouteConstraint

例子：”{controller=Home}/{action=Index}/{id:long}”

4.约束为double：double，对应的类为DoubleRouteConstraint

例子：”{controller=Home}/{action=Index}/{id:double }”

5.约束为decimal：decimal，对应的类为DecimalRouteConstraint

例子：”{controller=Home}/{action=Index}/{id:decimal }”

6.约束为布尔值：bool，对应的类为BoolRouteConstraint

例子：”{controller=Home}/{action=Index}/{id:bool}”

7.约束为字母：alpha，对应的类为AlphaRouteConstraint

例子：”{controller=Home}/{action=Index}/{id:alpha}”

8.约束为时间日期：datetime，对应的类为DateTimeRouteConstraint

例子：”{controller=Home}/{action=Index}/{id:datetime}”

9.约束为GUID：guid，对应的类为GuidRouteConstraint

例子：”{controller=Home}/{action=Index}/{id:bool}”

10.约束长度：length(len)，对应的类为LengthRouteConstraint(len)

例子：”{controller=Home}/{action=Index}/{id:length(5) }”

11.约束最小长度：minlength(len)，对应的类为MinLengthRouteConstraint(len)

例子：”{controller=Home}/{action=Index}/{id:minlength(5)}”

12.约束最大长度：maxlength(len)，对应的类为MaxLengthRouteConstraint(len)

例子：”{controller=Home}/{action=Index}/{id:maxlength(10)}”

13.约束长度范围：length(min,max)，对应的类为LengthRouteConstraint(min,max)

例子：”{controller=Home}/{action=Index}/{id:length(5,10)}”

14.约束最小值：min(v)，对应的类为MinRouteConstraint(v)

例子：”{controller=Home}/{action=Index}/{id:min(0)}”

15.约束最大值：max(v)，对应的类为MaxRouteConstraint(v)

例子：”{controller=Home}/{action=Index}/{id:max(1000)}”

16.约束范围：range(min,max)，对应的类为RangeRouteConstraint(min,max)

例子：”{controller=Home}/{action=Index}/{id:range(0,1000) }”

17.约束正则表达式：regex(exp)，对应的类为RegexRouteConstraint(exp)

例子：”{controller:regex(^My.\*)=Home }/{action=Index}/{id?}”

## **0x03 创建自定义约束**

除了使用内建的约束外，还可以自定义约束。只要实现IRouteConstraint接口即可。这个接口定义了一个方法：

bool Match(

HttpContext httpContext, //Http上下文信息

IRouter route, //当前正在测试的模板信息

string routeKey, //当前正在测试约束的变量名称

RouteValueDictionary values, //模板中变量提取出的值

RouteDirection routeDirection) //路由方向，枚举，值包括IncominRequestg和UrlGeneration

这个方法返回ture说明符合约束，返回false说明违反约束。下面我们通过实现这个接口定义一个约束，把id约束为偶数。

[IMG_260](https://www.cnblogs.com/durow/p/javascript:void(0);)

public class EvenRouteConstraint : IRouteConstraint

{

public bool Match(HttpContext httpContext, IRouter route, string routeKey, RouteValueDictionary values, RouteDirection routeDirection)

{

int id;

if (!int.TryParse(values[routeKey].ToString(), out id))

return false;

return id % 2 == 0;

}

}

[IMG_261](https://www.cnblogs.com/durow/p/javascript:void(0);)

然后这样配置路由来约束id：

[IMG_262](https://www.cnblogs.com/durow/p/javascript:void(0);)

routes.MapRoute(

name: "default",

template: "{controller}/{action}/{id?}",

defaults: new

{

controller = "Home",

action = "Index",

},

constraints: new

{

id = new EvenRouteConstraint()

});

[IMG_263](https://www.cnblogs.com/durow/p/javascript:void(0);)

这样配置后

空Path路由到HomeController的Index，id为空

Home/Index/12路由到HomeController的Index，id为12，是偶数

Home/Index/13匹配失败，id为13，不是偶数。

除了使用constraints参数添加自定义路由约束外，也可以把自定义路由约束映射为字符串，用于路由模板中。

[IMG_264](https://www.cnblogs.com/durow/p/javascript:void(0);)

public void ConfigureServices(IServiceCollection services)

{

services.Configure<RouteOptions>(options=>

{

options.ConstraintMap.Add("even",typeof(EvenRouteConstraint));

});

// Add framework services. services.AddMvc();

}

[IMG_265](https://www.cnblogs.com/durow/p/javascript:void(0);)

这样配置后路由模板写成：

routes.MapRoute(

name: "default",

template: "{controller=Home}/{action=Index}/{id:even?}");

});

可以达到同样的效果。把约束映射为字符串后，还可以方便的在Route特性中使用，例如：

[Route("durow/Test/[action]/{id:even?}")]

http://www.cnblogs.com/durow/p/5992926.html