Веб-программирование

Часть 1

Юрий Литвинов
y.litvinov@spbu.ru

23.11.2022

Веб-приложения

Как оно вообще работает

- Пользователь заходит браузером на определённый URL
  - На самом деле, выполняя HTTP GET-запрос на порт 80 или 443 (обычно)
- ОС сервера перенаправляет запрос запущенному там веб-серверу
  - Например, Apache, IIS, нынче популярны self-hosted сервисы, например, Kestrel
- Веб-сервер — отдельный процесс, в рамках которого запущено несколько веб-приложений, веб-сервер по URL запроса определяет, какому веб-приложению он адресован, и передаёт запрос ему
- Веб-приложение формирует ответ и отправляет его обратно по HTTP в виде HTML-страницы
- Эта страница и показывается пользователю в браузере

Веб-сервисы

- *Веб-сервис* — это примерно то же самое, но не для пользователя, а для других приложений
- Нужны для создания распределённых приложений
- Общаются не с помощью HTML, а посредством специализированных протоколов
  - Например, SOAP
    - Использует синтаксис XML, может использовать HTTP как транспорт
  - Также популярны REST (это, правда, не протокол), gRPC
- Содержат механизм публикации метаинформации
  - Например, WSDL для SOAP
  - OpenAPI (Swagger) для REST
- Реализуются посредством технологий, например, ASP.NET Web APIs

Веб-приложения и .NET

- Веб-сервер — IIS (Internet Information Services), IIS Express, Kestrel
  - Есть “из коробки” в Windows, IIS Express поставляется с Visual Studio и используется для отладки
  - Kestrel поставляется как часть ASP.NET
- Технология для разработки веб-приложений и веб-сервисов — ASP.NET
- Работа с базами данных — MS SQL Server (SQL Server Express)
- ORM — Entity Framework (Entity Framework Core)
- Фронтенд — TypeScript
- Контейнеризация — Docker под WSL
- Облачный хостинг — Azure

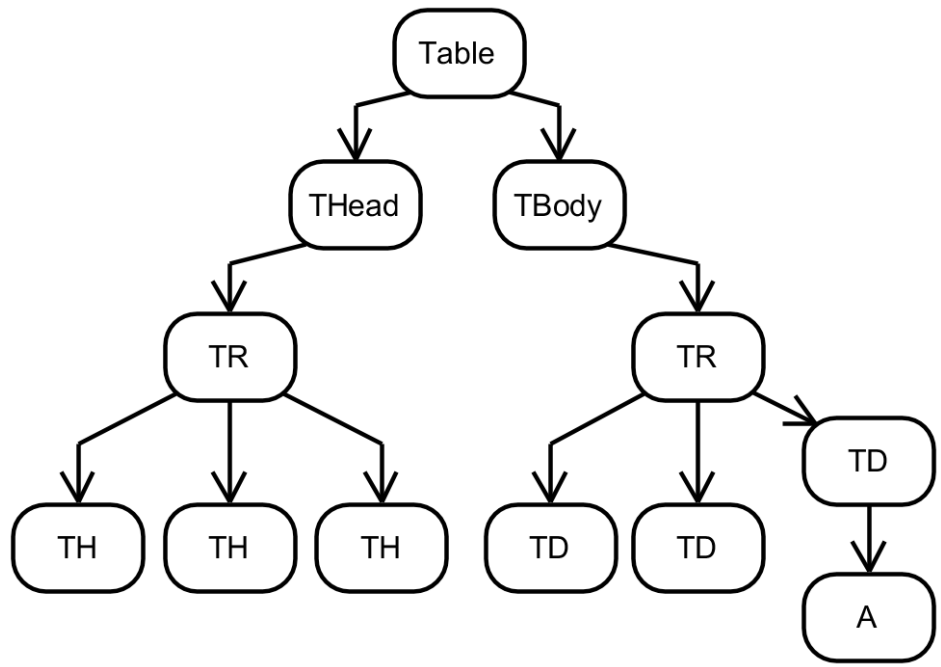
Браузерная часть

- HTML (HyperText Markup Language) — используется для задания содержимого и структуры веб-страницы
  - Тут размечаются параграфы, заголовки, списки, таблицы и т.д.
  - Тут же обычно определяются способы идентифицировать элементы
- CSS (Cascading Style Sheet) — используется для задания внешнего вида, оформления и расположения элементов
- JavaScript — используется для задания поведения веб-страницы на клиенте
  - Интерпретируется браузером
  - Полноценный язык программирования

DOM

- DOM (Document Object Model) — представление HTML-документа в виде дерева объектов и API для доступа к нему
- JavaScript может манипулировать DOM-деревом, браузер рендерит его “на лету”, что и даёт интерактивность

```
<table class="listing">
  <thead>
    <tr class="odd">
      <th>Выпускник</th>
      <th>Научный руководитель</th>
      <th>Текст</th>
    </tr>
  </thead>
  <tbody>
    <tr class="odd">
      <td>Акбаров Артур Александрович</td>
      <td>д.т.н., проф. Д.В. Кознов</td>
      <td><a href="bmo/441-Akbarov-report.pdf">Текст</a></td>
    </tr>
  </tbody>
</table>
```



HTML-формы

- Способ получить ввод от пользователя
- Возможность организовать POST-запрос (GET по умолчанию)

```
<form method="post">
  First name:<br>
  <input type="text" name="firstName"><br>
  Last name:<br>
  <input type="text" name="lastName"><br><br>
  <input type="radio" name="gender" value="male" checked>Male<br>
  <input type="radio" name="gender" value="female">Female<br>
  <input type="submit" value="Submit">
</form>
```

CSS

- Способ задать внешний вид элементов

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      body {background-color: powderblue;}
      h1   {color: blue;}
      p    {color: red;}
    </style>
  </head>
  <body>
    <h1>This is a heading</h1>
    <p>This is a paragraph.</p>
  </body>
</html>
```

© https://www.w3schools.com

Или, что более типично

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="styles.css">
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

© https://www.w3schools.com

Селекторы

```
<p id="p01">I am different</p>
<p class="error">Error message</p>
```

```
#p01 {
    color: blue;
}

p.error {
    color: red;
}
```

© https://www.w3schools.com

Немного JavaScript-а

```
<!DOCTYPE html>
<html>
<body>

<h1>My First JavaScript</h1>

<button type="button"
onclick="document.getElementById('demo').innerHTML = Date()">
Click me to display Date and Time.</button>

<p id="demo"></p>

</body>
</html>
```

© https://www.w3schools.com

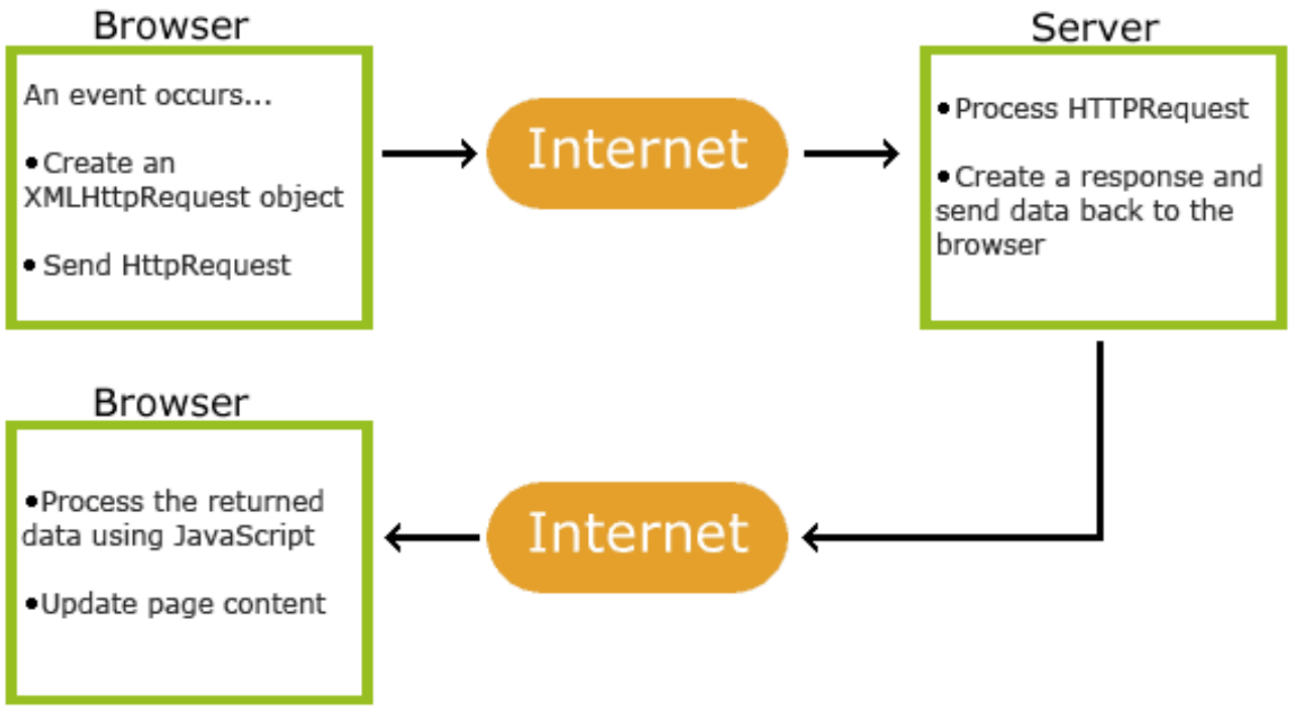
Или так

```
<!DOCTYPE html>
<html>
<head>
<script>
function doSomething() {
  document.getElementById("demo").style.fontSize = "25px";
  document.getElementById("demo").style.color = "red";
  document.getElementById("demo").style.backgroundColor = "yellow";
}
</script>
</head>
<body>

<button type="button" id="demo" onclick="doSomething()">Click me!</button>

</body>
</html>
```

© https://www.w3schools.com

AJAX

Asynchronous JavaScript And XML



Пример

```
<!DOCTYPE html>
<html>
<body>
<div id="demo">
  <h2>The XMLHttpRequest Object</h2>
  <button type="button" onclick="loadDoc()">Change Content</button>
</div>
<script>
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("demo").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();
}
</script>
</body>
</html>
```

Fetch API

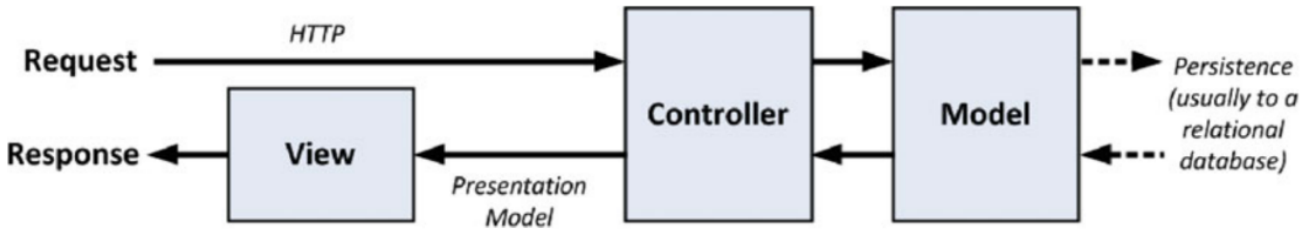
```
const data = { username: 'example' };

fetch('https://example.com/profile', {
  method: 'POST', // or 'PUT'
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify(data),
})
.then(response => response.json())
.then(data => {
  console.log('Success:', data);
})
.catch((error) => {
  console.error('Error:', error);
});
```

© https://developer.mozilla.org/

ASP.NET

- Появился в 2002 году (вместе с самим .NET), переписан под .NET Core в 2016
  - Вообще говоря, правильно говорить “ASP.NET Core”
- Кроссплатформенный, open-source, лицензия MIT
- Предполагает несколько моделей разработки:
  - многостраничное приложение
    - с использованием паттерна MVC
    - с использованием Razor Pages (по сути, паттерна, Model-View)
    - на Server-side Blazor
  - одностраничное приложение (SPA)
    - на клиентских JS-библиотеках и Web APIs
    - на Client-side Blazor и Web APIs
- На самом деле, “внутренности” одинаковы, отличаются шаблоны и подходы

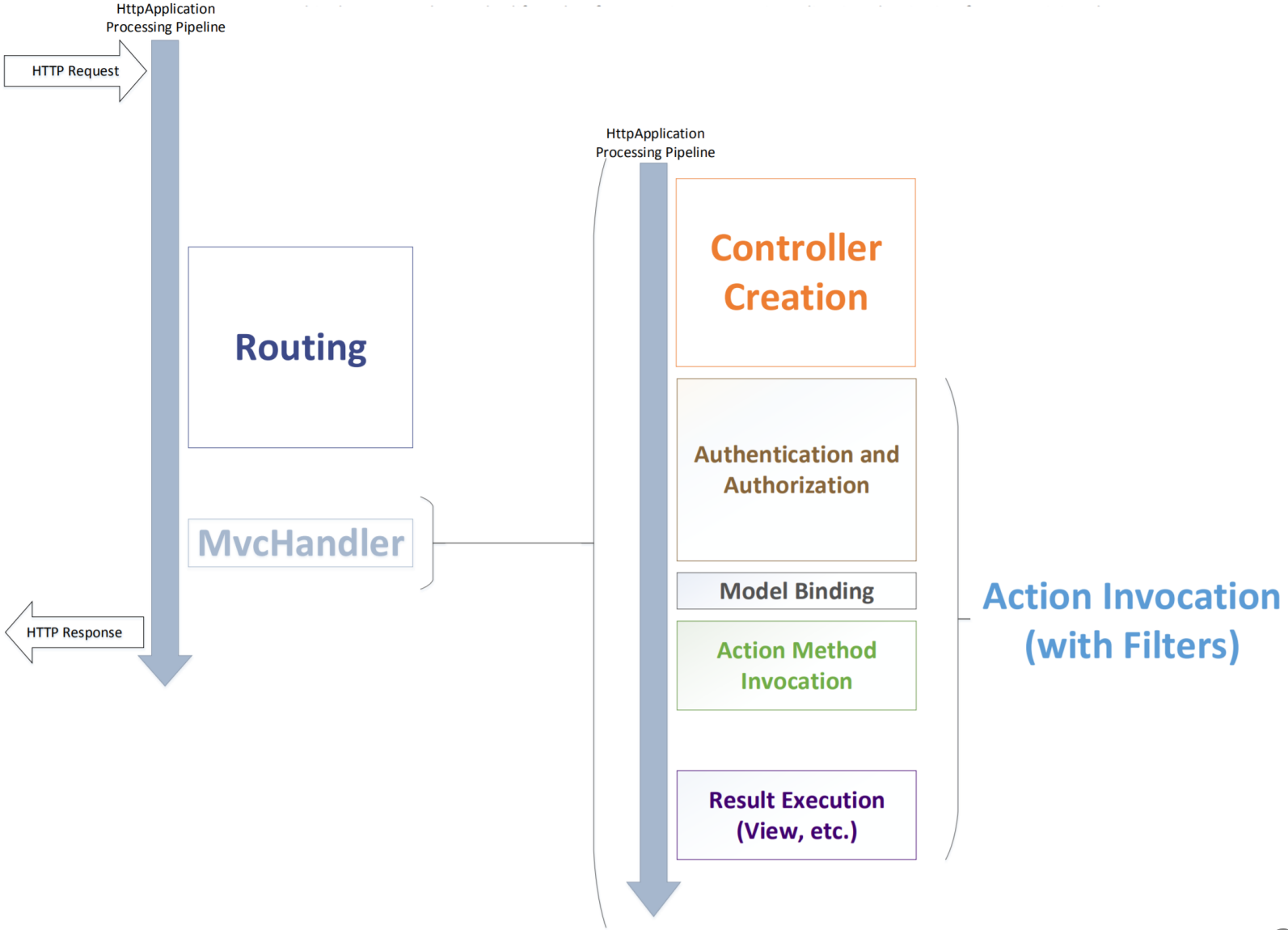
ASP.NET MVC, основные понятия



© A. Freeman, Pro ASP.NET Core MVC

- **Модель** содержит или представляет данные, с которыми работает приложение
  - **Domain model** содержит объекты предметной области вместе с бизнес-логикой, механизмами сериализации и т.д.
  - **View Model** содержит классы, удобные для отображения во View, без бизнес-логики
- **Представление** (View) отвечает за показ данных из модели пользователю
  - Работает в браузере, но генерится на сервере
- **Контроллер** отвечает за обработку входящих запросов, преобразование моделей и формирование данных для видов

Конвейер обработки запроса



© MSDN

Типичная структура проекта ASP.NET

- На самом деле, два шаблона проекта:
  - Web Application — Razor Pages, “облегчённая версия”
  - Web Application (Model-View-Controller) — “классическая” версия
- **wwwroot** — статические ресурсы приложения (то, что можно включать в html-страницу), отправляются клиенту как есть
  - favicon.ico — картинка, показывающаяся в заголовке вкладки
- **Controllers** — контроллеры, что неудивительно
  - Принцип Convention-over-configuration
- **Models** — доменные и view-модели
- **Views** — шаблоны HTML-страниц для представлений
  - Подпапки должны соответствовать контроллерам
  - Частичные представления
- **appsettings.json** — конфигурация приложения
- **Program.cs** — конфигурирует хост и запускает приложение
- **Startup.cs** — конфигурирует сервисы и конвейер обработки запроса

Razor

- Язык задания правил генерации
  - Обычно используется для генерации веб-страниц, но может использоваться и независимо
- Состоит из текста на целевом языке (в нашем случае html), кода на C# и Razor-разметки, которая собирает всё это воедино
- Сервер исполняет Razor-код, используя данные *модели* для генерации итоговой html-ки, которую и отправляет клиенту

Синтаксис Razor

- HTML-разметка пишется как есть
- Блоки кода заключаются в @{ }
- Один оператор можно писать без скобок:
  The time is @DateTime.Now
  - Обратите внимание, Razor-код выполняется на сервере!
- Выражения можно заключать в круглые скобки:
  @(someValue * 10)
- Всё, что выводится через @, кодируется вызовом HttpServerUtility.HtmlEncode

Пример

```
@page

<h1>Cthulhu fhtagn!</h1>

@for (int i = 0; i < 300; ++i)
{
    <p>Cthulhu fhtagn!</p>
}
```

Синтаксис Razor (2)

- Razor сам пытается угадать, где разметка, а где код
  - Но у него не всегда получается
- После @ и до пробела (или до конца оператора) — код
- После открывающего тэга — разметка
- После @: — HTML-разметка
- @* ... *@ — комментарии (серверные, не отправляются клиенту)
- @@ — @ в HTML (escaping)

Хелперы

- Функции, которые генерируют HTML-код
- Есть много стандартных
  - Html.BeginForm
  - Html.TextBox
  - Html.CheckBox
  - ...
  - Html.ActionLink
- Ещё бывают TagHelper-ы:
  @addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers

Пример:

```
<img src="~/images/asplogo.png" asp-append-version="true">
```

Сгенерится вот такое:

```
<img src="/images/asplogo.png?
  v=KI_dqr9NVtnMdsM2MUg4qthUnWZm5T1fCEimBPWDNgM">
```

Модели

- Всё, что выше, не сильно полезнее статической HTML
- Реальное веб-приложение использует данные из модели

```
@page
@using RazorPagesIntro.Pages
@model IndexModel2

<h2>Separate page model</h2>
<p>
  @Model.Message
</p>
```

Code behind

```
using Microsoft.AspNetCore.Mvc.RazorPages;
using System;

namespace RazorPagesIntro.Pages
{
    public class IndexModel2 : PageModel
    {
        public string Message { get; private set; } = "PageModel in C#";

        public void OnGet()
        {
            Message += $" Server time is { DateTime.Now }";
        }
    }
}
```

Модель в MVC

```
public IActionResult Error()
{
    return View(new ErrorViewModel {
        RequestId = Activity.Current?.Id
            ?? HttpContext.TraceIdentifier
    });
}
```

И View

```
@model ErrorViewModel

<h1 class="text-danger">Error.</h1>
<h2 class="text-danger">An error occurred
    while processing your request.</h2>

@if (Model?.ShowRequestId ?? false)
{
    <p>
        <strong>Request ID:</strong>
        <code>@Model?.RequestId</code>
    </p>
}
```

Роутинг

Или как ASP.NET находит по запросу страницу

- Convention over configuration — пока вы выполняете соглашения об именовании, по умолчанию всё происходит за вас
- Есть возможность конфигурировать роутинг вручную
- Соглашения Razor Pages:
  - URL вида “адрес сайта/” или “адрес сайта/Index” отображаются в /Pages/Index.cshtml
  - /Pages/Contact.cshtml — URL вида “адрес сайта/Contact”
  - /Pages/Store/Contact.cshtml — “адрес сайта/Store/Contact”
  - /Pages/Store/Index.cshtml — “адрес сайта/Store” или “адрес сайта/Store/Index”

Глобальное переопределение роутов

Razor Pages:

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

// Add services to the container.
builder.Services.AddRazorPages(
    options => options.Conventions.AddPageRoute("/Index", "/Ololo"));
```

MVC:

```
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");
```

Локальное переопределение роутов

Только для MVC

```
[Route("Ololo")]
public class HomeController : Controller
{
    [Route("")] // Прибавляется к роуту до контроллера,
                // чтобы получилось /Ololo
    [Route("Ololo")] // Прибавляется к роуту, чтобы
                // получилось "Ololo/Ololo"
    [Route("/")] // Никуда не прибавляется,
                // определяет роут ""
    public IActionResult Index() {}
}
```

Model binding

Или как передать в Code Behind параметры

```
@page
@model WebApplication1.Pages.IndexModel
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers

<html>
<body>
    <p>
        Enter your name:
    </p>
    <form method="post">
        Name: <input type="text" name="name" />
        <input type="submit" />
    </form>
</body>
</html>
```

(GET тоже будет работать)

Способ 1: в параметрах обработчика

```
namespace WebApplication1.Pages
{
    public class IndexModel : PageModel
    {
        public void OnGet()
        {

        }

        public void OnPost(string name)
        {
            Console.WriteLine(name);
        }
    }
}
```

Способ 2: вручную

```
public void OnPost()
{
    var name = Request.Form["name"];
    Console.WriteLine(name);
}
```

Способ 3: через свойства

```
[BindProperty]
public string Name { get; set; }

public void OnPost()
{
    Console.WriteLine(Name);
}
```