# 前言

本节我们来讲讲ASP**[.NET](http://lib.csdn.net/base/dotnet" \o ".NET知识库" \t "http://blog.csdn.net/ceg6648/article/details/_blank)** Core中的路由，在讲路由之前我们首先回顾下之前所讲在ASP**[.Net](http://lib.csdn.net/base/dotnet" \o ".NET知识库" \t "http://blog.csdn.net/ceg6648/article/details/_blank)** Core中的模型绑定这其中有一个问题是我在项目当中遇见的，我们下面首先来看看这个问题。

## 回顾ASP.NET Core模型绑定

我们有这样一个场景：修改个人资料中的各个属性，此时每个属性的值的类型肯定是不一样的，所以我们将值定义为object，如下model。

public class BlogViewModel

{

public string prop { get; set; }

public object value { get; set; }

}

我们在前台进行提交，如下：

<script type="text/javascript">

$(function () {

var model = { prop: "Jeffcky", value: "cnblogs" };

$("#btn").on("click", function () {

$.ajax({

url: "/home/index",

type: "post",

contentType: "application/x-www-form-urlencoded;charset=utf-8",

dataType: "json",

data: model,

success: function (data) {

}

});

});

});</script>

我们在后台接收传递的对象。

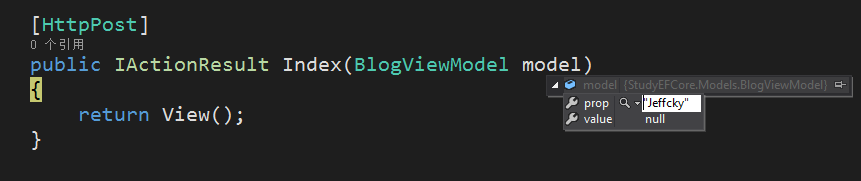
[HttpPost]

public IActionResult Index(BlogViewModel model)

{

return View();

}



此时我们看到将值类型定义为object根本接收不到值，当我将其修改为string肯定是好使的。之前以为利用表单接收任何对象都是好使的，在实际应用时却不如我们所期望的那样（对于此种情况暂时未找到解决方案）。在这种情况下我只好利用contentType:"application/json"来解决，如下：

$(function () {

var model = { prop: "Jeffcky", value: "cnblogs" };

$("#btn").on("click", function () {

$.ajax({

url: "/home/index",

type: "post",

contentType: "application/json;charset=utf-8",

dataType: "json",

data: JSON.stringify(model),

success: function (data) {

}

});

});

});

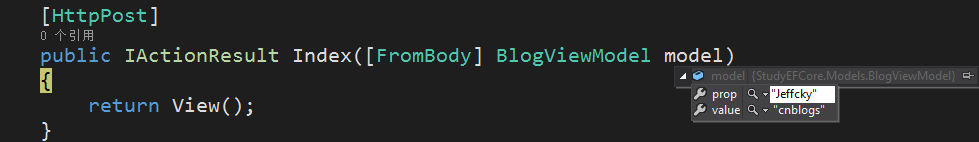
[HttpPost]

public IActionResult Index([FromBody] BlogViewModel model)

{

return View();

}



只能转换思维来解决，好了到了这里我们对于对象利用模型绑定遇到的问题到此结束，接下来我们开始讲述路由。

## ASP.NET Core MVC/WebAPi路由作用

无论是在ASP.NET MVC/WebAPi还是ASP.NET Core MVC/WebAPi，我们都一直在讲路由，为什么每次都要拿出来说呢，因为它重要，从前台到后台的数据交互需要通过路由解决，所以我们需要在ASP.NET Core重点拿出来讲，我们得好好说说路由的好处是什么呢，请往下看。

### （1）SEO

SEO即Search Engine Optimization，我们翻译过来就是**[搜索引擎](http://lib.csdn.net/base/searchengine" \o "搜索引擎知识库" \t "http://blog.csdn.net/ceg6648/article/details/_blank)**优化，引入路由就是为了呈现友好的SEO，一个友好的SEO即能够非常准确的描述需要请求的资源，对于用户而言它能更好的去获取想要的内容，可以增加网址访问的次数，当然了也可以是网址易于收录到搜索引擎中。

### （2）URL不需要映射到文件

如果没有路由的话，如果过来一个请求此时需要映射到磁盘上真实存在的物理文件，有了路由我们可以完全控制过来的请求，当一个确定的HTTP请求过来时我们可以引导到其到某个控制器上对应的action方法。

### （3）长的URL和扩展都将会被淘汰

我们能够知道当我们访问一个页面时有些直接以扩展名为.aspx或者html呈现，现在我们有了路由，完全不需要扩展名并且会缩短URL的长度。

好了我们讲完路由的好处，我们就要看看在ASP.NET Core中的路由到底是怎样的呢。

## ASP.NET Core MVC/WebAPi路由方式

### 创建默认路由

这个就不用多讲了，当我们创建ASP.NET Core应用程序时在Startup中就会创建默认的路由如下：

public class Startup

{

public void ConfigureServices(IServiceCollection services)

{

services.AddMvc();

}

public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory)

{

app.UseMvc(routes =>

{

routes.MapRoute(

name: "default",

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

});

}

}

这是默认给我们创建的，看着有点不太舒服，我还是习惯将其修改如下等同的路由配置。

app.UseMvc(routes =>

{

routes.MapRoute(

name: "default",

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

defaults: new { controller = "Home", action = "Index" });

});

### 扩展默认路由

上述只是我们创建的默认路由，当我们需要额外进行自定义扩展对于特定需求，所以我们如上类似再来扩展一个路由。

app.UseMvc(routes =>

{  
　　　　　　　　　//扩展路由

routes.MapRoute(

name: "about-route",

template: "about",

defaults: new { controller = "Home", action = "About" });

routes.MapRoute(

name: "default",

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

defaults: new { controller = "Home", action = "Index" });

});

### 使用路由特性

我们也可以配置路由特性针对控制器和Action方法，如下：

[Route("[controller]")]

public class HomeController : Controller

{

private IBlogRepository \_blogRepository;

public HomeController(IBlogRepository blogRepository)

{

\_blogRepository = blogRepository;

}

[Route("[action]")]

public IActionResult Index()

{

return View();

}

[Route("Index")]

[HttpPost]

public IActionResult Index([FromBody] BlogViewModel model)

{

return View();

}

}

上述[Route("[controller]")]和[Route("[action]")])都是引用的声明的控制器名称和方法名称，即Home控制器和Index方法，所以上述两个action方法都是匹配的/Home/Index只是请求方式不同而已。在官网中有使用路由中间件的方式，有需要的可以参考一下：【https://docs.microsoft.com/en-us/aspnet/core/fundamentals/routing】

### RESTful风格路由

对于RESTful风格路由我们更多的是用在请求ASP.NET Core WebAPi接口上，如下：

[Route("api/[controller]")]

public class Blog : Controller

{

[HttpGet] // api/blog

public IEnumerable<string> Get()

{

return new string[] { "hello", "world!" };

}

[HttpPost("Post")] // api/blog/post

public IActionResult PostBlog([FromBody] BlogViewModel model)

{

return View();

}

}

那么问题来了，如果是有参数我们该进行如何传递呢，当然有解决方案，如下：

[HttpPost("{id}")]

public void PostUpdate(int id, [FromBody] string value)

{

}

此时前台请求如下：

$("#btn").on("click", function () {

$.ajax({

url: "/api/blog/2",

type: "post",

contentType: "application/json;charset=utf-8",

dataType: "json",

data: JSON.stringify("Jeffcky"),

success: function (data) {

}

});

});

### 使用约束

对于参数类型是否非法，我们通过对参数进行约束来达到我们得目的，如下：

[HttpPost("{id:int}")]

public void PostUpdate(int id, [FromBody] string value)

{

}

或者对参数约束为guid，如下：

[HttpPost("{id:guid}")]

public void PostUpdate(string id, [FromBody] string value)

{

}

当然还有其他类型参数约束就不一一列举了。

## 总结

本节我们讲解了ASP.NET Core中的路由使用，比较基础性的东西，更深入的东西参看官网资料，如若有不太理解和有所疑惑的地方请在评论中指出看到会马上回复。

http://blog.csdn.net/ceg6648/article/details/54647048