[**net core 3.1引用 Swagger**](https://zhuanlan.zhihu.com/p/451550080)

[对待丶](https://www.zhihu.com/people/civilization-71)

[**对待丶**](https://www.zhihu.com/people/civilization-71)

小菜鸟工程师

2 人赞同了该文章

**前言**

目前的项目基本都是前后端分离，API 功能的演变是不可避免的，但维护 API 文档的头痛不是必须的。Swagger 工具将繁重的工作从生成和维护您的 API 文档中解脱出来，确保您的文档随着 API 的发展而保持最新。

一、Swagger是什么？

Swagger 是一个规范且完整的框架，用于生成、描述、调用和可视化 RESTful 风格的 Web 服务。

Swagger 的目标是对 REST API 定义一个标准且和语言无关的接口，可以让人和计算机拥有无须访问源码、文档或网络流量监测就可以发现和理解服务的能力。当通过 Swagger 进行正确定义，用户可以理解远程服务并使用最少实现逻辑与远程服务进行交互。与为底层编程所实现的接口类似，Swagger 消除了调用服务时可能会有的猜测。

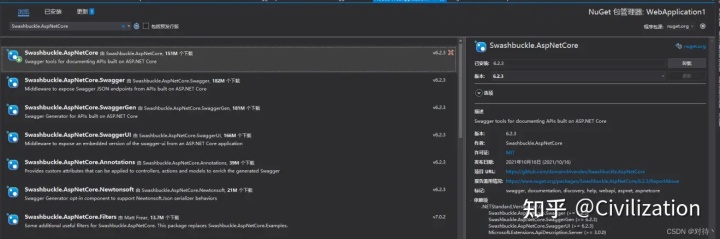
swagger官网：[https://swagger.io](https://link.zhihu.com/?target=https%3A//swagger.io)

**二、使用步骤**

**本次演示源码上传至gitee**  
地址：[https://gitee.com/huang945617/netcore3—using-swagger](https://link.zhihu.com/?target=https%3A//gitee.com/huang945617/netcore3%25E2%2580%2594using-swagger)

**1.引入库**

通过“管理NuGet程序包”下载 Swashbuckle.AspNetCore包：



2.代码

编辑 Startup.cs：

1.将swagger生成器添加到ConfigureServices方法中：

**public** **void** ConfigureServices(IServiceCollection services)

{

services.AddControllers();

**#region Swagger服务**

services.AddSwaggerGen(o =>

{

o.SwaggerDoc("v1", **new** OpenApiInfo

{

Title = "WebApi",

Version = "v1",

Contact = **new** OpenApiContact

{

Name = "对待丶",

Email = **string**.Empty,

Url = **new** Uri("http://127.0.0.1:5000/swagger/index.html")

},

Description = "API描述",

License = **new** OpenApiLicense

{

Name = "对待丶",

Url = **new** Uri("http://127.0.0.1:5000/swagger/index.html")

}

});

});

**#endregion**

}

2.将swagger中间件添加到Configure方法中：

**#region Swagger 中间件**

app.UseSwagger();

app.UseSwaggerUI(o =>

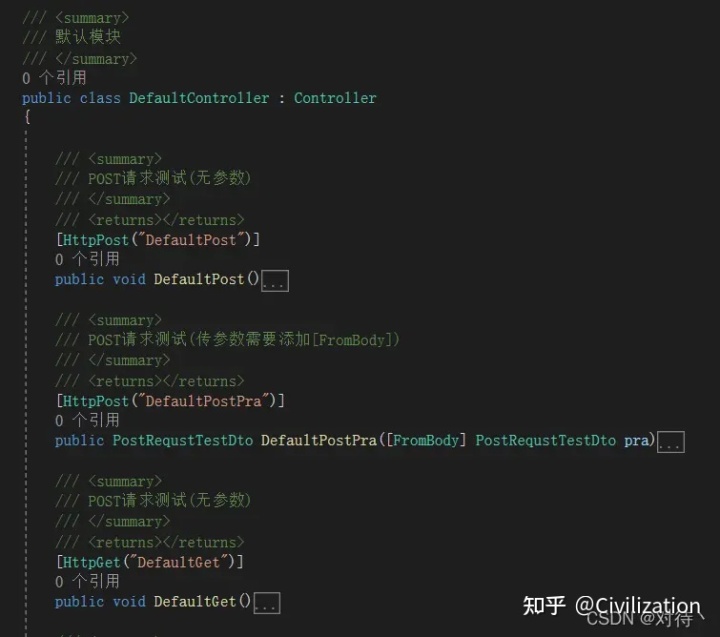
{

o.SwaggerEndpoint("/swagger/v1/swagger.json", "WebApi");

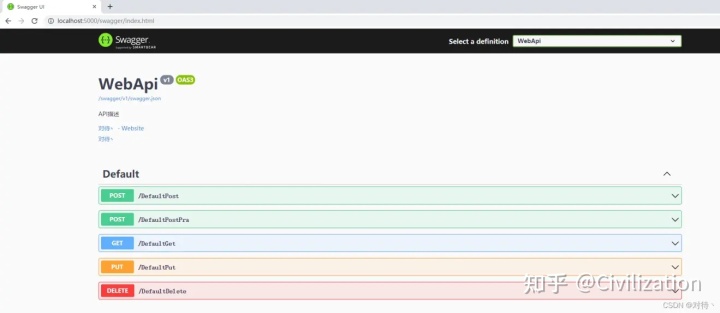
});

**#endregion**

**添加一个测试控制器：**



添加后我们启动项目看下效果：http://localhost:/swagger/index.html

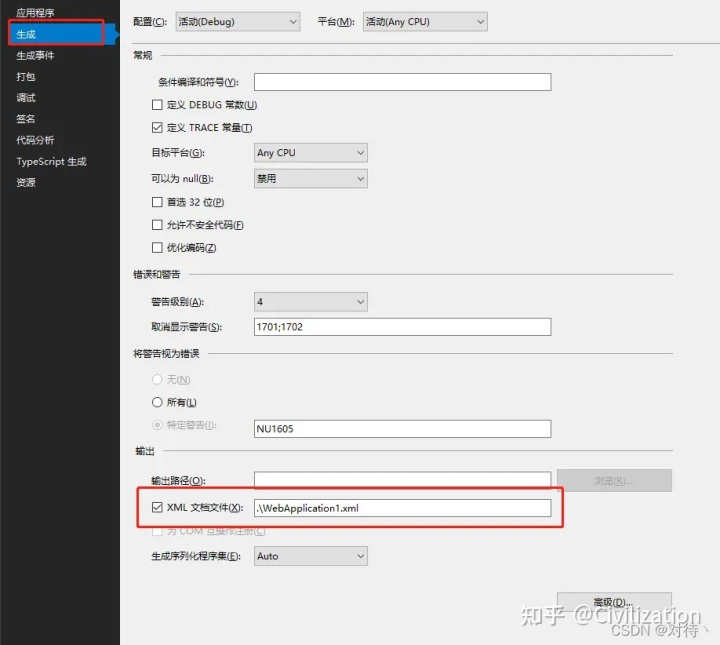


启动后虽然可以正常的浏览，但是我们对方法/参数的注释却无法展示出来，这样明显无法达到我们想要的效果。

接下来我们可通过启用XML来实现注释

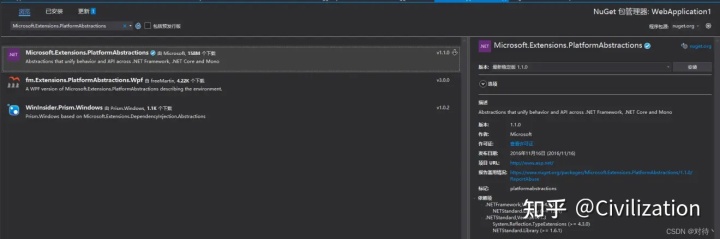
方法一：

“解决方案资源管理器”选择项目右键单击->“属性”，设置XML文档文件的路径



方法二：

通过“管理NuGet程序包”下载 Microsoft.Extensions.PlatformAbstractions包



使用以上方法的其中一种后，更改ConfigureServices方法中的AddSwaggerGen

**public** **void** ConfigureServices(IServiceCollection services)

{

services.AddControllers();

**#region Swagger服务**

services.AddSwaggerGen(o =>

{

o.SwaggerDoc("v1", **new** OpenApiInfo

{

Title = "WebApi",

Version = "v1",

Contact = **new** OpenApiContact

{

Name = "对待丶",

Email = **string**.Empty,

Url = **new** Uri("http://127.0.0.1:5000/swagger/index.html")

},

Description = "API描述",

License = **new** OpenApiLicense

{

Name = "对待丶",

Url = **new** Uri("http://127.0.0.1:5000/swagger/index.html")

}

});

**var** assemblyName = Assembly.GetEntryAssembly().GetName().Name;

*//控制器 方法/参数注释*

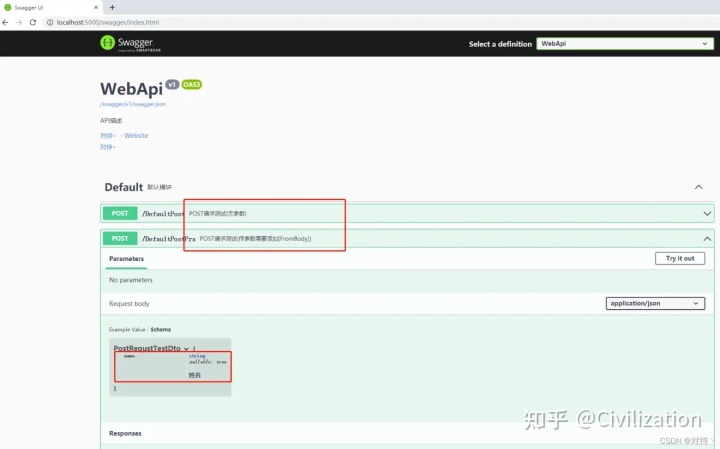
options.IncludeXmlComments(Path.Combine(AppContext.BaseDirectory, $"{assemblyName}.xml"), **true**);

});

**#endregion**

}

更改后，在启动项目浏览看下效果



这样我们添加的注释就可以全部显示出来了。

3.进阶用法

当项目越来越大时，功能模块也越来越多，全部展示在一个页面，不方便对接人员的查找及阅读，如果我们按照我们对应的功能模块来分组是否会更好呢？接下来简单的修改下

1.新建一个类，里面放的是分组对应模块的信息

**public** **static** **class** **SwaggerApi**

{

*/// <summary>*

*/// Swagger分组信息，将进行遍历使用*

*/// </summary>*

**public** **static** **readonly** List<SwaggerApiInfo> ApiInfos = **new** List<SwaggerApiInfo>()

{

**new** SwaggerApiInfo

{

UrlPrefix = SwaggerGroupingModel.GROUPNAME\_TEST,

Name = "测试模块",

OpenApiInfo = **new** OpenApiInfo

{

Version = "v1",

Title = "Api-Test",

Description = "Api-测试模块"

}

},

**new** SwaggerApiInfo

{

UrlPrefix = SwaggerGroupingModel.GROUPNAME\_COMM,

Name = "公共模块",

OpenApiInfo = **new** OpenApiInfo

{

Version = "v1",

Title = "WebApi-Comm",

Description = "WebApi-公共模块"

}

}

};

*/// <summary>*

*/// 分组信息*

*/// </summary>*

pullic **class** **SwaggerApiInfo**

{

*/// <summary>*

*/// URL前缀*

*/// </summary>*

**public** **string** UrlPrefix { **get**; **set**; }

*/// <summary>*

*/// 名称*

*/// </summary>*

**public** **string** Name { **get**; **set**; }

*/// <summary>*

*/// <see cref="Microsoft.OpenApi.Models.OpenApiInfo"/>*

*/// </summary>*

**public** OpenApiInfo OpenApiInfo { **get**; **set**; }

}

}

*/// <summary>*

*/// Swagerr分组模块*

*/// </summary>*

**public** **static** **class** **SwaggerGroupingModel**

{

*/// <summary>*

*/// 公共数据模块*

*/// </summary>*

**public** **const** **string** GROUPNAME\_TEST = "test";

*/// <summary>*

*/// 公共数据模块*

*/// </summary>*

**public** **const** **string** GROUPNAME\_COMM = "comm";

}

2.新建一个swagger的扩展类Extensions

*/// <summary>*

*/// Swagger 扩展*

*/// </summary>*

**public** **static** **partial** **class** **Extensions**

{

*/// <summary>*

*/// 配置自定义Swagger服务*

*/// </summary>*

*/// <param name="builder">应用程序生成器</param>*

**public** **static** IApplicationBuilder UseDefaultSwagger(**this** IApplicationBuilder builder)

{

builder.UseSwagger();

builder.UseSwaggerUI(options =>

{

options.DocExpansion(DocExpansion.None);

*//通过循环将加载信息*

SwaggerApi.ApiInfos.ForEach(x =>

{

options.SwaggerEndpoint($"/swagger/{x.UrlPrefix}/swagger.json", x.Name);

});

**#region 以下参数可自行设置**

*// 模型的默认扩展深度，设置为 -1 完全隐藏模型*

options.DefaultModelsExpandDepth(-1);

*// API文档仅展开标记*

options.DocExpansion(DocExpansion.List);

*// API前缀设置为空*

options.RoutePrefix = **string**.Empty;

options.DocumentTitle = "WebApi文档";

*//return;*

**#region**

});

**return** builder;

}

*/// <summary>*

*/// 初始化Swagger*

*/// </summary>*

*/// <param name="services"></param>*

**public** **static** **void** InitSwagger(**this** IServiceCollection services)

{

**var** assemblyName = Assembly.GetEntryAssembly().GetName().Name;

*//添加Swagger*

services.AddSwaggerGen(options =>

{

*//通过循环将加载信息*

SwaggerApi.ApiInfos.ForEach(x =>

{

options.SwaggerDoc(x.UrlPrefix, x.OpenApiInfo);

});

*//解决相同类名会报错的问题*

options.CustomSchemaIds(type => type.FullName);

*//控制器注释*

options.IncludeXmlComments(Path.Combine(AppContext.BaseDirectory, $"{assemblyName}.xml"), **true**);

});

}

}

3.修改Startup类，通过直接调用swagger的扩展类Extensions来添加服务、中间件

**public** **void** ConfigureServices(IServiceCollection services)

{

services.AddControllers();

**#region Swagger服务注入**

services.InitSwagger();

**#endregion**

}

**public** **void** Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

**#region Swagger 中间件**

app.UseSwagger();

app.UseDefaultSwagger();

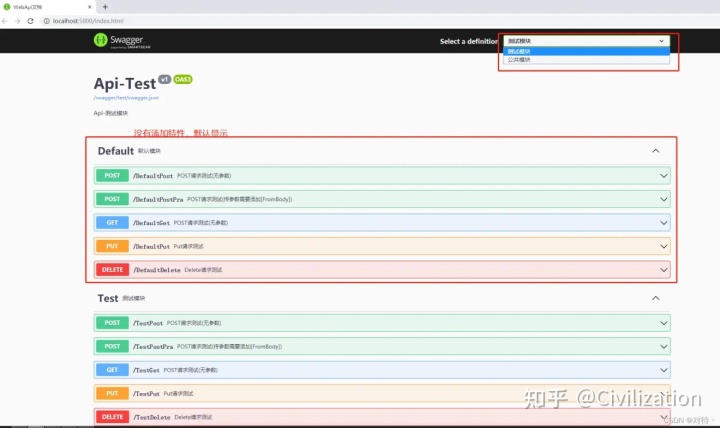
**#endregion**

}

在创建2个测试控制器，在控制器上添加特性[ApiExplorerSettings("")]，没有添加此特性的控制器默认是在每个分组中都显示



启动在浏览下效果，可以看到，控制器都有按照我们的分组展示出来，这样对接人员可以更加方便的来查看对应的接口。



4.接口调试

Swagger除了可以提供接口文档之外，还可以直接进行调试。

选择接口，点击【Try it out】，如果有参数的情况，编辑参数值，点击【Execute】执行，就可以调试了。

