Основы ASP.NET Core

Запуск приложения. Класс Program

В любом типе проектов ASP.NET Core, как и в проекте консольного приложения, мы можем найти файл **Program.cs**, в котором определен одноименный класс Program и с которого по сути начинается выполнение приложения. В ASP.NET Core 3 этот файл выглядит следующим образом:

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

namespace HelloApp
{
    public class Program
    {
        public static void Main(string[] args)
        {
            CreateHostBuilder(args).Build().Run();
        }

        public static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .ConfigureWebHostDefaults(webBuilder =>
                {
                    webBuilder.UseStartup<Startup>();
                });
    }
}
```

Чтобы запустить приложение ASP.NET Core, необходим объект **IHost**, в рамках которого развертывается веб-приложение. Для создания IHost применяется объект **IHostBuilder**.

В программе по умолчанию в статическом методе `CreateHostBuilder` как раз создается и настраивается IHostBuilder. Непосредственно создание IHostBuilder производится с помощью метода **Host.CreateDefaultBuilder(args)**.

Данный метод выполняет ряд задач.

- Устанавливает корневой каталог (для этого используется свойство Directory.GetCurrentDirectory). Корневой каталог представляет папку, где будет производиться поиск различного содержимого, например, представлений.
- Устанавливает конфигурацию хоста. Для этого загружаются переменные среды с префиксом "DOTNET_" и аргументы командной строки
- Устанавливает конфигурацию приложения. Для этого загружается содержимое из файлов appsettings.json и appsettings.{Environment}.json, а также переменные среды и аргументы командной строки. Если приложение в статусе разработки, то также используются данные Secret Manager (менеджера секретов), который позволяет сохранить конфиденциальные данные, используемые при разработке.
- Добавляет провайдеры логирования
- Если проект в статусе разработки, то также обеспечивает валидацию сервисов

Далее вызывается метод **ConfigureWebHostDefaults()**. Этот метод призван выполнять конфигурацию параметров хоста, а именно:

- Загружает конфигурацию из переменных среды с префиксом "ASPNETCORE_"
- Запускает и настраивает веб-сервер Kestrel, в рамках которого будет разворачиваться приложение
- Добавляет компонент `Host Filtering`, который позволяет настраивать адреса для веб-сервера Kestrel
- Если переменная окружения `ASPNETCORE_FORWARDEDHEADERS_ENABLED` равна `true`, добавляет компонент Forwarded Headers, который позволяет считывать из запроса заголовки "X-Forwarded-"
- Если для работы приложения требуется IIS, то данный метод также обеспечивает интеграцию с IIS

Метод `ConfigureWebHostDefaults()` в качестве параметра принимает делегат `Action<IWebHostBuilder&`. А помощью последовательного вызова цепочки методов у объекта

IWebHostBuilder производится инициализация веб-сервера для развертывания веб-приложения. В частности, в данном случае у IWebHostBuilder вызывается метод **UseStartup()**:

```
webBuilder.UseStartup<Startup>()
```

Этим вызовом устанавливается стартовый класс приложения - класс Startup, с которого и будет начинаться обработка входящих запросов.

В методе Main вызывается метод у созданного объекта IHostBuilder вызывается метод **Build()**, который собственно создает хост - объект **IHost**, в рамках которого развертывается веб-приложение. А затем для непосредственного запуска у IHost вызывается метод **Run**:

```
CreateHostBuilder(args).Build().Run();
```

После этого приложение запущено, и веб-сервер начинает прослушивать все входящие HTTP-запросы.