Конспект «ASP.NET Core in Action»

Теги: `#edu/summary` `#dev`

Ссылки

- Manning Publications — книга
- Obsidian — markdown редактор

Содержание

Структура ASP.NET Core приложения

Program.cs

- Application settings (connection strings, credentials, ...)
- Logging
- HTTP server (Kestrel)
- Content root
- IIS integration

Редко изменяется (инфраструктура, конфигурация запуска, ...).

The `IHostBuilder` created in `Program` calls `ConfigureServices` and then `Configure`.

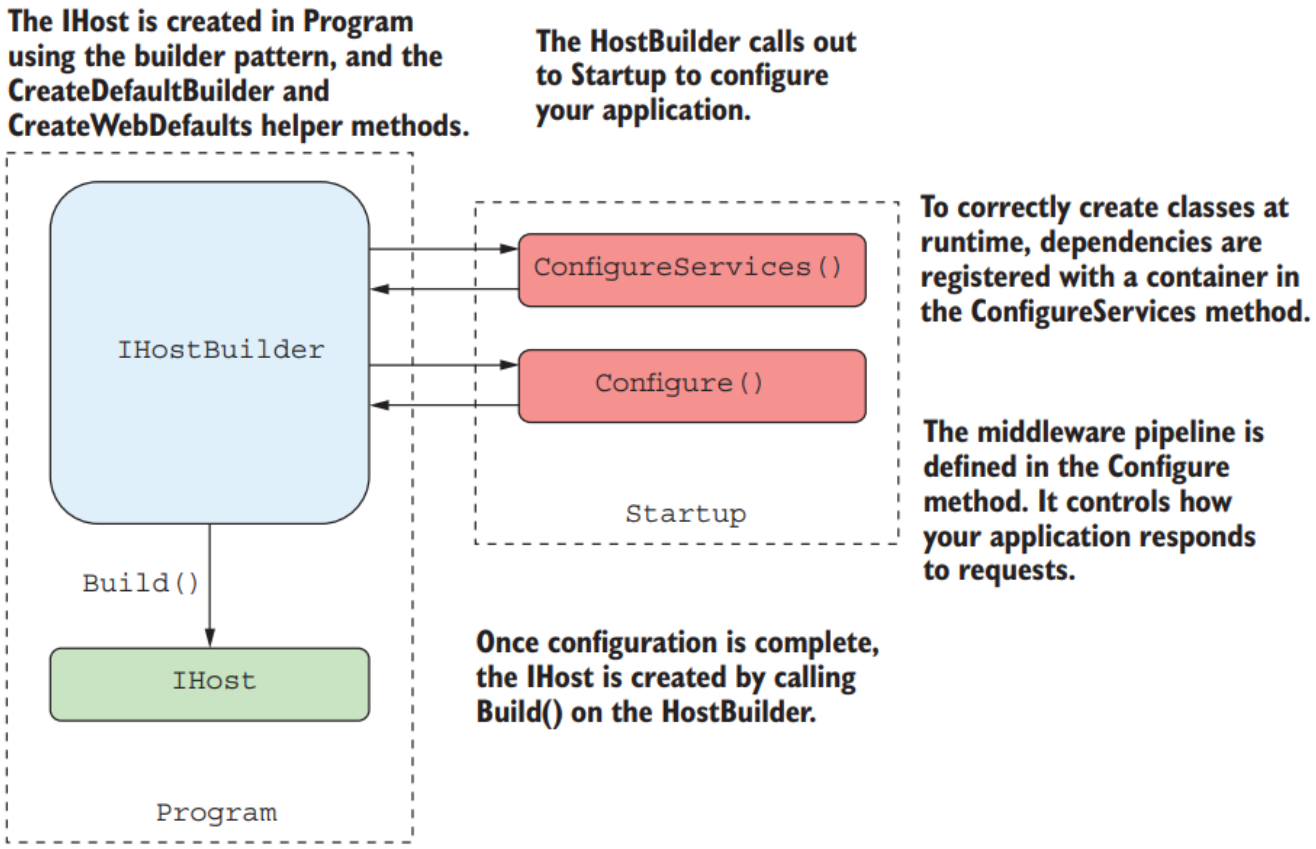


Figure 2.10 The `IHostBuilder` is created in Program.cs and calls methods on `Startup` to configure the application's services and middleware pipeline. Once configuration is complete, the `IHost` is created by calling `Build()` on the `IHostBuilder`.

Startup.cs

- Service registration (dependency injection)
- Middleware pipeline (middleware and endpoints)
- Endpoint configuration

Меняется при добавлении новых фич и корректировке кастомного поведения программы.

`Startup` doesn't implement an interface as such. Instead, the methods are invoked by using reflection.

Listing 2.6 `Startup.Configure`: defining the middleware pipeline

```
public class Startup
{
    public void Configure(
        IApplicationBuilder app,
        IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }
        else
        {
            app.UseExceptionHandler("/Error");
            app.UseHsts();
        }

        app.UseHttpsRedirection();

        app.UseStaticFiles();

        app.UseRouting();
        app.UseAuthorization();

        app.UseEndpoints(endpoints =>
        {
            endpoints.MapRazorPages();
        });
    }
}
```

IApplicationBuilder is used to build the middleware pipeline.

Other services can be accepted as parameters.

Different behavior when in development or production

Only runs in a development environment

Only runs in a production environment

Adds the static-file middleware

Adds the endpoint routing middleware, which determines which endpoint to execute

Adds the authorization middleware, which can block access to specific pages as required

Adds the endpoint middleware, which executes a Razor Page to generate an HTML response

Razor Pages

Client → Request → IIS → Routing → Dynamic Razor Page (C# + HTML = .cshtml) + Static CSS and other resources → Rendering → HTML + CSS + other resources → Response body → IIS → Client.

Middleware

Middleware are C# classes that can handle an HTTP request or response via `HttpContext` object. Middleware can:

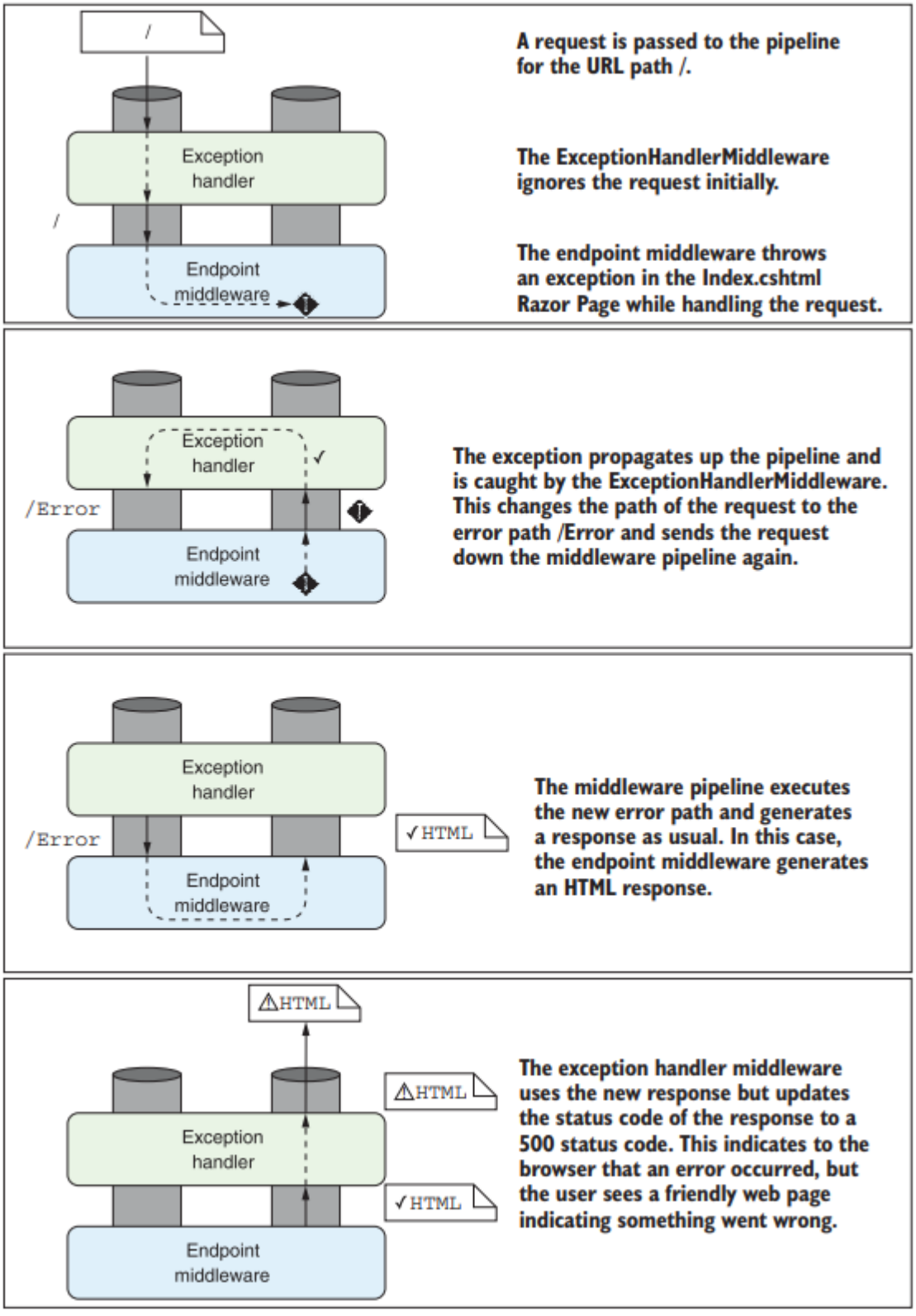
- Handle an incoming HTTP request by generating an HTTP response
- Process an incoming HTTP request, modify it, and pass it on to another piece of middleware
- Process an outgoing HTTP response, modify it, and pass it on to either another piece of middleware or the ASP.NET Core web server

Обработка исключений

Изначально исключение прерывает ход выполнения конвеера на middleware, в котором возникло и возвращает (происходит re-throw из низших middlewares в высшие) пользователю ответ со статус-кодом (4xx для проблем на стороне клиента, 5xx — на стороне сервера) и пустым телом. Пустое тело ⇒ браузер отобразит стандартное окно об ошибке с указанием статус-кода.

Чтобы интегрировать отображение ошибок в UI/UX приложения, используется re-executing: когда возникает исключение, оно пробрасывается по конвееру снизу вверх и ловится try-catch-конструкцией в middleware, ответственном за обработку исключений. Middleware, в свою очередь, убирает статус-код ошибки из формирующегося ответа, очищает тело ответа (которое могло быть наполнено сгенерированным низшими middlewares контентом) и присваивает адресу запроса адрес, ответственный за отображение ошибок (например, `/Error`). Таким образом, middleware, обрабатывающий исключение «перезапускает» конвеер, заново заставляя его обработать запрос, только уже нацеленный не на адрес, обработка которого может выбросить исключение, а условный `/Error` с оформленной под стиль сайта страницей отображения user-friendly описания ошибки.

Если не только executing, но и re-executing выбросил исключение (например, во время обработки запроса `/Error`), обрабатывающий исключения middleware вернёт клиенту ответ со статус-кодом ошибки 500 без тела (что, соответственно, заставит браузер выдать пользователю свою дефолтную страницу ошибки). Рекурсии не произойдёт.



Ошибка 404 генерируется низшим middleware — неявной заглушкой, которая заложена в конвеер по дефолту и при получении запроса очищает его тело и устанавливает статус-код в 404.

- `app.UseDeveloperExceptionPage();` — страница с подробной (техническая) информацией об исключении (используется исключительно при отладке);

- `app.UseExceptionHandler("/Error");` — редирект исключений на адрес страницы с кратким user-friendly (без технических деталей) описанием ошибки (используется на продакшене);
- `app.UseStatusCodePagesWithReExecute("/{0}");` — редирект статус-кодов вида 4xx и 5xx на адрес `/{0}`, где плейсхолдер `{0}` будет замещён статус-кодом.

91 — To be continued...