# **Введение в ASP NET CORE**

**ASP.NET Core** – программная платформа, кот.разраб.Microsoft и предназнач.для разраб.веб-прилож. Явл.развитием OWIN (The Open Web Interface). Katana – OWIN-совестимый хост, раб.с микрософт.

**.NET CORE –** альтернатива .NET, своя реализация CLR (CoreCLR) и своя библ (CoreFX).

**ASP.NET Core** – фреймв.с откр. кодом и его м.всегда скачать с гитхаба, где есть полностью исх.код.

Последняя версия – 31 (2019г)

Прилож., кот.разраб.на этой платформе могут работать как .NET CORE так и над полной .NET.

Если прилож.раб.над NET Core, то обесп.кроссплатф.над Windows, Linux, Mac OS.

Прилож. ASP .NET Core могут работать под IIS (Windows) или под web-С Kastrel (кроссплатф.варик)

|  |  |  |
| --- | --- | --- |
| ASP.NET 4.6 | ASP .NET Core 2.0(3.1) | |
| .NET Framework 4.6 (Windows) | | .Net Core 2.0(3.1) (Lnix, Windows, Mac OS) |
| .NET FCL | | .Net Core Library |
| Runtime Components – компоненты собственной ОС | | |

Модульность обеспеч.то, что все отдельные компоненты загруж.через NuGet и нет необх.примен. библ.System.Web.dll (осн.библ.для прилож. ASP.NET) => обеспеч.кроссплатформенность, т.к.NuGet раб.на различн.платформах.

МБ разраб.след.типы прилож:

* MVC UI, Web API, Web Pages (не нах.шир.примен)
* Web Form не поддерживается

**ASP .NET Core** поддерж.вижлой начиная с 2015 года, а также вижул студио код (кроссплатф.инстр)

Работая с core мы разрабатываем wev-придожение с помощью таких языков, как c#, html, css, js.

Проверить установку .NET Core (ком.строка от админа):

* dotnet --version – проверить версию
* создание приложения – dotnet new console –output FirstApp
  + создается файлик .csproj – предназ.для вижлы и позвол раб.в вижле с проектом
* зайти в папку с приложением – cd FirstApp
* выполнить приложение – dotnet run – все сформируется, скомпилируется и выведет рез.

*Создание веб-приложения:*

Visual C# => Веб => веб-приложение ASP.NET Core => делаем его пустым для того, чтобы посмотреть что там есть в начальном приекте.

*Program.cs:*

|  |  |
| --- | --- |
|  | создается хост  исп.стрелочная ф-ция, кот запуск.цепочку ф-ций и создает хост и сод.станд. Kastrel http сервер  CreateDefaultBuilder – построение хоста. |

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

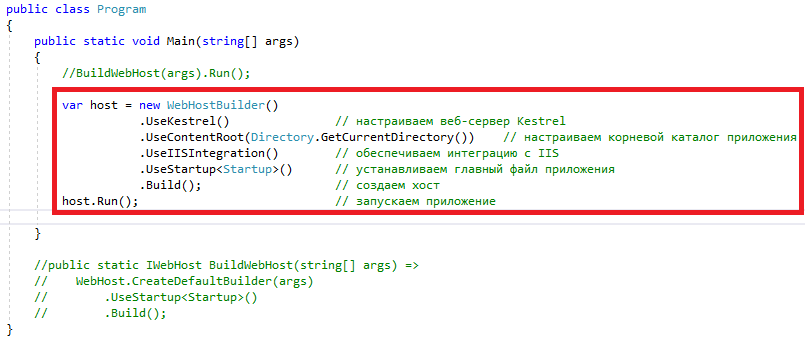
Метод ConfigureWebHostDefaults() в качестве параметра принимает делегат Action<IWebHostBuilder&. А помощью последовательного вызова цепочки методов у объекта IWebHostBuilder производится инициализация веб-сервера для развертывания веб-приложения. В частности, в данном случае у IWebHostBuilder вызывается метод **UseStartup()**

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**

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

Подробнее о CreateDefaultBuilder:

****

*Startup.cs* – класс, кот сод.логик обработки вх.запросов – производит конфигурацию приложения, настраивает сервисы, которые прилож будет исп, устанавл.компоненты для обработки запроса или middleware.

|  |  |
| --- | --- |
|  | Startup – класс, кот явл главным файлом веб прилож  \*м.сод.необяз.к-р  2 метода:  ConfigureService – необяз., вызыв.после к-ра и примен для регистрации сервисов  Configure – обяз., вызыв после ConfigureService – обработка сообщ кот прих.по протоколу http (запускаем run и формируем response => на все запросы отвечаем “Hello World”) |

Разберем подробно:

* 1. Выражение ***if (env.IsDevelopment())*** проверяет, находится ли приложение в состоянии/статусе разработки. Что это значит? Для проекта можно указать, например, через настройки, что он находится в состоянии разработки. Вообще условно есть три состояния или стадии проекта: в состоянии разработки (Development), в состоянии подготовки к развертыванию (Staging) и в состоянии полноценного использования (Production), когда он уже развернут на каком-нибудь сервере, и пользователи могут к нему обращаться. По умолчанию Visual Studio устанавливает для проекта состояние разработки. И данное выражение как раз проверяет состояние.

Если проект находится в состоянии разработки, то, возможно, мы захотим применять некоторые действия, которые не нужны, когда приложение уже развернуто. Так, по умолчанию вызывается метод *app.UseDeveloperExceptionPage();* который выводит подробные сообщения об ошибках. Подобные сообщения нежелательны и могут раскрывать некоторые чувствительные данные, когда приложение уже развернуто на сервере и с ним могут работать пользователи, поэтому они подобные сообщения по умолчанию выводятся только в состоянии разработки.

* 1. **Следующий шаг – run()**

Метод Run представляет собой простейший способ для добавления компонентов middleware в конвейер. Однако компоненты, определенные через метод Run, не вызывают никакие другие компоненты и дальше обработку запроса не передают.

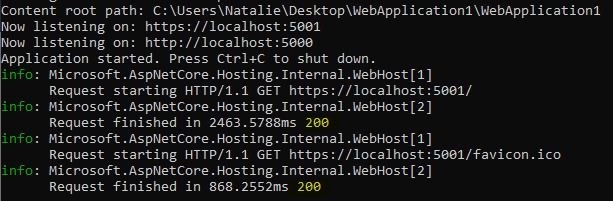
Он принимает в качестве параметра контекст запроса HttpContext и возвращает объект Task. Поэтому в методе Run делегат в качестве параметра context принимает контекст запроса - объект HttpContext.

Данный метод определяет один единственный делегат запроса, который обрабатывает все запросы к приложению. Суть этого делегата заключается в отправке в ответ на запросы сообщения "Hello World!". Причем так как данный метод не передает обработку запроса далее по конвейеру, то его следует помещать в самом конце. До него же могут быть помещены другие методы.

В итоге при запуске проекта по типу Empty мы увидим в браузере строку "Hello World!".

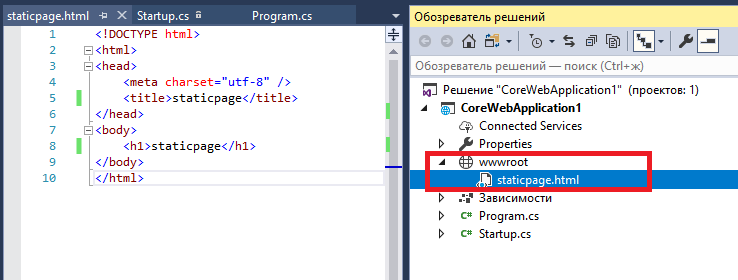
Проект мб запущен в 2 режимах:

* CoreWebApp – запускаем как консольное приложение – как хостинг?
  + Стартует как консольное приложение
  + Выбает протокол, чтобы отследить какие запросы обраб.

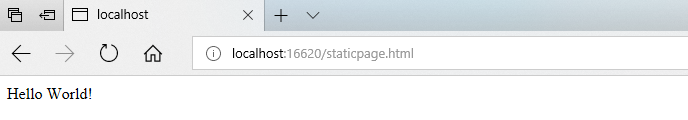


* IIS Express – через ИИС

***Wwwroot*** – папка для статич.контекта (html, css, js); стат.контект не отображ.по умолч.



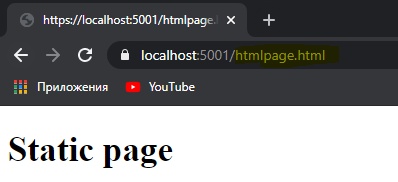
Если перейдем в браузере localhost:16620/staticpage.html – она его будет искать в wwwroot найдет и выдаст HelloWorld



**Для отображения статического контента** необх.подключить NuGet пакеты; при создании net core проекта автоматически подключается Micrisoft.AspNetCore.All в котором есть все

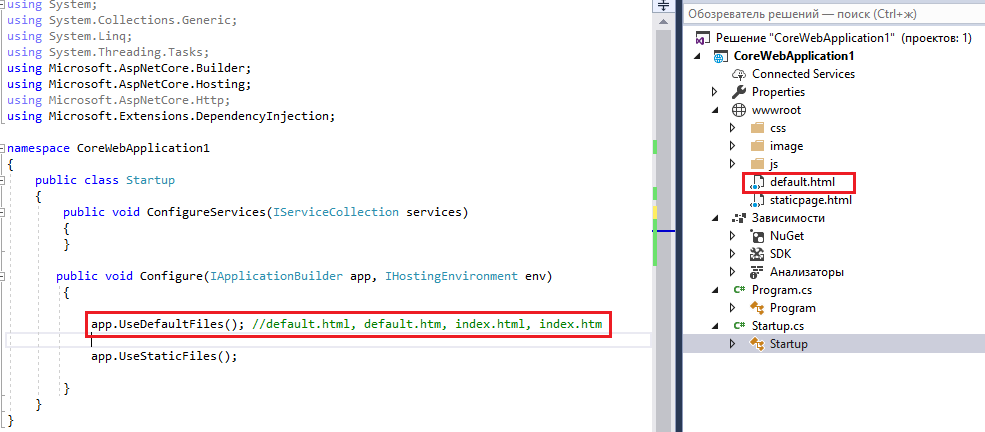
Для того, чтобы указать, что мы хотим использовать статич.файл => Startup.cs => в методе Configure пишем только 1 строку => app.UseStaticFile() – оно работало по умолч., а это более строгое определение и вызывает именно статический файл.;

Делаем localhost:16620/staticpage.html

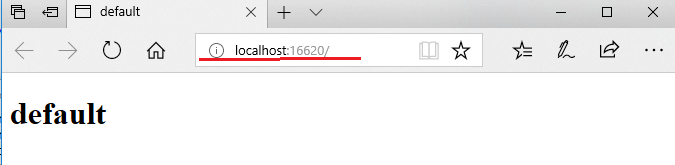
  
Также можно добавить картинку в wwroot и вызвать в html странице:

<img src=”название\_картинки” width=”500”, height = “500”/>

Можно **создать стартовые страницы** по умолчанию. Для этого исп. app.UseDefaultFiles(), кот.позвол устан. имена дефолтных файлов(в этом случае – dafault.html/.htm, index.html/.htm) – именно в таком порядке будут отыскиватсья стартовые файлы в wwwroot

****

Тогда по запросу localhost:16620 выведется содержимое дефолтного файла

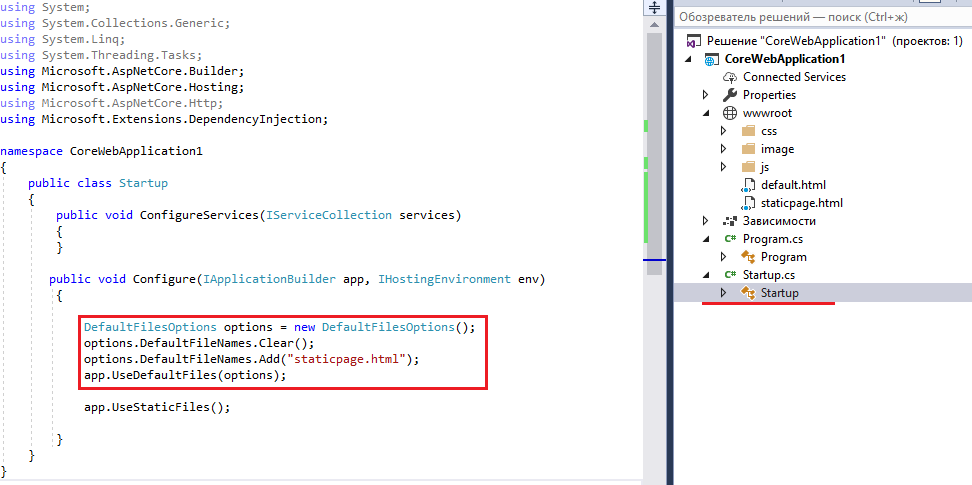
****

Также можно самим составить список дефолтных файлов:

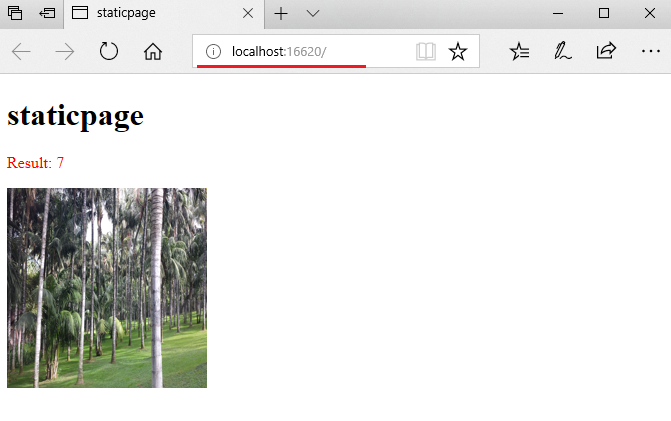
\*создаем объект

\*чистим старый список

\*добавляем свой элемент

****

Запускаем приложение:

****

**Файлы для скачивания**

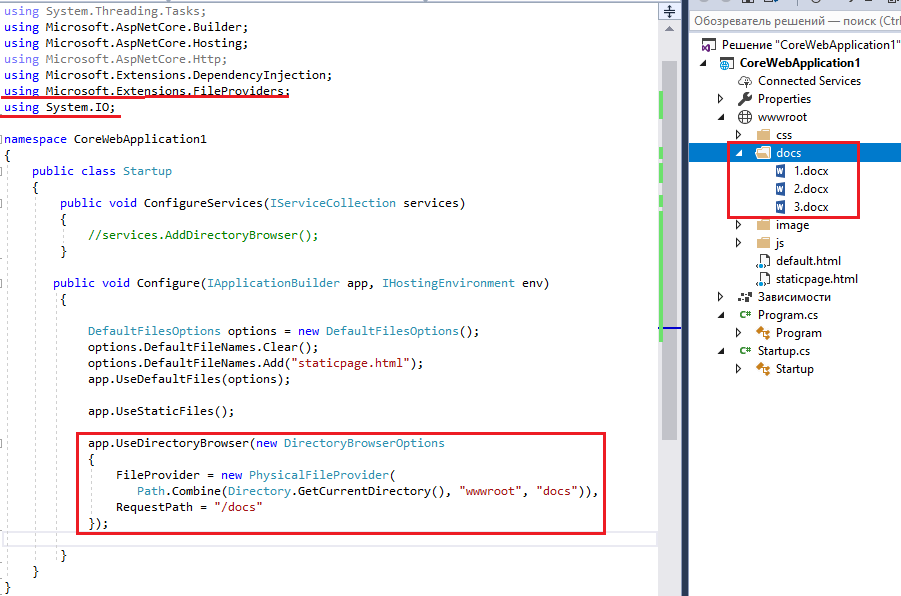
Необходимо добавить нужные namespacы:

* Using System.IO
* Using Microsoft.Extensions.FileProviders

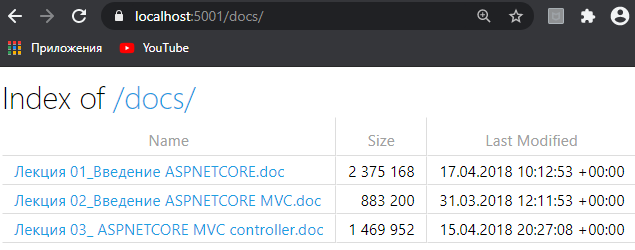
Пишем следующий код и указываем где нах.те файлы, к кот мы хотим дать юзерам доступ

\*указали что папка для скачивания – wwroot/docs

\* указали, что в uri надо указать docs чтобы добраться к этим файлам

  
В папку docs мы закинули пару документов и по запросу выдается:

(если нажмем на файл – он скачается)



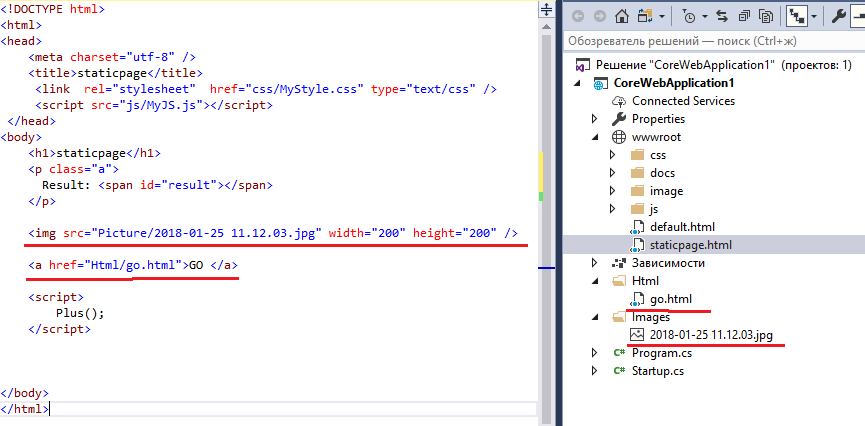
Кроме wwwroot мы можем назначить и другие директории **для хранения стат.файлов**

\* указ отдельно 2 папки

\* указали по какому uri будет доступ к статическому контенту (html/picture)

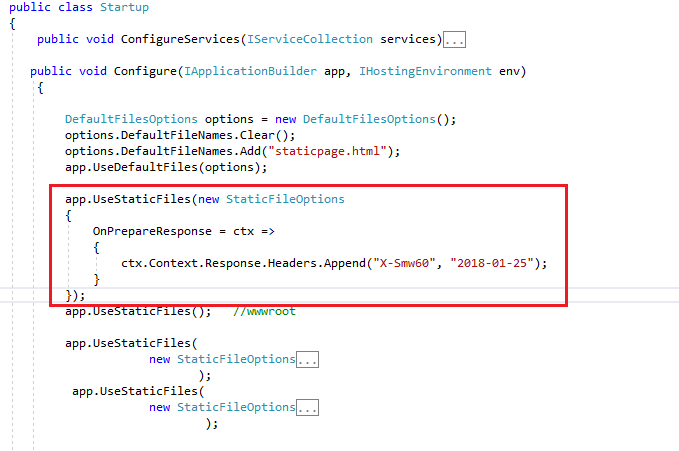
****

дальше в нашем стат.файле в wwwroot м.исп-ть эти файлы в body:

****

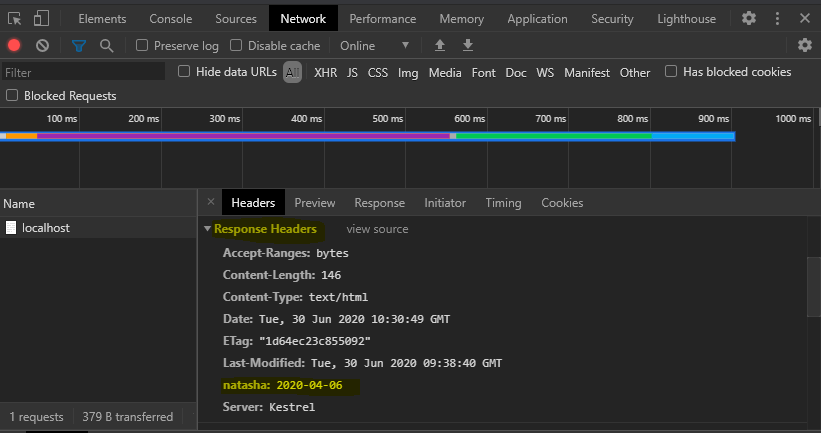
Можем **добавить http-заголовок**, при этом UseStaticFiles должен быть первым

\* через стрелочную функцию добавляем заголовок и значение

****

Чтобы проверить заголовок:

\* необходимо сделать запрос еще раз, чтобы появился «localhost» во вкладке network



# **Введение в ASP.NETCore MVC**

**MVC** (Model View Controllet) – шаблон / архитектурный паттерн; включает 3 компоненты:

* Модель – данные;
* Представление – отображение модели;
* Контроллер – обработка запросов, координация взаимодействия модели и представления.
* Маршрутизатор – выбор контроллера и действия

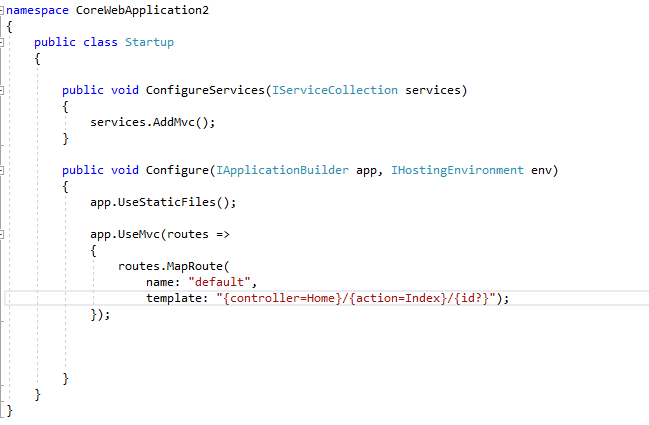
Поступает запрос – запрос поступает на маршрутизатор – создается контроллер – контроллер формирует респонс – контроллер отрабатывает, использует модель и формируется представление, которое формирует респонс – респонс уходит на сторону клиента.



*Создание веб-приложения:*

Visual C# => Веб => веб-приложение ASP.NET Core => MVC

Смотрим Startup.cs:

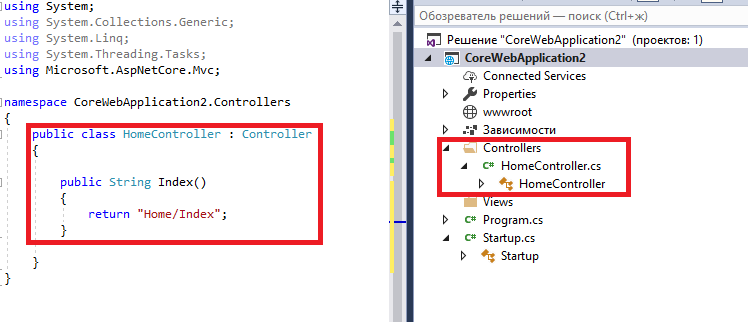
****

\* для того, чтобы исп.MVC используем метод AddMVC(), т.е. подключить сервис MVC

\* В методе Configure подключаем статические файлы, а также используем MVC и здесь же определяем маршрутизатор, в котором используем MapRoute (при написании шаблона маршрута указываем контроллер и его дефолное значение, а также дефолтное зрначение action.

Разрабатываем контроллеры

* У нас это класс, который наследует от Controller
* В .net 2 разных базовых класса для контроллеров mvc и web-api, а также разные маршрутизаторы
* В asp .net core все контроллеры имеют общие контроллеры, а также общие маршрутизаторы (те же для mvc и web-api)

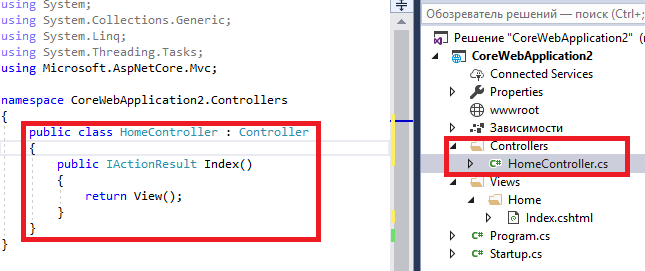
****

Обработка запроса

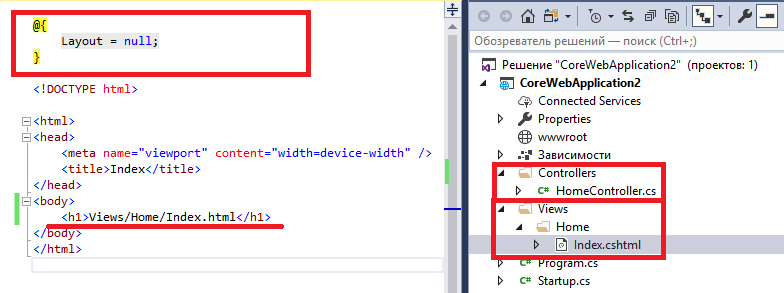
****

**Также можно использовать view**

\* здесь вызывается view по умолч., имя кот.совпад.с именем action

****

Содержимое view:

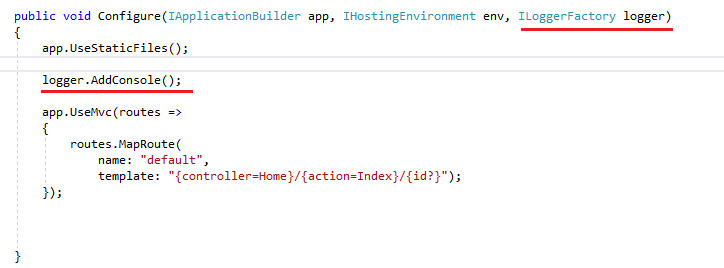
****

Результат обработки:

****

Для того, чтобы выводить все в **журнал консоль –** необходимо подключить логгер – AddConsole()

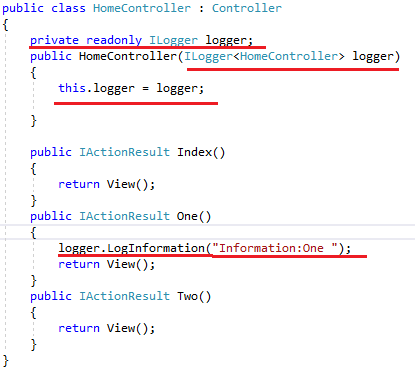
* В результате будет выводится дополнительная инфа в консоль
* ILoggerFactory – через встроенное DI подставит объект, который позволяет создать такой логгер



\*используем логгер

\*передаем его в контроллер, при этом авто исп.внедрение зависимости

\* можем выдавать некоторую инфу в консоль

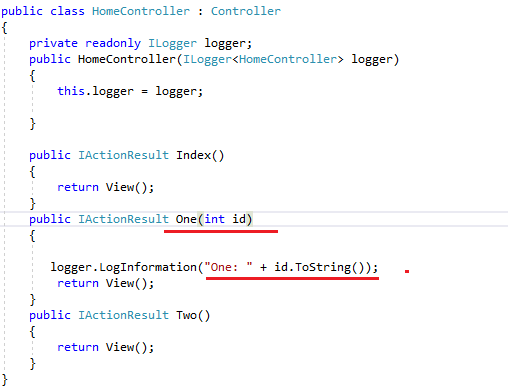


Т.е.логгер включается для того, чтобы использовать какую-то трассировку в нашем прилож.

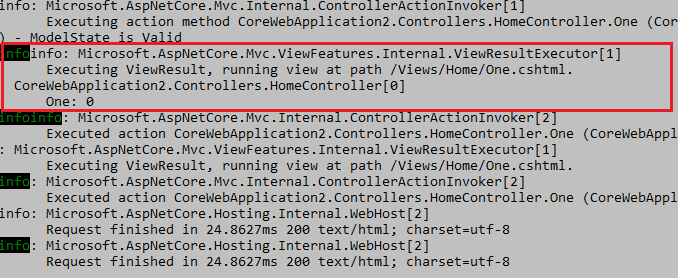
**Передача параметров**

Есть акция One и пар-р id

Можем сразу записывать его в логгер

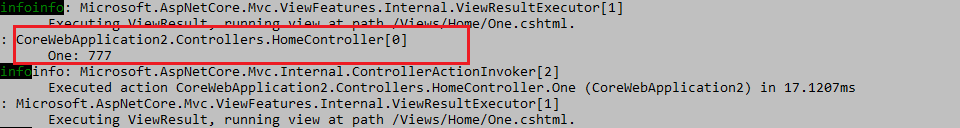
****

При вызове One (localhost:7484/Home/One) получаем страницу, а в консоль вывдится:

****

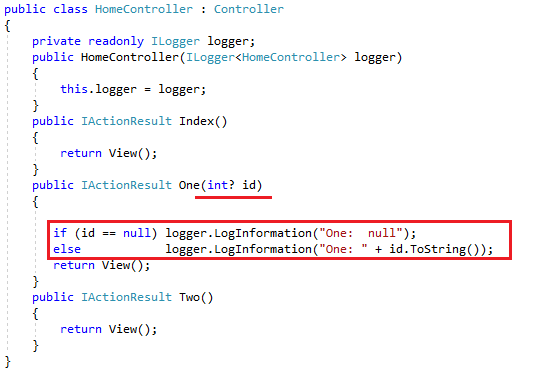
0 – т.к.стоит параметр, а мы ничего не передавали при вызове

Если бы вызов был localhost:7484/Home/One/777 => в консоль вывелось бы

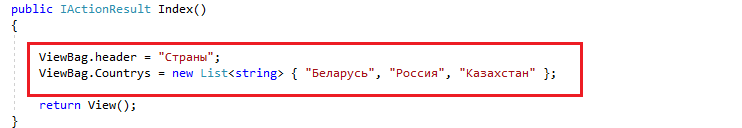
****

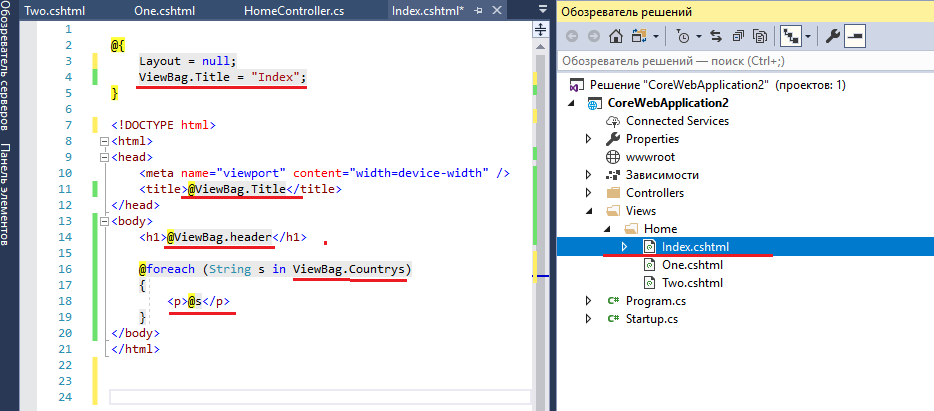
Если мы после int поставим знак вопроса, то вместо 0 выведется null

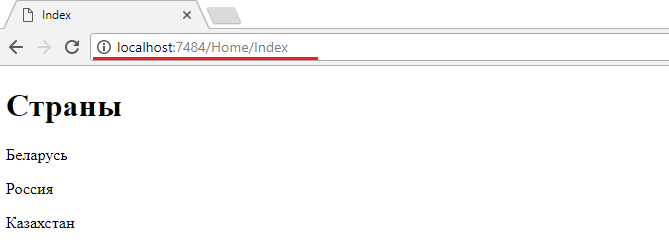
* Это делается для того, если дефолтное значение стоит 0, чтобы отличить его от null



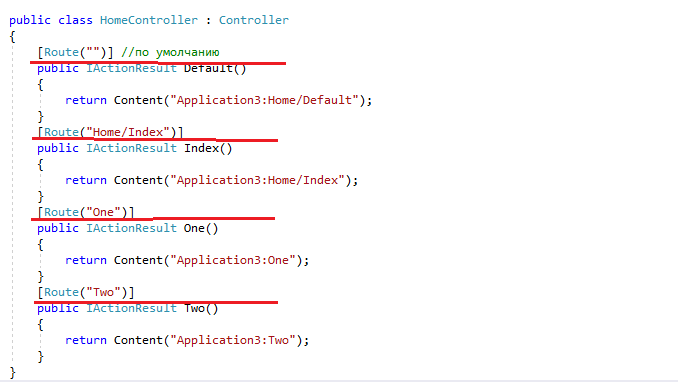
М.исп-ть **viewbag** – объект с динамическими св-ми.

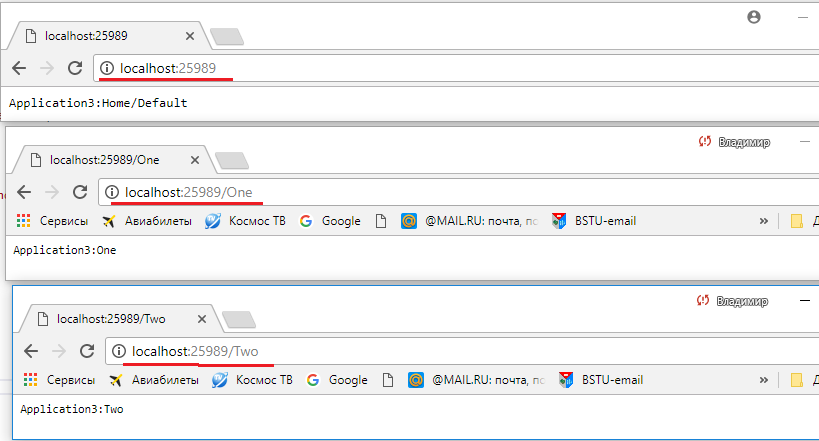
****

****

****

**атрибут Route**  - если мы хотим делать маршрутизацию без таблиц маршрутизации, то мы просто указываем маршруты явно для к.action с помощью атрибута Route

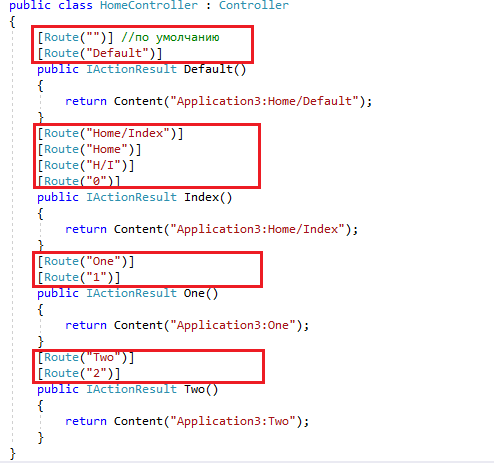
****

****

Также с помощью атрибута Route можно указывать **множественные маршруты.**

Т.е.при переходе по Home/Index, Home, H/I, 0 результат будет 1 и тем же.

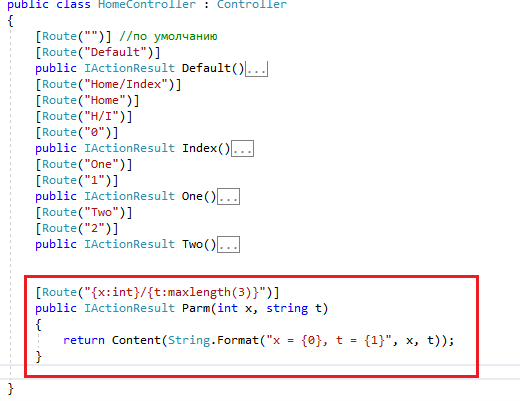
Т.е.к 1 actionу мы м.прописать много маршрутов

****

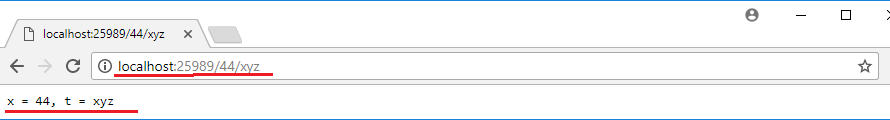
**Обработка сложных маршрутов:**

Используем параметризованный uri и обрабатываем значения как параметры.

Описывается маршрут, кот.обяз.д.сдерж.целочисл.знач, и строку макс из 3 символов

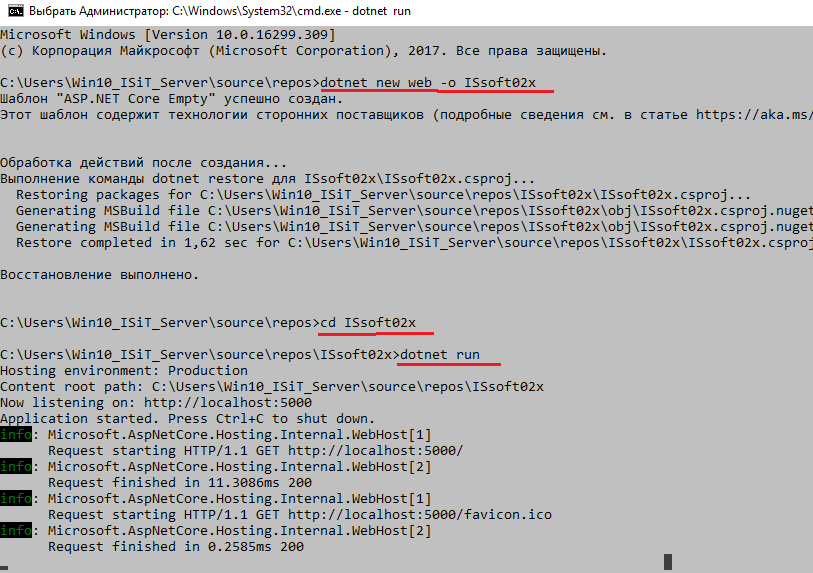
****

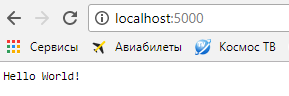
Обработка такого маршрута:

****

**Разработка web-приложения с помощью CLI (ком.строка)**

\* dotnet new wev –o ISsoft02x – создаст готовое веб-приложение

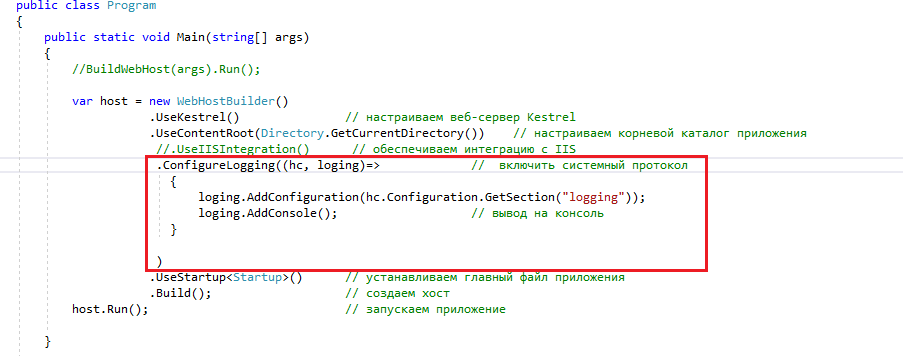
****

****

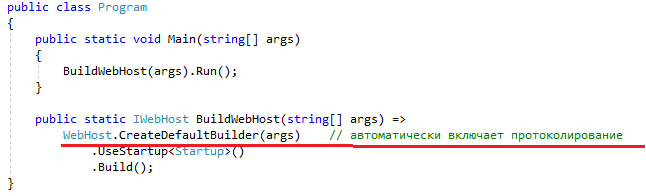
М. изменить приложение и также запустить через ком.строку.

# **ASP.NETCore MVC Controller**

М.включить протоколирование

****

Используем протоколирование (CReateDefaultBuilder – авто включает протоколирование)

****

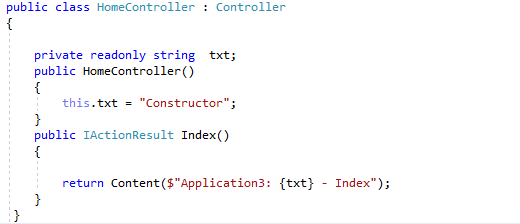
**MVC Controller -** компонент приложения разработанного в соответствии с паттерном MVC. В одном приложении может быть несколько контроллеров.

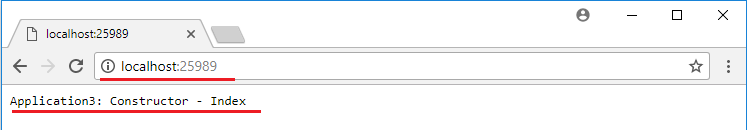
**MVC Controller –** программный класс, производный от одного из абстрактных классов: **Controller** или **ControllerBase**.

**ControllerBase** не поддерживает view**. Controller** поддерживает view

М. применять **конструкторы** и применять стандартные методы

\*Content – выводить д-е в редима text/plain

****

****

Существуют особенности по поиску контроллеров в core.

Существует redlection-поиск контроллера. По каким признакам?

* неабстрактный public-класс;
* не generic;
* не вложен в другие типы;
* не декорирован [NonController];
* наследуется от Control, либо имя класса имеет суффикс Control.

**М.создать контроллер без суффикса Controller**

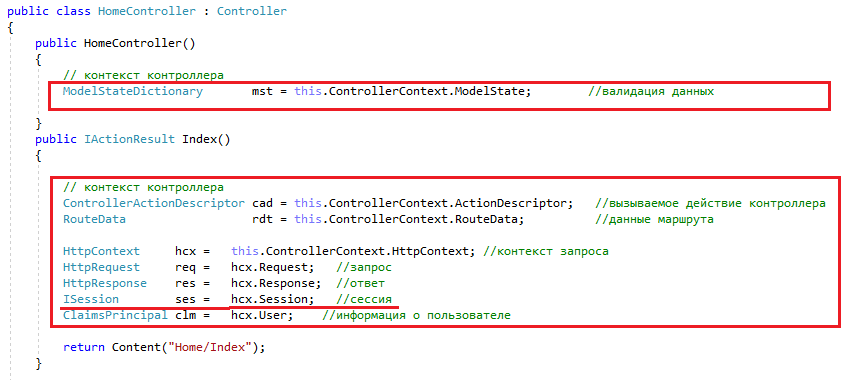
**контекст контроллера** – св-во, в кот.входят другие св-ва кот.м.нас заинетресовать

ModelState – объект, кот предназнач.для валидации д-х.

В любом mvc приложении есть 3 типа моделей – для view, для action и для данных.

Модель для экшн образуется из набора пар-ров, кот приходят в экшн.

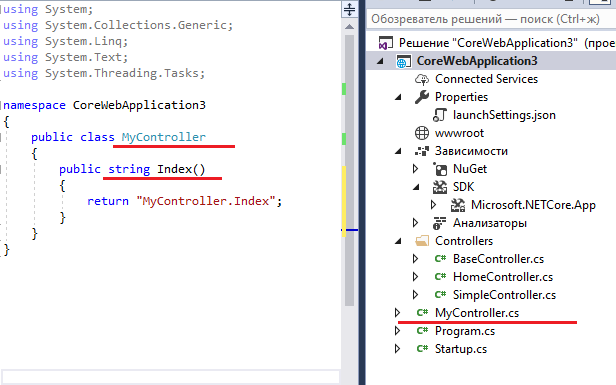
ModelState и предназначена для валидации д-х, кот приходят в кач-ве пар-ров.

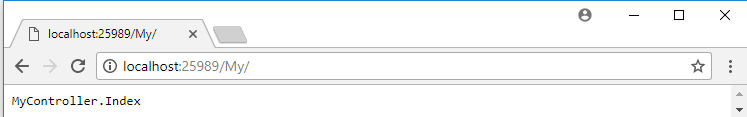
****

POCO-controller - controller не наследник класса Controller; имя с суффиксом Controller;

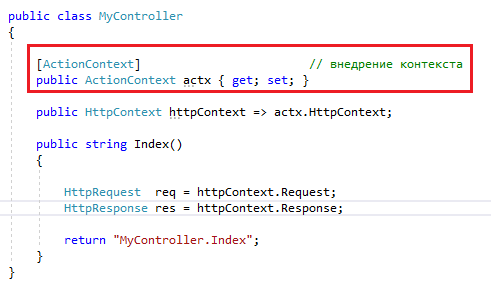
**может находиться в отдельной сборке** **(не получилось);**

**пробовал в отдельную сборку подключать Microsoft.AspnetCore.Mvc – не помогло.**

****

****

Здесь можем внедрить контекст, т.к.тот контекст кот.дб прийти от контроллера не пришел, поэтому можем внедрить свой

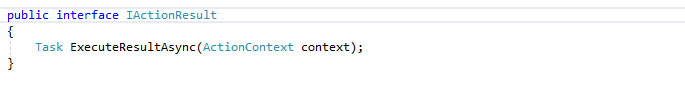
****

**MVC Controller** по умолч.не поддерживает сессию, адля поддержки объекта **Session** исп.AddSession.

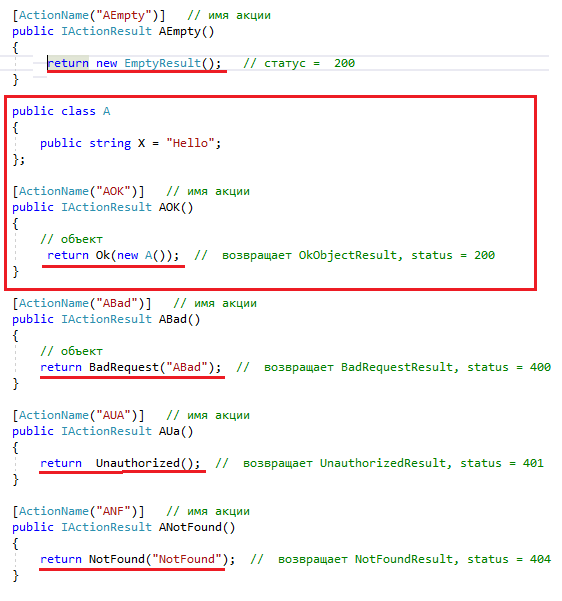
**Требования к action (методам контроллера)**

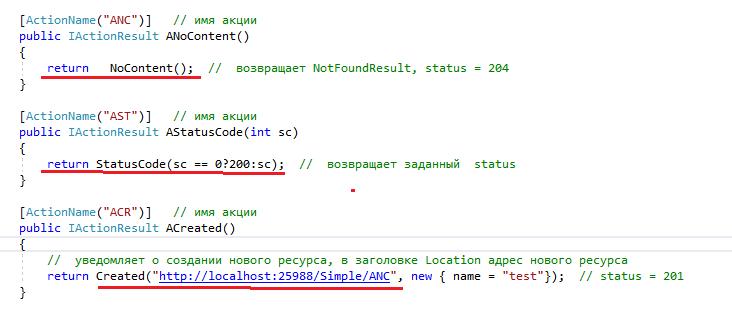
* нестатический публичный метод котроллера;
* не является generic;
* не помечен атрибутом [NonAction];
* перегруженные методы могут быть только отмеченные атрибутами [Action], [ActionName], [AcceptVerbs], [HttpGet], [HttpPost]; - т.е.если мы помечаем данными атрибутами методы, то они могут быть перегружены
* ref и out–параметры в действиях не допустимы;
* возвращает IActionResult – преобразует ответ к этому типу (возвращаем стринг - преобразуется)

интерфейс IActionResult.- асинхронный вызов

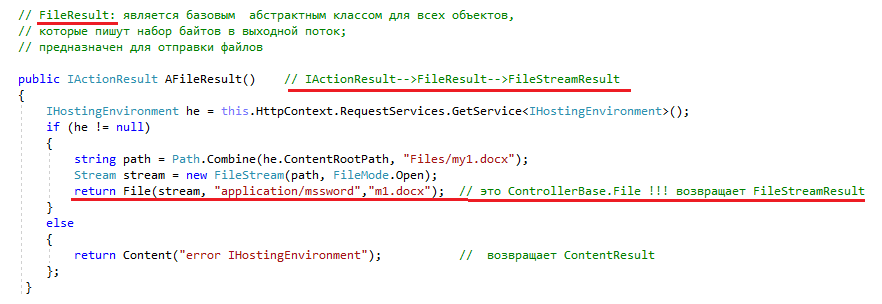
****

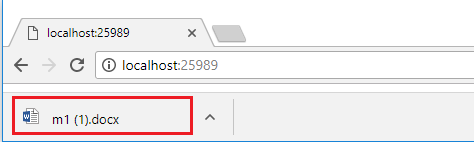
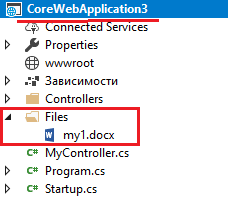
**возврат кодов**

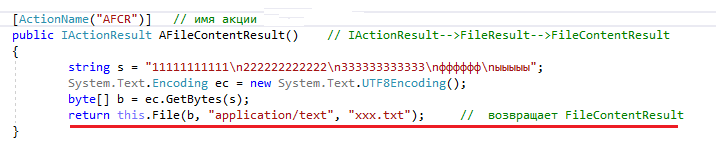
****

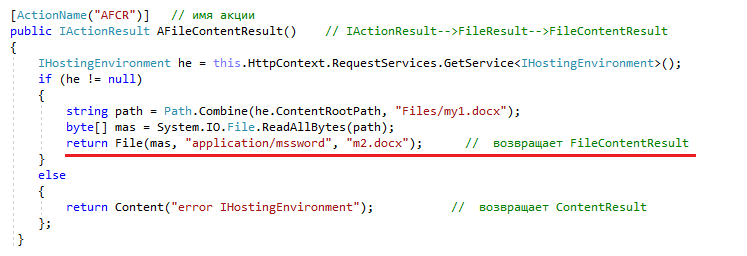
****

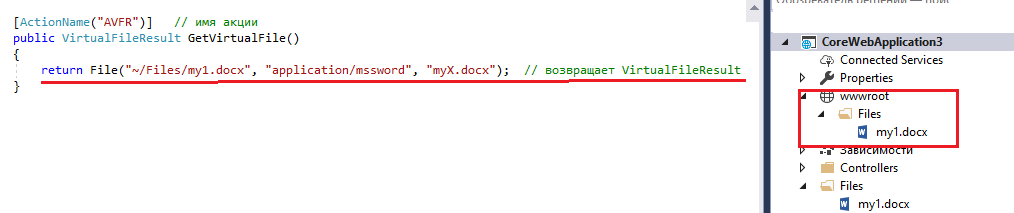
**MVC Controller:** отправка файлов

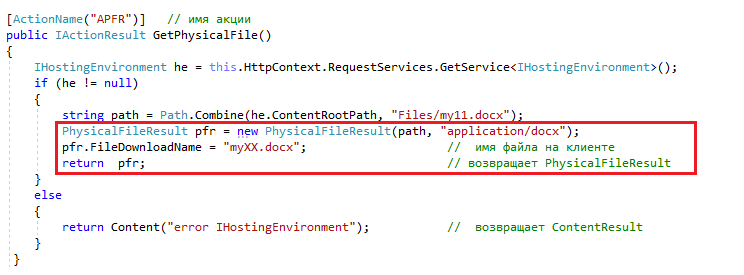
****

****

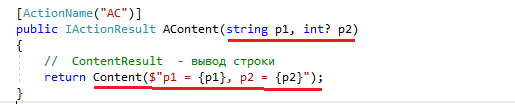
****

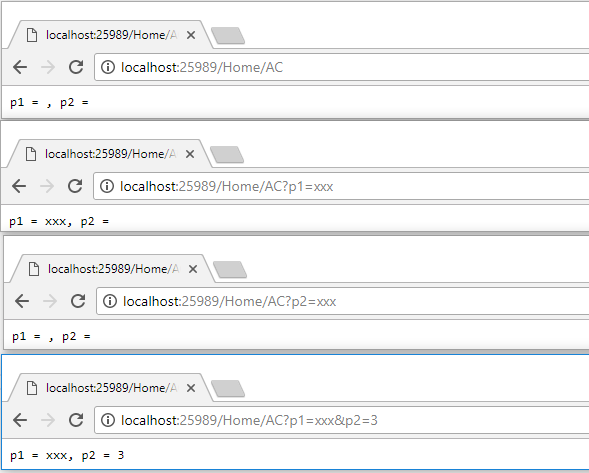
****

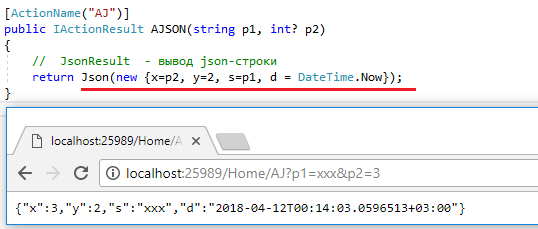
****

****

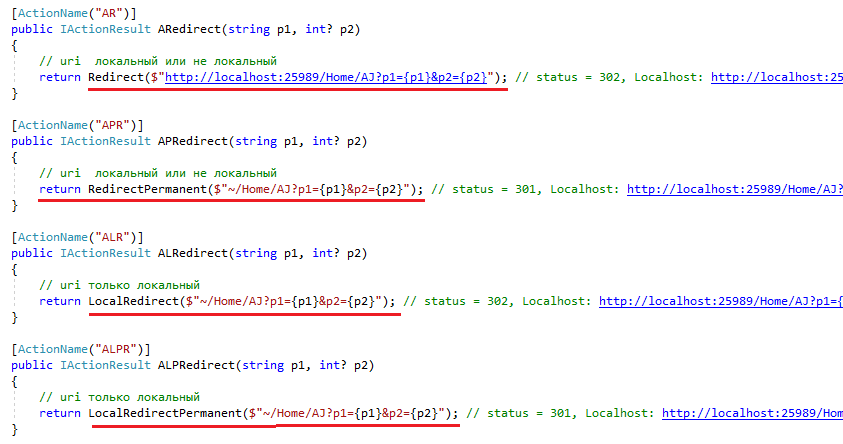
1. **MVC Controller:** вывод строки или JSON-строки

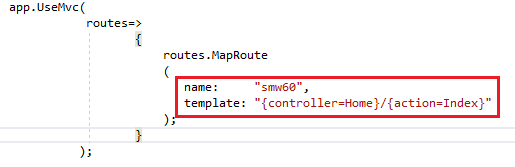
****

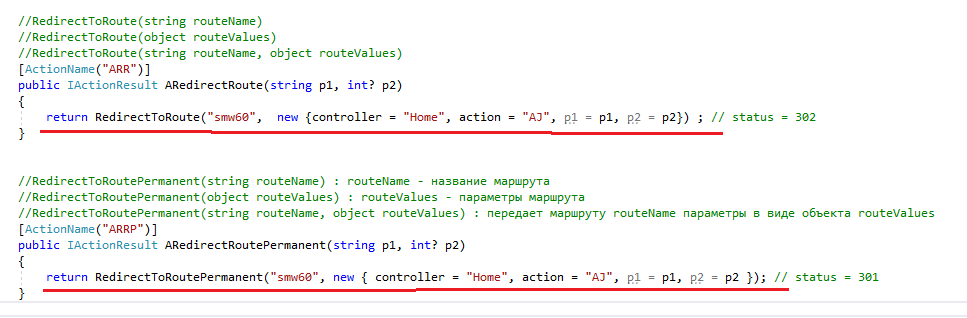
****

****

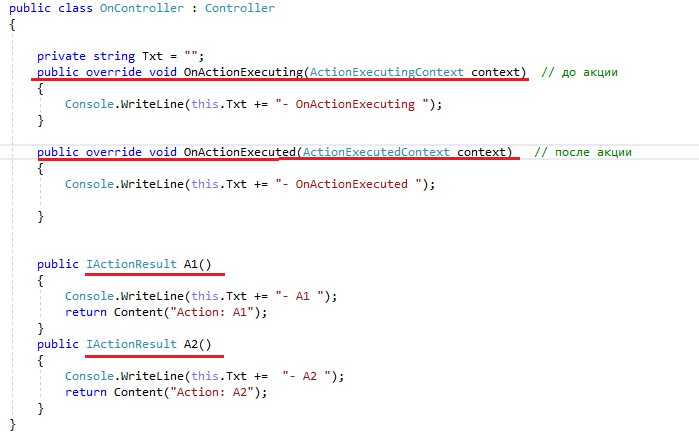
1. **MVC Controller:** переадресация

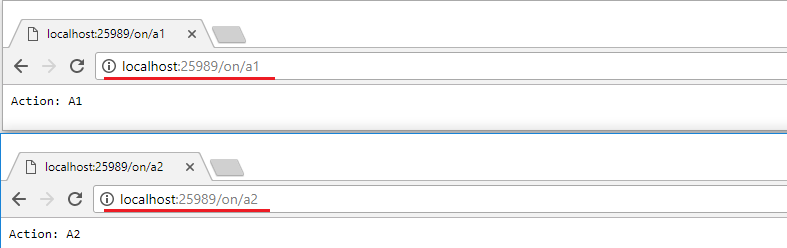




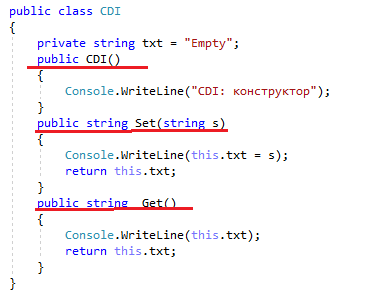


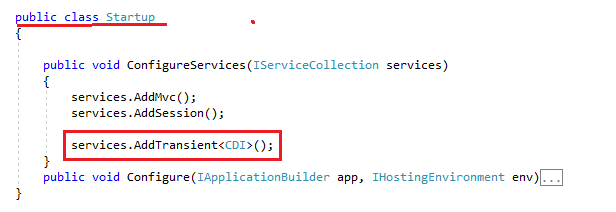
1. **MVC Controller:** OnAction…

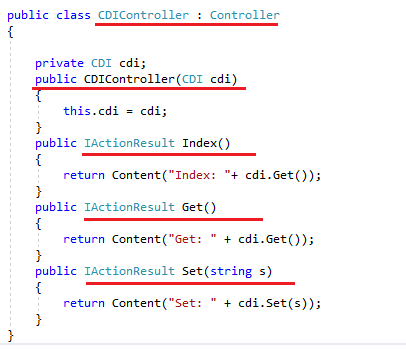
****

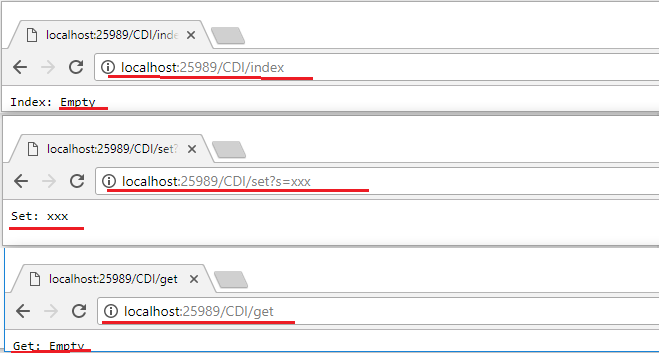
****

1. **MVC Controller:** Dependency Injection (внедрение зависимостей)

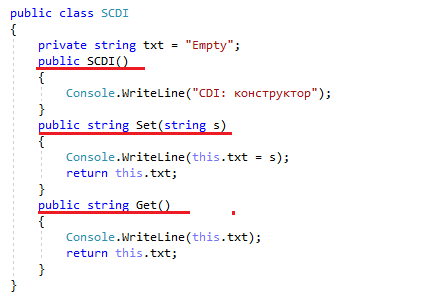
****

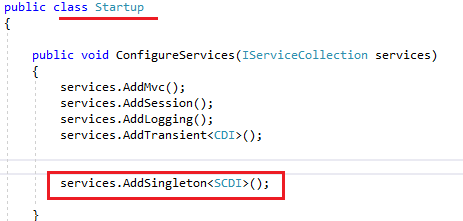
****

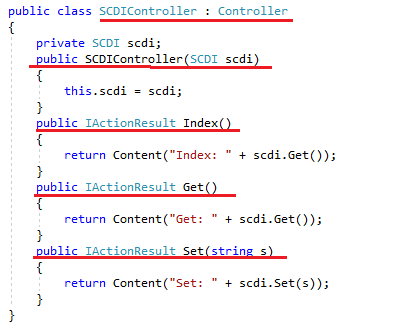
****

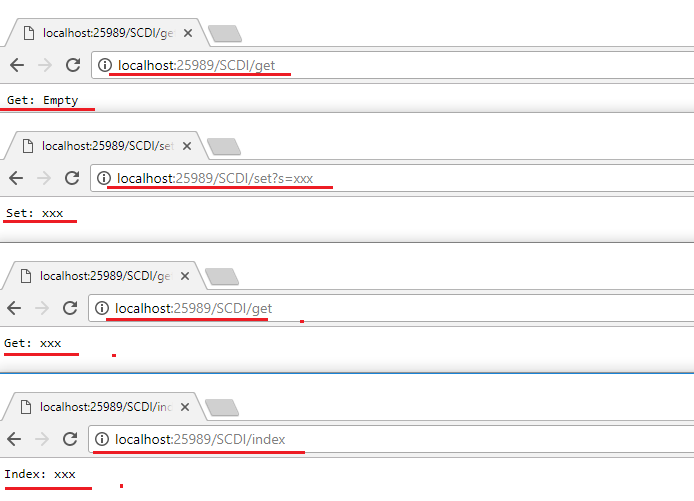
****

1. **MVC Controller:** Dependency Injection (внедрение зависимостей) Singleton

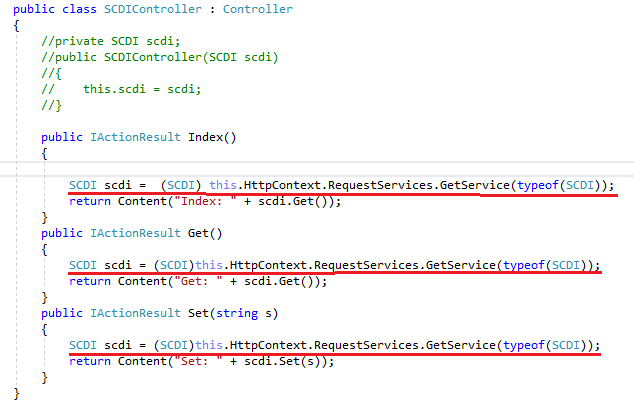
****

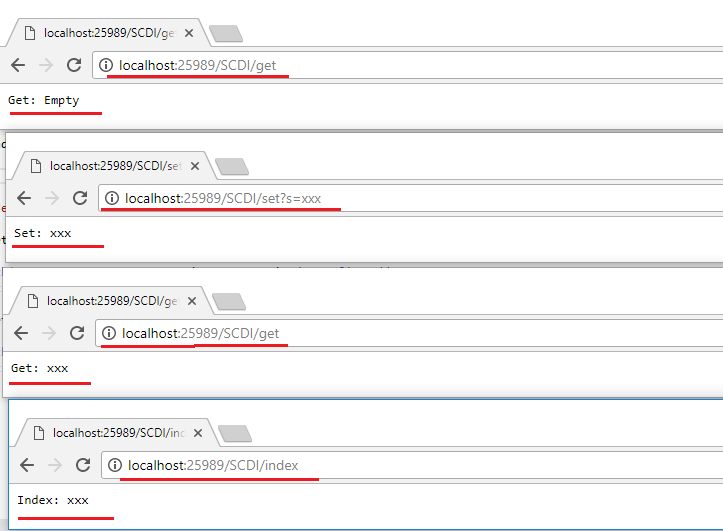
****

****

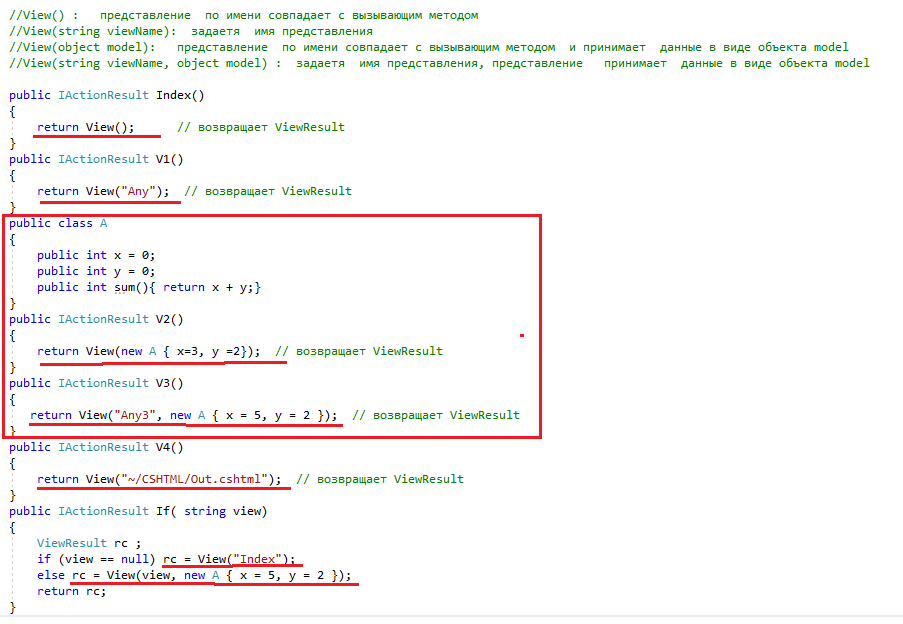
****

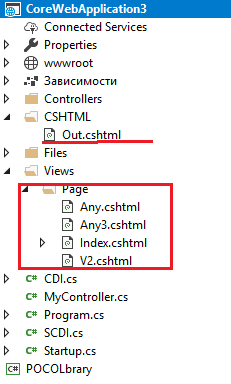
1. **MVC Controller:** Dependency Injection (внедрение зависимостей) Singleton динамический метод внедрения

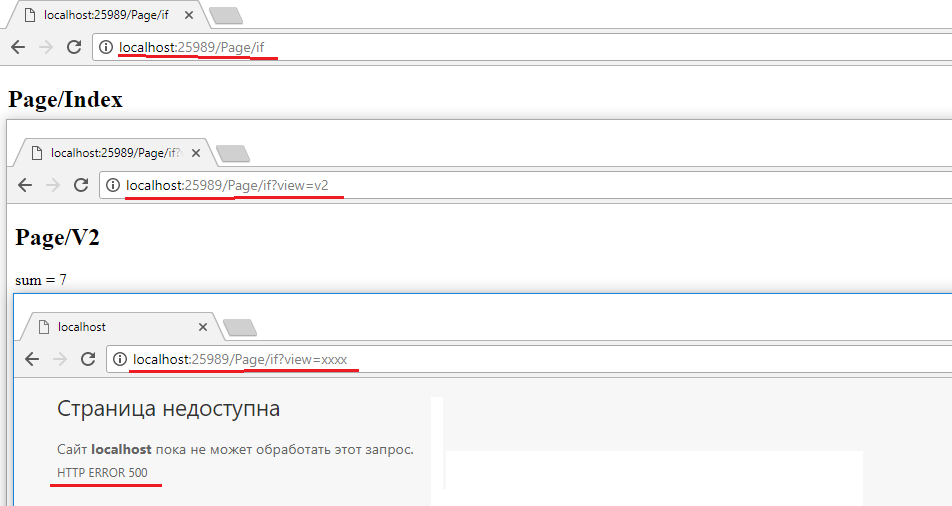


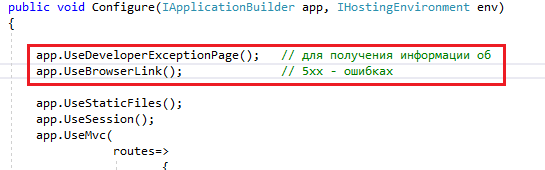


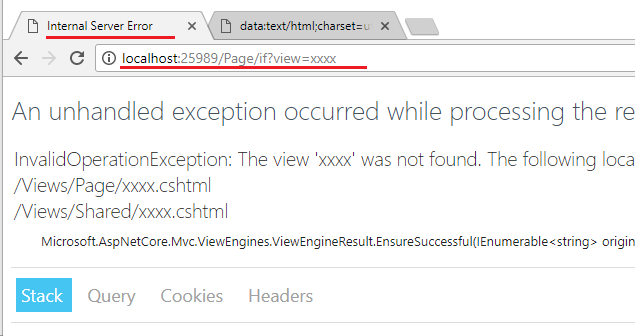
1. **MVC Controller:** методырендеринга представления в web-страницу

****

****

****

****

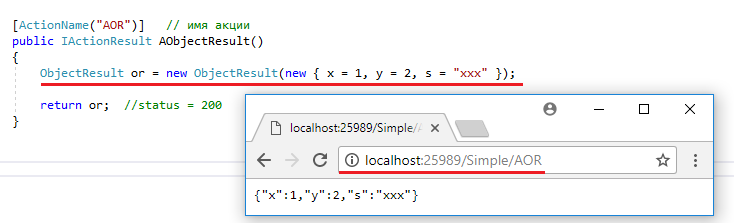
****

1. **MVC Controller:** возвращаемые инфраструктурой значения

**IActionResult** –стандартное значение;

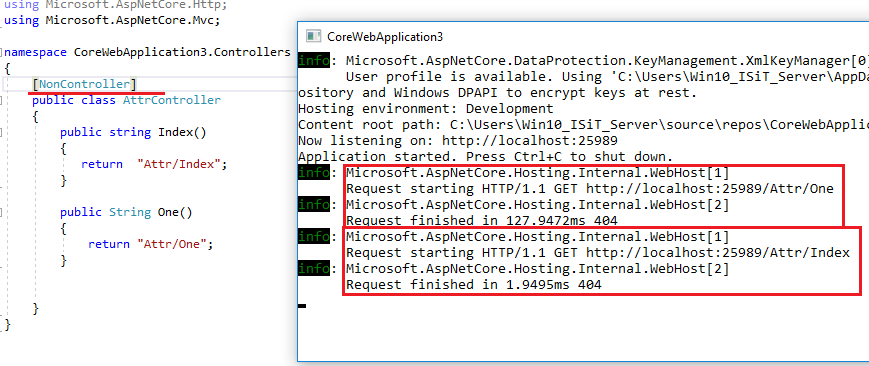
**Task или void** – заменяется на **EmptyResult**;

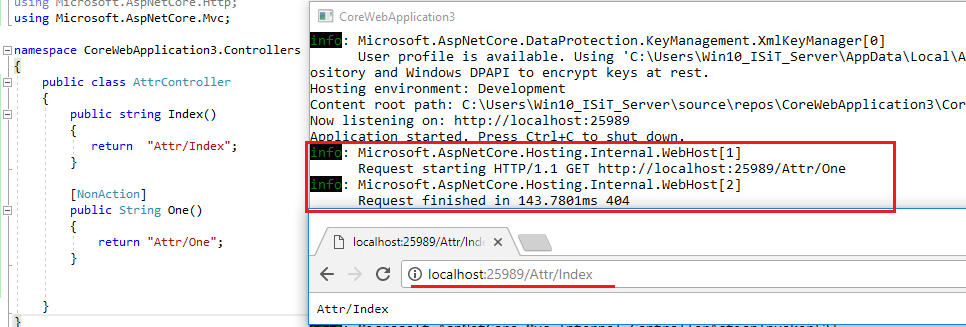
остальное – заменяется на **ObjectResult**.

****

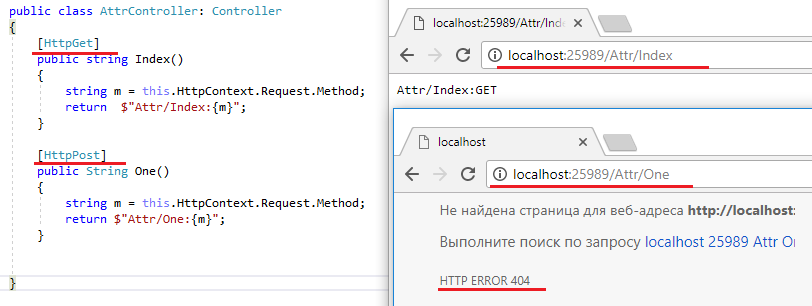
1. **MVC Controller:** атрибуты [NonAction], [NonController]

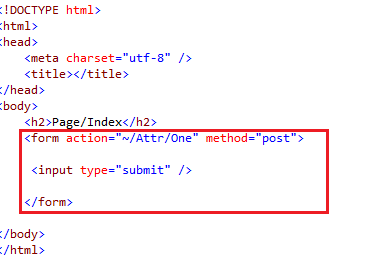
****

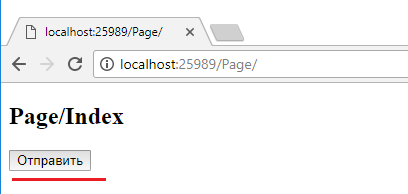
****

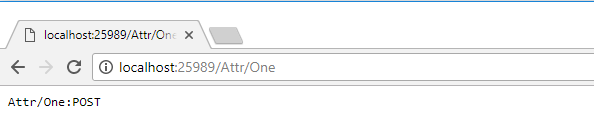
****

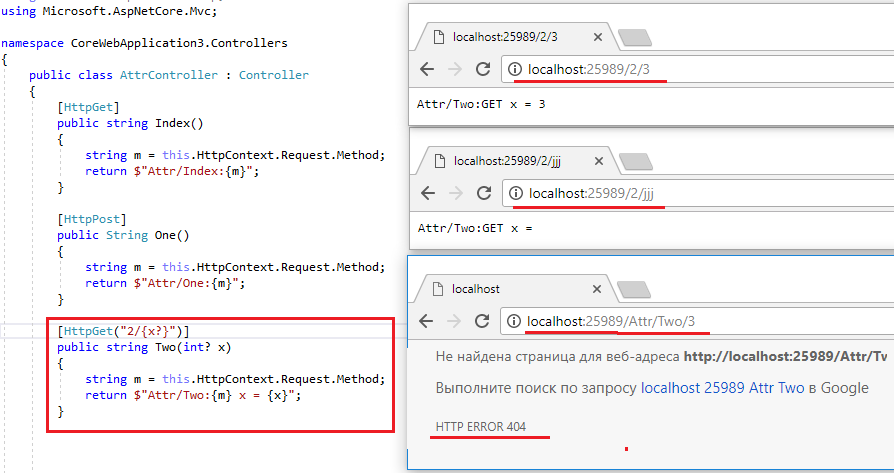
1. **MVC Controller:** атрибуты [HttpGet], [HttpPost], [AccepVerb]…

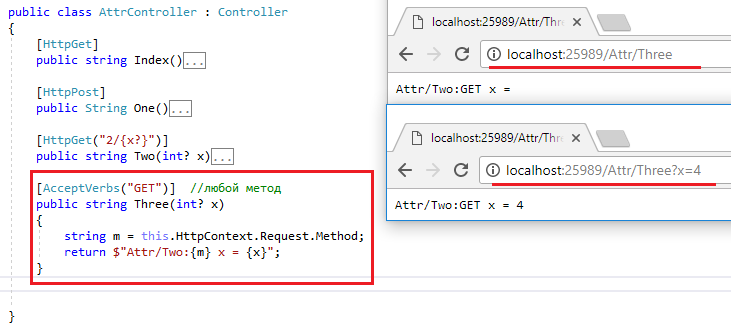
****

****

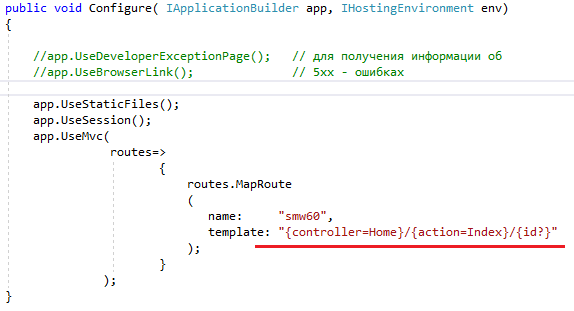
****

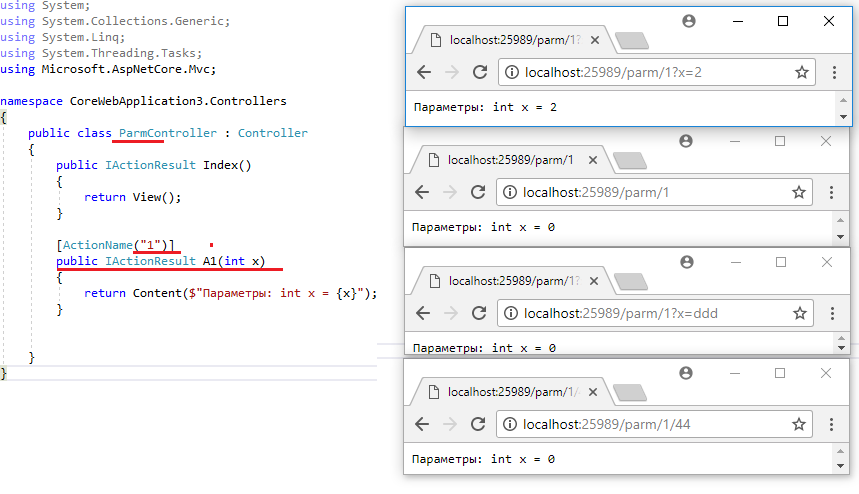
****

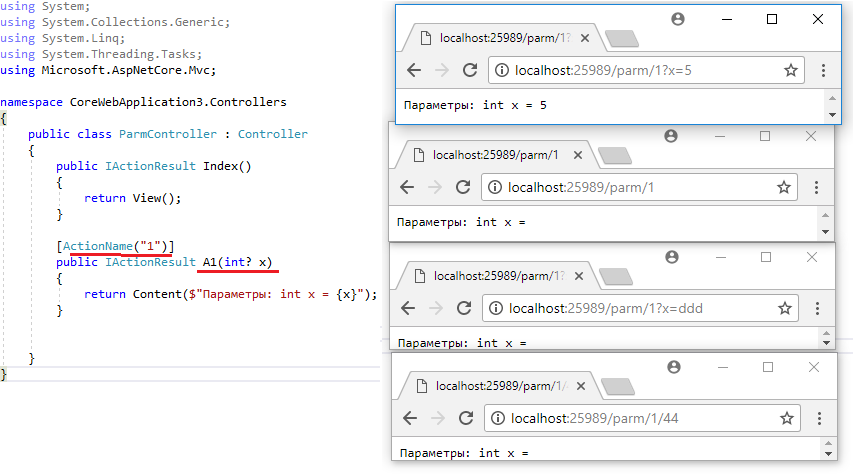
****

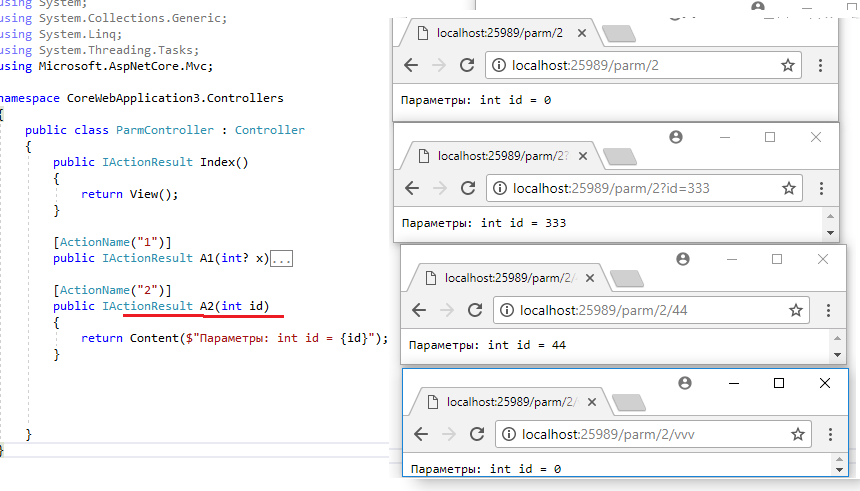
****

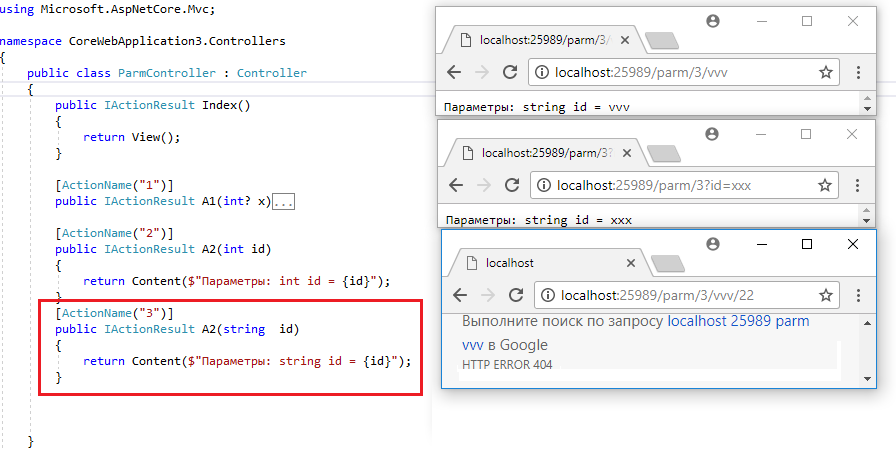
1. **MVC Controller:** параметры действий (акций)

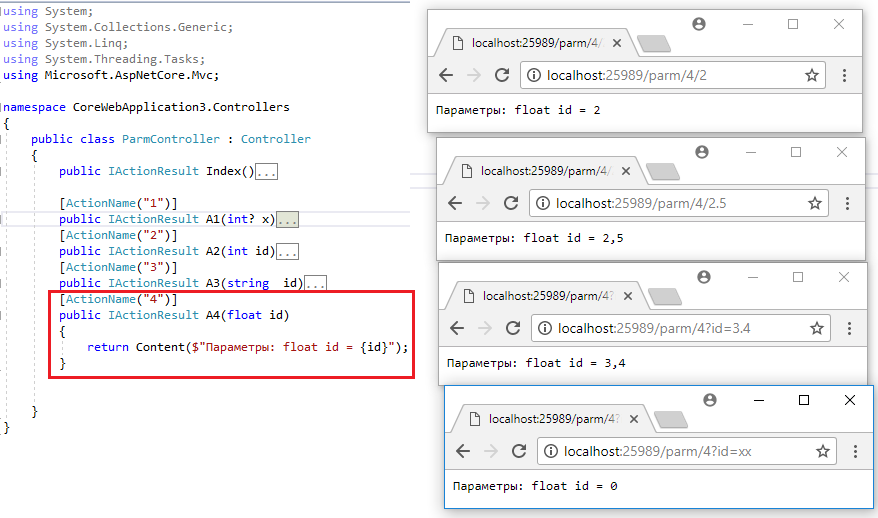
****

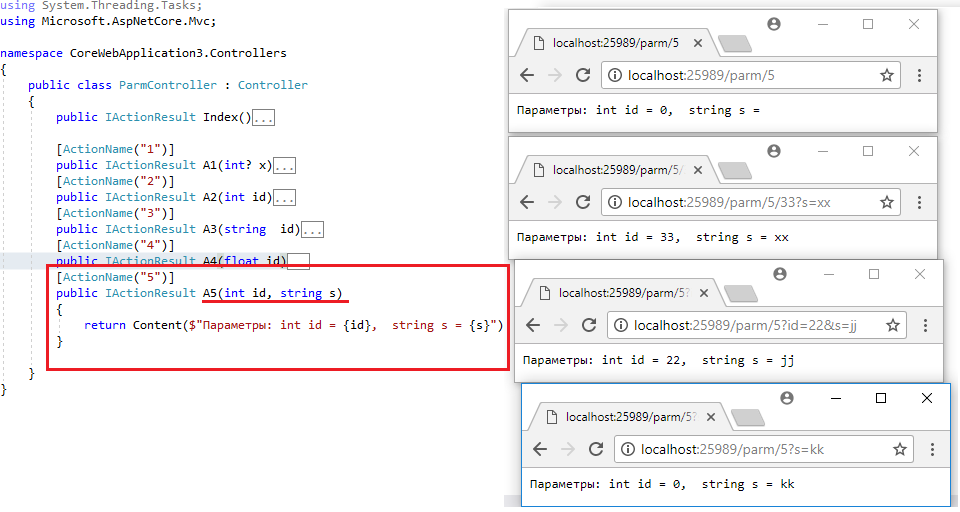
****

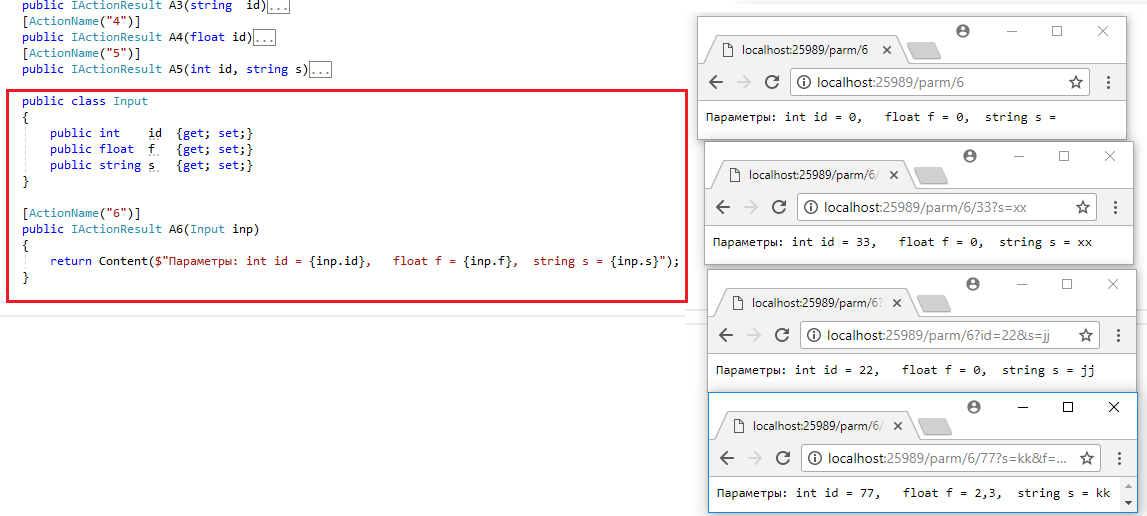
****

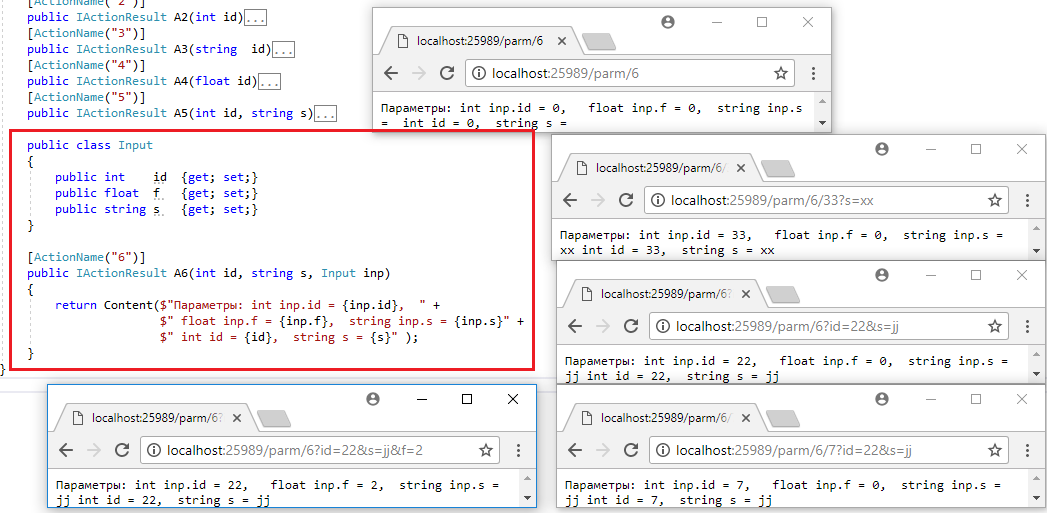
****

****

****

****

****

****

# **ДОП ИНФА из метанита:**

**Компоненты middleware, кот встроены в ASP.NETCore**

* Authentication: предоставляет поддержку аутентификации
* Cookie Policy: отслеживает согласие пользователя на хранение связанной с ним информации в куках
* CORS: обеспечивает поддержку кроссдоменных запросов
* Diagnostics: предоставляет страницы статусных кодов, функционал обработки исключений, страницу исключений разработчика
* Forwarded Headers: перенаправляет зголовки запроса
* Health Check: проверяет работоспособность приложения asp.net core
* HTTP Method Override: позволяет входящему POST-запросу переопределить метод
* HTTPS Redirection: перенаправляет все запросы HTTP на HTTPS
* HTTP Strict Transport Security (HSTS): для улучшения безопасности приложения добавляет специальный заголовок ответа
* MVC: обеспечивает функционал фреймворка MVC
* Request Localization: обеспечивает поддержку локализации
* Response Caching: позволяет кэшировать результаты запросов
* Response Compression: обеспечивает сжатие ответа клиенту
* URL Rewrite: предоставляет функциональность URL Rewriting
* Endpoint Routing: предоставляет механизм маршрутизации
* Session: предоставляет поддержку сессий
* Static Files: предоставляет поддержку обработки статических файлов
* WebSockets: добавляет поддержку протокола WebSockets

**ЖЦ middleware**

Метод Configure выполняется один раз при создании объекта класса Startup, и компоненты middleware создаются один раз и живут в течение всего жизненного цикла приложения. То есть для последующей обработки запросов используются одни и те же компоненты.

Допустим в классе Startuo определен так:

*public void Configure(IApplicationBuilder app)*

*{*

*int x = 2;*

*app.Run(async (context) =>*

*{*

*x = x \* 2; // 2 \* 2 = 4*

*await context.Response.WriteAsync($"Result: {x}");*

*});*

*}*

При запуске результат будет 4, однако при последующих запросах он будет больше - 8,16 и т.д.

**Структура ASP.NET Core MVC приложения**

* **Dependencies**: все добавленные в проект пакеты и библиотеки
* **wwwroot**: этот узел (на жестком диске ему соответствует одноименная папка) предназначен для хранения статических файлов - изображений, скриптов javascript, файлов css и т.д., которые используются приложением. Цель добавления этой папки в проект по сравнению с другими версиями ASP.NET, состоит в разграничении доступа к статическим файлам, к которым разрешен доступ со стороны клиента и к которым доступ запрещен.
* **Controllers**: папка для хранения контроллеров, используемых приложением
* **Models**: каталог для хранения моделей
* **Views**: каталог для хранения представлений
* **appsettings.json**: хранит конфигурацию приложения
* **Program.cs**: файл, определяющий класс Program, который инициализирует и запускает хост с приложением.
* **Startup.cs**: файл, определяющий класс Startup, с которого начинается работа приложения. То есть это входная точка в приложение.