

**Agenda: Working with Middlewares**

- What is Middleware?
- Use, Run, and Map Methods
- Middleware Ordering
- Built-in Middleware
- Writing Custom Middleware
- Work with static files in ASP.NET Core
- Enable directory browsing
- Serve a default document
- Use Fileserver
- Migrating Http Modules to Middleware
- Migrating Http Handler to Middleware

**What is Middleware?**

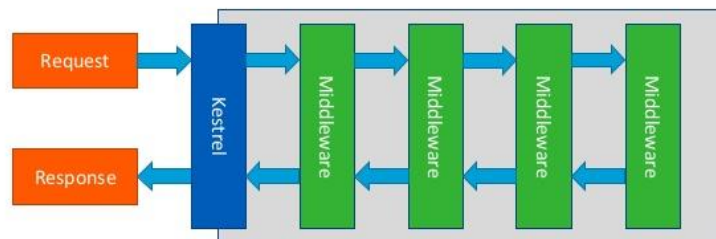
Middleware is software that's assembled into an application pipeline to handle requests and responses.

The ASP.NET Core request pipeline consists of a sequence of request delegates, called one after the other.

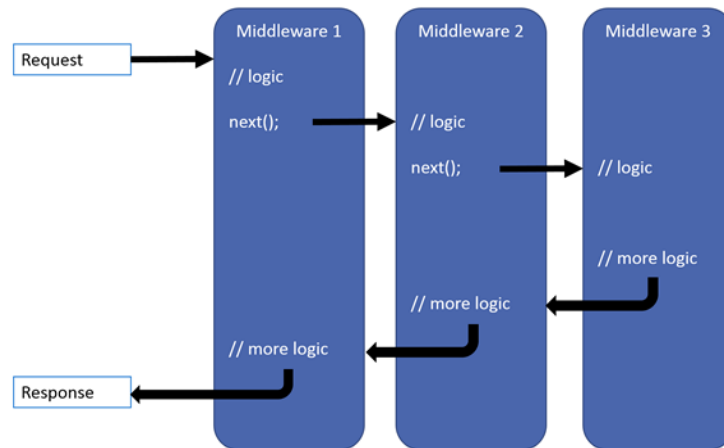
Asp.net Core web application



Kestrel passes the request to a middleware pipeline



© 2016 Microsoft & Co. All Rights Reserved.



#### Each component:

- Can perform work **before** and **after** the next component in the pipeline.
- Chooses whether or **not to pass** the request to the next component in the pipeline (**short-circuiting**). Short-circuiting is often desirable because it avoids unnecessary work. For example, the static file middleware can return a request for a static file and short-circuit the rest of the pipeline.
- Exception-handling delegates need to be called early in the pipeline, so they can catch.

#### Use, Run, and Map

The **Use** method can short-circuit the pipeline (that is, if it doesn't call a **next** request delegate).

**Run** is a convention, and some middleware components may expose `Run[Middleware]` methods that run at the end of the pipeline.

```

public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    app.Use(async (context, next) =>
    {
        // Do work that doesn't write to the Response.
        await context.Response.WriteAsync("Before from Use1!");
        await next.Invoke();
        await context.Response.WriteAsync("After from Use1--");
        // Do logging or other work that doesn't write to the Response.
    });
    app.Use(async (context, next) =>
    {

```

```
// Do work that doesn't write to the Response.
await context.Response.WriteAsync("Before from Use2--");
await next.Invoke();
await context.Response.WriteAsync("After from Use2--");
//await next.Invoke();
// Do logging or other work that doesn't write to the Response.
});
app.Run(async context =>
{
    await context.Response.WriteAsync("From Run--");
});
}
```

**Map\*** extensions are used as a convention **for branching** the pipeline. Map branches the request pipeline based on matches of the given request path. If **the request path starts with the given path, the branch is executed**.

```
public class Startup
{
    private static void HandleMapTest1(IApplicationBuilder app)
    {
        app.Run(async context =>
        {
            await context.Response.WriteAsync("Map Test 1");
        });
    }

    public void Configure(IApplicationBuilder app)
    {
        app.Map("/map1", HandleMapTest1);
        app.Map("/map2", app1 => {
            app1.Run(async context1 =>
            {
                await context1.Response.WriteAsync("Map Test 2");
            });
        });
        app.Run(async context =>
        {

```

```
        await context.Response.WriteAsync("Hello from non-Map delegate. <p>");  
    });  
}  
}
```

The following table shows the requests and responses from `http://localhost:1234` using the previous code:

Request	Response
localhost:1234	Hello from non-Map delegate.
localhost:1234/map1	Map Test 1
localhost:1234/map2	Map Test 2
localhost:1234/map2/demo	Map Test 2
localhost:1234/map3	Hello from non-Map delegate.

### Middleware Ordering

The order that middleware components are added in the `Configure` method defines the order in which they're invoked on requests, and the reverse order for the response. This ordering is critical for security, performance, and functionality.

The `Configure` method (shown below) adds the following middleware components:

1. Exception/error handling
2. Static file server
3. Authentication
4. MVC

```
public void Configure(IApplicationBuilder app)  
{  
    app.UseExceptionHandler("/Home/Error"); // Call first to catch exceptions thrown in the following middleware.  
    app.UseStaticFiles(); // Return static files and end pipeline.  
    app.UseAuthentication(); // Authenticate before you access secure resources.  
    app.UseMvcWithDefaultRoute(); // Add MVC to the request pipeline.  
}
```

Static files are not compressed with this ordering of the middleware.

```
public void Configure(IApplicationBuilder app)  
{  
    app.UseStaticFiles(); // Static files not compressed by middleware.  
    app.UseResponseCompression();  
}
```

```
app.UseMvcWithDefaultRoute();  
}
```

### Writing Custom Middleware

Middleware is generally encapsulated in a class and exposed with an extension method.

```
public class LogURLMiddleware  
{  
    private readonly RequestDelegate _next;  
    public LogURLMiddleware(RequestDelegate next, object o1, object o2)  
    {  
        _next = next;  
    }  
    public Task InvokeAsync(HttpContext context)  
    {  
        //Write code here to Save the URL in database or File  
        // Call the next delegate/middleware in the pipeline  
        return this._next(context);  
    }  
}  
  
public static class LogURLMiddlewareExtensions  
{  
    public static IApplicationBuilder UseLogUrl(this IApplicationBuilder app, object ob1, object ob2)  
    {  
        return app.UseMiddleware<LogURLMiddleware>(ob1, ob2);  
    }  
}
```

### Edit Configure Method

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)  
{  
    //...  
    app.UseLogUrl();  
    //...  
}
```

**Built-in Middleware**

ASP.NET Core ships with the following middleware components, as well as a description of the order in which they should be added:

Middleware	Description	Order
<a href="#">Authentication</a>	Provides authentication support.	Before <code>HttpContext.User</code> is needed.
<a href="#">CORS</a>	Configures Cross-Origin Resource Sharing.	Before components that use CORS.
<a href="#">Diagnostics</a>	Configures diagnostics.	Before components that generate errors.
<a href="#">ForwardedHeaders/HttpOverrides</a>	Forwards proxied headers onto the current request.	Before components that consume the updated fields (examples: Scheme, Host, ClientIP, Method).
<a href="#">Response Caching</a>	Provides support for caching responses.	Before components that require caching.
<a href="#">Response Compression</a>	Provides support for compressing responses.	Before components that require compression.
<a href="#">RequestLocalization</a>	Provides localization support.	Before localization sensitive components.
<a href="#">Routing</a>	Defines and constrains request routes.	Terminal for matching routes.
<a href="#">Session</a>	Provides support for managing user sessions.	Before components that require Session.
<a href="#">Static Files</a>	Provides support for serving static files and directory browsing.	Terminal if a request matches files.
<a href="#">URL Rewriting</a>	Provides support for rewriting URLs and redirecting requests.	Before components that consume the URL.
<a href="#">WebSockets</a>	Enables the WebSockets protocol.	Before components that are required to accept WebSocket requests.

**Work with static files in ASP.NET Core**

Static files are stored within your project's **webroot** directory. The default directory is `<content_root>/wwwroot`, but it can be changed via the [UseWebRoot](#) method. The `WebHost.CreateDefaultBuilder` method sets the content root to the current directory:

If targeting .NET Framework, add the **Microsoft.AspNetCore.StaticFiles** package to your project.

If targeting .NET Core, the **Microsoft.AspNetCore.All** metapackage includes this package.

Invoke the [UseStaticFiles](#) method within `Startup.Configure`

#### To Serve files outside of web root

```
app.UseStaticFiles(new StaticFileOptions
{
    FileProvider = new PhysicalFileProvider(Path.Combine(Directory.GetCurrentDirectory(), "MyStaticFiles")),
    RequestPath = "/StaticFiles"
});
```

#### Enable directory browsing

Directory browsing allows users of your web app to see a directory listing and files within a specified directory.

Directory browsing is **disabled** by default for security reasons

```
app.UseDirectoryBrowser(new DirectoryBrowserOptions
{
    FileProvider = new PhysicalFileProvider(Path.Combine(Directory.GetCurrentDirectory(), "wwwroot", "images")),
    RequestPath = "/MyImages"
});
```

Add required services by invoking the `AddDirectoryBrowser` method from `Startup.ConfigureServices`:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDirectoryBrowser();
}
```

#### Serve a default document

Setting a default home page provides visitors a logical starting point when visiting your site.

```
public void Configure(IApplicationBuilder app)
{
    app.UseDefaultFiles();
    app.UseStaticFiles();
}
```

With `UseDefaultFiles`, requests to a folder search for:

- *default.htm*
- *default.html*
- *index.htm*
- *index.html*

The following code changes the default file name to *mydefault.html*:

```
public void Configure(IApplicationBuilder app)
{
    // Serve my app-specific default file, if present.
    DefaultFilesOptions options = new DefaultFilesOptions();
    options.DefaultFileNames.Clear();
    options.DefaultFileNames.Add("mydefault.html");
    app.UseDefaultFiles(options);
    app.UseStaticFiles();
}
```

### UseFileServer

**UseFileServer** combines the functionality of

1. `UseStaticFiles`
2. `UseDefaultFiles`
3. `UseDirectoryBrowser`

```
app.UseFileServer(enableDirectoryBrowsing: true);
```

The following code enables static files, default files, and directory browsing of `MyStaticFiles`:

```
app.UseFileServer(new FileServerOptions
{
    FileProvider = new PhysicalFileProvider(Path.Combine(Directory.GetCurrentDirectory(), "MyStaticFiles")),
    RequestPath = "/StaticFiles",
    EnableDirectoryBrowsing = true
});
```

Note: **AddDirectoryBrowser** must be called when the **EnableDirectoryBrowsing** property value is true:

```
public void ConfigureServices(IServiceCollection services)
```



```
{  
    services.AddDirectoryBrowser();  
}
```

### Exception Middleware

To configure an app to display a page that shows detailed information about exceptions, use the *Developer Exception Page*.

```
if (env.IsDevelopment())  
{  
    app.UseDeveloperExceptionPage(); //Developer Exception Page  
}  
else  
{  
    app.UseExceptionHandler("/error"); //Custom Error Handling Page  
}
```

### Status Code Middleware

By default, an app doesn't provide a rich status code page for HTTP status codes, such as *404 Not Found*. To provide status code pages, use Status Code Pages Middleware.

```
app.UseStatusCodePages(async context =>  
{  
    context.HttpContext.Response.ContentType = "text/plain";  
    await context.HttpContext.Response.WriteAsync(  
        "Status code page, status code: " +  
        context.HttpContext.Response.StatusCode);  
});  
  
app.UseStatusCodePagesWithRedirects("/error/{0}");  
  
app.MapWhen(context => context.Request.Path == "/missingpage", builder => {  
    builder.Run(async context =>  
    {  
        context.Response.Redirect("/home ");  
    });  
});
```

### Migrating Http Modules with Middleware

**Sample Custom Module**

```
public class MyModule : IHttpModule
{
    public void Init(HttpApplication application)
    {
        application.BeginRequest += (new EventHandler(this.Application_BeginRequest));
        application.EndRequest += (new EventHandler(this.Application_EndRequest));
    }

    private void Application_BeginRequest(Object source, EventArgs e)
    {
        HttpContext context = ((HttpApplication)source).Context;

        // Do something with context near the beginning of request processing.
    }

    private void Application_EndRequest(Object source, EventArgs e)
    {
        HttpContext context = ((HttpApplication)source).Context;

        // Do something with context near the end of request processing.
    }
}
```

**Middleware equivalent of above Module**

```
public class MyMiddleware
{
    private readonly RequestDelegate _next;

    public MyMiddleware(RequestDelegate next)
    {
        _next = next;
    }

    public async Task Invoke(HttpContext context)
    {
        // Do something with context near the beginning of request processing.
        await _next.Invoke(context);
        // Clean up.
    }
}
```

```
    }  
}  
  
public static class MyMiddlewareExtensions  
{  
    public static IApplicationBuilder UseMyMiddleware(this IApplicationBuilder builder)  
    {  
        return builder.UseMiddleware<MyMiddleware>();  
    }  
}
```

### Migrating Http Handler code to middleware

```
public class MyHandler : IHttpHandler  
{  
    public bool IsReusable { get { return true; } }  
    public void ProcessRequest(HttpContext context)  
    {  
        string response = string.Format("Title of the report: {0}", context.Request.QueryString["title"]);  
  
        context.Response.ContentType = "text/plain";  
        context.Response.Output.Write(response);  
    }  
    // ...  
    private string GenerateResponse(HttpContext context)  
    {  
        string title = context.Request.QueryString["title"];  
        return string.Format("Title of the report: {0}", title);  
    }  
}
```

Following is equivalent Middleware

```
public class MyHandlerMiddleware  
{  
    // Must have constructor with this signature, otherwise exception at run time
```

```
public MyHandlerMiddleware(RequestDelegate next)
{
    // This is an HTTP Handler, so no need to store next
}

public async Task Invoke(HttpContext context)
{
    var response = string.Format("Title of the report: {0}", context.Request.Query["title"]);
    context.Response.ContentType = "text/plain";
    await context.Response.WriteAsync(response);
}
}

public static class MyHandlerExtensions
{
    public static IApplicationBuilder UseMyHandler(this IApplicationBuilder builder)
    {
        return builder.UseMiddleware<MyHandlerMiddleware>();
    }
}
```

### Insert Middleware into the request pipeline

```
// Create branch to the MyHandlerMiddleware.
// All requests ending in .report will follow this branch.
app.MapWhen(
    context => context.Request.Path.ToString().EndsWith(".report"),
    appBranch =>
    {
        // ... optionally add more middleware to this branch
        appBranch.UseMyHandler();
    });
```