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

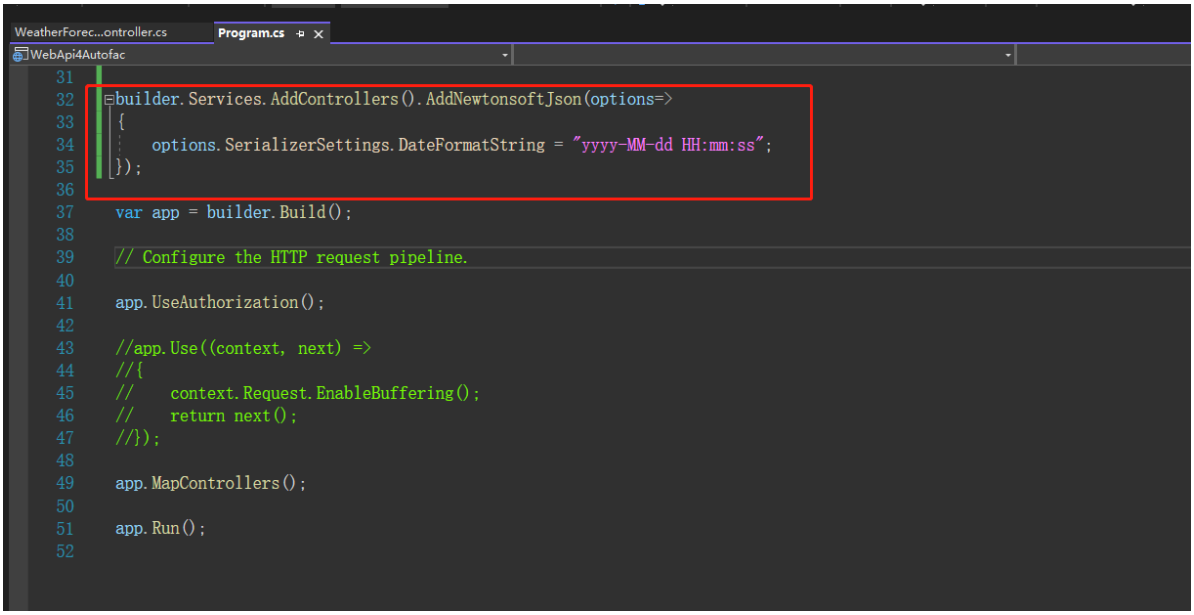
解决方式

- 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

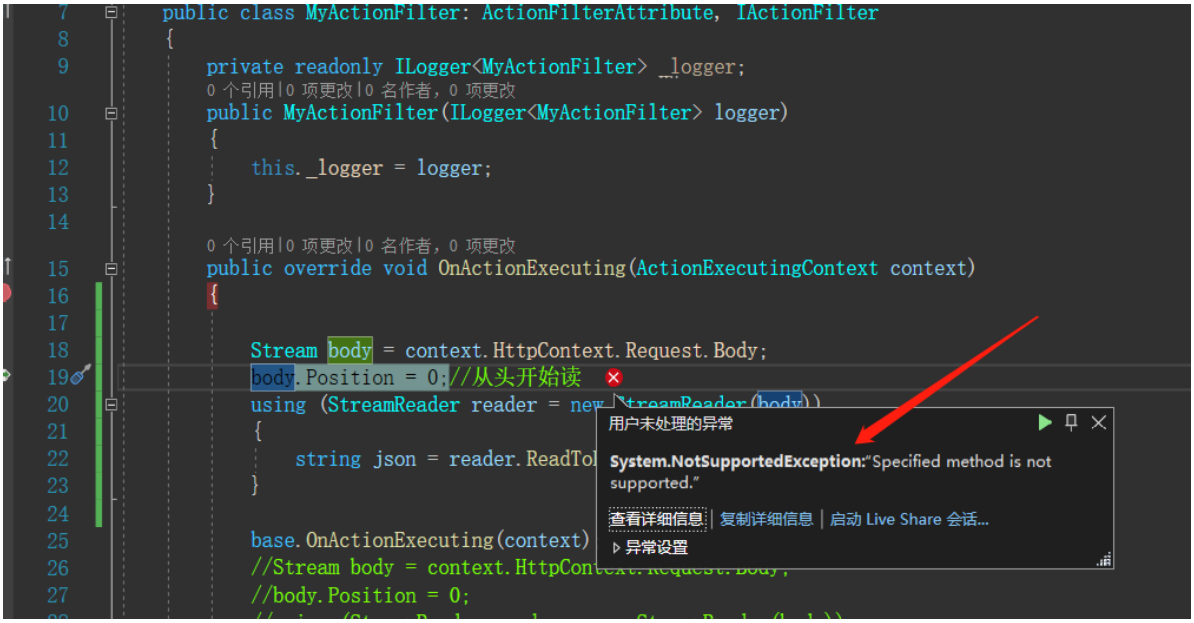
  ```
  builder.Services.AddControllers().AddNewtonsoftJson(options=>
  {
      options.SerializerSettings.DateFormatString = "yyyy-MM-dd HH:mm:ss";
  });
  ```



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

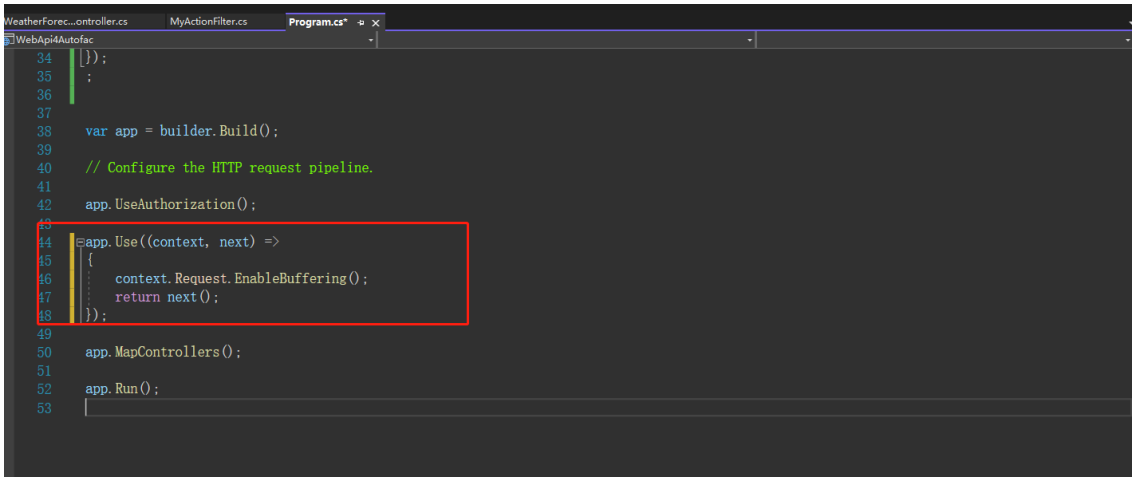
```
Stream body = context.HttpContext.Request.Body;
body.Position = 0;//从头开始读
using (StreamReader reader = new StreamReader(body))
{
    string json = reader.ReadToEnd();
}
```



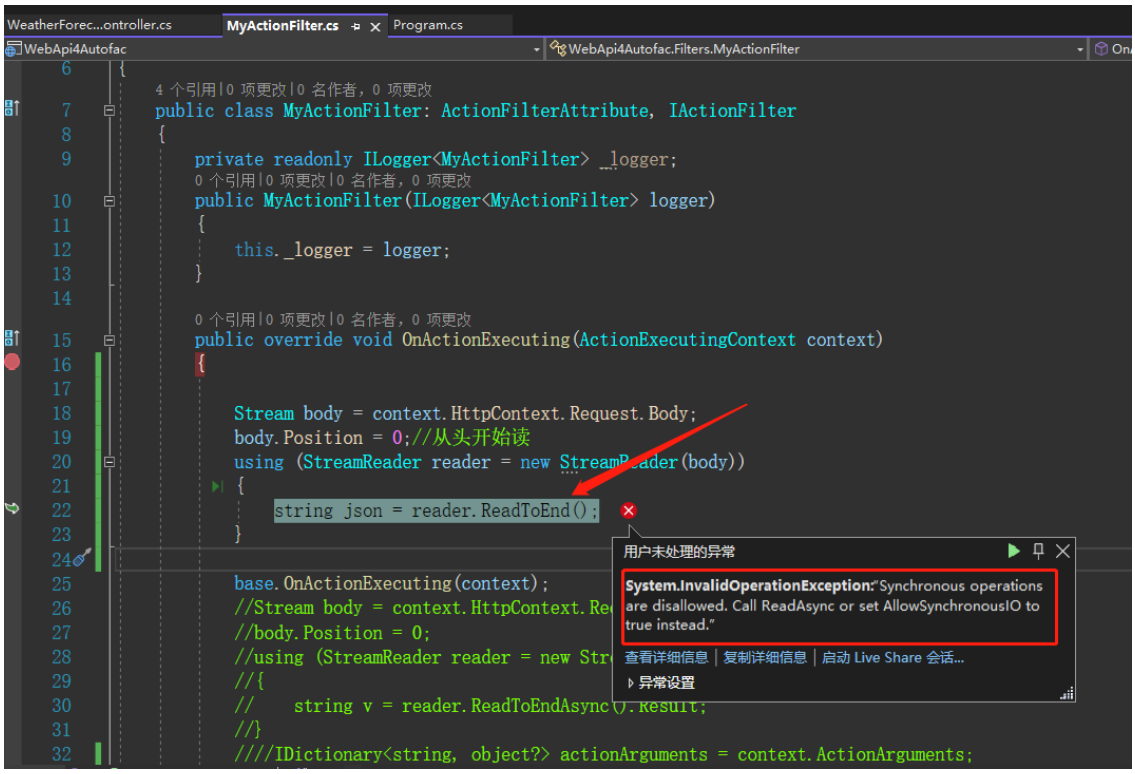
解决方法:

- 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();
});
```



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



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;
}
```

```
            this._logger = logger;
        }

        0 个引用|0 项更改|0 名作者，0 项更改
        public override void OnActionExecuting(ActionExecutingContext context)
        {

            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;
            }
```

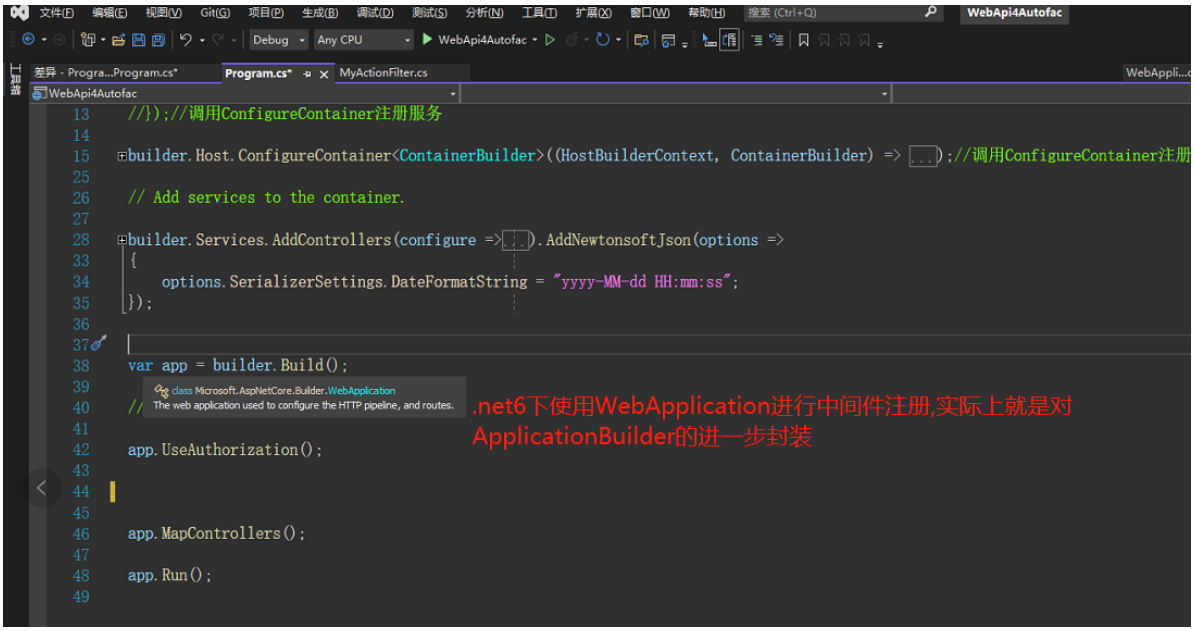
已用时间 <= 1ms

json 查看 "{\"StuId\":\"1\",\"StuName\":\"he_zhw\",\"date\":\"2021-11-29\"}"

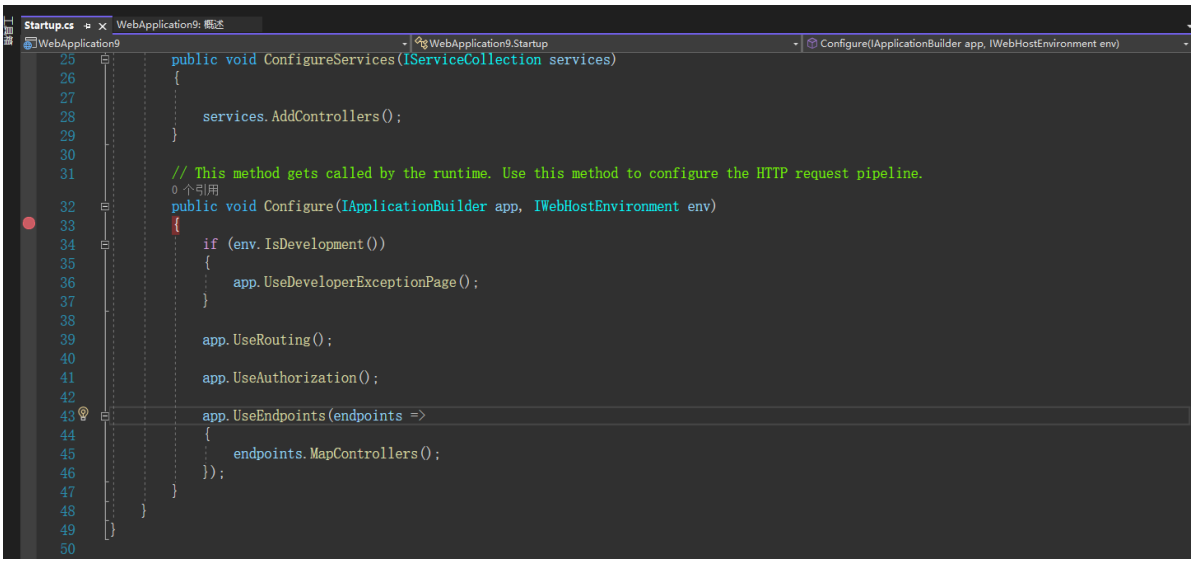
.net6与.net5下WebApi模板的差异

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

.net6下webapi的模板



.net5下的webapi的模板



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这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.MapControlle

app.Run();
```

```
(扩展) IApplicationBuilder IApplicationBuilder.UseAuthorization()
Adds the Microsoft.AspNetCore.Authorization.AuthorizationMiddleware to the specified IApplicationBuilder, which enables authorization capabilities.
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.
返回结果:
A reference to app after the operation has completed.
```

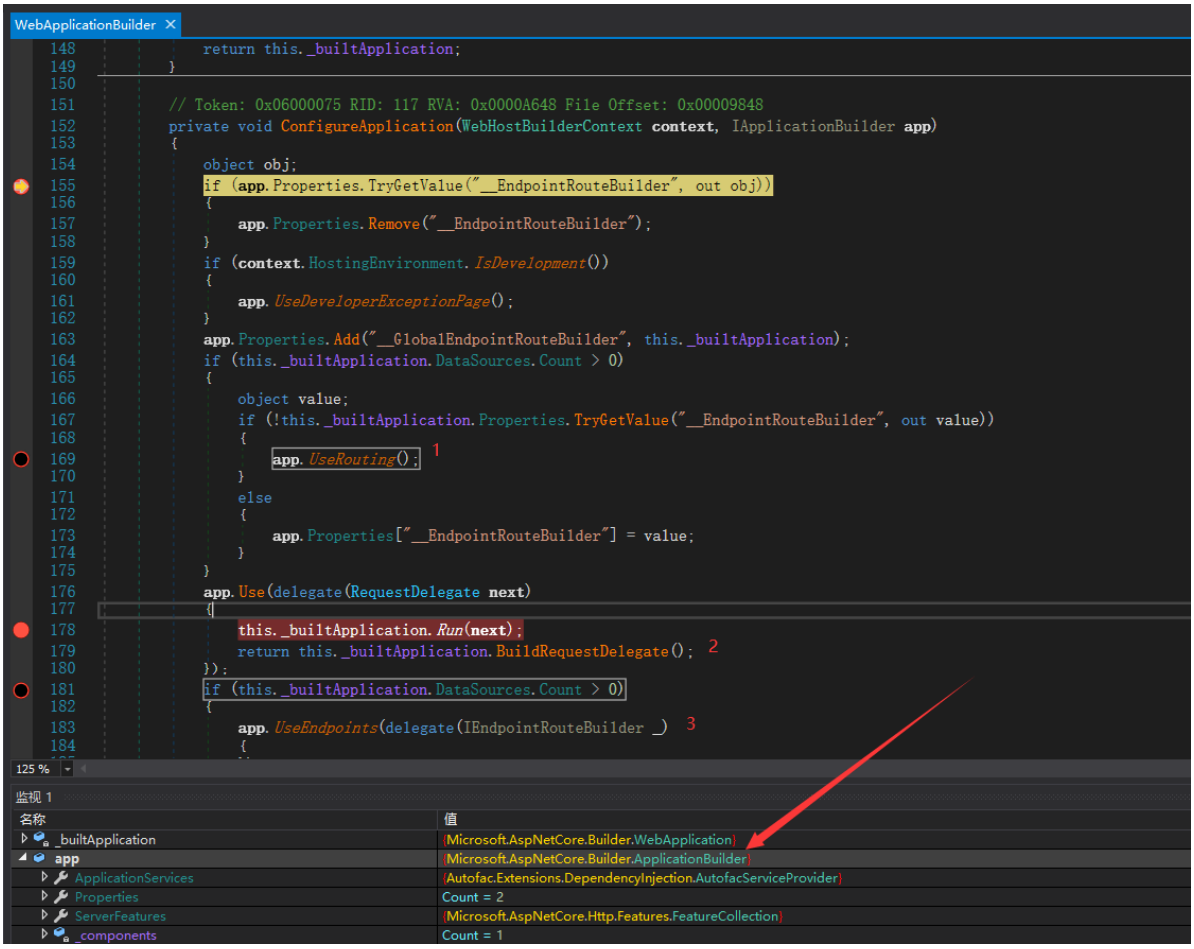
.net6调用源码，当WebApplication调用Run()方法后会执行以下代码

Microsoft.AspNetCore.Builder.WebApplicationBuilder中的ConfigureApplication方法

WebApplicationBuilder

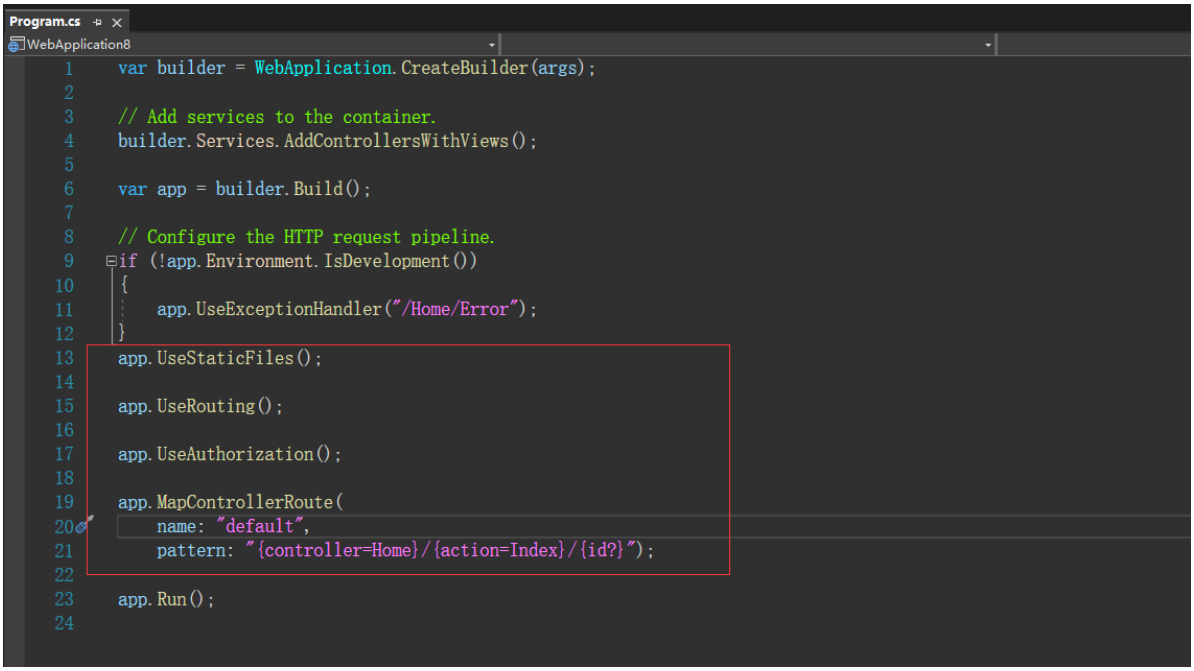
```
        {
            object obj;
            if (app.Properties.TryGetValue("__EndpointRouteBuilder", out obj))
            {
                app.Properties.Remove("__EndpointRouteBuilder");
            }
            if (context.HostingEnvironment.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            app.Properties.Add("__GlobalEndpointRouteBuilder", this._builtApplication);
            if (this._builtApplication.DataSources.Count > 0)
            {
                object value;
                if (!this._builtApplication.Properties.TryGetValue("__EndpointRouteBuilder", out value))
                {
                    app.UseRouting(); 判断如果没有注册过路由中间件则进行注册
                }
                else
                {
                    app.Properties["__EndpointRouteBuilder"] = value;
                }
            }
            app.Use(delegate(RequestDelegate next)
            {
                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
                return this._builtApplication.BuildRequestDelegate(); 这一步是执行WebApplication中注册的中间件
            });
            if (this._builtApplication.DataSources.Count > 0)
            {
                app.UseEndpoints(delegate(IEndpointRouteBuilder _)
                {
                });
            }
            foreach (KeyValuePair<string, object> keyValuePair in this._builtApplication.Properties)
            {
                app.Properties[keyValuePair.Key] = keyValuePair.Value;
            }
            app.Properties.Remove("__GlobalEndpointRouteBuilder");
            if (obj != null)
            {
                app.Properties["__EndpointRouteBuilder"] = obj;
            }
        }

        // Token: 0x0400001A RID: 26
        private const string EndpointRouteBuilderKey = "__EndpointRouteBuilder";
```



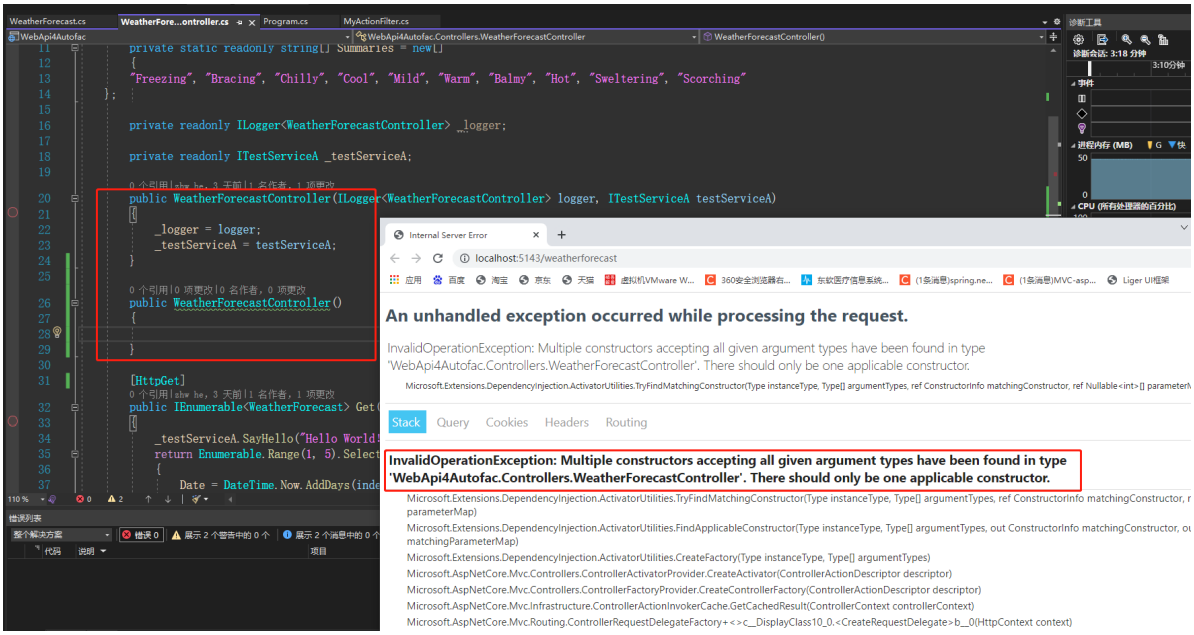
当请求进来时会先依次执行外层的中间件，再执行WebApplication注册的中间件，**相当于WebApi中的所有由WebApplication注册的中间件都包含在`app.UseRouting()`与`app.UseEndpoints()`之间执行**

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



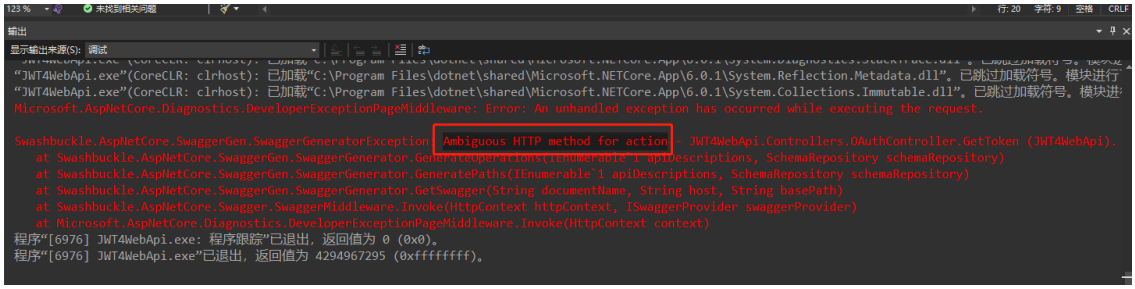
多个构造函数问题

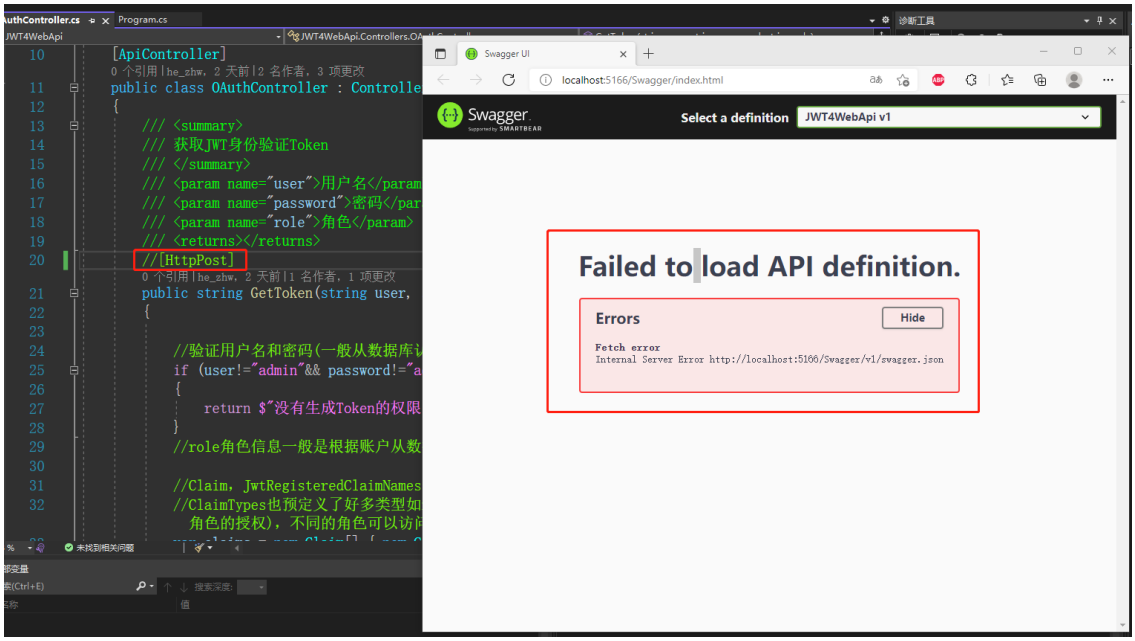
进行构造函数注入时,只能有一个构造函数,否则会报错



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)]