

# 注意事项

## Newtonsoft.Json时间格式化

```
public class Student
{
    public string StuId { get; set; }
    public string StuName { get; set; }
    public DateTime date { get; set; }
}
```

以上Student类型中date序列化出来的时间是默认是ISO8601格式



解决方式

- 自定义一个时间格式化

```
IsoDateTimeConverter timeConverter = new IsoDateTimeConverter();
//这里使用自定义日期格式，如果不使用的话，默认是ISO8601格式
timeConverter.DateTimeFormat = "yyyy-MM-dd HH:mm:ss";
string json = JsonConvert.SerializeObject(student, timeConverter);
```

## WebApi中的时间格式化

WebApi或mvc中DateTime的序列化也是ISO8601格式

解决方式

- nuget引用Microsoft.AspNetCore.Mvc.NewtonsoftJson包

```
Install-Package Microsoft.AspNetCore.Mvc.NewtonsoftJson -Version 6.0.0
```

- 在注册控制器是指定时间格式

```
builder.Services.AddControllers().AddNewtonsoftJson(options=>
{
    options.SerializerSettings.DateFormatString = "yyyy-MM-dd HH:mm:ss";
});
```

```
WeatherForec...ontroller.cs Program.cs x
WebApi4Autofac
31
32 builder.Services.AddControllers().AddNewtonsoftJson(options=>
33 {
34     options.SerializerSettings.DateFormatString = "yyyy-MM-dd HH:mm:ss";
35 });
36
37 var app = builder.Build();
38
39 // Configure the HTTP request pipeline.
40
41 app.UseAuthorization();
42
43 //app.Use((context, next) =>
44 //{
45 //    context.Request.EnableBuffering();
46 //    return next();
47 //});
48
49 app.MapControllers();
50
51 app.Run();
52
```

## WebApi通过数据流获取请求的原始报文问题

在.net core比较高版本中以下用法会报错,旧版本可以使用

```
Stream body = context.HttpContext.Request.Body;
body.Position = 0; //从头开始读
using (StreamReader reader = new StreamReader(body))
{
    string json = reader.ReadToEnd();
}
```

```
7 public class MyActionFilter: ActionFilterAttribute, IActionFilter
8 {
9     private readonly ILogger<MyActionFilter> _logger;
10     0 个引用 | 0 项更改 | 0 名作者, 0 项更改
11     public MyActionFilter(ILogger<MyActionFilter> logger)
12     {
13         this._logger = logger;
14     }
15     0 个引用 | 0 项更改 | 0 名作者, 0 项更改
16     public override void OnActionExecuting(ActionExecutingContext context)
17     {
18         Stream body = context.HttpContext.Request.Body;
19         body.Position = 0; //从头开始读
20         using (StreamReader reader = new StreamReader(body))
21         {
22             string json = reader.ReadToEnd();
23         }
24
25         base.OnActionExecuting(context)
26         //Stream body = context.HttpContext.Request.Body;
27         //body.Position = 0;
28         //using (StreamReader reader = new StreamReader(body))
29         {
30             string json = reader.ReadToEnd();
31         }
32     }
33 }
```

用户未处理的异常  
System.NotSupportedException: "Specified method is not supported."  
查看详细信息 | 复制详细信息 | 启动 Live Share 会话...  
异常设置

解决方法:

- 中间件注册中加上以下代码即可,要在终结点中间件之前添加

```
app.Use((context, next) =>
{
    context.Request.EnableBuffering();
    return next();
});
```

```
WeatherForecastController.cs | MyActionFilter.cs | Program.cs
WebApi4Autofac
34 | });
35 | ;
36 |
37 |
38 | var app = builder.Build();
39 |
40 | // Configure the HTTP request pipeline.
41 |
42 | app.UseAuthorization();
43 |
44 | app.Use((context, next) =>
45 | {
46 |     context.Request.EnableBuffering();
47 |     return next();
48 | });
49 |
50 | app.MapControllers();
51 |
52 | app.Run();
53 |
```

- .net core3.0之后中读取请求流使用同步方法会报错，要使用异步方法读取数据流

```
WeatherForecastController.cs | MyActionFilter.cs | Program.cs
WebApi4Autofac | WebApi4Autofac.Filters.MyActionFilter
6 | {
7 |     4 个引用 | 0 项更改 | 0 名作者, 0 项更改
8 |     public class MyActionFilter: ActionFilterAttribute, IActionFilter
9 |     {
10 |         private readonly ILogger<MyActionFilter> _logger;
11 |         0 个引用 | 0 项更改 | 0 名作者, 0 项更改
12 |         public MyActionFilter(ILogger<MyActionFilter> logger)
13 |         {
14 |             this._logger = logger;
15 |         }
16 |
17 |         0 个引用 | 0 项更改 | 0 名作者, 0 项更改
18 |         public override void OnActionExecuting(ActionExecutingContext context)
19 |         {
20 |             Stream body = context.HttpContext.Request.Body;
21 |             body.Position = 0; //从头开始读
22 |             using (StreamReader reader = new StreamReader(body))
23 |             {
24 |                 string json = reader.ReadToEnd();
25 |             }
26 |
27 |             base.OnActionExecuting(context);
28 |             //Stream body = context.HttpContext.Re
29 |             //body.Position = 0;
30 |             //using (StreamReader reader = new Str
31 |             //{
32 |                 string v = reader.ReadToEndAsync().Result;
33 |             }
34 |             ///IDictionary<string, object?> actionArguments = context.ActionArguments;
```

用户未处理的异常

**System.InvalidOperationException:** Synchronous operations are disallowed. Call ReadAsync or set AllowSynchronousIO to true instead.

查看详细信息 | 复制详细信息 | 启动 Live Share 会话... | 异常设置

正确用法

```
Stream body = context.HttpContext.Request.Body;
body.Position = 0; //从头开始读
using (StreamReader reader = new StreamReader(body))
{
    string json = reader.ReadToEndAsync().Result;
}
```

```
12         this._logger = logger;
13     }
14
15     0 个引用 | 0 项更改 | 0 名作者, 0 项更改
16     public override void OnActionExecuting(ActionExecutingContext context)
17     {
18         Stream body = context.HttpContext.Request.Body;
19         body.Position = 0; // 从头开始读
20         using (StreamReader reader = new StreamReader(body))
21         {
22             string json = reader.ReadToEndAsync().Result;
23             // 已用时间 <= 1ms  json  查看 - "{ \"Stuid\": \"1\", \"StuName\": \"The_zhw\", \"date\": \"2021-11-29\" }"
24         }
25     }
26     base.OnActionExecuting(context);
27 }
```

## .net6与.net5下WebApi模板的差异

源码位置 aspnetcore-6.0.0\src\DefaultBuilder\src\WebApplication.cs

.net6下webapi的模板

```
13 // 调用ConfigureContainer注册服务
14
15 builder.Host.ConfigureContainer<ContainerBuilder>((HostBuilderContext, ContainerBuilder) => { }); // 调用ConfigureContainer注册
25
26 // Add services to the container.
27
28 builder.Services.AddControllers(configure => { }).AddNewtonsoftJson(options =>
33 {
34     options.SerializerSettings.DateFormatString = "yyyy-MM-dd HH:mm:ss";
35 });
36
37
38 var app = builder.Build();
39
40 // The web application used to configure the HTTP pipeline, and routes.
41 app.UseAuthorization();
42
43
44
45
46 app.MapControllers();
47
48 app.Run();
49
```

.net6下使用WebApplication进行中间件注册,实际上就是对ApplicationBuilder的进一步封装

.net5下的webapi的模板

```
25 public void ConfigureServices(IServiceCollection services)
26 {
27
28     services.AddControllers();
29 }
30
31 // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
32 public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
33 {
34     if (env.IsDevelopment())
35     {
36         app.UseDeveloperExceptionPage();
37     }
38
39     app.UseRouting();
40
41     app.UseAuthorization();
42
43     app.UseEndpoints(endpoints =>
44     {
45         endpoints.MapControllers();
46     });
47 }
48
49
50
```

对比以上两个模板可以看到, .net6中 `app.UseAuthorization()` 中间件没有在 `app.UseRouting()` 与 `app.UseEndpoints()` 之间注册, 而中间件是必须要在两者之间进行注册的

```
builder.Services.AddControllers(configure => { }).AddNewtonsoftJson(options =>
{
    options.SerializerSettings.DateFormatString = "yyyy-MM-dd HH:mm:ss";
});

var app = builder.Build();

// Configure the HTTP request pipeline.

app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    template: "{controller}/{action}/{id}",
    defaults: new { id = 1 }
);

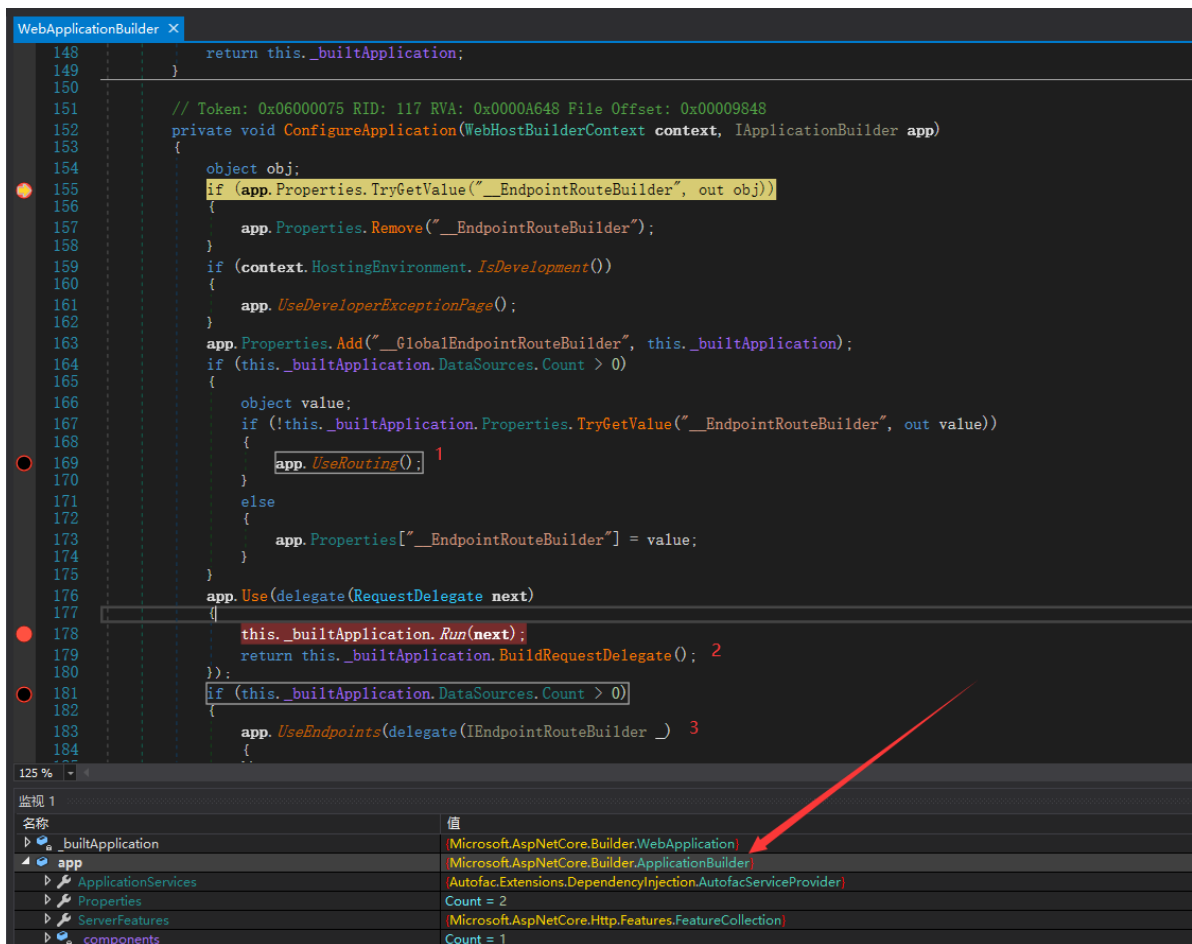
app.Run();
```

添加 Microsoft.AspNetCore.Authorization.AuthorizationMiddleware 到指定的 IApplicationBuilder，以启用授权功能。  
When authorizing a resource that is routed using endpoint routing, this call must appear between the calls to app.UseRouting() and app.UseEndpoints(...) for the middleware to function correctly.

.net6调用源码，当WebApplication调用Run()方法后会执行以下代码

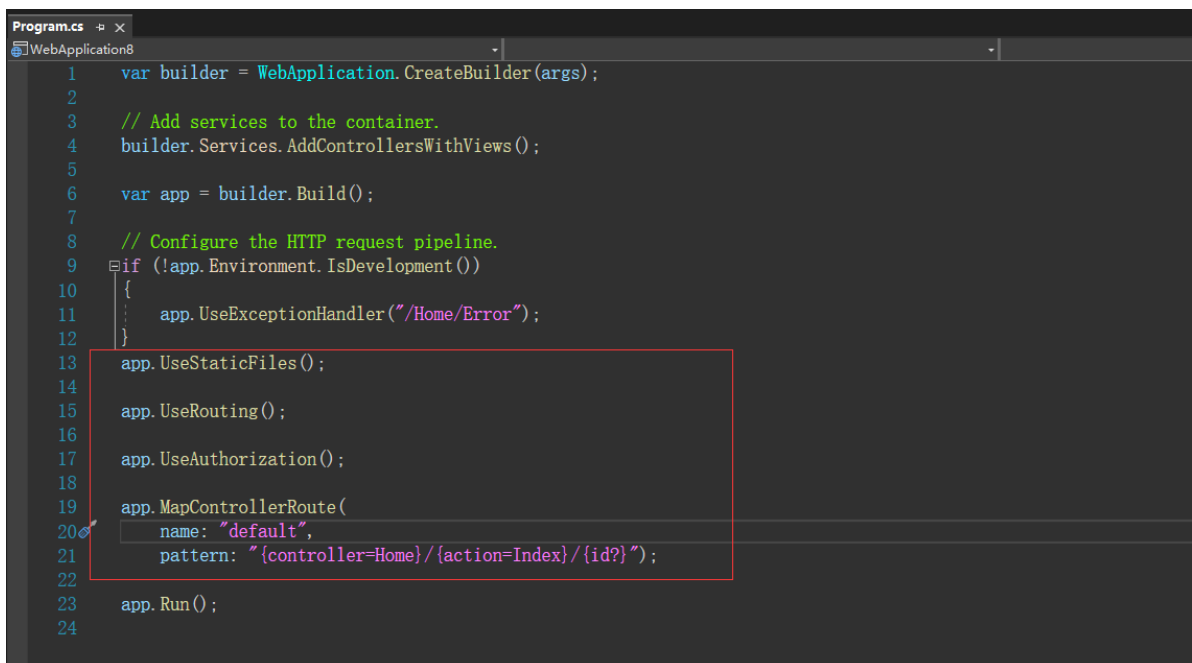
Microsoft.AspNetCore.Builder.WebApplicationBuilder中的ConfigureApplication方法

```
WebApplicationBuilder X
{
    153
    154     object obj;
    155     if (app.Properties.TryGetValue("__EndpointRouteBuilder", out obj))
    156     {
    157         app.Properties.Remove("__EndpointRouteBuilder");
    158     }
    159     if (context.HostingEnvironment.IsDevelopment())
    160     {
    161         app.UseDeveloperExceptionPage();
    162     }
    163     app.Properties.Add("__GlobalEndpointRouteBuilder", this._builtApplication);
    164     if (this._builtApplication.DataSources.Count > 0)
    165     {
    166         object value;
    167         if (!this._builtApplication.Properties.TryGetValue("__EndpointRouteBuilder", out value))
    168         {
    169             app.UseRouting(); 判断如果没有注册过路由由中间件则进行注册
    170         }
    171         else
    172         {
    173             app.Properties["__EndpointRouteBuilder"] = value;
    174         }
    175     }
    176     app.Use(delegate(RequestDelegate next)
    177     {
    178         this._builtApplication.Run(next); 这一步是将WebApplication中注册过的中间件加入到Application实例的中间件管道中
    179         return this._builtApplication.BuildRequestDelegate(); 这一步是执行WebApplication中注册的中间件
    180     });
    181     if (this._builtApplication.DataSources.Count > 0)
    182     {
    183         app.UseEndpoints(delegate(IEndpointRouteBuilder _)
    184         {
    185             });
    186         foreach (KeyValuePair<string, object> keyValuePair in this._builtApplication.Properties)
    187         {
    188             app.Properties[keyValuePair.Key] = keyValuePair.Value;
    189         }
    190         app.Properties.Remove("__GlobalEndpointRouteBuilder");
    191         if (obj != null)
    192         {
    193             app.Properties["__EndpointRouteBuilder"] = obj;
    194         }
    195     }
    196 }
    197
    198 // Token: 0x0400001A RID: 26
    199 private const string EndpointRouteBuilderKey = "EndpointRouteBuilder";
```



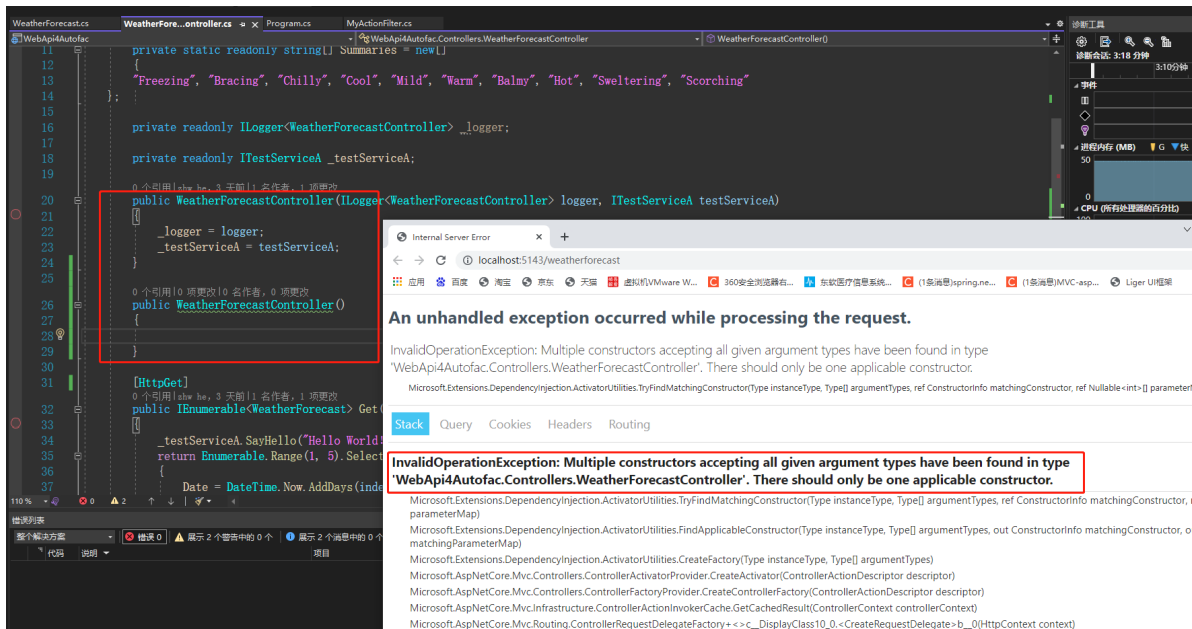
当请求进来时会先依次执行外层的中间件，再执行WebApplication注册的中间件，**相当于WebApi中的所有由WebApplication注册的中间件都包含在 app.UseRouting() 与 app.UseEndpoints() 之间执行**

再看.net6下创建的mvc模板,因为已经在在WebApplication中注册过路由中间件,所以在Microsoft.AspNetCore.Builder.WebApplicationBuilder中的ConfigureApplication方法就不再进行注册,所以请求进来时会按照在WebApplication中的中间件顺序执行



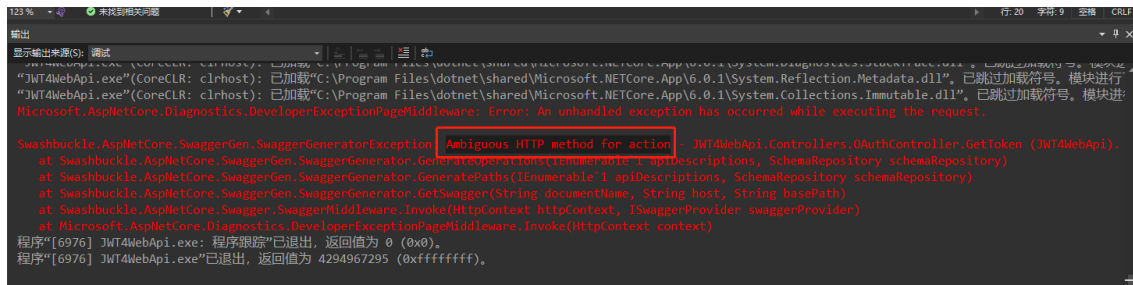
## 多个构造函数问题

进行构造函数注入时,只能有一个构造函数,否则会报错



## 使用Swagger注意事项

- 当Action不标记Http谓词时启动会报错



## 当Swagger与异常中间件一起使用时注意事项

- 异常中间件不能使用Http谓词，否则异常发生时不会进入到自定义的异常API,详见官网[处理ASP.NET Core Web API 中的错误 | Microsoft Docs](#)这样就与Swagger必须使用http谓词冲突

### ⚠ 警告

仅当应用程序在开发环境中运行时才启用开发人员异常页。当应用在生产环境中运行时，请勿公开详细的异常信息。有关配置环境的详细信息，请参阅在 ASP.NET Core 中使用多个环境。

不要使用 HTTP 方法属性（如 `HttpGet`）标记错误处理程序操作方法。显式谓词可阻止某些请求访问操作方法。如果未经身份验证的用户应看到错误，则允许匿名访问此方法。

## 异常处理程序

在非开发环境中，可使用[异常处理中间件](#)来生成错误负载：

1. 在 `Startup.Configure` 中，调用 `UseExceptionHandler` 以使用中间件：

```
C#  
  
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)  
{  
    if (env.IsDevelopment())  
    {  
        app.UseDeveloperExceptionPage();  
    }  
    else  
    {  
        app.UseExceptionHandler("/error");  
    }  
  
    app.UseHttpsRedirection();  
    app.UseRouting();  
    app.UseAuthorization();  
    app.UseEndpoints(endpoints =>  
    {  
        endpoints.MapControllers();  
    });  
}
```

- 解决方式,异常中间件指向的API不开放在Swagger上，在指定Action上标记 `[ApiExplorerSettings(IgnoreApi = true)]`

```
using Microsoft.AspNetCore.Http;  
using Microsoft.AspNetCore.Mvc;  
namespace JWT4WebApi.Controllers  
{  
    [Route("error")]  
    [ApiExplorerSettings(IgnoreApi = true)]  
    public IActionResult Error()  
    {  
        return Problem();  
    }  
}  
  
{  
    app.UseSwagger();  
    app.UseSwaggerUI();  
    app.UseExceptionHandler("/error");  
    // Configure the HTTP request pipeline.  
    app.UseAuthentication(); // 启用鉴权中间件  
    app.UseAuthorization();  
    app.MapControllers();  
    app.Run();  
}
```