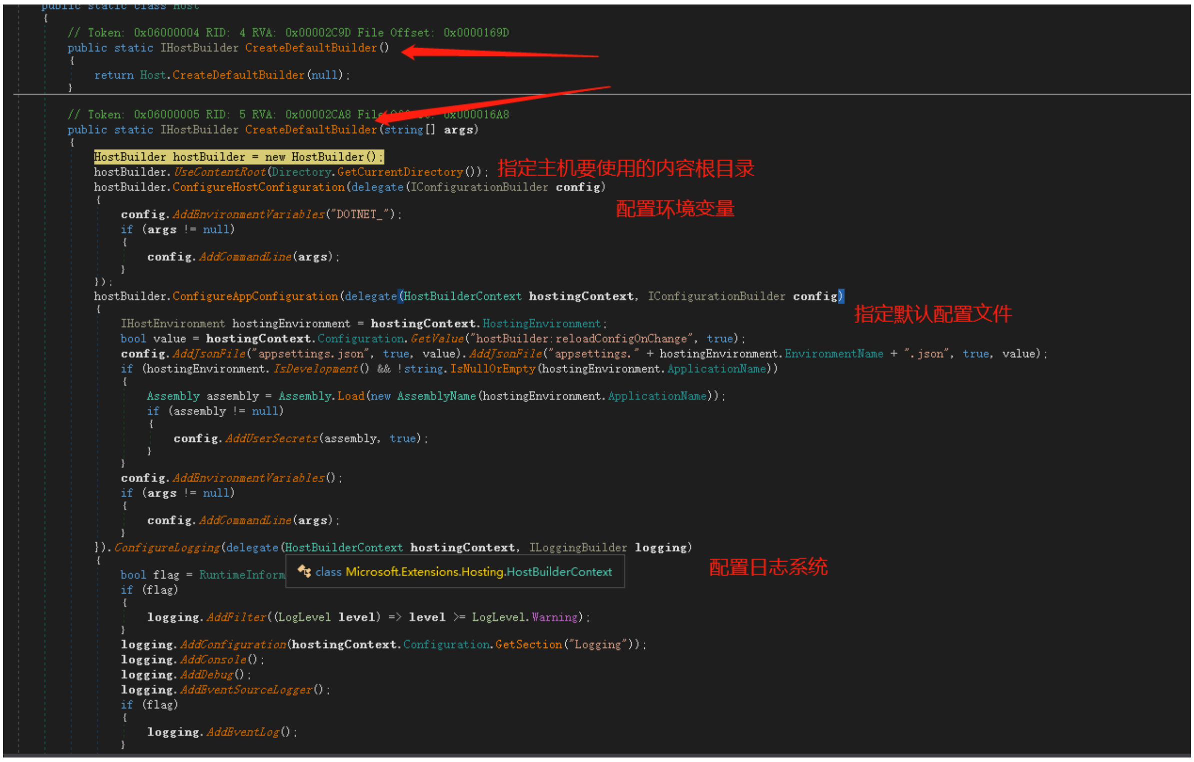
启动流程

一.调用`Microsoft.Extensions.Hosting.dll`下的静态方法`Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder()`注册通用主机



```
public static class Host
{
    // Token: 0x06000004 RID: 4 RVA: 0x00002C9D File Offset: 0x0000169D
    public static IHostBuilder CreateDefaultBuilder()
    {
        return Host.CreateDefaultBuilder(null);
    }

    // Token: 0x06000005 RID: 5 RVA: 0x00002CA8 Fil  0x000016A8
    public static IHostBuilder CreateDefaultBuilder(string[] args)
    {
        HostBuilder hostBuilder = new HostBuilder();
        hostBuilder.UseContentRoot(Directory.GetCurrentDirectory());
        hostBuilder.ConfigureHostConfiguration(delegate(IConfigurationBuilder config)
        {
            config.AddEnvironmentVariables("DOTNET_");
            if (args != null)
            {
                config.AddCommandLine(args);
            }
        });
        hostBuilder.ConfigureAppConfiguration(delegate(HostBuilderContext hostingContext, IConfigurationBuilder config)
        {
            IHostEnvironment hostingEnvironment = hostingContext.HostingEnvironment;
            bool value = hostingContext.Configuration.GetValue("hostBuilder:reloadConfigOnChange", true);
            config.AddJsonFile("appsettings.json", true, value).AddJsonFile("appsettings." + hostingEnvironment.EnvironmentName + ".json", true, value);
            if (hostingEnvironment.IsDevelopment() && !string.IsNullOrEmpty(hostingEnvironment.ApplicationName))
            {
                Assembly assembly = Assembly.Load(new AssemblyName(hostingEnvironment.ApplicationName));
                if (assembly != null)
                {
                    config.AddUserSecrets(assembly, true);
                }
            }
            config.AddEnvironmentVariables();
            if (args != null)
            {
                config.AddCommandLine(args);
            }
        }).ConfigureLogging(delegate(HostBuilderContext hostingContext, ILoggingBuilder logging)
        {
            bool flag = RuntimeInform
            if (flag)
            {
                logging.AddFilter((LogLevel level) => level >= LogLevel.Warning);
            }
            logging.AddConfiguration(hostingContext.Configuration.GetSection("Logging"));
            logging.AddConsole();
            logging.AddDebug();
            logging.AddEventSourceLogger();
            if (flag)
            {
                logging.AddEventLog();
            }
```

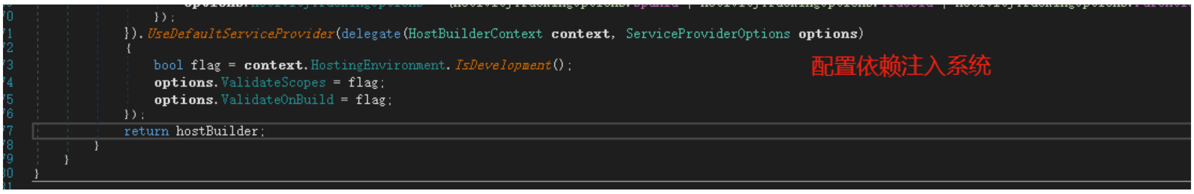
class Microsoft.Extensions.Hosting.HostBuilderContext

指定主机要使用的内容根目录

配置环境变量

指定默认配置文件

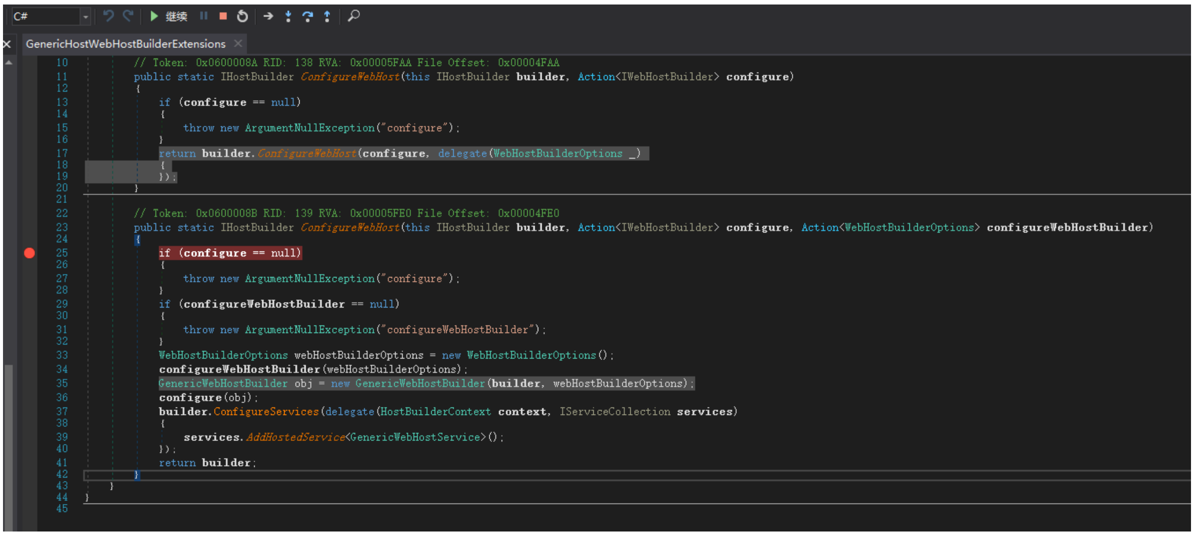
配置日志系统



```
                });
            }).UseDefaultServiceProvider(delegate(HostBuilderContext context, ServiceProviderOptions options)
            {
                bool flag = context.HostingEnvironment.IsDevelopment();
                options.ValidateScopes = flag;
                options.ValidateOnBuild = flag;
            });
            return hostBuilder;
        }
    }
}
```

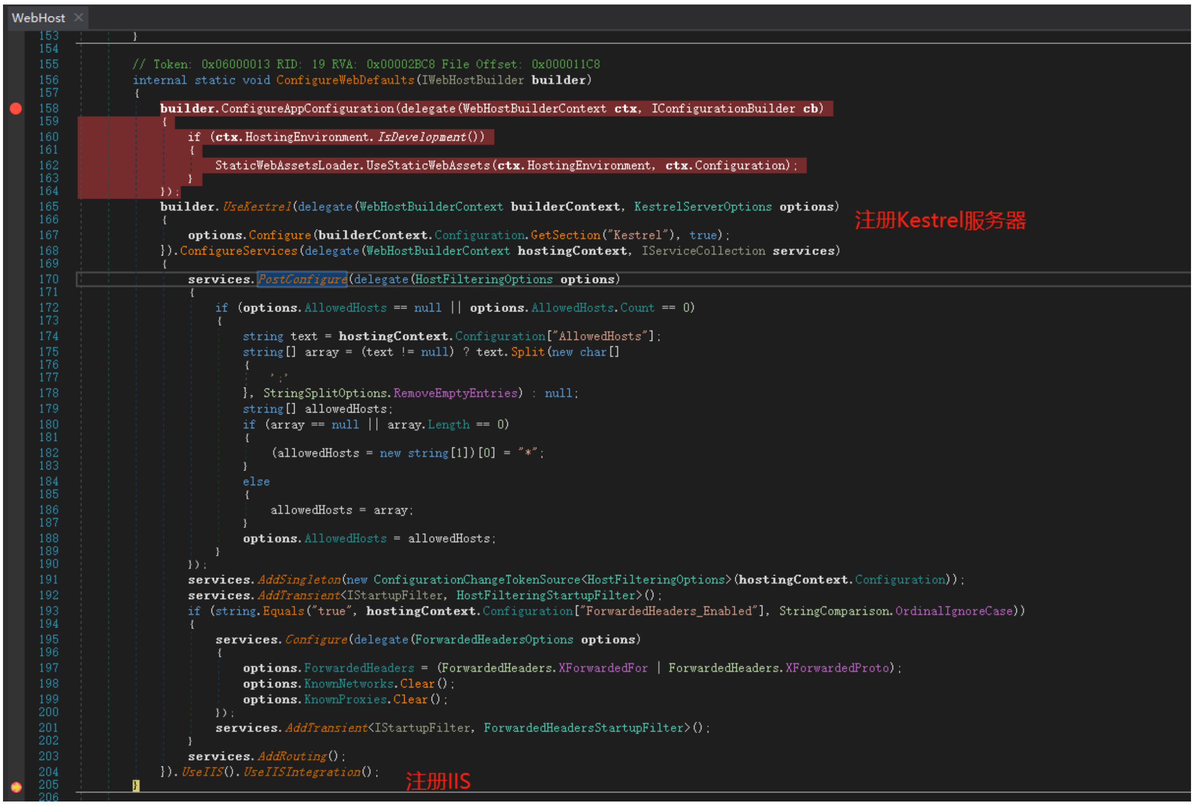
配置依赖注入系统

二、调用`Microsoft.AspNetCore.Hosting.dll`的静态方法Microsoft.Extensions.Hosting.GenericHostWebHostBuilderExtensions.*ConfigureWebHost*()注册web主机



```
        // Token: 0x0600008A RID: 138 RVA: 0x00005FAA File Offset: 0x00004FAA
        public static IHostBuilder ConfigureWebHost(this IHostBuilder builder, Action<IWebHostBuilder> configure)
        {
            if (configure == null)
            {
                throw new ArgumentNullException("configure");
            }
            return builder.ConfigureWebHost(configure, delegate(WebHostBuilderOptions _)
            {
            });
        }

        // Token: 0x0600008B RID: 139 RVA: 0x00005FE0 File Offset: 0x00004FE0
        public static IHostBuilder ConfigureWebHost(this IHostBuilder builder, Action<IWebHostBuilder> configure, Action<WebHostBuilderOptions> configureWebHostBuilder)
        {
            if (configure == null)
            {
                throw new ArgumentNullException("configure");
            }
            if (configureWebHostBuilder == null)
            {
                throw new ArgumentNullException("configureWebHostBuilder");
            }
            WebHostBuilderOptions webHostBuilderOptions = new WebHostBuilderOptions();
            configureWebHostBuilder(webHostBuilderOptions);
            GenericWebHostBuilder obj = new GenericWebHostBuilder(builder, webHostBuilderOptions);
            configure(obj);
            builder.ConfigureServices(delegate(HostBuilderContext context, IServiceCollection services)
            {
                services.AddHostedService<GenericWebHostService>();
            });
            return builder;
        }
    }
}
```

最后调用`Microsoft.AspNetCore.dll`下的静态方法`Microsoft.AspNetCore.WebHost.ConfigureWebDefaults(IWebHostBuilder **builder**)`

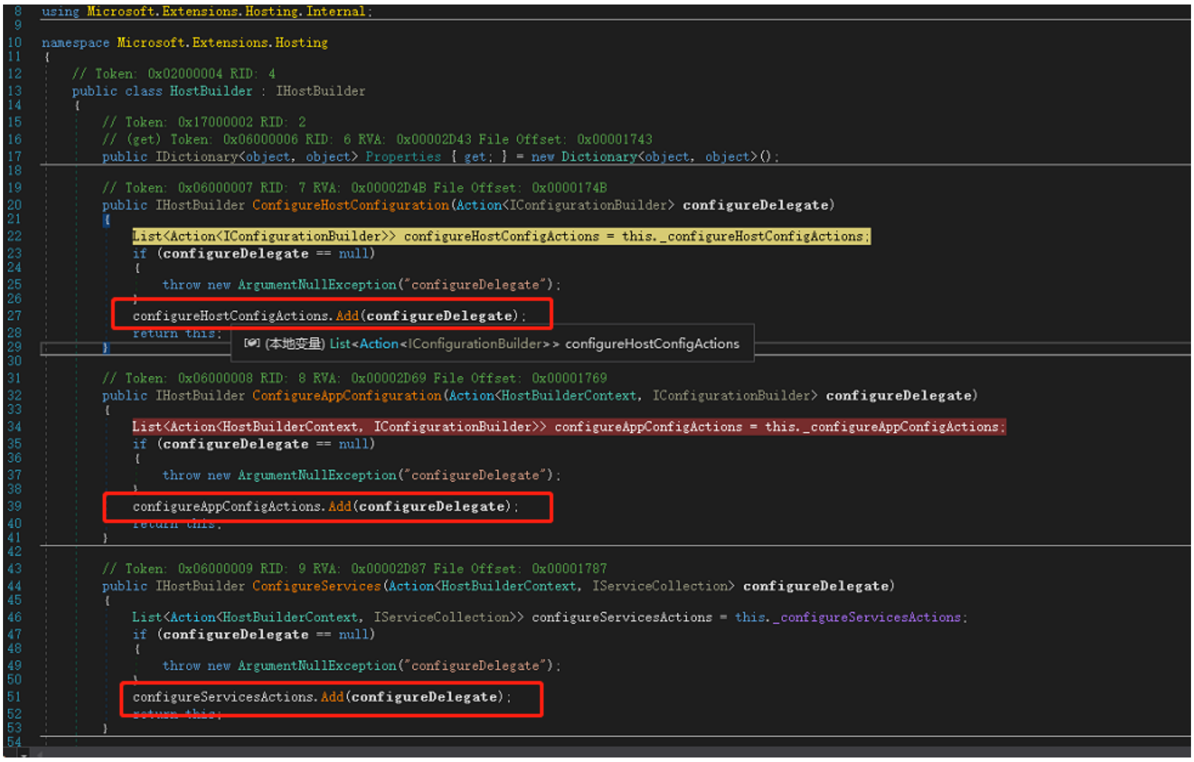


```
        }

        // Token: 0x06000013 RID: 19 RVA: 0x00002BC8 File Offset: 0x000011C8
        internal static void ConfigureWebDefaults(IWebHostBuilder builder)
        {
            builder.ConfigureAppConfiguration(delegate(WebHostBuilderContext ctx, IConfigurationBuilder cb)
            {
                if (ctx.HostingEnvironment.IsDevelopment())
                {
                    StaticWebAssetsLoader.UseStaticWebAssets(ctx.HostingEnvironment, ctx.Configuration);
                }
            });
            builder.UseKestrel(delegate(WebHostBuilderContext builderContext, KestrelServerOptions options)
            {
                options.Configure(builderContext.Configuration.GetSection("Kestrel"), true);
            }).ConfigureServices(delegate(WebHostBuilderContext hostingContext, IServiceCollection services)
            {
                services.PostConfigure(delegate(HostFilteringOptions options)
                {
                    if (options.AllowedHosts == null || options.AllowedHosts.Count == 0)
                    {
                        string text = hostingContext.Configuration["AllowedHosts"];
                        string[] array = (text != null) ? text.Split(new char[]
                        {
                            ','
                        }, StringSplitOptions.RemoveEmptyEntries) : null;
                        string[] allowedHosts;
                        if (array == null || array.Length == 0)
                        {
                            (allowedHosts = new string[1])[0] = "*";
                        }
                        else
                        {
                            allowedHosts = array;
                        }
                        options.AllowedHosts = allowedHosts;
                    }
                });
                services.AddSingleton(new ConfigurationChangeTokenSource<HostFilteringOptions>(hostingContext.Configuration));
                services.AddTransient<IStartupFilter, HostFilteringStartupFilter>();
                if (string.Equals("true", hostingContext.Configuration["ForwardedHeaders_Enabled"], StringComparison.OrdinalIgnoreCase))
                {
                    services.Configure(delegate(ForwardedHeadersOptions options)
                    {
                        options.ForwardedHeaders = (ForwardedHeaders.XForwardedFor | ForwardedHeaders.XForwardedProto);
                        options.KnownNetworks.Clear();
                        options.KnownProxies.Clear();
                    });
                    services.AddTransient<IStartupFilter, ForwardedHeadersStartupFilter>();
                }
                services.AddRouting();
            }).UseIIS().UseIISIntegration();
        }
```

注册Kestrel服务器

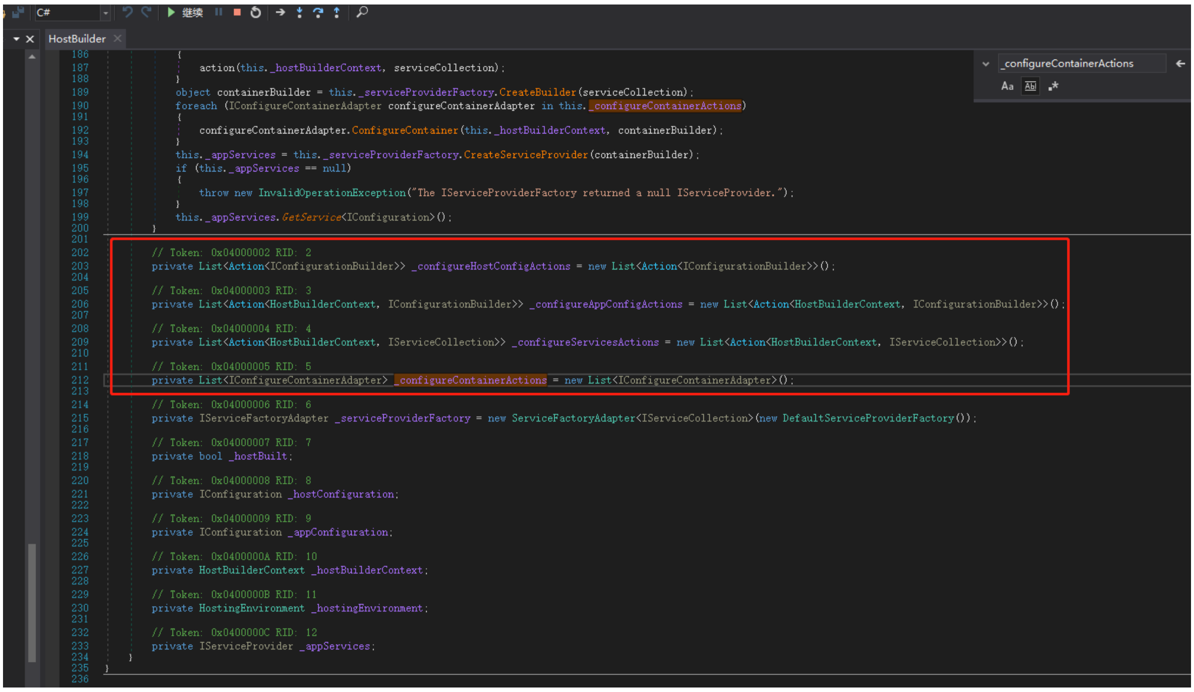
注册IIS

三、注册主机的各种委托会放到`Microsoft.Extensions.Hosting.dll`下的主机`Microsoft.Extensions.Hosting.HostBuilder`的本地集合中



```
using Microsoft.Extensions.Hosting.Internal;

namespace Microsoft.Extensions.Hosting
{
    // Token: 0x02000004 RID: 4
    public class HostBuilder : IHostBuilder
    {
        // Token: 0x17000002 RID: 2
        // (get) Token: 0x06000006 RID: 6 RVA: 0x00002D43 File Offset: 0x00001743
        public IDictionary<object, object> Properties { get; } = new Dictionary<object, object>();

        // Token: 0x06000007 RID: 7 RVA: 0x00002D4B File Offset: 0x0000174B
        public IHostBuilder ConfigureHostConfiguration(Action<IConfigurationBuilder> configureDelegate)
        {
            List<Action<IConfigurationBuilder>> configureHostConfigActions = this._configureHostConfigActions;
            if (configureDelegate == null)
            {
                throw new ArgumentNullException("configureDelegate");
            }
            configureHostConfigActions.Add(configureDelegate);
            return this;
        }

        // Token: 0x06000008 RID: 8 RVA: 0x00002D69 File Offset: 0x00001769
        public IHostBuilder ConfigureAppConfiguration(Action<HostBuilderContext, IConfigurationBuilder> configureDelegate)
        {
            List<Action<HostBuilderContext, IConfigurationBuilder>> configureAppConfigActions = this._configureAppConfigActions;
            if (configureDelegate == null)
            {
                throw new ArgumentNullException("configureDelegate");
            }
            configureAppConfigActions.Add(configureDelegate);
            return this;
        }

        // Token: 0x06000009 RID: 9 RVA: 0x00002D87 File Offset: 0x00001787
        public IHostBuilder ConfigureServices(Action<HostBuilderContext, IServiceCollection> configureDelegate)
        {
            List<Action<HostBuilderContext, IServiceCollection>> configureServicesActions = this._configureServicesActions;
            if (configureDelegate == null)
            {
                throw new ArgumentNullException("configureDelegate");
            }
            configureServicesActions.Add(configureDelegate);
            return this;
        }
```

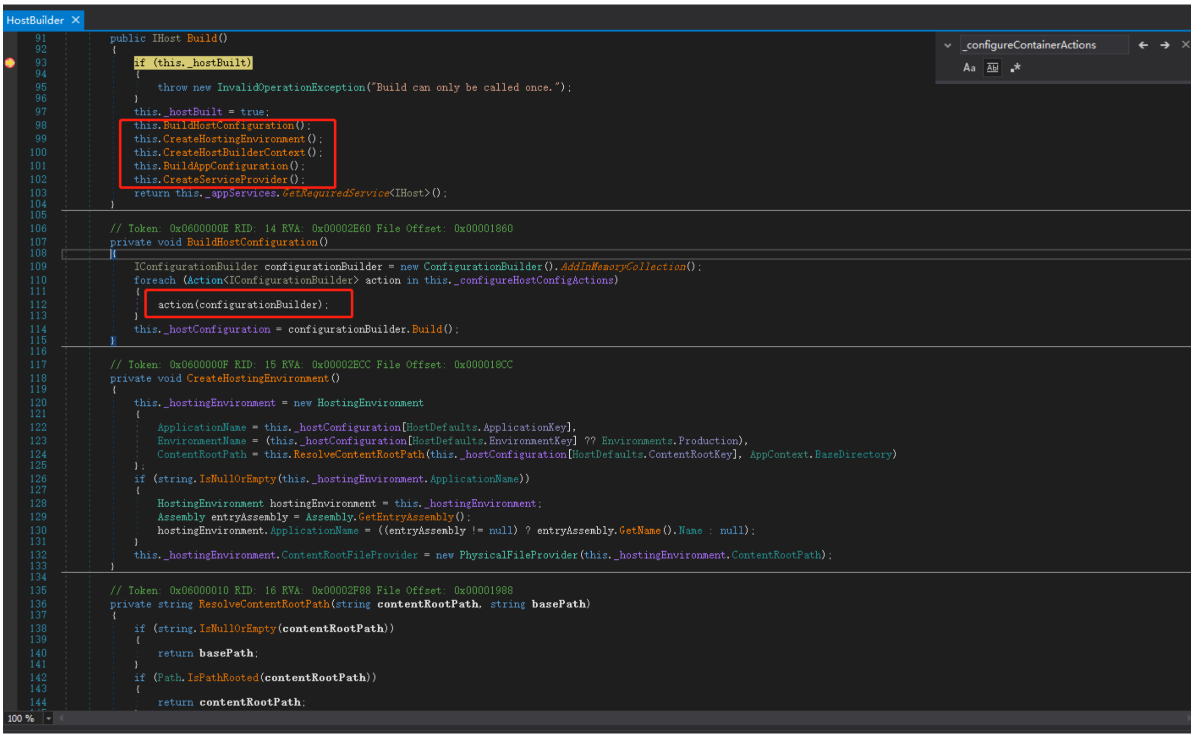
(本地变量) List<Action<IConfigurationBuilder>> configureHostConfigActions



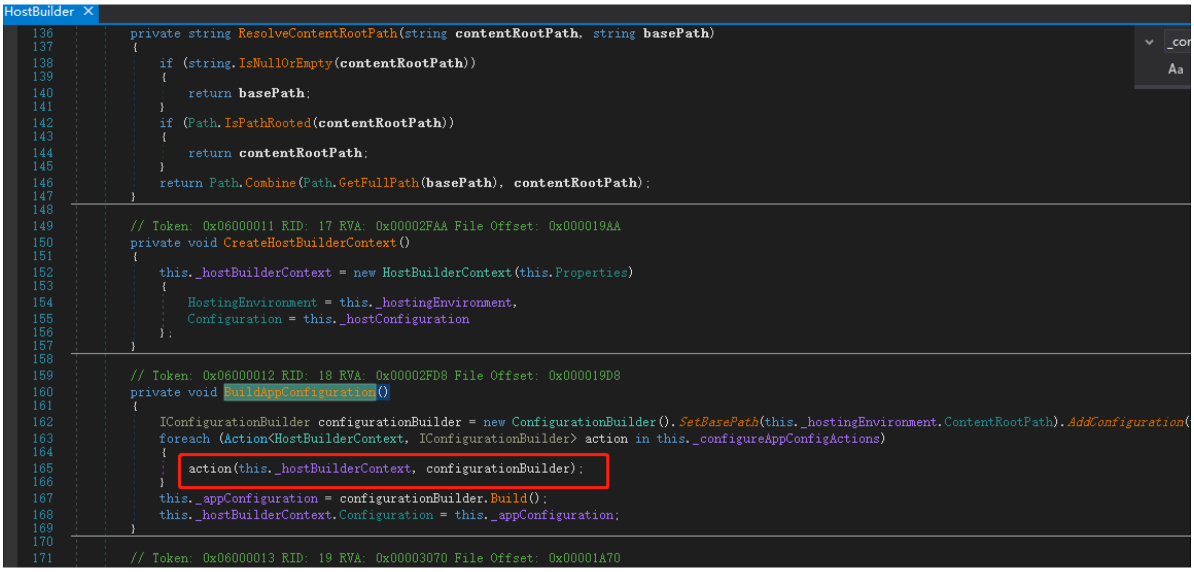
C#
继续
HostBuilder
_configureContainerActions
Aa

```
                {
                    action(this._hostBuilderContext, serviceCollection);
                }
                object containerBuilder = this._serviceProviderFactory.CreateBuilder(serviceCollection);
                foreach (IConfigureContainerAdapter configureContainerAdapter in this._configureContainerActions)
                {
                    configureContainerAdapter.ConfigureContainer(this._hostBuilderContext, containerBuilder);
                }
                this._appServices = this._serviceProviderFactory.CreateServiceProvider(containerBuilder);
                if (this._appServices == null)
                {
                    throw new InvalidOperationException("The IServiceProviderFactory returned a null IServiceProvider.");
                }
                this._appServices.GetService<IConfiguration>();
            }

            // Token: 0x04000002 RID: 2
            private List<Action<IConfigurationBuilder>> _configureHostConfigActions = new List<Action<IConfigurationBuilder>>();

            // Token: 0x04000003 RID: 3
            private List<Action<HostBuilderContext, IConfigurationBuilder>> _configureAppConfigActions = new List<Action<HostBuilderContext, IConfigurationBuilder>>();

            // Token: 0x04000004 RID: 4
            private List<Action<HostBuilderContext, IServiceCollection>> _configureServicesActions = new List<Action<HostBuilderContext, IServiceCollection>>();

            // Token: 0x04000005 RID: 5
            private List<IConfigureContainerAdapter> _configureContainerActions = new List<IConfigureContainerAdapter>();

            // Token: 0x04000006 RID: 6
            private IServiceFactoryAdapter _serviceProviderFactory = new ServiceFactoryAdapter<IServiceCollection>(new DefaultServiceProviderFactory());

            // Token: 0x04000007 RID: 7
            private bool _hostBuilt;

            // Token: 0x04000008 RID: 8
            private IConfiguration _hostConfiguration;

            // Token: 0x04000009 RID: 9
            private IConfiguration _appConfiguration;

            // Token: 0x0400000A RID: 10
            private HostBuilderContext _hostBuilderContext;

            // Token: 0x0400000B RID: 11
            private HostingEnvironment _hostingEnvironment;

            // Token: 0x0400000C RID: 12
            private IServiceProvider _appServices;
        }
}
```

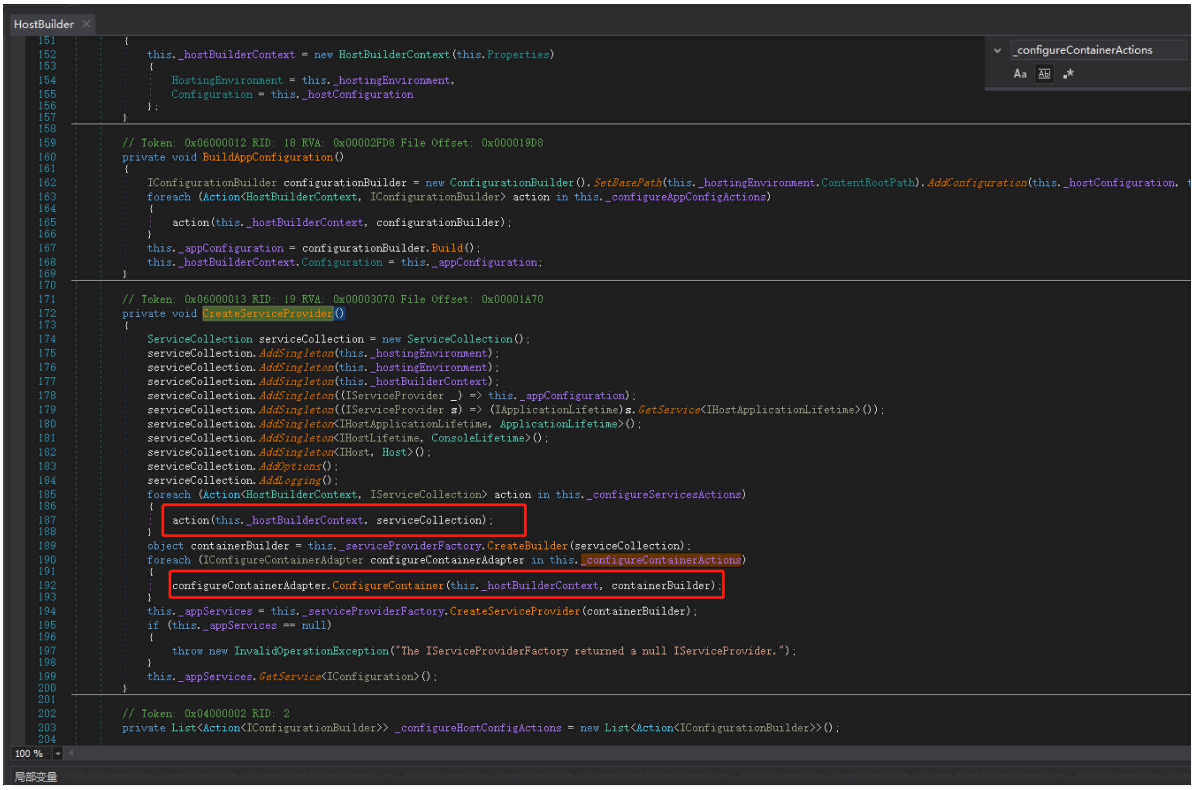
四、紧接着调用主机的Build()方法将注册的委托都执行一遍,初始化Startup类型，并调用Startup的`ConfigureServices`方法



```
public IHost Build()
{
    if (this._hostBuilt)
    {
        throw new InvalidOperationException("Build can only be called once.");
    }
    this._hostBuilt = true;
    this.BuildHostConfiguration();
    this.CreateHostingEnvironment();
    this.CreateHostBuilderContext();
    this.BuildAppConfiguration();
    this.CreateServiceProvider();
    return this._appServices.GetRequiredService<IHost>();
}

// Token: 0x0600000E RID: 14 RVA: 0x00002E60 File Offset: 0x00001860
private void BuildHostConfiguration()
{
    IConfigurationBuilder configurationBuilder = new ConfigurationBuilder().AddInMemoryCollection();
    foreach (Action<IConfigurationBuilder> action in this._configureHostConfigActions)
    {
        action(configurationBuilder);
    }
    this._hostConfiguration = configurationBuilder.Build();
}

// Token: 0x0600000F RID: 15 RVA: 0x00002ECC File Offset: 0x000018CC
private void CreateHostingEnvironment()
{
    this._hostingEnvironment = new HostingEnvironment
    {
        ApplicationName = this._hostConfiguration[HostDefaults.ApplicationKey],
        EnvironmentName = (this._hostConfiguration[HostDefaults.EnvironmentKey] ?? Environments.Production),
        ContentRootPath = this.ResolveContentRootPath(this._hostConfiguration[HostDefaults.ContentRootKey], AppContext.BaseDirectory)
    };
    if (string.IsNullOrEmpty(this._hostingEnvironment.ApplicationName))
    {
        HostingEnvironment hostingEnvironment = this._hostingEnvironment;
        Assembly entryAssembly = Assembly.GetEntryAssembly();
        hostingEnvironment.ApplicationName = ((entryAssembly != null) ? entryAssembly.GetName().Name : null);
    }
    this._hostingEnvironment.ContentRootFileProvider = new PhysicalFileProvider(this._hostingEnvironment.ContentRootPath);
}

// Token: 0x06000010 RID: 16 RVA: 0x00002F88 File Offset: 0x00001988
private string ResolveContentRootPath(string contentRootPath, string basePath)
{
    if (string.IsNullOrEmpty(contentRootPath))
    {
        return basePath;
    }
    if (Path.IsPathRooted(contentRootPath))
    {
        return contentRootPath;
```

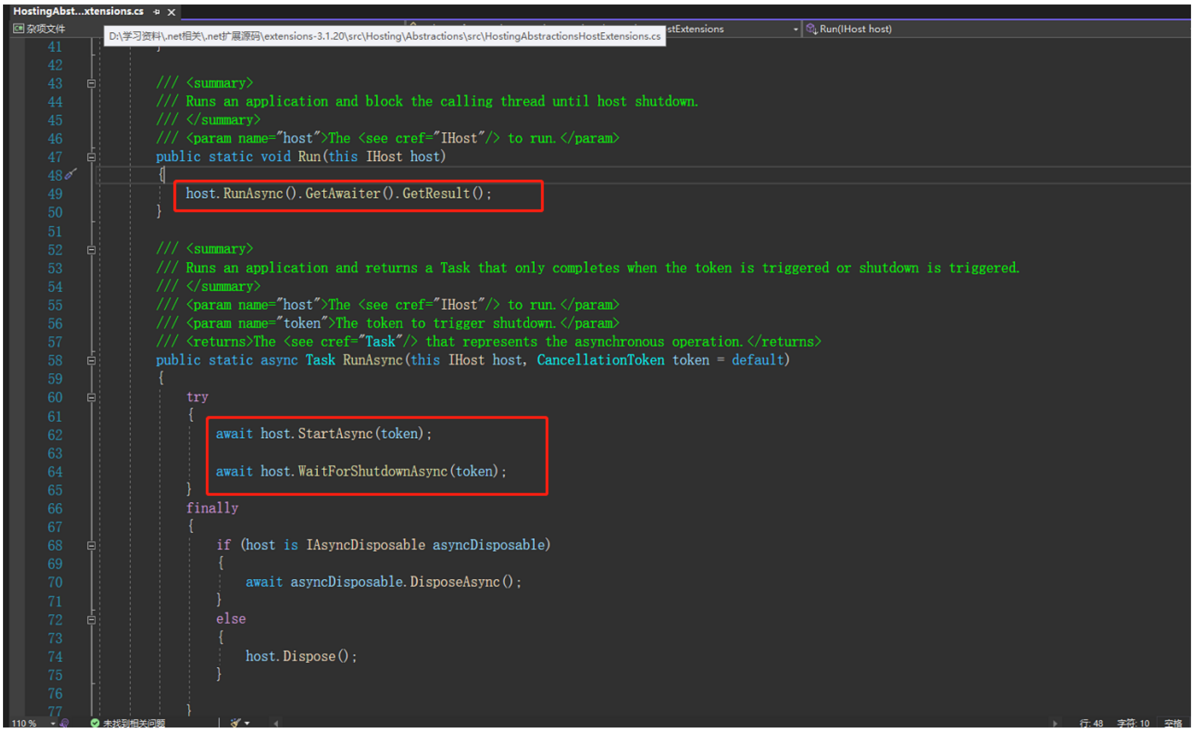


```
private string ResolveContentRootPath(string contentRootPath, string basePath)
{
    if (string.IsNullOrEmpty(contentRootPath))
    {
        return basePath;
    }
    if (Path.IsPathRooted(contentRootPath))
    {
        return contentRootPath;
    }
    return Path.Combine(Path.GetFullPath(basePath), contentRootPath);
}

// Token: 0x06000011 RID: 17 RVA: 0x00002FAA File Offset: 0x000019AA
private void CreateHostBuilderContext()
{
    this._hostBuilderContext = new HostBuilderContext(this.Properties)
    {
        HostingEnvironment = this._hostingEnvironment,
        Configuration = this._hostConfiguration
    };
}

// Token: 0x06000012 RID: 18 RVA: 0x00002FD8 File Offset: 0x000019D8
private void BuildAppConfiguration()
{
    IConfigurationBuilder configurationBuilder = new ConfigurationBuilder().SetBasePath(this._hostingEnvironment.ContentRootPath).AddConfiguration(
    foreach (Action<HostBuilderContext, IConfigurationBuilder> action in this._configureAppConfigActions)
    {
        action(this._hostBuilderContext, configurationBuilder);
    }
    this._appConfiguration = configurationBuilder.Build();
    this._hostBuilderContext.Configuration = this._appConfiguration;
}

// Token: 0x06000013 RID: 19 RVA: 0x00003070 File Offset: 0x00001A70
```

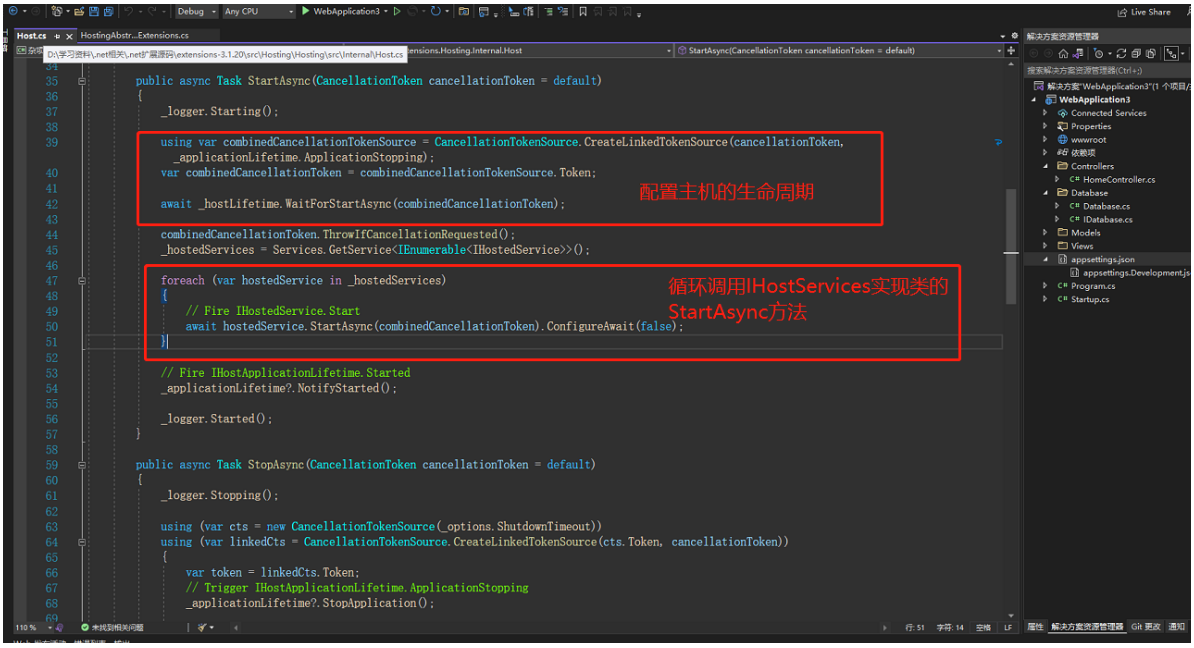


```
    {
        this._hostBuilderContext = new HostBuilderContext(this.Properties)
        {
            HostingEnvironment = this._hostingEnvironment,
            Configuration = this._hostConfiguration
        };
    }

    // Token: 0x06000012 RID: 18 RVA: 0x00002FD8 File Offset: 0x000019D8
    private void BuildAppConfiguration()
    {
        IConfigurationBuilder configurationBuilder = new ConfigurationBuilder().SetBasePath(this._hostingEnvironment.ContentRootPath).AddConfiguration(this._hostConfiguration, t
        foreach (Action<HostBuilderContext, IConfigurationBuilder> action in this._configureAppConfigActions)
        {
            action(this._hostBuilderContext, configurationBuilder);
        }
        this._appConfiguration = configurationBuilder.Build();
        this._hostBuilderContext.Configuration = this._appConfiguration;
    }

    // Token: 0x06000013 RID: 19 RVA: 0x00003070 File Offset: 0x00001A70
    private void CreateServiceProvider()
    {
        ServiceCollection serviceCollection = new ServiceCollection();
        serviceCollection.AddSingleton(this._hostingEnvironment);
        serviceCollection.AddSingleton(this._hostingEnvironment);
        serviceCollection.AddSingleton(this._hostBuilderContext);
        serviceCollection.AddSingleton((IServiceProvider _) => this._appConfiguration);
        serviceCollection.AddSingleton((IServiceProvider s) => (IApplicationLifetime)s.GetService<IHostApplicationLifetime>());
        serviceCollection.AddSingleton<IHostApplicationLifetime, ApplicationLifetime>();
        serviceCollection.AddSingleton<IHostLifetime, ConsoleLifetime>();
        serviceCollection.AddSingleton<IHost, Host>();
        serviceCollection.AddOptions();
        serviceCollection.AddLogging();
        foreach (Action<HostBuilderContext, IServiceCollection> action in this._configureServicesActions)
        {
            action(this._hostBuilderContext, serviceCollection);
        }
        object containerBuilder = this._serviceProviderFactory.CreateBuilder(serviceCollection);
        foreach (IConfigureContainerAdapter configureContainerAdapter in this._configureContainerActions)
        {
            configureContainerAdapter.ConfigureContainer(this._hostBuilderContext, containerBuilder);
        }
        this._appServices = this._serviceProviderFactory.CreateServiceProvider(containerBuilder);
        if (this._appServices == null)
        {
            throw new InvalidOperationException("The IServiceProviderFactory returned a null IServiceProvider.");
        }
        this._appServices.GetService<IConfiguration>();
    }

    // Token: 0x04000002 RID: 2
    private List<Action<IConfigurationBuilder>> _configureHostConfigActions = new List<Action<IConfigurationBuilder>>();
```

五、执行`Microsoft.Extensions.Hosting.Abstractions.dll`中`Microsoft.Extensions.Hosting.HostingAbstractionsHostExtensions`的*Run*方法运行应用并阻止调用线程，直到关闭主机


HostingAbst...xtensions.cs
杂项文件
D:\学习资料\.net相关\.net扩展源码\extensions-3.1.20\src\Hosting\Abstractions\src\HostingAbstractionsHostExtensions.cs
stExtensions
Run(IHost host)
110 %
未找到相关问题
行: 48
字符: 10
空格

```
        }

        /// <summary>
        /// Runs an application and block the calling thread until host shutdown.
        /// </summary>
        /// <param name="host">The <see cref="IHost"/> to run.</param>
        public static void Run(this IHost host)
        {
            host.RunAsync().GetAwaiter().GetResult();
        }

        /// <summary>
        /// Runs an application and returns a Task that only completes when the token is triggered or shutdown is triggered.
        /// </summary>
        /// <param name="host">The <see cref="IHost"/> to run.</param>
        /// <param name="token">The token to trigger shutdown.</param>
        /// <returns>The <see cref="Task"/> that represents the asynchronous operation.</returns>
        public static async Task RunAsync(this IHost host, CancellationToken token = default)
        {
            try
            {
                await host.StartAsync(token);

                await host.WaitForShutdownAsync(token);
            }
            finally
            {
                if (host is IAsyncDisposable asyncDisposable)
                {
                    await asyncDisposable.DisposeAsync();
                }
                else
                {
                    host.Dispose();
                }

            }
```

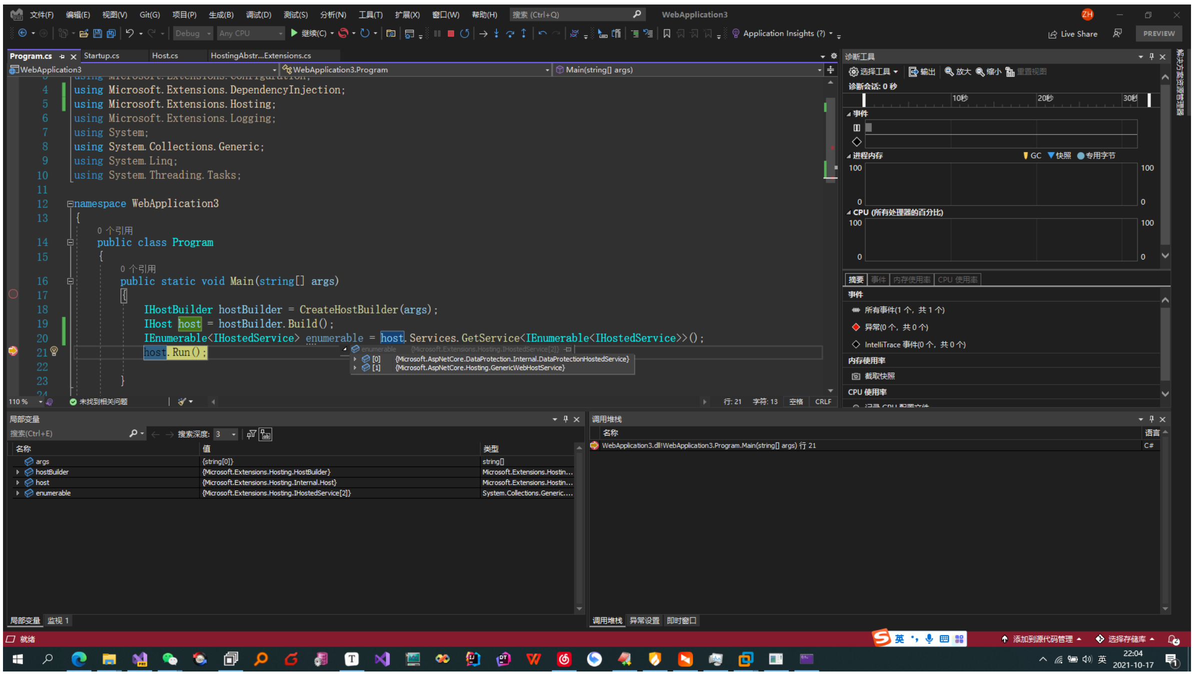


```
        public async Task StartAsync(CancellationToken cancellationToken = default)
        {
            _logger.Starting();

            using var combinedCancellationTokenSource = CancellationTokenSource.CreateLinkedTokenSource(cancellationToken,
              _applicationLifetime.ApplicationStopping);
            var combinedCancellationToken = combinedCancellationTokenSource.Token;

            await _hostLifetime.WaitForStartAsync(combinedCancellationToken);

            combinedCancellationToken.ThrowIfCancellationRequested();
            _hostedServices = Services.GetService<IEnumerable<IHostedService>>();

            foreach (var hostedService in _hostedServices)
            {
                // Fire IHostedService.Start
                await hostedService.StartAsync(combinedCancellationToken).ConfigureAwait(false);
            }

            // Fire IHostApplicationLifetime.Started
            _applicationLifetime?.NotifyStarted();

            _logger.Started();
        }

        public async Task StopAsync(CancellationToken cancellationToken = default)
        {
            _logger.Stopping();

            using (var cts = new CancellationTokenSource(_options.ShutdownTimeout))
            using (var linkedCts = CancellationTokenSource.CreateLinkedTokenSource(cts.Token, cancellationToken))
            {
                var token = linkedCts.Token;
                // Trigger IHostApplicationLifetime.ApplicationStopping
                _applicationLifetime?.StopApplication();
```

配置主机的生命周期

循环调用IHostServices实现类的StartAsync方法

Host.cs
HostingAbstr...Extensions.cs
D:\学习资料\.net相关\.net扩展源码\extensions-3.1.20\src\Hosting\Hosting\src\Internal\Host.cs
StartAsync(CancellationToken cancellationToken = default)
解决方案资源管理器
解决方案"WebApplication3"(1 个项目)
WebApplication3
Connected Services
Properties
wwwroot
依赖项
Controllers
HomeController.cs
Database
Database.cs
IDatabase.cs
Models
Views
appsettings.json
appsettings.Development.json
Program.cs
Startup.cs
110 %
未找到相关问题
行: 51 字符: 14 空格 LF



```
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace WebApplication3
{
    0 个引用
    public class Program
    {
        0 个引用
        public static void Main(string[] args)
        {
            IHostBuilder hostBuilder = CreateHostBuilder(args);
            IHost host = hostBuilder.Build();
            IEnumerable<IHostedService> enumerable = host.Services.GetService<IEnumerable<IHostedService>>();
            host.Run();

        }
```

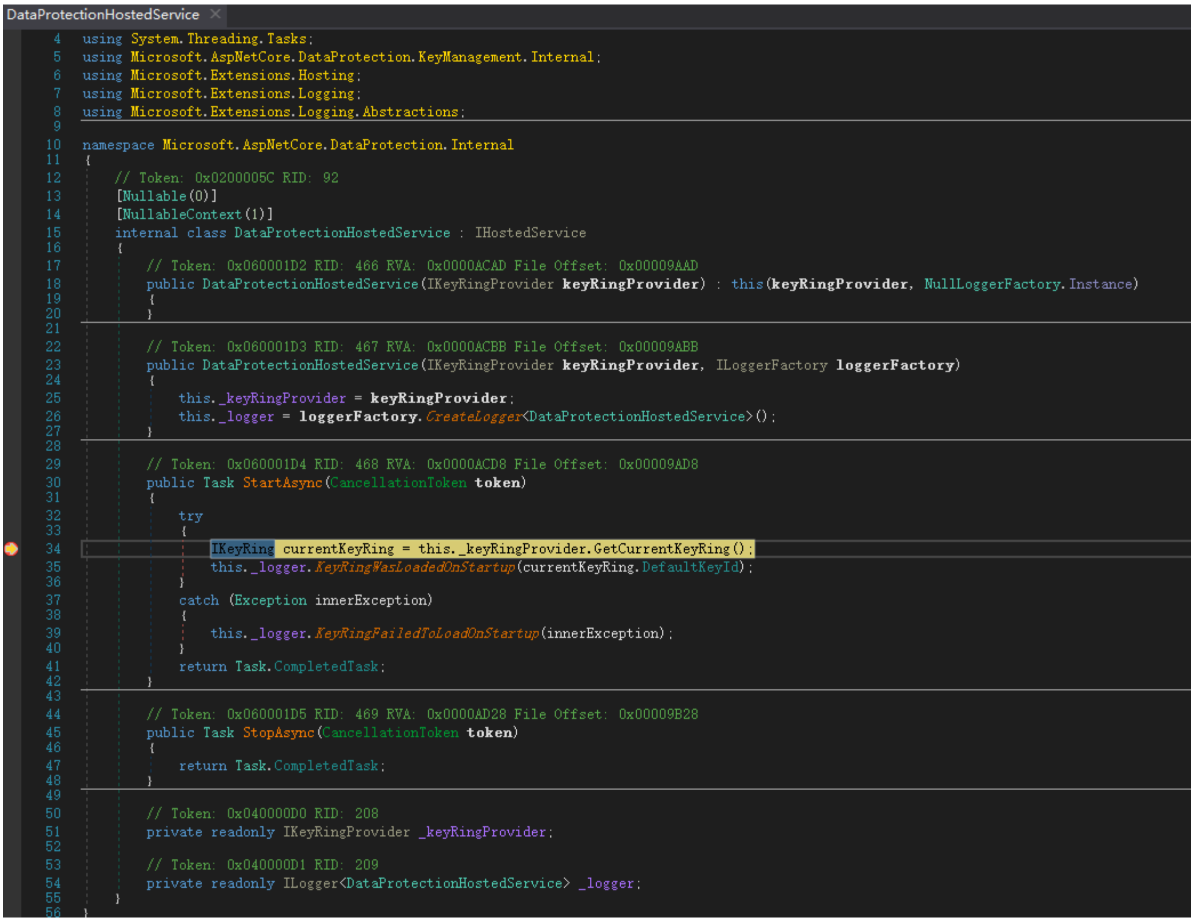
```
enumerable  {Microsoft.Extensions.Hosting.IHostedService[2]}
  [0]  {Microsoft.AspNetCore.DataProtection.Internal.DataProtectionHostedService}
  [1]  {Microsoft.AspNetCore.Hosting.GenericWebHostService}
```

局部变量

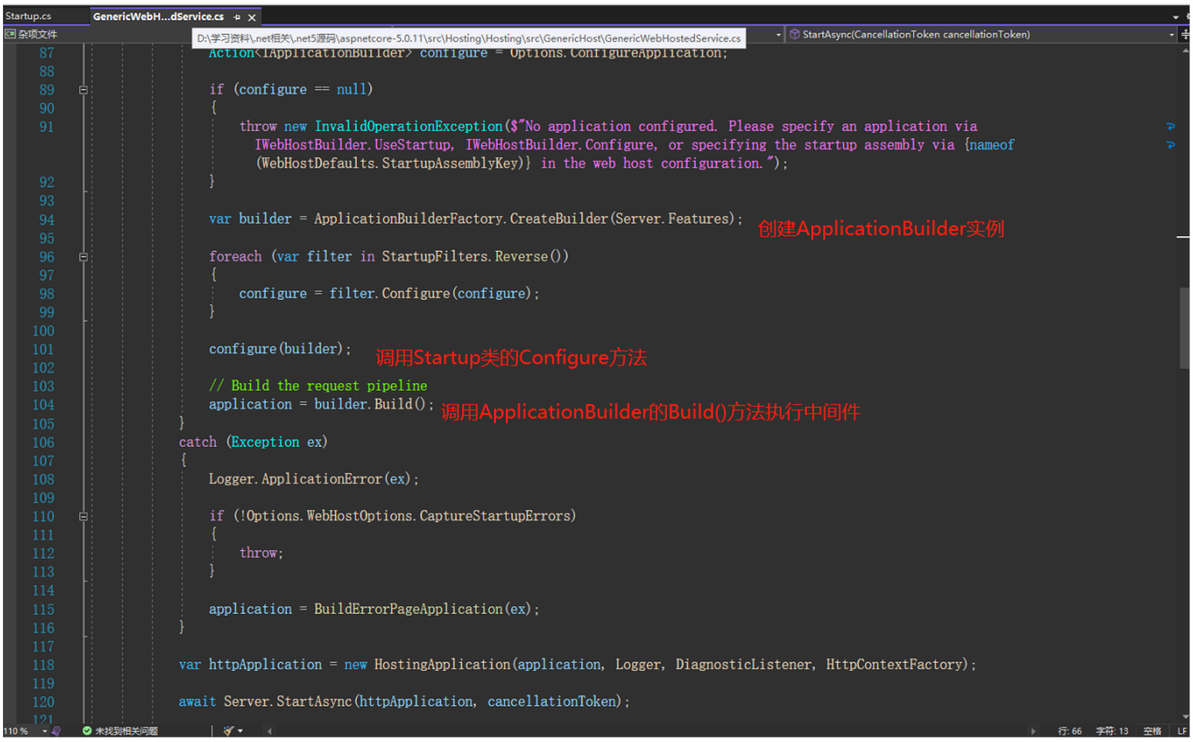
名称	值	类型
args	{string[0]}	string[]
hostBuilder	{Microsoft.Extensions.Hosting.HostBuilder}	Microsoft.Extensions.Hostin...
host	{Microsoft.Extensions.Hosting.Internal.Host}	Microsoft.Extensions.Hostin...
enumerable	{Microsoft.Extensions.Hosting.IHostedService[2]}	System.Collections.Generic....

调用堆栈

名称	语言
WebApplication3.dll!WebApplication3.Program.Main(string[] args) 行 21	C#



```csharp
using System.Threading.Tasks;
using Microsoft.AspNetCore.DataProtection.KeyManagement.Internal;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Abstractions;

namespace Microsoft.AspNetCore.DataProtection.Internal
{
    // Token: 0x0200005C RID: 92
    [Nullable(0)]
    [NullableContext(1)]
    internal class DataProtectionHostedService : IHostedService
    {
        // Token: 0x060001D2 RID: 466 RVA: 0x0000ACAD File Offset: 0x00009AAD
        public DataProtectionHostedService(IKeyRingProvider keyRingProvider) : this(keyRingProvider, NullLoggerFactory.Instance)
        {
        }

        // Token: 0x060001D3 RID: 467 RVA: 0x0000ACBB File Offset: 0x00009ABB
        public DataProtectionHostedService(IKeyRingProvider keyRingProvider, ILoggerFactory loggerFactory)
        {
            this._keyRingProvider = keyRingProvider;
            this._logger = loggerFactory.CreateLogger<DataProtectionHostedService>();
        }

        // Token: 0x060001D4 RID: 468 RVA: 0x0000ACD8 File Offset: 0x00009AD8
        public Task StartAsync(CancellationToken token)
        {
            try
            {
                IKeyRing currentKeyRing = this._keyRingProvider.GetCurrentKeyRing();
                this._logger.KeyRingWasLoadedOnStartup(currentKeyRing.DefaultKeyId);
            }
            catch (Exception innerException)
            {
                this._logger.KeyRingFailedToLoadOnStartup(innerException);
            }
            return Task.CompletedTask;
        }

        // Token: 0x060001D5 RID: 469 RVA: 0x0000AD28 File Offset: 0x00009B28
        public Task StopAsync(CancellationToken token)
        {
            return Task.CompletedTask;
        }

        // Token: 0x040000D0 RID: 208
        private readonly IKeyRingProvider _keyRingProvider;

        // Token: 0x040000D1 RID: 209
        private readonly ILogger<DataProtectionHostedService> _logger;
    }
}
```

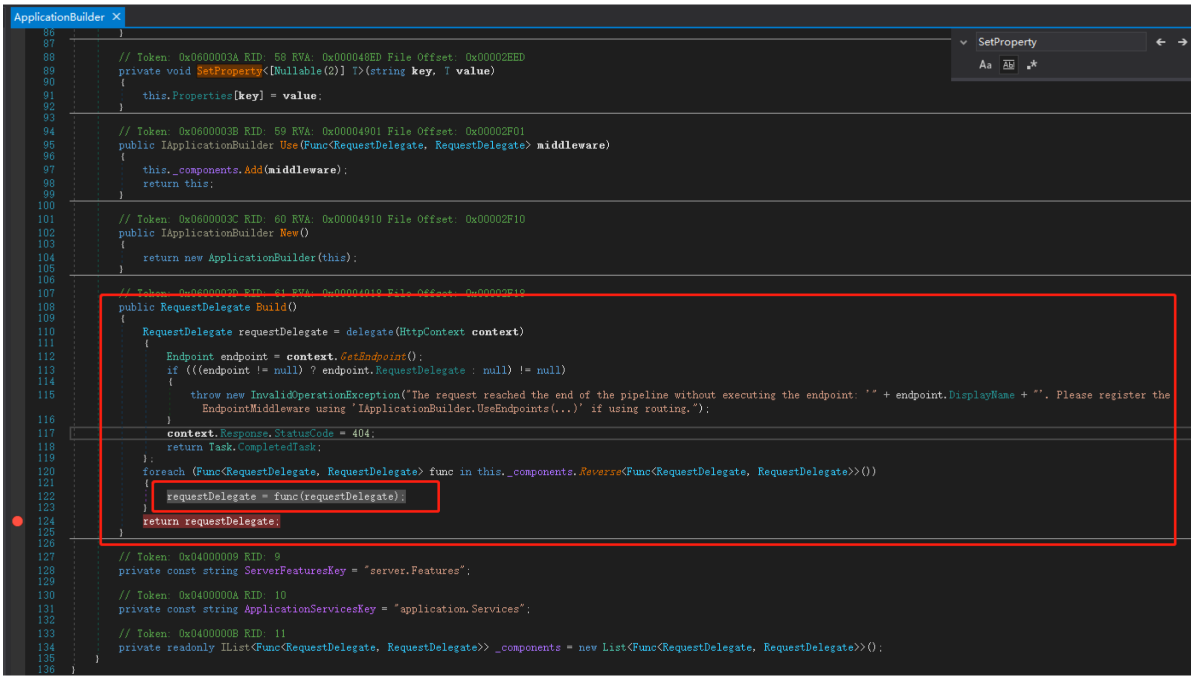

Startup.cs
GenericWebH...dService.cs
杂项文件
D:\学习资料\.net相关\.net5源码\aspnetcore-5.0.11\src\Hosting\Hosting\src\GenericHost\GenericWebHostedService.cs
StartAsync(CancellationToken cancellationToken)
110 %
未找到相关问题
行: 66
字符: 13
空格
LF

```csharp
Action<IApplicationBuilder> configure = Options.ConfigureApplication;

if (configure == null)
{
    throw new InvalidOperationException($"No application configured. Please specify an application via 
      IWebHostBuilder.UseStartup, IWebHostBuilder.Configure, or specifying the startup assembly via {nameof
      (WebHostDefaults.StartupAssemblyKey)} in the web host configuration.");
}

var builder = ApplicationBuilderFactory.CreateBuilder(Server.Features);

foreach (var filter in StartupFilters.Reverse())
{
    configure = filter.Configure(configure);
}

configure(builder);

// Build the request pipeline
application = builder.Build();
}
catch (Exception ex)
{
    Logger.ApplicationError(ex);

    if (!Options.WebHostOptions.CaptureStartupErrors)
    {
        throw;
    }

    application = BuildErrorPageApplication(ex);
}

var httpApplication = new HostingApplication(application, Logger, DiagnosticListener, HttpContextFactory);

await Server.StartAsync(httpApplication, cancellationToken);
```

创建ApplicationBuilder实例

调用Startup类的Configure方法

调用ApplicationBuilder的Build()方法执行中间件



```
        }

        // Token: 0x0600003A RID: 58 RVA: 0x000048ED File Offset: 0x00002EED
        private void SetProperty<[Nullable(2)] T>(string key, T value)
        {
            this.Properties[key] = value;
        }

        // Token: 0x0600003B RID: 59 RVA: 0x00004901 File Offset: 0x00002F01
        public IApplicationBuilder Use(Func<RequestDelegate, RequestDelegate> middleware)
        {
            this._components.Add(middleware);
            return this;
        }

        // Token: 0x0600003C RID: 60 RVA: 0x00004910 File Offset: 0x00002F10
        public IApplicationBuilder New()
        {
            return new ApplicationBuilder(this);
        }

        // Token: 0x0600003D RID: 61 RVA: 0x00004918 File Offset: 0x00002F18
        public RequestDelegate Build()
        {
            RequestDelegate requestDelegate = delegate(HttpContext context)
            {
                Endpoint endpoint = context.GetEndpoint();
                if (((endpoint != null) ? endpoint.RequestDelegate : null) != null)
                {
                    throw new InvalidOperationException("The request reached the end of the pipeline without executing the endpoint: '" + endpoint.DisplayName + "'. Please register the
                      EndpointMiddleware using 'IApplicationBuilder.UseEndpoints(...)' if using routing.");
                }
                context.Response.StatusCode = 404;
                return Task.CompletedTask;
            };
            foreach (Func<RequestDelegate, RequestDelegate> func in this._components.Reverse<Func<RequestDelegate, RequestDelegate>>())
            {
                requestDelegate = func(requestDelegate);
            }
            return requestDelegate;
        }

        // Token: 0x04000009 RID: 9
        private const string ServerFeaturesKey = "server.Features";

        // Token: 0x0400000A RID: 10
        private const string ApplicationServicesKey = "application.Services";

        // Token: 0x0400000B RID: 11
        private readonly IList<Func<RequestDelegate, RequestDelegate>> _components = new List<Func<RequestDelegate, RequestDelegate>>();
    }
}
```