Contents

[Generic host (HostBuilder) //TODO 1](#_Toc158826339)

[Configuration //TODO 1](#_Toc158826340)

[Dependency injection //TODO 1](#_Toc158826341)

[Options pattern 3](#_Toc158826342)

[Logging //TODO 3](#_Toc158826343)

[Hosted service // TODO 3](#_Toc158826344)

[ASP .NET Core 3](#_Toc158826345)

[ASP .NET Core vs ASP .NET Core MVC 3](#_Toc158826346)

[Kestrel vs IIS 4](#_Toc158826347)

[Фильтры 4](#_Toc158826348)

[Свой фильтр 5](#_Toc158826349)

[Глобальные фильтры 5](#_Toc158826350)

[Установка отдельно для контроллеров: 5](#_Toc158826351)

[Filter vs Middleware 6](#_Toc158826352)

[Отличия MVC от ASP .NET 6](#_Toc158826353)

Generic host (HostBuilder) //TODO

Просто все отсюда: <https://docs.microsoft.com/en-us/dotnet/core/extensions/generic-host>

Configuration //TODO

<https://docs.microsoft.com/en-us/dotnet/core/extensions/configuration>

Appsettings - <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration/?view=aspnetcore-6.0>

Dependency injection //TODO

**Construction Injection**

Зависимости нужны на этапе создания, чтобы проинициализировать объекты. Т.е. конструктор использует параметры для внедрения зависимостей.

Плюсы/минусы

зависимость можно сделать immutable (неизменяемой), чтобы предотвратить повторное внедрение зависимости

все зависимости должны быть переданы в конструкторе

**Setter Injection**

Для внедрения сложных зависимостей в указанный момент

Плюсы/минусы

не нужно все прописывать в конструкторе

всегда надо отслеживать, какая зависимость нужна, чтоб подгрузить ее

**Interface Based Injection**

…

DI - <https://docs.microsoft.com/en-us/dotnet/core/extensions/dependency-injection>

Типы регистрации сервисов:

<https://docs.microsoft.com/en-us/dotnet/core/extensions/dependency-injection#service-registration-methods>

| **Method** | **Automatic object disposal** | **Multiple implementations** | **Pass args** |
| --- | --- | --- | --- |
| Add{LIFETIME}<{SERVICE}, {IMPLEMENTATION}>()  Example:  services.AddSingleton<IMyDep, MyDep>(); | Yes | Yes | No |
| Add{LIFETIME}<{SERVICE}>(sp => new {IMPLEMENTATION})  Examples:  services.AddSingleton<IMyDep>(sp => new MyDep()); services.AddSingleton<IMyDep>(sp => new MyDep(99)); | Yes | Yes | Yes |
| Add{LIFETIME}<{IMPLEMENTATION}>()  Example:  services.AddSingleton<MyDep>(); | Yes | No | No |
| AddSingleton<{SERVICE}>(new {IMPLEMENTATION})  Examples:  services.AddSingleton<IMyDep>(new MyDep()); services.AddSingleton<IMyDep>(new MyDep(99)); | No | Yes | Yes |
| AddSingleton(new {IMPLEMENTATION})  Examples:  services.AddSingleton(new MyDep()); services.AddSingleton(new MyDep(99)); |  |  |  |

IOC - <https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/architectural-principles#dependency-inversion>

* [TryAdd](https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.dependencyinjection.extensions.servicecollectiondescriptorextensions.tryadd)
* [TryAddTransient](https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.dependencyinjection.extensions.servicecollectiondescriptorextensions.tryaddtransient)
* [TryAddScoped](https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.dependencyinjection.extensions.servicecollectiondescriptorextensions.tryaddscoped)
* [TryAddSingleton](https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.dependencyinjection.extensions.servicecollectiondescriptorextensions.tryaddsingleton)

один интерфейс можно регистрировать для разных сервисов

Options pattern

<https://docs.microsoft.com/en-us/dotnet/core/extensions/options>

Logging //TODO

<https://docs.microsoft.com/en-us/dotnet/core/extensions/logging?tabs=command-line>

Hosted service // TODO

<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/host/hosted-services?view=aspnetcore-6.0&tabs=visual-studio>

ASP .NET Core

ASP .NET Core vs ASP .NET Core MVC

<https://docs.microsoft.com/en-us/dotnet/architecture/porting-existing-aspnet-apps/middleware-modules-handlers>

You must remember how you differentiate between a normal MVC Controller and a Web API Controller...

WebAPI Controllers enforces this annotation [ApiController] and must inherits from ControllerBase:

[ApiController]

public class PeopleController : ControllerBase {

//Your API methods here

}

A normal MVC Controller only inherits from Controller base class:

public class PeopleController : Controller {

//Your Action methods here...

}

Those already create configuration for your APP which becomes easier for you Hosting environment to know what is going and what to return when.

Kestrel vs IIS

<https://www.tutorialspoint.com/what-is-kestrel-and-how-does-it-differ-from-iis-asp-net>

Фильтры

IAuthorizationFilter

Запускается первым из всех. Проверяет, есть ли у юзера все права для доступа к методу. (проверяет авторизацию)

IResourceFilter

wЗапускается вторым.

Методы

OnResourceExecuting - код в нем запустится после AuthorizationFilter, но до других фильтров.

OnResourceExecuted - выполнится последним, после выполнения всего пайплайна приложения.

IActionFilter

OnActionExecuting - вызвется во время выполнения Action-а

OnActionExecuted - после выполнения

IPageFilter

IExceptionFilter

Фильтрует ошибки до того, как они заносятся в тело ответа.

Может отловить все ошибки кроме авторизационных.

IResultFilter

Запускается сразу до и после Action метода.

Сам фильтр запускается только если Action метод выполнен успешно

Глобальные фильтры выполняются первыми (заданные в Startup),

затем фильтры контроллера,

затем фильтры Action-а

Для смены очередности фильтров, необходимо в фильтре реализовать интерфейс IOrderedFilter

В нем public int Order => <номер>

Свой фильтр

public class SimpleResourceFilter : Attribute, IResourceFilter

{

public void OnResourceExecuting(ResourceExecutingContext context)

{

var a = context.ActionArguments["name"];

if (a == "test")

<do something>

context.HttpContext.Response.Cookies.Append("LastVisit", DateTime.Now.ToString("dd/MM/yyyy hh-mm-ss"));

}

public void OnResourceExecuted(ResourceExecutedContext context)

{

// реализация отсутствует

}

}

Глобальные фильтры

public void ConfigureServices(IServiceCollection services)

{

// глобально - все сервисы MVC - и контроллеры, и Razor Page

services.AddMvc(options =>

{

options.Filters.Add(typeof(SimpleResourceFilter)); // подключение по типу

// альтернативный вариант подключения

//options.Filters.Add(new SimpleResourceFilter()); // подключение по объекту

});

}

Установка отдельно для контроллеров:

public void ConfigureServices(IServiceCollection services)

{

services.AddControllersWithViews(options=>

{

options.Filters.Add(new SimpleResourceFilter()); // подключение по объекту

options.Filters.Add(typeof(SimpleResourceFilter)); // подключение по типу

});

}

Чтобы прервать пайплайн фильтров, надо присвоить значение Result

public void OnResourceExecuting(ResourceExecutingContext context)

{

context.Result = new ContentResult { Content = "Ресурс не найден" };

}

Конвейер обработки запроса и middleware

Компоненты конвейера, которые отвечают за обработку запроса, называются middleware.

Для подключения компонентов middleware используется метод Configure из класса Startup.

Компоненты middleware конфигурируются с помощью методов расширений Run, Map и Use объекта IApplicationBuilder, который передается в метод Configure() класса Startup

Свой middleware

Класс middleware должен иметь конструктор, который принимает параметр типа RequestDelegate. Через этот параметр можно получить ссылку на тот делегат запроса, который стоит следующим в конвейере обработки запроса.

Также в классе должен быть определен метод, который должен называться либо Invoke, либо InvokeAsync. Причем этот метод должен возвращать объект Task и принимать в качестве параметра контекст запроса - объект HttpContext. Данный метод собственно и будет обрабатывать запрос.

public class TokenMiddleware

{

private readonly RequestDelegate \_next;

public TokenMiddleware(RequestDelegate next)

{

this.\_next = next;

}

public async Task InvokeAsync(HttpContext context)

{

var token = context.Request.Query["token"];

if (token!="12345678")

{

context.Response.StatusCode = 403;

await context.Response.WriteAsync("Token is invalid");

}

else

{

await \_next.Invoke(context); <---------- Вызов следующего мидлвеера

}

}

}

public class Startup

{

public void Configure(IApplicationBuilder app)

{

app.UseMiddleware<TokenMiddleware>(); <------------- добавление в пайплайн

app.Run(async (context) =>

{

await context.Response.WriteAsync("Hello World");

});

}

}

Filter vs Middleware

Middleware operate on the level of ASP.NET Core and can act on every single request that comes in to the application.

Filters only run for requests that come to the endpoint that has the filter attached to it.

Отличия MVC от ASP .NET

The HttpRequestMessage exists only inside the "hosting" environment, web browser client does not know anything about that.

In the "hosting" world, IIS app pool is running the code you have built and deployed which knows very well wich framewok you are using as your code also contains the using assemblies you listed, System.Web... or System.Net...

Consider that even if you have shown separation between hosting, Controller and Action, all of that is running in same App Pool in IIS which, again, runs your code so knows what it is about as your IL assemblies were built from your specific source code.

IActionResult

Объекты типа IActionResult непосредственно предназначены для генерации результата действия.

Интерфейс IActionResult находится в пространстве имен Microsoft.AspNetCore.Mvc и определяет один метод:

public interface IActionResult

{

Task ExecuteResultAsync(ActionContext context);

}

Свой ActionResult

public class HtmlResult : IActionResult

{

string htmlCode;

public HtmlResult(string html)

{

htmlCode = html;

}

public async Task ExecuteResultAsync(ActionContext context)

{

string fullHtmlCode = "<!DOCTYPE html><html><head>";

fullHtmlCode += "<title>Главная страница</title>";

fullHtmlCode += "<meta charset=utf-8 />";

fullHtmlCode += "</head> <body>";

fullHtmlCode += htmlCode;

fullHtmlCode += "</body></html>";

await context.HttpContext.Response.WriteAsync(fullHtmlCode);

}

}