

ASP.NET Core Middleware

Антон Марченко, 2019

Работа web-приложения

- Клиент посылает запрос к серверу
- Сервер обрабатывает запрос
- Сервер посылает ответ клиенту

Обработка запроса

Сложный процесс

Включает различные шаги в зависимости от запроса

В ASP.NET Core это - конвейер обработки запросов и ответов Middleware pipeline

Контекст обработки хранится в **HttpContext**

HttpContext

```
public abstract class HttpContext
{
    public abstract ICollection<Feature> Features { get; }

    public abstract HttpRequest Request { get; }

    public abstract HttpResponse Response { get; }

    public abstract ConnectionInfo Connection { get; }

    public abstract WebSocketManager WebSockets
    { get; }
```

```
    public abstract ClaimsPrincipal User { get; set; }

    public abstract IDictionary<object, object> Items
    { get; set; }

    public abstract IServiceProvider RequestServices
    { get; set; }

    public abstract CancellationToken RequestAborted
    { get; set; }

    public abstract string TraceIdentifier { get; set; }

    public abstract ISession Session { get; set; }

    public abstract void Abort();
}
```

Middleware

- Промежуточный слой (в переводе)
- Цепочка компонент, где каждая
 - по необходимости передаёт запрос дальше
 - выполняет действия до и после вызова следующих компонент

Совсем простой пример

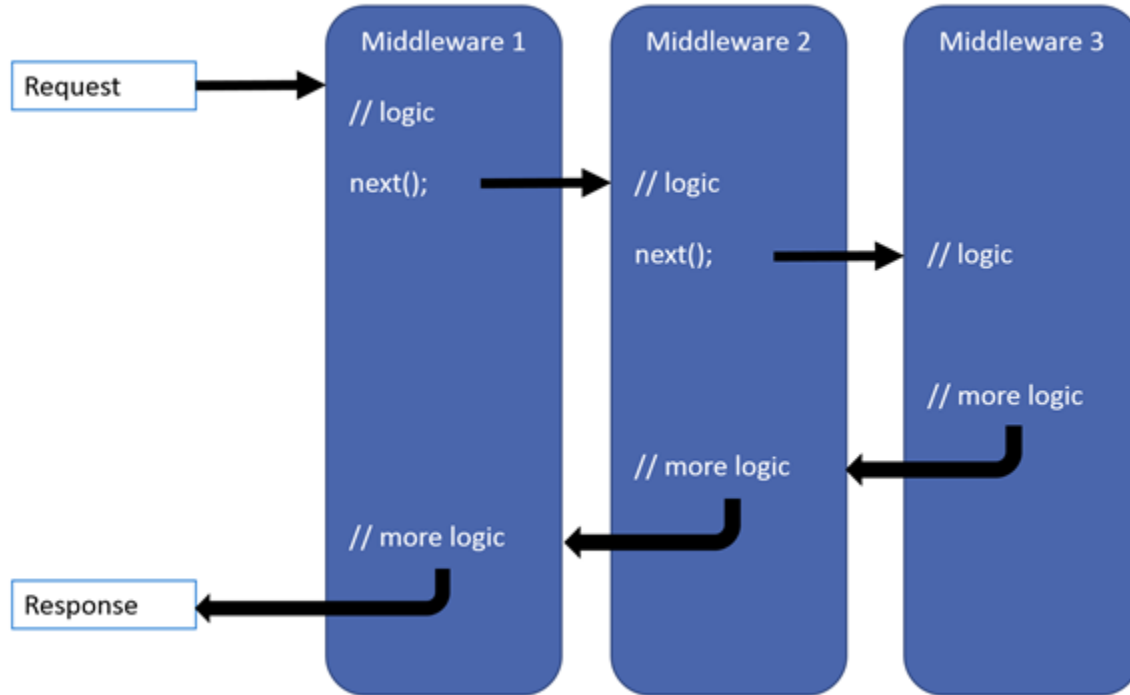
```
public class Startup
{
    public void Configure(IApplicationBuilder app)
    {
        app.Run(async context =>
        {
            await context.Response.WriteAsync("Hello, World!");
        });
    }
}
```

Чуть сложнее

```
public class Startup
```

```
{  
    public void Configure(IApplicationBuilder app)  
    {  
        app.Use(async (context, next) =>  
        {  
            // Do work that doesn't write to the Response.  
            await next.Invoke();  
            // Do logging or other work that doesn't write to the Response.  
        });  
  
        app.Run(async context =>  
        {  
            await context.Response.WriteAsync("Hello from 2nd delegate.");  
        });  
    }  
}
```

Middleware Pipeline



Конвейер

Связный(двусвязный) список компонентов

Каждая содержит ссылку на следующий или является терминальной

Каждая ответственна за:

- обработку своей части запроса
- передачи обработки дальше или замыкание конвейера

Выстраивание конвейера

Происходит в методе **Configure**

Используя **IApplicationBuilder**

- добавляя логику в методах **Use, Map, Run**
- добавляя типизированные компоненты с помощью расширения **UseMiddleware<T>**

С помощью делегата

```
public delegate Task RequestDelegate(HttpContext context);
```

Методы Use, Map, Run

Use - соединяет делегаты в цепочку

Map - разветвляет конвейер по совпадению
пути запроса

MapWhen - разветвляет конвейер по
предикату `Func<HttpContext, bool>`

Run - завершает конвейер

Метод Use

- Основной метод ApplicationBuilder для выстраивания конвейера
- Остальные используют его

```
public IApplicationBuilder Use(  
    Func<RequestDelegate,  
    RequestDelegate> middleware)  
{  
    this.componentsList.Add(middleware);  
    return (IApplicationBuilder) this;  
}
```

Метод Run

```
public static class RunExtensions
{
    public static void Run(this IApplicationBuilder app, RequestDelegate handler)
    {
        if (app == null)
            throw new ArgumentNullException(nameof (app));
        if (handler == null)
            throw new ArgumentNullException(nameof (handler));
        app.Use((Func<RequestDelegate, RequestDelegate>) (_ => handler));
    }
}
```

Метод Map

```
public static class MapExtensions
{
    public static IApplicationBuilder Map( this IApplicationBuilder app, PathString pathMatch, Action<IApplicationBuilder> configuration)
    {
        if (app == null)
            throw new ArgumentNullException(nameof (app));
        if (configuration == null)
            throw new ArgumentNullException(nameof (configuration));
        if (pathMatch.HasValue && pathMatch.Value.EndsWith("/", StringComparison.Ordinal))
            throw new ArgumentException("The path must not end with a '/'", nameof (pathMatch));
        IApplicationBuilder applicationBuilder = app.New();
        configuration(applicationBuilder);
        RequestDelegate requestDelegate = applicationBuilder.Build();
        MapOptions options = new MapOptions()
        {
            Branch = requestDelegate,
            PathMatch = pathMatch
        };

        return app.Use((Func<RequestDelegate, RequestDelegate>)
            (next => new RequestDelegate(new MapMiddleware(next,options).Invoke)));
    }
}
```

Map – пример

```
public class Startup
{
    private static void HandleMapTest1(IApplicationBuilder app)
    {
        app.Run(async context =>
        {
            await context.Response.WriteAsync("Map Test 1");
        });
    }

    private static void HandleMapTest2(IApplicationBuilder app)
    {
        app.Run(async context =>
        {
            await context.Response.WriteAsync("Map Test 2");
        });
    }

    public void Configure(IApplicationBuilder app)
    {
        app.Map("/map1", HandleMapTest1); // localhost:1234/map1
        app.Map("/map2", HandleMapTest2); // localhost:1234/map2
        app.Run(async context =>         // any other
        {
            await context.Response.WriteAsync("Hello from non-Map delegate. <p>");
        });
    }
}
```

Метод MapWhen

```
public static class MapWhenExtensions
{
    public static IApplicationBuilder MapWhen( this IApplicationBuilder app, Func<HttpContext, bool> predicate,
                                                Action<IApplicationBuilder> configuration)
    {
        if (app == null)
            throw new ArgumentNullException(nameof (app));
        if (predicate == null)
            throw new ArgumentNullException(nameof (predicate));
        if (configuration == null)
            throw new ArgumentNullException(nameof (configuration));
        IApplicationBuilder applicationBuilder = app.New();
        configuration(applicationBuilder);
        RequestDelegate requestDelegate = applicationBuilder.Build();
        MapWhenOptions options = new MapWhenOptions()
        {
            Predicate = predicate,
            Branch = requestDelegate
        };
        return app.Use((Func<RequestDelegate, RequestDelegate>)
            (next => new RequestDelegate(new MapWhenMiddleware(next, options).Invoke)));
    }
}
```


MapWhen – пример

```
public class Startup
{
    private static void HandleBranch(IApplicationBuilder app)
    {
        app.Run(async context =>
        {
            var branchVer = context.Request.Query["branch"];
            await context.Response.WriteAsync($"Branch used = {branchVer}");
        });
    }

    public void Configure(IApplicationBuilder app)
    {
        app.MapWhen(context => context.Request.Query.ContainsKey("branch"),
            HandleBranch); // localhost:1234/?branch=master
        app.Run(async context =>
        {
            await context.Response.WriteAsync("Hello from non-Map delegate. <p>");
        });
    }
}
```

Предупреждение

Аккуратней с отправкой ответа!

После записи `context.Response` его нельзя
изменять

- может нарушиться формат результата или протокол при несоответствии `Content-Length`

Запись в `Response` – только в терминальной
компоненте

Кастомные компоненты Middleware

Отдельные классы с логикой обработки запроса и управления конвейером

Подключаются в Configure, используя `builder.UseMiddleware`

Класс должен содержать

- конструктор, принимающий `RequestDelegate`
- метод `Invoke` или `InvokeAsync`, принимающий `HttpContext` и возвращающий `Task`

Дополнительные параметры

Кастомные компоненты Middleware могут содержать дополнительные параметры в конструкторе, которые подставляются согласно DI

Главное, чтобы присутствовал параметр RequestDelegate next

UseMiddleware<T> также может принимать дополнительные параметры напрямую.

Упрощение добавления

```
public static class StaticFileExtensions
{
    public static IApplicationBuilder UseStaticFiles(
        this IApplicationBuilder app)
    {
        if (app == null)
            throw new ArgumentNullException(nameof (app));
        return app.UseMiddleware<StaticFileMiddleware>();
    }
}
```

```
public static IApplicationBuilder UseStaticFiles(
    this IApplicationBuilder app,
    string requestPath)
{
    if (app == null)
        throw new ArgumentNullException(nameof (app));
    IApplicationBuilder app1 = app;
    StaticFileOptions options = new StaticFileOptions();
    options.RequestPath = new PathString(requestPath);
    return app1.UseStaticFiles(options);
}
```

```
public static IApplicationBuilder UseStaticFiles(
    this IApplicationBuilder app,
    StaticFileOptions options)
{
    if (app == null)
        throw new ArgumentNullException(nameof (app));
    if (options == null)
        throw new ArgumentNullException(nameof (options));
    return app.UseMiddleware<StaticFileMiddleware>
        ((object)Microsoft.Extensions.Options.Options.Create<StaticFileOptions>(options));
}
```

Пример пользовательского middleware

```
public class RequestCultureMiddleware
{
    private readonly RequestDelegate _next;

    public RequestCultureMiddleware(RequestDelegate next)
    {
        _next = next;
    }

    public async Task InvokeAsync(HttpContext context)
    {
        var cultureQuery = context.Request.Query["culture"];
        if (!string.IsNullOrEmpty(cultureQuery))
        {
            var culture = new CultureInfo(cultureQuery);

            CultureInfo.CurrentCulture = culture;
            CultureInfo.CurrentUICulture = culture;
        }

        // Call the next delegate/middleware in the pipeline
        await _next(context);
    }
}
```

Порядок вызова

Конвейер Middleware - двусвязный список

Порядок обработки фиксируется

ApplicationBuilder'ом

С помощью порядка обеспечивается
безопасность, производительность,
функциональность

Параметр для Invoke

```
public class CustomMiddleware
{
    private readonly RequestDelegate _next;

    public CustomMiddleware(RequestDelegate next)
    {
        _next = next;
    }

    // IMyScopedService is injected into Invoke
    public async Task Invoke(HttpContext httpContext, IMyScopedService svc)
    {
        svc.MyProperty = 1000;
        await _next(httpContext);
    }
}
```


Обратные вызовы

Сперва вызывается код до
`next.Invoke(context);`

После обработки всех последующих
компонент, вызывается код после
`next.Invoke(context);`

Middleware обработки ошибок

Сначала передаёт запрос дальше по цепочке

После их обработки анализирует
`context.Response` и в случае ошибки выдаёт
соответствующую информацию

// Enable the Exception Handler Middleware to catch exceptions thrown in the following middlewares.

```
app.UseExceptionHandler("/Error");
```

// Use HTTPS Redirection Middleware to redirect HTTP requests to HTTPS.

```
app.UseHttpsRedirection();
```

// Return static files and end the pipeline.

```
app.UseStaticFiles();
```

// Use Cookie Policy Middleware to conform to EU General Data Protection Regulation (GDPR) regulations.

```
app.UseCookiePolicy();
```

// Authenticate before the user accesses secure resources.

```
app.UseAuthentication();
```

// If the app uses session state, call Session Middleware after Cookie Policy Middleware and before MVC Middleware.

```
app.UseSession();
```

// Add MVC to the request pipeline.

```
app.UseMvc();
```

Примеры стандартных компонент

Добавляются до использующих их компонент

- CORS
- Кэширование
- Сжатие
- Локализация
- WebSockets

Middleware vs Services

Middleware - получают и обрабатывают запросы пользователя

Service - вспомогательные компоненты, могут использоваться в других, обычно внедряются с помощью DI

Но это уже совсем другая история

Маршрутизация в ASP.NET Core

`{controller=Home}/{action=Index}/{id?}`

- Маршрутизация использует *конечные точки* (Endpoint) для представления логических конечных точек в приложении
- Конечная точка определяет делегат для обработки запросов и коллекцию произвольных метаданных.

Маршрутизация в ASP.NET Core

- Синтаксис шаблона маршрута используется для определения маршрутов с помощью параметров размеченного маршрута.
- Допускается конфигурация конечной точки в стандартном стиле и с атрибутами.
- [IRouteConstraint](#) используется для определения того, содержит ли параметр URL-адреса допустимое значение для ограничения заданной конечной точки.
- Модели приложения, такие как MVC и Razor Pages, регистрируют все свои конечные точки.
- Реализация маршрутизации принимает решения о маршрутизации в нужном месте конвейера ПО промежуточного слоя.

Маршрутизация в ASP.NET Core

- Можно перечислить все конечные точки в приложении в любом месте конвейера ПО промежуточного слоя.
- Приложение может использовать маршрутизацию для формирования URL-адресов на основе сведений о конечной точке, что позволяет избежать жесткого задания URL-адресов и упрощает поддержку.
- Формирование URL-адреса основано на адресах, которые поддерживают произвольную расширяемость:
 - API генератора ссылок ([LinkGenerator](#)) может быть разрешено в любом месте с помощью [внедрения зависимостей \(DI\)](#) для формирования URL-адреса.
 - Если API генератора ссылок недоступно через внедрение зависимостей, [IUrlHelper](#) предлагает методы для создания URL-адреса.

Свойства RouteData

- [RouteData.Values](#) — это словарь *значений маршрута*, полученных из маршрута.
- [RouteData.DataTokens](#) — это контейнер свойств с дополнительными данными, связанными с соответствующим маршрутом. Эти значения определяются разработчиком и никоим образом **не** влияют на поведение маршрутизации. Кроме того, значения, спрятанные в токенах.
- [RouteData.Routers](#) — это список маршрутов, которые участвовали в успешном сопоставлении с запросом. Маршруты могут быть вложены друг в друга. Свойство [Routers](#) отражает путь по логическому дереву маршрутов, который привел к совпадению. Последний элемент в свойстве [Routers](#) — это соответствующий обработчик маршрутов.

Создание URL-адреса с помощью LinkGenerator

[LinkGenerator](#) - генератор ссылок. Внедряется с помощью зависимостей. Может установить связь с действиями и страницами MVC и Razor Pages с помощью следующих методов расширения:

- [GetPathByAction](#)
- [GetUriByAction](#)
- [GetPathByPage](#)
- [GetUriByPage](#)

Маршрутизация конечных точек

Маршрутизация конечной точки интегрируется с ПО промежуточного слоя с использованием двух методов расширения:

- [UseRouting](#) добавляет соответствие маршрута в конвейер ПО промежуточного слоя
- [UseEndpoints](#) добавляет выполнение конечной точки в конвейер ПО промежуточного слоя

Пример

```
public void Configure(IApplicationBuilder app)
{
    // Matches request to an endpoint.
    app.UseRouting();

    // Endpoint aware middleware.
    // Middleware can use metadata from the matched endpoint.
    app.UseAuthorization();

    // Execute the matched endpoint.
    app.UseEndpoints(endpoints =>
    {
        // Configuration of app endpoints.
        endpoints.MapRazorPages();
        endpoints.MapGet("/", context => context.Response.WriteAsync("Hello world"));
        endpoints.MapHealthChecks("/healthz");
    });
}
```

Пример в middleware

```
public class ProductsLinkMiddleware
{
    private readonly LinkGenerator _linkGenerator;

    public ProductsLinkMiddleware(RequestDelegate next, LinkGenerator linkGenerator)
    {
        _linkGenerator = linkGenerator;
    }

    public async Task InvokeAsync(HttpContext httpContext)
    {
        var url = _linkGenerator.GetPathByAction("ListProducts", "Store");

        httpContext.Response.ContentType = "text/plain";

        await httpContext.Response.WriteAsync($"Go to {url} to see our products.");
    }
}
```

Создание маршрутов

```
routes.MapRoute(  
    name: "default",  
    template: "{controller=Home}/{action=Index}/{id?}");
```

```
routes.MapRoute(  
    name: "us_english_products",  
    template: "en-US/Products/{id}",  
    defaults: new { controller = "Products", action = "Details" },  
    constraints: new { id = new IntRouteConstraint() },  
    dataTokens: new { locale = "en-US" });
```

Регистрация маршрутов

```
var trackPackageRouteHandler = new RouteHandler(context =>
{
    var routeValues = context.GetRouteData().Values;
    return context.Response.WriteAsync(
        $"Hello! Route values: {string.Join(", ", routeValues)}");
});
```

```
var routeBuilder = new RouteBuilder(app, trackPackageRouteHandler);
```

```
routeBuilder.MapRoute(
    "Track Package Route",
    "package/{operation:regex(^track|create$)}/{id:int}");
```

```
routeBuilder.MapGet("hello/{name}", context =>
{
    var name = context.GetRouteValue("name");
    // The route handler when HTTP GET "hello/<anything>" matches
    // To match HTTP GET "hello/<anything>/<anything>",
    // use routeBuilder.MapGet("hello/{*name}")
    return context.Response.WriteAsync($"Hi, {name}!");
});
```

```
var routes = routeBuilder.Build();
app.UseRouter(routes);
```

Шаблон маршрутов

{ ... } – параметры маршрута

{controller=Home}{action=Index} – некорректно (нет разделителя)

files/{filename}.{ext?} – подходит и для маршрута с расширением и без («.» – опциональна):

- /files/myFile.txt
- /files/myFile

blog/{**slug} – для любого продолжения после «blog/»

Можно создавать ограничения на параметры маршрута:

blog/{article:minlength(10)}

См. <https://docs.microsoft.com/ru-ru/aspnet/core/fundamentals/routing?view=aspnetcore-3.1#route-constraint-reference>

Зарезервированные имена: action, area, controller, handler, page

Можно использовать регулярные выражения

Обработка ошибок

```
if (env.IsDevelopment())
{
    app.UseDeveloperExceptionPage();
}
else
{
    app.UseExceptionHandler("/Error");
    app.UseHsts();
}
```

Для MVC:

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

Lambda для исключений

```
app.UseExceptionHandler(errorApp =>
{
    errorApp.Run(async context =>
    {
        context.Response.StatusCode = 500;
        context.Response.ContentType = "text/html";

        await context.Response.WriteAsync("<html lang='en'><body>\r\n");
        await context.Response.WriteAsync("ERROR!<br><br>\r\n");

        var exceptionHandlerPathFeature =
            context.Features.Get<IExceptionHandlerPathFeature>();

        // Use exceptionHandlerPathFeature to process the exception (for example,
        // logging), but do NOT expose sensitive error information directly to
        // the client.

        if (exceptionHandlerPathFeature?.Error is FileNotFoundException)
            await context.Response.WriteAsync("File error thrown!<br><br>\r\n");

        await context.Response.WriteAsync("<a href='/'>Home</a><br>\r\n");
        await context.Response.WriteAsync("</body></html>\r\n");
        await context.Response.WriteAsync(new string(' ', 512)); // IE padding
    });
});
```

Обработка ошибок HTTP

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    // обработка ошибок HTTP
    app.UseStatusCodePages("text/plain", "Error. Status code : {0}");
    // можно выполнить редирект
    // app.UseStatusCodePagesWithRedirects("/error?code={0}");
    // но лучше так:
    // app.UseStatusCodePagesWithReExecute("/error", "?code={0}");

    app.Map("/hello", ap => ap.Run(async (context) =>
    {
        await context.Response.WriteAsync($"Hello ASP.NET Core");
    }));
}
```

Static files

- **Wwwroot**
 - **css**
 - **images**
 - **js**
- **MyStaticFiles**
 - **images**
 - *banner1.svg*

```
public void Configure(IApplicationBuilder app)
{
    app.UseStaticFiles(); // For the wwwroot folder

    app.UseStaticFiles(new StaticFileOptions
    {
        FileProvider = new PhysicalFileProvider(
            Path.Combine(Directory.GetCurrentDirectory(), "MyStaticFiles")),
        RequestPath = "/StaticFiles"
    });
}
```

```

```

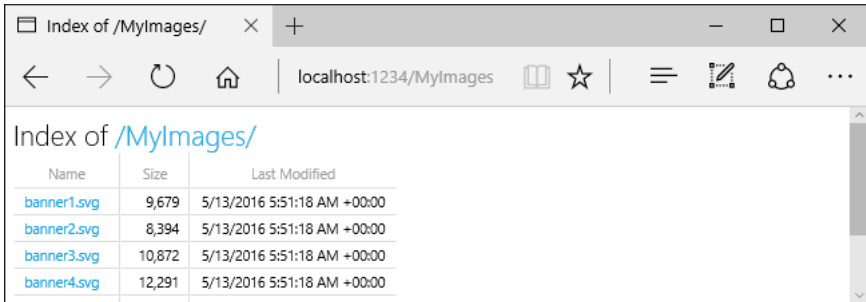
Directory browsing

```
public void Configure(IApplicationBuilder app)
{
    app.UseStaticFiles(); // For the wwwroot folder

    app.UseStaticFiles(new StaticFileOptions
    {
        FileProvider = new PhysicalFileProvider(
            Path.Combine(Directory.GetCurrentDirectory(), "wwwroot", "images")),
        RequestPath = "/MyImages"
    });

    app.UseDirectoryBrowser(new DirectoryBrowserOptions
    {
        FileProvider = new PhysicalFileProvider(
            Path.Combine(Directory.GetCurrentDirectory(), "wwwroot", "images")),
        RequestPath = "/MyImages"
    });
}
```

`http://<server_address>/MyImages`



Файл по умолчанию

```
public void Configure(IApplicationBuilder app)
{
    app.UseDefaultFiles();
    app.UseStaticFiles();
}
```

- default.htm
- default.html
- index.htm
- index.html

Можно объединить:

```
public void Configure(IApplicationBuilder app)
{
    app.UseStaticFiles(); // For the wwwroot folder

    app.UseFileServer(new FileServerOptions
    {
        FileProvider = new PhysicalFileProvider(
            Path.Combine(Directory.GetCurrentDirectory(), "MyStaticFiles")),
        RequestPath = "/StaticFiles",
        EnableDirectoryBrowsing = true
    });
}
```

Состояние приложения

```
public void Configure(IApplicationBuilder app)
{
    app.Use(async (context, next) =>
    {
        context.Items["text"] = "Text from HttpContext.Items";
        await next.Invoke();
    });

    app.Run(async (context) =>
    {
        context.Response.ContentType = "text/html; charset=utf-8";
        await context.Response.WriteAsync($"Текст: {context.Items["text"]}");
    });
}
```

Cookie

```
public void Configure(IApplicationBuilder app)
{
    app.Run(async (context) =>
    {
        if (context.Request.Cookies.ContainsKey("name"))
        {
            string name = context.Request.Cookies["name"];
            await context.Response.WriteAsync($"Hello {name}!");
        }
        else
        {
            context.Response.Cookies.Append("name", "Tom");
            await context.Response.WriteAsync("Hello World!");
        }
    });
}
```


Сессии

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDistributedMemoryCache();
    services.AddSession();
}
public void Configure(IApplicationBuilder app)
{
    app.UseSession();
    app.Run(async (context) =>
    {
        if (context.Session.Keys.Contains("name"))
            await context.Response.WriteAsync($"Hello {context.Session.GetString("name")}!");
        else
        {
            context.Session.SetString("name", "Tom");
            await context.Response.WriteAsync("Hello World!");
        }
    });
}
```

Хранение сложных объектов

```
public static class SessionExtensions
{
    public static void Set<T>(this ISession session, string key, T value)
    {
        session.SetString(key, JsonSerializer.Serialize<T>(value));
    }

    public static T Get<T>(this ISession session, string key)
    {
        var value = session.GetString(key);
        return value == null ? default(T) : JsonSerializer.Deserialize<T>(value);
    }
}
```

Глобализация и локализация

System.Globalization. CultureInfo

культура-субкультура ("en-US" и "en-GB")

```
public string GetCulture(string code = "")
{
    if (!String.IsNullOrEmpty(code))
    {
        CultureInfo.CurrentCulture = new CultureInfo(code);
        CultureInfo.CurrentUICulture = new CultureInfo(code);
    }
    return $"CurrentCulture:{CultureInfo.CurrentCulture.Name}, CurrentUICulture:{CultureInfo.CurrentUICulture.Name}";
}
```

Свой middleware

```
public class CultureMiddleware
{
    private readonly RequestDelegate _next;

    public CultureMiddleware(RequestDelegate next)
    {
        this._next = next;
    }

    // продолжение ---->>>>>
```

```
public async Task Invoke(HttpContext context)
{
    var lang = context.Request.Query["lang"].ToString();
    if (!string.IsNullOrEmpty(lang))
    {
        try
        {
            CultureInfo.CurrentCulture = new CultureInfo(lang);
            CultureInfo.CurrentUICulture = new CultureInfo(lang);
        }
        catch (CultureNotFoundException) { }
    }
    await _next.Invoke(context);
}

public static class CultureExtensions
{
    public static IApplicationBuilder UseCulture(this IApplicationBuilder
builder)
    {
        return builder.UseMiddleware<CultureMiddleware>();
    }
}
```

RequestLocalizationMiddleware

```
public void Configure(IApplicationBuilder app)
{
    ...
    var supportedCultures = new[]
    {
        new CultureInfo("en-US"),
        new CultureInfo("en-GB"),
        new CultureInfo("en"),
        new CultureInfo("ru-RU"),
        new CultureInfo("ru"),
        new CultureInfo("de-DE"),
        new CultureInfo("de")
    };
    app.UseRequestLocalization(new RequestLocalizationOptions
    {
        DefaultRequestCulture = new RequestCulture("ru-RU"),
        SupportedCultures = supportedCultures,
        SupportedUICultures = supportedCultures
    });
    ...
}
```

Локализация строк. IStringLocalizer

```
public class CustomStringLocalizer : IStringLocalizer
```

```
{
    Dictionary<string, Dictionary<string, string>> resources;
    // ключи ресурсов
    const string HEADER = "Header";
    const string MESSAGE = "Message";

    public CustomStringLocalizer()
    {
        // словарь для английского языка
        Dictionary<string, string> enDict = new Dictionary<string, string>
        {
            {HEADER, "Welcome" },
            {MESSAGE, "Hello World!" }
        };
        // словарь для русского языка
        Dictionary<string, string> ruDict = new Dictionary<string, string>
        {
            {HEADER, "Добо пожаловать" },
            {MESSAGE, "Привет мир!" }
        };
        // словарь для немецкого языка
        Dictionary<string, string> deDict = new Dictionary<string, string>
        {
            {HEADER, "Willkommen" },
            {MESSAGE, "Hallo Welt!" }
        };
        // создаем словарь ресурсов
        resources = new Dictionary<string, Dictionary<string, string>>
        {
            {"en", enDict },
            {"ru", ruDict },
            {"de", deDict }
        };
    }
}
```

```
// по ключу выбираем для текущей культуры нужный ресурс
```

```
public LocalizedString this[string name]
{
    get
    {
        var currentCulture = CultureInfo.CurrentUICulture;
        string val = "";
        if (resources.ContainsKey(currentCulture.Name))
        {
            if (resources[currentCulture.Name].ContainsKey(name))
            {
                val = resources[currentCulture.Name][name];
            }
        }
        return new LocalizedString(name, val);
    }
}

public LocalizedString this[string name, params object[] arguments] => throw
new NotImplementedException();

public IEnumerable<LocalizedString> GetAllStrings(bool includeParentCultures)
{
    throw new NotImplementedException();
}

public IStringLocalizer WithCulture(CultureInfo culture)
{
    return this;
}
}
```

Ресурсы и локализация в контроллерах

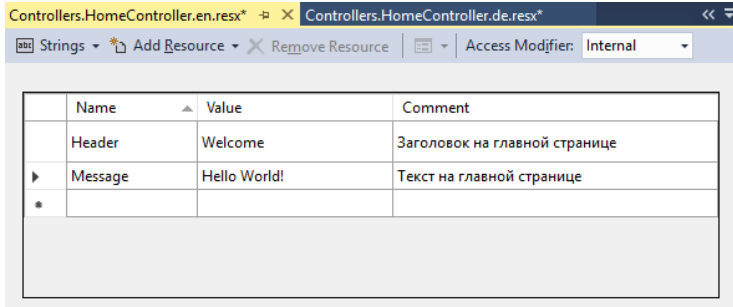
Файлы ресурсов должны называться по имени файла с учетом пространства имен, но без названия проекта, если оно совпадает с именем сборки. Например, ресурс для класса Startup должен представлять файл *Startup.[код_культуры].resx*.

Add→ Resource Files:

Resources/Controllers.HomeController.ru.resx

Resources/Controllers/HomeController.ru.resx

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddLocalization(options => options.ResourcesPath = "Resources");
    services.AddControllersWithViews();
}
```



The screenshot shows the Visual Studio Resource Designer interface. At the top, there are tabs for 'Controllers.HomeController.en.resx' and 'Controllers.HomeController.de.resx'. Below the tabs is a toolbar with buttons for 'Strings', 'Add Resource', 'Remove Resource', and 'Access Modifier' (set to 'Internal'). The main area contains a table with columns 'Name', 'Value', and 'Comment'.

Name	Value	Comment
Header	Welcome	Заголовок на главной странице
Message	Hello World!	Текст на главной странице
*		

```
private readonly IStringLocalizer<HomeController> _localizer;
public HomeController(IStringLocalizer<HomeController> localizer)
{
    _localizer = localizer;
}
public IActionResult Index()
{
    ViewData["Title"] = _localizer["Header"];
    ViewData["Message"] = _localizer["Message"];
    return View();
}
```

Локализация аннотаций данных

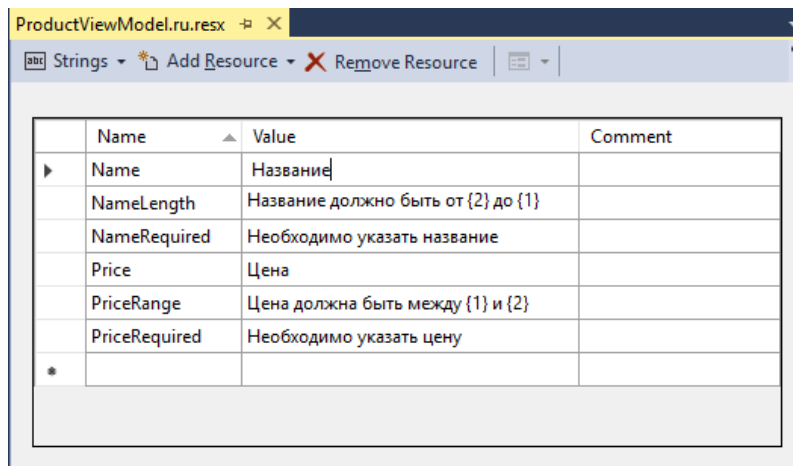
```
using System.ComponentModel.DataAnnotations;
```

```
namespace LocalizationApp.ViewModels
{
    public class ProductViewModel
    {
        [Required(ErrorMessage = "NameRequired")]
        [StringLength(20, ErrorMessage = "NameLength", MinimumLength = 6)]
        [Display(Name = "Name")]
        public string Name { get; set; }

        [Required(ErrorMessage = "PriceRequired")]
        [Range(10, 100, ErrorMessage = "PriceRange")]
        [Display(Name = "Price")]
        public int Price { get; set; }
    }
}
```

Resources/ViewModels/ProductViewModel.ru.resx
Resources/ViewModels/ProductViewModel.en.resx
Resources/ViewModels/ProductViewModel.de.resx

```
services.AddControllersWithViews()
        .AddDataAnnotationsLocalization()
        .AddViewLocalization();
```



The screenshot shows the Visual Studio Resource Manager for the file `ProductViewModel.ru.resx`. The table displays the following data:

	Name	Value	Comment
▶	Name	Название	
	NameLength	Название должно быть от {2} до {1}	
	NameRequired	Необходимо указать название	
	Price	Цена	
	PriceRange	Цена должна быть между {1} и {2}	
	PriceRequired	Необходимо указать цену	
*			

Общие ресурсы локализации

```
public class SharedResource
{
}
```

SharedResource.ru.resx
SharedResource.en.resx
SharedResource.de.resx

```
public class HomeController : Controller
{
    private readonly IStringLocalizer<HomeController> _localizer;
    private readonly IStringLocalizer<SharedResource> _sharedLocalizer;
    public HomeController(IStringLocalizer<HomeController> localizer,
        IStringLocalizer<SharedResource> sharedLocalizer)
    {
        _localizer = localizer;
        _sharedLocalizer = sharedLocalizer;
    }
    public string Test()
    {
        // получаем ресурс Message
        string message = _sharedLocalizer["Message"];
        return message;
    }
}
```

Хранение ресурсов в базе данных

Стандартно описывается модель, контекст.
См.

<https://metanit.com/sharp/aspnet5/28.9.php>

Ссылки

<https://docs.microsoft.com/ru-ru/aspnet/core/fundamentals/middleware/>

Книга ASP.NET Core in Action - ANDREW LOCK