# ASP.NET Core Web Api入门

## 课前准备

1. 学员基础：熟悉C#编程，了解ASP.NET
2. 演示环境：Windows 10 + Visual Studio 2022 + .NET 6
3. Visual Studio 2022：<https://visualstudio.microsoft.com/zh-hans/>
4. .NET 6 : <https://dotnet.microsoft.com/download>
5. 课程源码：<https://github.com/zmrbak/WebApi>
6. 参考资料：<https://docs.microsoft.com/zh-cn/aspnet/core/?view=aspnetcore-6.0>
7. 课程类别：演示、实操教学
8. 相关书籍：不做推荐，都挺好的，但是需要具备相应的知识储备。

## 第一个Web Api项目

curl -X "GET" "https://localhost:7037/WeatherForecast" -H "accept: text/plain"

Windows 10系统中运行前，把单引号换成双引号，删除\，将其写成一行。

Linux中，两种都可以。

## 将模板项目改造成实用的Web Api（1）

[HttpGet]

curl -X 'GET' \

'https://localhost:44319/WeatherForecast' \

-H 'accept: text/plain'

[HttpGet("{index}")]

curl -X 'GET' \

'https://localhost:44319/WeatherForecast/**1**' \

-H 'accept: text/plain'

[HttpPost]

public WeatherForecast PostWeatherForecast(**int index**)

{

return weatherForecasts.ElementAt(index);

}

curl -X 'POST' \

'https://localhost:44319/WeatherFor**ecast?in**dex=1' \

-H 'accept: text/plain' \

-d ''

1. Microsoft.AspNetCore.Mvc.HttpDeleteAttribute
2. Microsoft.AspNetCore.Mvc.HttpGetAttribute
3. Microsoft.AspNetCore.Mvc.HttpHeadAttribute
4. Microsoft.AspNetCore.Mvc.HttpOptionsAttribute
5. Microsoft.AspNetCore.Mvc.HttpPatchAttribute
6. Microsoft.AspNetCore.Mvc.HttpPostAttribute
7. Microsoft.AspNetCore.Mvc.HttpPutAttribute

GET：获取资源，安全，幂等

POST：创建资源，不安全，非幂等

PUT：更新/创建特定资源（替换一个资源），不安全，幂等

**DELETE：删除特定资源，不安全，幂等**

PATCH：对资源进行部分更新（修改某个资源中的一部分，打补丁），不安全，幂等

HEAD：与GET相同，**但不返回Body**，安全，幂等

OPTIONS：获取指定资源所支持的操作，安全，幂等

## 将模板项目改造成实用的Web Api（2）

## 将模板项目改造成实用的Web Api（3）

## 将Web Api发布到IIS服务器

1、安装IIS，使用默认配置

2、ASP.NET Core Runtime Hosting Bundle

3、将发布的Web Api复制到Web根目录

4、测试

curl -X "GET" ^

"http://localhost/WeatherForecast" ^

-H "accept: text/plain"

curl http://localhost/WeatherForecast

## 在IIS服务器上启用HTTPS(1)

Programe.cs

app.UseHttpsRedirection();

launchSettings.json

"sslPort": 44396

"applicationUrl": <https://localhost:7291;http://localhost:5291>

## 在IIS服务器上启用HTTPS(2)

## 将Web Api发布到Linux服务器

**安装aspnetcore运行时**

dotnet

yum makecache

yum search dotnet

yum search aspnetcore

yum install -y aspnetcore-runtime-6.0

dotnet

dotnet --info

**复制文件**

**运行程序**

cd publish/

dotnet WebApi9.dll

6.0.0-rc.2.21470.23

**修改配置**

vi WebApi9.runtimeconfig.json

6.0.0-rc.2.21470.23

**运行程序**

dotnet WebApi9.dll

**测试**

curl http://localhost:5000/WeatherForecast/

ip a

curl <http://192.168.240.134:5000/WeatherForecast/>

**设置端口**

dotnet WebApi 9.dll --urls="http://\*:5001"

curl <http://192.168.240.134:5001/WeatherForecast/>

**外部访问**

curl <http://192.168.240.134:5001/WeatherForecast/>

systemctl stop firewalld

curl <http://192.168.240.134:5001/WeatherForecast/>

systemctl start firewalld

curl <http://192.168.240.134:5001/WeatherForecast/>

## 使用Nginx来发布Web Api

Kestrel

Nginx

**安装nginx代理**

yum install -y nginx

systemctl enable nginx

/usr/lib/systemd/system/nginx.service

systemctl start nginx

**配置nginx**

**https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/linux-nginx?view=aspnetcore-6.0**

vi /etc/nginx/nginx.conf

location / {

proxy\_pass http://127.0.0.1:5000;

proxy\_http\_version 1.1;

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection keep-alive;

proxy\_set\_header Host $host;

proxy\_cache\_bypass $http\_upgrade;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Proto $scheme;

}

nginx -s reload

**添加防火墙策略**

firewall-cmd --add-port=80/tcp --permanent

firewall-cmd --reload

**SELinux设置**

setenforce 0

curl http://192.168.240.134/WeatherForecast/

sestatus

setenforce 1

**SELinux排错工具**

yum install -y setroubleshoot

cat /var/log/audit/audit.log

tail -n 3 /var/log/audit/audit.log

tail -n 3 /var/log/audit/audit.log | tee /tmp/audit

audit2why -i /tmp/audit

setsebool -P httpd\_can\_network\_connect 1

## 将Web Api以Linux中的服务来运行

**创建服务配置文件**

vi /etc/systemd/system/kestrel-WebApi009.service

**添加内容**

[Unit]

Description=Example .NET Web API App running on Ubuntu

[Service]

WorkingDirectory=/root/publish/

ExecStart=/usr/bin/dotnet /root/publish/WebApi009.dll

Restart=always

# Restart service after 10 seconds if the dotnet service crashes:

RestartSec=10

KillSignal=SIGINT

SyslogIdentifier=dotnet-example

User=root

Environment=ASPNETCORE\_ENVIRONMENT=Production

Environment=DOTNET\_PRINT\_TELEMETRY\_MESSAGE=false

[Install]

WantedBy=multi-user.target

**启用服务**

systemctl enable kestrel-WebApi009

**启动服务**

systemctl start kestrel-WebApi009

**参考文档**

https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/linux-nginx?view=aspnetcore-6.0

## Web Api中的Route

## Web Api中的Action

## Web Api中的数据绑定

## Web Api中的数据注解与数据验证

## Web Api中的过滤器(Filter)（1）

## Web Api中的过滤器(Filter)（2）

## Web Api中的过滤器(Filter)（3）

## 最小化的Web Api（1）

<Project Sdk="Microsoft.NET.Sdk.Web">

<PropertyGroup>

<TargetFramework>net6.0</TargetFramework>

<Nullable>enable</Nullable>

<ImplicitUsings>enable</ImplicitUsings>

</PropertyGroup>

**<ItemGroup>**

**<PackageReference Include="Swashbuckle.AspNetCore" Version="6.2.3" />**

**</ItemGroup>**

</Project>

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle

**builder.Services.AddEndpointsApiExplorer();**

**builder.Services.AddSwaggerGen();**

var app = builder.Build();

// Configure the HTTP request pipeline.

**if (app.Environment.IsDevelopment())**

**{**

**app.UseSwagger();**

**app.UseSwaggerUI();**

**}**

var summaries = new[]

{

"Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"

};

app.MapGet("/weatherforecast", () =>

{

var forecast = Enumerable.Range(1, 5).Select(index =>

new WeatherForecast

(

DateTime.Now.AddDays(index),

Random.Shared.Next(-20, 55),

summaries[Random.Shared.Next(summaries.Length)]

))

.ToArray();

return forecast;

})

**.WithName("GetWeatherForecast");**

app.Run();

internal record WeatherForecast(DateTime Date, int TemperatureC, string? Summary)

{

public int TemperatureF => 32 + (int)(TemperatureC / 0.5556);

}

"operationId": "GetWeatherForecast",

http://localhost:5242/swagger/v1/swagger.json

## 最小化的Web Api（2）

## 启用ASP.NET Core源码调试

源码来源：

1. github

<https://github.com/dotnet/aspnetcore>

1. 在VS中直接调试ASP.NET Core源码

<https://docs.microsoft.com/en-us/aspnet/core/test/debug-aspnetcore-source>

In **Tools -> Options -> Debugging -> General**, un-check **Enable Just My Code**

Verify **Enable Source Link support** is checked.

In **Tool -> Options -> Debugging -> Symbols**, enable **Microsoft Symbol Servers**.

如果没有生效，请重启VS

## Web Api中的配置参数

程序运行环境

IsDevelopment

IsProduction

IsStaging

**IsEnvironment**

配置参数的优先级

0、系统默认配置

1、appsettings.json

2、appsettings.{environment}.json

3、secrets.json(用户机密)

C:\Users\lib\AppData\Roaming\Microsoft\UserSecrets\8ea232c1-7d20-463e-9c47-bd938758a6b8

<UserSecretsId>8ea232c1-7d20-463e-9c47-bd938758a6b8</UserSecretsId>

4、环境变量

系统环境变量

用户环境变量

当前设置的环境变量

5、命令行参数

## Web Api中的配置文件

## ASP.NET Core中的日志信息（1）

builder.Logging.**ClearProviders**();

Microsoft.Extensions.Logging.Console.**ConsoleLoggerProvider**

Microsoft.Extensions.Logging.Debug.**DebugLoggerProvider**

Microsoft.Extensions.Logging.EventSource.**EventSourceLoggerProvider**

Microsoft.Extensions.Logging.EventLog.**EventLogLoggerProvider**

public WeatherForecastController(**ILogger<WeatherForecastController>** logger)

{

\_logger = logger;

}

public enum LogLevel

{

Trace,

Debug,

Information,

Warning,

Error,

Critical,

None

}

logging.AddFilter<EventLogLoggerProvider>(level => level >= LogLevel.Warning);

logging.AddConfiguration(hostingContext.Configuration.GetSection("Logging"));

{

"**Logging**": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*"

}

logging.AddConsole();

logging.AddDebug();

logging.AddEventSourceLogger();

logging.AddEventLog();

<https://github.com/dotnet/runtime/blob/main/src/libraries/Microsoft.Extensions.Logging.Debug/src/DebugLoggerFactoryExtensions.cs>

NLog.Web.AspNetCore

<https://github.com/NLog/NLog.Web>

Serilog.AspNetCore

<https://github.com/serilog/serilog-aspnetcore>

Microsoft.Extensions.Logging.Log4Net.AspNetCore

<https://github.com/huorswords/Microsoft.Extensions.Logging.Log4Net.AspNetCore>

AWS.Logger.AspNetCore

<https://github.com/aws/aws-logging-dotnet>

## ASP.NET Core中的日志信息（2）

## ASP.NET Core中的日志信息（3）

## ASP.NET Core中的日志信息（4）

## 向Web Api 添加EF的Api控制器（1）

#nullable disable

public string? Name { get; set; }

public string Name { get; set; }=null!;

public string Name { get; set; }=default!;

**appsettings.json**

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*",

"ConnectionStrings": {

"WebApi28Context": "**filename=test.db**"

}

}

**Program.cs**

builder.Services.AddDbContext<WebApi28Context>(

options =>

options.**UseSqlite**(builder.Configuration.GetConnectionString("WebApi28Context")

?? throw new InvalidOperationException("Connection string 'WebApi28Context' not found."))

);

**Microsoft.Data.Sqlite**

**Microsoft.EntityFrameworkCore.Sqlite.Core**

Microsoft.EntityFrameworkCore.Tools

Add-Migration a1

Update-Database

Microsoft.VisualStudio.Web.CodeGeneration.Design

return CreatedAtAction("GetStudent", new { id = student.Id }, student);

## 向Web Api 添加EF的Api控制器（2）

## Web Api中的日期自定义格式化

builder.Services.AddControllers()

.AddJsonOptions(options =>

{

options.JsonSerializerOptions.Converters.Add(new JsonCustomData());

options.JsonSerializerOptions.Converters.Add(new JsonCustomDataString());

});

internal class JsonCustomData : JsonConverter<DateTime>

{

public override DateTime Read(ref Utf8JsonReader reader, Type typeToConvert, JsonSerializerOptions options)

{

return reader.GetDateTime();

}

public override void Write(Utf8JsonWriter writer, DateTime value, JsonSerializerOptions options)

{

//writer.WriteStringValue(value.ToString());

//writer.WriteStringValue(value);

writer.WriteStringValue(value.ToString("yyyy-MM-dd HH:mm:ss"));

}

}

## ASP.NET Core 的Docker映像操作（1）

Windows Docker

<https://hub.docker.com/>

CentOS

<https://docs.docker.com/engine/install/centos/>

sudo yum remove docker \

docker-client \

docker-client-latest \

docker-common \

docker-latest \

docker-latest-logrotate \

docker-logrotate \

docker-engine

sudo yum install -y yum-utils

sudo yum-config-manager \

--add-repo \

https://download.docker.com/linux/centos/docker-ce.repo

sudo yum install docker-ce docker-ce-cli containerd.io

sudo systemctl start docker

sudo systemctl enable docker

docker -v

创建项目

命令行

VS

docker build -f "C:\Users\lib\source\repos\WebApi31\WebApi31\Dockerfile"

--force-rm

-t webapi31:dev

--target base

--label "com.microsoft.created-by=visual-studio"

--label "com.microsoft.visual-studio.project-name=WebApi31"

"C:\Users\lib\source\repos\WebApi31"

docker run

-dt

-v "C:\Users\lib\vsdbg\vs2017u5:/remote\_debugger:rw"

-v "C:\Users\lib\source\repos\WebApi31\WebApi31:/app"

-v "C:\Users\lib\source\repos\WebApi31:/src/"

-v "C:\Users\lib\.nuget\packages\:/root/.nuget/fallbackpackages"

-e "ASPNETCORE\_LOGGING\_\_CONSOLE\_\_DISABLECOLORS=true"

-e "ASPNETCORE\_ENVIRONMENT=Development"

-e "DOTNET\_USE\_POLLING\_FILE\_WATCHER=1"

-e "NUGET\_PACKAGES=/root/.nuget/fallbackpackages"

-e "NUGET\_FALLBACK\_PACKAGES=/root/.nuget/fallbackpackages"

-P

--name WebApi31

--entrypoint tail webapi31:dev

-f /dev/null

docker exec -i 6ef2d8510406c49fc05905e66050950960fef3e875f75a5fde2c41631b57d77f /bin/sh

-c "if PID=$(pidof dotnet); then kill $PID; fi"

docker ps

docker ps -a

docker images

docker save -o webapi31 webapi31

docker rmi webapi31:dev

docker load -i webapi31

<https://docs.microsoft.com/zh-cn/aspnet/core/host-and-deploy/docker/building-net-docker-images?view=aspnetcore-6.0>

Linux dotnet

yum install -y dotnet-sdk-6.0

dotnet new webapi -o api31

cd api31

dotnet publish -c Release -o published

dotnet published/api31.dll

curl <http://127.0.0.1:5000/weatherforecast>

curl -I <http://127.0.0.1:5000/weatherforecast>

curl <https://127.0.0.1:5001/weatherforecast>

curl -k <https://127.0.0.1:5001/weatherforecast>

[root@CentOS8 api31]# cat Dockerfile

FROM mcr.microsoft.com/dotnet/aspnet:6.0 AS runtime

WORKDIR /app

COPY published/\* ./

ENTRYPOINT ["dotnet", "api31.dll"]

docker build -t api31\_linux .

docker run -p 80:80 --name api31\_1 api31\_linux

docker stop f

## ASP.NET Core 的Docker映像操作（2）

## 为ASP.NET Core添加后台服务（1）

**IHostedService**

public class MyHostedService : IHostedService

{

private readonly ILogger<MyHostedService> logger;

public MyHostedService(ILogger<MyHostedService> logger)

{

this.logger = logger;

logger.LogInformation(DateTime.Now + "\tMyHostedService 构造函数");

}

public Task StartAsync(CancellationToken cancellationToken)

{

logger.LogInformation(DateTime.Now + "\tMyHostedService StartAsync");

return Task.Factory.StartNew( async () => {

while (!cancellationToken.IsCancellationRequested)

{

logger.LogInformation(DateTime.Now + "\tMyHostedService ...");

try

{

await Task.Delay(2000, cancellationToken);

}

catch { }

}

}, cancellationToken);

}

public Task StopAsync(CancellationToken cancellationToken)

{

logger.LogInformation(DateTime.Now + "\tMyHostedService StopAsync");

return Task.CompletedTask;

}

}

builder.Services.AddSingleton<IHostedService, MyHostedService>();

**BackgroundService**

public class MyBackgroundService : BackgroundService

{

private readonly ILogger<MyBackgroundService> logger;

public MyBackgroundService(ILogger<MyBackgroundService> logger)

{

this.logger = logger;

logger.LogInformation(DateTime.Now + "\tMyBackgroundService 构造函数");

}

protected override async Task ExecuteAsync(CancellationToken stoppingToken)

{

logger.LogInformation(DateTime.Now + "\tMyBackgroundService StartAsync");

await Task.Factory.StartNew(async () =>

{

while (!stoppingToken.IsCancellationRequested)

{

logger.LogInformation(DateTime.Now + "\tMyBackgroundService ...");

try

{

await Task.Delay(2000, stoppingToken);

}

catch { }

}

}, stoppingToken);

}

}

builder.Services.AddHostedService<MyBackgroundService>();

<https://github.com/dotnet/runtime/blob/main/src/libraries/Microsoft.Extensions.Hosting.Abstractions/src/BackgroundService.cs>

var host = Host.CreateDefaultBuilder()

.ConfigureServices(services =>

{

//services.AddHostedService<MyBackgroundService>();

services.AddSingleton<IHostedService, MyHostedService>();

})

.Build();

//host.Run();

await host.RunAsync();

## 为ASP.NET Core添加后台服务（2）

## 为ASP.NET Core添加自定义中间件（1）

http://localhost:5125/weatherforecast

curl <http://localhost:5125/weatherforecast>

var builder = WebApplication.CreateBuilder(args);

var app = builder.Build();

app.MapGet("/weatherforecast", () => { return "Hello World"; });

app.Run();

## 为ASP.NET Core添加自定义中间件（2）

public static IApplicationBuilder UseSwagger(this IApplicationBuilder app, SwaggerOptions options)

{

return app.UseMiddleware<SwaggerMiddleware>(new object[1] { options });

}

public class StopwatchMiddleware

{

private readonly RequestDelegate next;

public StopwatchMiddleware(RequestDelegate next)

{

this.next = next;

}

public async Task Invoke(HttpContext httpContext)

{

var stopWatch = new Stopwatch();

stopWatch.Start();

await next(httpContext);

stopWatch.Stop();

Console.WriteLine(stopWatch.ElapsedTicks);

}

}

public static class StopwatchExtensions

{

public static IApplicationBuilder UseStopWatch(this IApplicationBuilder app)

{

return app.UseMiddleware<StopwatchMiddleware>();

}

}

app.UseMiddleware<StopwatchMiddleware>();

app.UseStopWatch();

## 使用MAP的流水线分支

public static

**IEndpointConventionBuilder**

MapPut(this IEndpointRouteBuilder endpoints, string pattern, **RequestDelegate requestDelegate**);

public static

**RouteHandlerBuilder**

MapPut(this IEndpointRouteBuilder endpoints, string pattern, **Delegate handler**);

return MapMethods(endpoints, pattern, **GetVerb**, requestDelegate);

app.**Map**( pattern: "/app1", () =>

{

return "/APP1";

});

app.**Map**(pathMatch: "/app2",

app2 =>

{

app2.Run(

async hander =>

{

await hander.Response.WriteAsync("/APP2");

}

);

}

);

## 使用MAPWHEN的流水线分支

app.MapWhen(context => context.Request.Method.ToUpper() == "POST",

app1 =>

{

//app1.Run(async content =>

//{

// await content.Response.WriteAsync("/POST");

//});

app1.MapWhen(context => context.Request.Query.ContainsKey("a1"),

app2 =>

{

app2.Run(async content =>

{

await content.Response.WriteAsync("/A1");

});

});

app1.MapWhen(context => context.Request.Query.ContainsKey("a2"),

app2 =>

{

app2.Run(async content =>

{

await content.Response.WriteAsync("/A2");

});

});

}

);