ASP.NET Core : Middleware

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

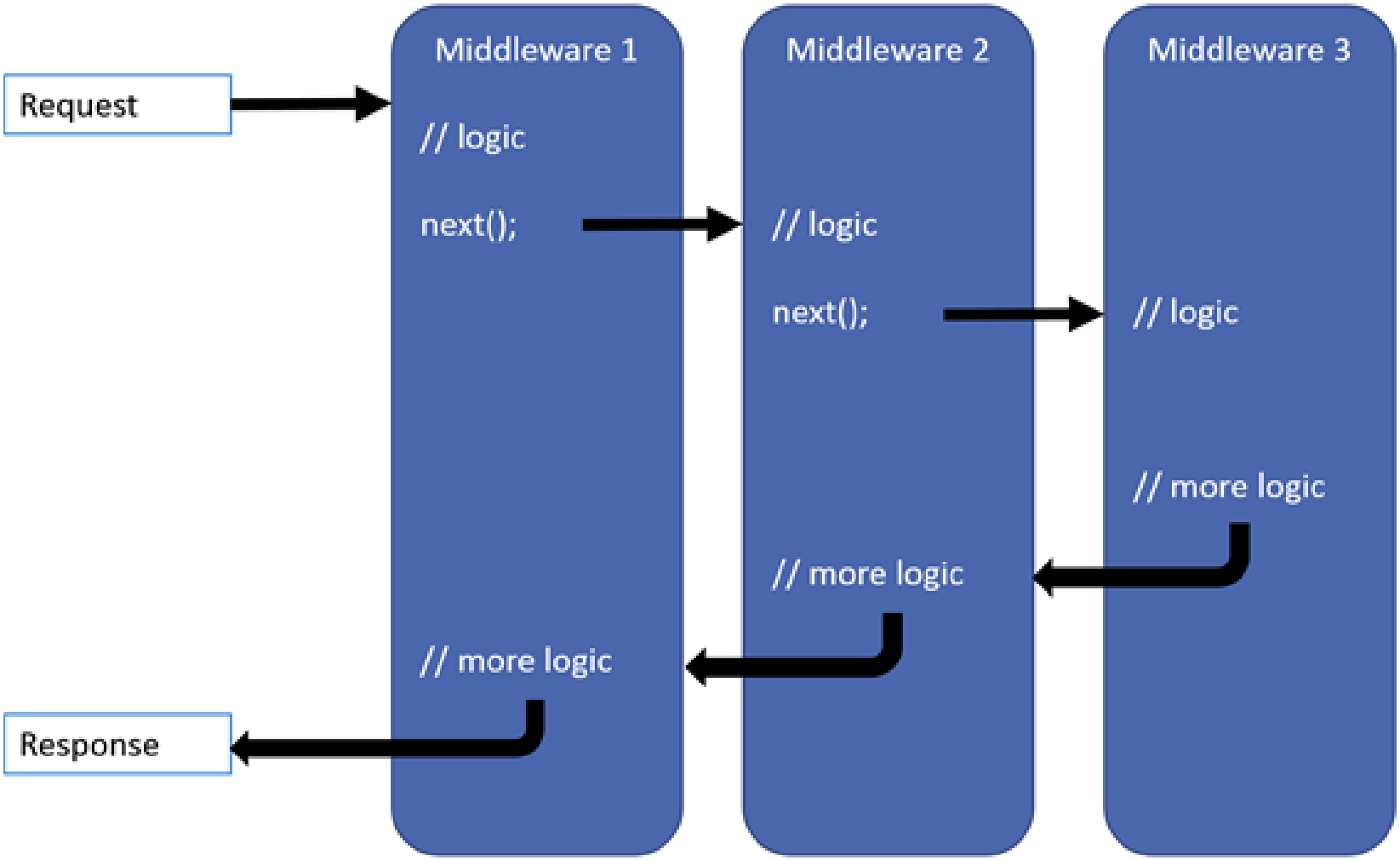
Plan

ASP.NET Core

Middleware

- Traduit souvent en français en `Intergiciel`.
- Composant = { classes **C#** }
- Ayant accès aux objets de type `Request` et `Response`.
- Pouvant appelé le middleware suivant dans le pipeline (désigné souvent par un délégué nommé `next`).
- Pouvant s'exécuter avant et/ou après le middleware suivant dans le pipeline.
- Configurable avec trois extensions : `Run`, `Use` ou `Map`.

ASP.NET Core



Source : *documentation officielle*

ASP.NET Core

Créons un nouveau projet

- Faire un clic droit sur la solution `CoursAspNetCore`
- Aller dans `Ajouter > Nouveau projet...`
- Choisir `C#` dans `Tous les langages`
- Sélectionner `ASP.NET Core Vide`
- Cliquer sur `Suivant`
- Saisir `CoursMiddleware` dans `Nom`
- Valider

ASP.NET Core

Code généré de `Program.cs`

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.MapGet("/", () => "Hello World!");

app.Run();
```

ASP.NET Core

Code généré de `Program.cs`

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.MapGet("/", () => "Hello World!");

app.Run();
```


Lancez le projet et vérifier que `Hello World!` s'affiche.

ASP.NET Core

Définissons notre premier middleware

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.Run(async context =>
{
    await context.Response.WriteAsync("Hello World!");
});

app.Run();
```

ASP.NET Core

Définissons notre premier middleware

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.Run(async context =>
{
    await context.Response.WriteAsync("Hello World!");
});

app.Run();
```

Relancez le projet et vérifier que `Hello World!` s'affiche.

ASP.NET Core

Explication

Le paramètre `context` contient les deux objets `Response` et `Request`.

ASP.NET Core

Définissons deux middlewares

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.Run(async context =>
{
    await context.Response.WriteAsync("Hello World! 1");
});

app.Run(async context =>
{
    await context.Response.WriteAsync("Hello World! 2");
});

app.Run();
```

ASP.NET Core

Définissons deux middlewares

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.Run(async context =>
{
    await context.Response.WriteAsync("Hello World! 1");
});

app.Run(async context =>
{
    await context.Response.WriteAsync("Hello World! 2");
});

app.Run();
```

Relancez le projet et vérifier que le seul message qui s'affiche est `Hello World! 1`.

ASP.NET Core

Explication

- Le middleware doit utiliser le délégué `next` pour appeler le deuxième middleware.
- Pour utiliser `next`, il faut remplacer `Run` par `Use`.

ASP.NET Core

Modifions le premier middleware

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.Use(async (context, next) =>
{
    await context.Response.WriteAsync("Hello World! 1");
    await next.Invoke();
});

app.Run(async context =>
{
    await context.Response.WriteAsync("Hello World! 2");
});

app.Run();
```

ASP.NET Core

Modifions le premier middleware

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.Use(async (context, next) =>
{
    await context.Response.WriteAsync("Hello World! 1");
    await next.Invoke();
});

app.Run(async context =>
{
    await context.Response.WriteAsync("Hello World! 2");
});

app.Run();
```

Relancez le projet et vérifier que `Hello World! 1Hello World! 2` s'affiche.

ASP.NET Core

`next.Invoke()` **peut être remplacé par le raccourci** `next()`

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.Use(async (context, next) =>
{
    await context.Response.WriteAsync("Hello World! 1");
    await next();
});

app.Run(async context =>
{
    await context.Response.WriteAsync("Hello World! 2");
});

app.Run();
```

ASP.NET Core

`next.Invoke()` **peut être remplacé par le raccourci** `next()`

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.Use(async (context, next) =>
{
    await context.Response.WriteAsync("Hello World! 1");
    await next();
});

app.Run(async context =>
{
    await context.Response.WriteAsync("Hello World! 2");
});

app.Run();
```

Relancez le projet et vérifier que `Hello World! 1Hello World! 2` s'affiche.

Un middleware peut exécuter d'autres instructions après la réponse d'un second middleware qu'il a appelé

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.Use(async (context, next) =>
{
    await context.Response.WriteAsync("Hello World! 1");
    await next();
    await context.Response.WriteAsync("Hello World! 3");
});

app.Run(async context =>
{
    await context.Response.WriteAsync("Hello World! 2");

});

app.Run();
```

Un middleware peut exécuter d'autres instructions après la réponse d'un second middleware qu'il a appelé

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.Use(async (context, next) =>
{
    await context.Response.WriteAsync("Hello World! 1");
    await next();
    await context.Response.WriteAsync("Hello World! 3");
});

app.Run(async context =>
{
    await context.Response.WriteAsync("Hello World! 2");

});

app.Run();
```

Relancez le projet et vérifier que `Hello World! 1Hello World! 2Hello World! 3` s'affiche.

ASP.NET Core

`Map` permet

- de créer une branche dans le pipeline,
- d'associer (mapper) un middleware à un chemin.

ASP.NET Core

Définissons un middleware qui s'exécute lorsque le chemin demandé est `check`

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.Map("/check", Check);

app.Run(async (context) =>
{
    await context.Response.WriteAsync("Hello World!");
});

app.Run();

static void Check(IApplicationBuilder app)
{
    app.Run(async (context) =>
    {
        if (context.Request.Query.ContainsKey("nom"))
        {
            await context.Response.WriteAsync($"Bonjour {context.Request.Query["nom"]}");
        }
        else
        {
            await context.Response.WriteAsync("Bonjour doe");
        }
    });
}
```

Java

Requête	Réponse
""	Hello World!
"/check?nom=wick"	Bonjour wick
"/check"	Bonjour doe

ASP.NET Core

Il est possible d'imbriquer les `Map`

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.Map("/check", level =>
{
    level.Map("/nom", CheckNom);
    level.Map("/prenom", CheckPrenom);
});

app.Run(async (context) =>
{
    await context.Response.WriteAsync("Hello World!");
});

app.Run();
```

ASP.NET Core

N'oublions pas de définir `CheckNom` **et** `CheckPrenom`

```
static void CheckNom(IApplicationBuilder app)
{
    app.Run(async (context) =>
    {
        if (context.Request.Query.ContainsKey("value"))
        {
            await context.Response.WriteAsync($"Bonjour {context.Request.Query["value"]}");
        }
        else
        {
            await context.Response.WriteAsync("Bonjour doe");
        }
    });
}
static void CheckPrenom(IApplicationBuilder app)
{
    app.Run(async (context) =>
    {
        if (context.Request.Query.ContainsKey("value"))
        {
            await context.Response.WriteAsync($"Bonjour {context.Request.Query["value"]}");
        }
        else
        {
            await context.Response.WriteAsync("Bonjour john");
        }
    });
}
```

Java

Requête	Réponse
"/nom?value=wick"	Bonjour wick
"/nom"	Bonjour doe
"/prenom?value=jack"	Bonjour jack
"/prenom"	Bonjour john

ASP.NET Core

`Map` permet

- de créer une branche dans le pipeline,
- d'associer (mapper) un middleware à une condition.

ASP.NET Core

Définissons un middleware qui s'exécute lorsque le chemin demandé contient le paramètre `nom`

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.MapWhen(context => context.Request.Query.ContainsKey("nom"), Check);

app.Run(async (context) =>
{
    await context.Response.WriteAsync("Hello World!");
});

app.Run();

static void Check(IApplicationBuilder app)
{
    app.Run(async (context) =>
    {
        await context.Response.WriteAsync($"Bonjour {context.Request.Query["nom"]}");
    });
}
```

Java

Requête	Réponse
""	Hello World!
"/check?nom=wick"	Bonjour wick
"?nom=wick"	Bonjour wick

ASP.NET Core

Middleware personnalisé

- Classe **C#**
- Ayant un constructeur `public` avec un paramètre de type `RequestDelegate`.
- Une méthode `public`
  - nommée `Invoke` ou `InvokeAsync`
  - retournant une `Task`.
  - acceptant un premier paramètre de type `HttpContext`.

ASP.NET Core

Middleware personnalisé

- Classe **C#**
- Ayant un constructeur `public` avec un paramètre de type `RequestDelegate`.
- Une méthode `public`
  - nommée `Invoke` ou `InvokeAsync`
  - retournant une `Task`.
  - acceptant un premier paramètre de type `HttpContext`.

Des paramètres supplémentaires pourront être rajoutés pour le constructeur et `Invoke`/`InvokeAsync` et seront remplis par Injection de dépendance.

ASP.NET Core

Créons un middleware

- Faire un clic droit sur le projet
- Aller dans `Ajouter > Nouvel élément...`
- Choisir `C#` dans `Tous les langages`
- Chercher puis sélectionner `Classe d'intergiciel (middleware)`
- Saisir `MonMiddleware` dans `Nom`
- Valider

Code généré

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Http;
using System.Threading.Tasks;

namespace CoursMiddleware
{
    public class MonMiddleware
    {
        private readonly RequestDelegate _next;

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

        public Task Invoke(HttpContext httpContext)
        {

            return _next(httpContext);
        }
    }

    // Extension method used to add the middleware to the HTTP request pipeline.
    public static class MonMiddlewareExtensions
    {
        public static IApplicationBuilder UseMonMiddleware(this IApplicationBuilder
           builder)
        {
            return builder.UseMiddleware<MonMiddleware>();
        }
    }
}
```

On peut remplacer `Invoke` **par** `InvokeAsync`

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Http;
using System.Threading.Tasks;

namespace CoursMiddleware
{
    public class MonMiddleware
    {
        private readonly RequestDelegate _next;

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

        public async Task InvokeAsync(HttpContext httpContext)
        {
            await _next(httpContext);
        }
    }

    // Extension method used to add the middleware to the HTTP request pipeline.
    public static class MonMiddlewareExtensions
    {
        public static IApplicationBuilder UseMonMiddleware(this IApplicationBuilder
           builder)
        {
            return builder.UseMiddleware<MonMiddleware>();
        }
    }
}
```

ASP.NET Core

Ajoutons deux messages à la réponse avant et après `_next`

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Http;
using System.Threading.Tasks;

namespace CoursMiddleware
{
    public class MonMiddleware
    {
        private readonly RequestDelegate _next;

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

        public async Task InvokeAsync(HttpContext httpContext)
        {
            await httpContext.Response.WriteAsync("Avant next de MonMiddleware");
            await _next(httpContext);
            await httpContext.Response.WriteAsync("Après next de MonMiddleware");
        }
    }

    // Extension method used to add the middleware to the HTTP request pipeline.
        // Le code précédent reste inchangé
}
```

ASP.NET Core

Utilisons `MonMiddleware` **dans** `Program.cs`

```
using CoursMiddleware;

var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.Use(async (context, next) =>
{
    await context.Response.WriteAsync("Hello World! 1");
    await next();
    await context.Response.WriteAsync("Hello World! 3");
});

app.UseMonMiddleware();

app.Run(async context =>
{
    await context.Response.WriteAsync("Hello World! 2");
});

app.Run();
```

ASP.NET Core

Utilisons `MonMiddleware` **dans** `Program.cs`

```
using CoursMiddleware;

var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.Use(async (context, next) =>
{
    await context.Response.WriteAsync("Hello World! 1");
    await next();
    await context.Response.WriteAsync("Hello World! 3");
});

app.UseMonMiddleware();

app.Run(async context =>
{
    await context.Response.WriteAsync("Hello World! 2");
});

app.Run();
```

Relancez le projet et vérifier que `Hello World! 1Avant next de MonMiddlewareHello World! 2Après next de MonMiddlewareHello World! 3` s'affiche.

Nous pouvons aussi ajouter un deuxième paramètre au constructeur qui sera injecté par ASP.NET

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Http;
using System.Threading.Tasks;

namespace CoursMiddleware
{
    public class MonMiddleware
    {
        private readonly RequestDelegate _next;
        private readonly ILogger _logger;

        public MonMiddleware(RequestDelegate next, ILoggerFactory logFactory)
        {
            _next = next;

            _logger = logFactory.CreateLogger("MonMiddleware");
        }

        public async Task InvokeAsync(HttpContext httpContext)
        {
            _logger.LogInformation("Avant next de MonMiddleware");
            await httpContext.Response.WriteAsync("Avant next de MonMiddleware");
            await _next(httpContext);
            _logger.LogInformation("Après next de MonMiddleware");
            await httpContext.Response.WriteAsync("Après next de MonMiddleware");
        }
    }

    // Extension method used to add the middleware to the HTTP request pipeline.
        // Le code précédent reste inchangé
}
```

Rien à changer dans `Program.cs`

```
using CoursMiddleware;

var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.Use(async (context, next) =>
{
    await context.Response.WriteAsync("Hello World! 1");
    await next();
    await context.Response.WriteAsync("Hello World! 3");
});

app.UseMonMiddleware();

app.Run(async context =>
{
    await context.Response.WriteAsync("Hello World! 2");
});

app.Run();
```

Rien à changer dans `Program.cs`

```
using CoursMiddleware;

var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.Use(async (context, next) =>
{
    await context.Response.WriteAsync("Hello World! 1");
    await next();
    await context.Response.WriteAsync("Hello World! 3");
});

app.UseMonMiddleware();

app.Run(async context =>
{
    await context.Response.WriteAsync("Hello World! 2");
});

app.Run();
```

Relancez le projet et vérifier que

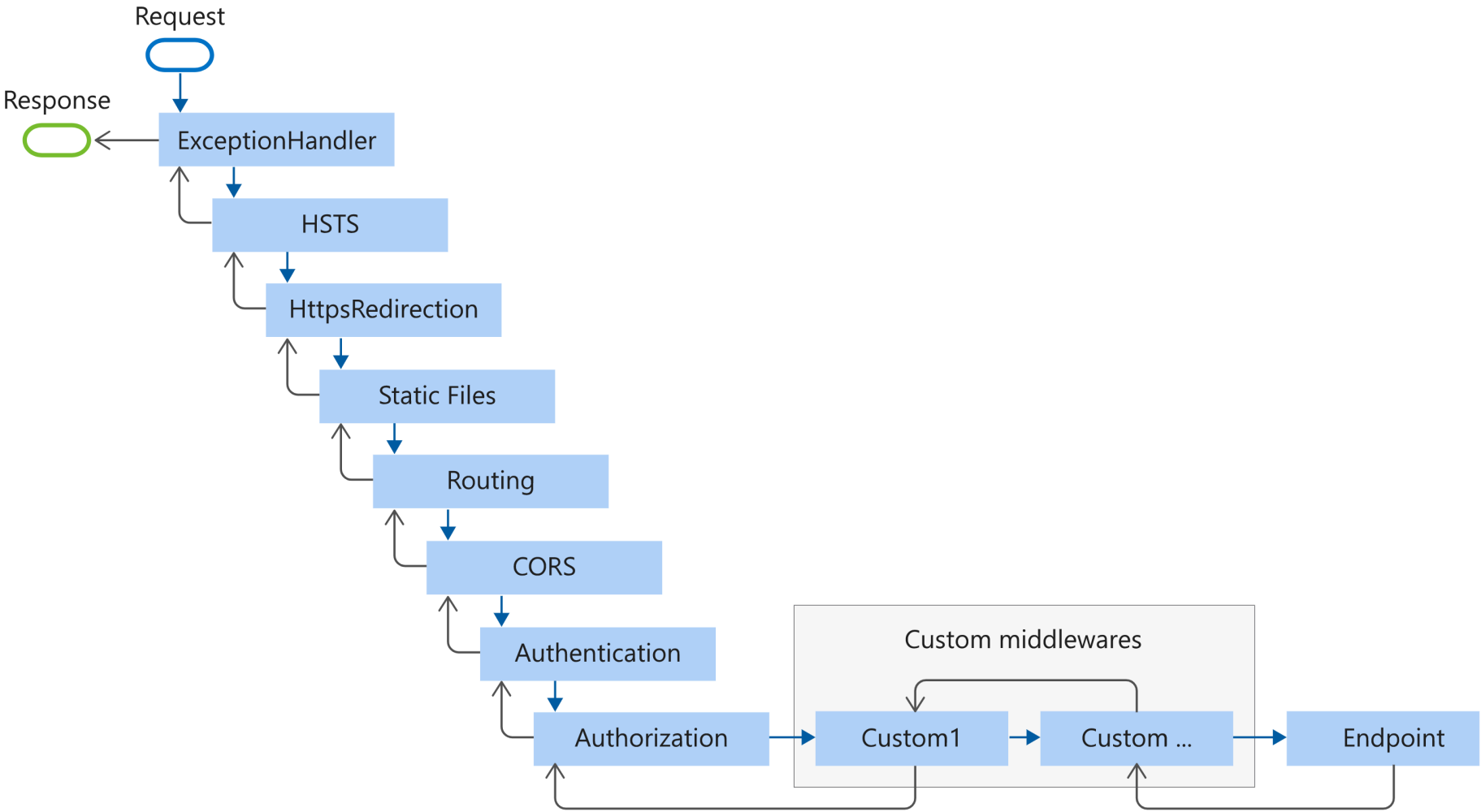
- `Hello World! 1Avant next de MonMiddlewareHello World! 2Après next de MonMiddlewareHello World! 3` s'affiche dans le navigateur, et
- `Avant next de MonMiddleware Après next de MonMiddleware` s'affiche dans la console.

ASP.NET Core

ASP.NET Core propose un ensemble de middleware prédéfinis

- `ExceptionHandler` : intercepte les exceptions levées dans les middlewares suivants.
- `Hsts` (**H**TTP **S**trict **T**ransport **S**ecurity) : améliore la sécurité en ajoutant un en-tête de réponse spécial (**Strict-Transport-Security**).
- `HttpsRedirection` : Redirige toutes les requêtes **HTTP** vers **HTTPS**.
- `Routing` : Contrôle les routes de requête.
- `Cors` : Configure le partage des ressources cross-origin (CORS).
- `Authentication` : Prend en charge l'authentification.
- `Authorization` : Fournit la prise en charge des autorisations.
- ...

ASP.NET Core



Source : *documentation officielle*