управление данными в микросервисах на C#

ASP.NET

web vs application server

web vs application sever

web server

- управление соединениями с клиентами
- реализация сетевых протоколов
- обслуживание сетевых запросов

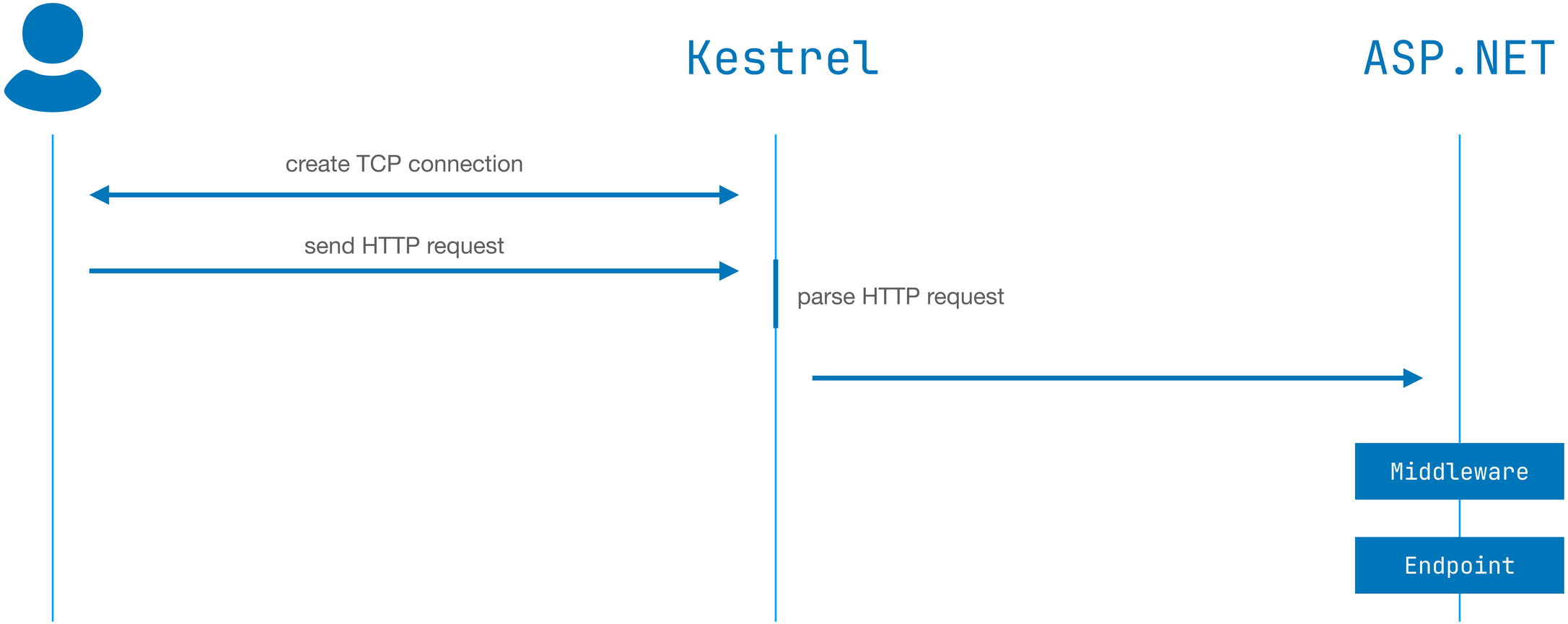
web vs application sever

application server

- обслуживает логику приложения
- обслуживает логику инфраструктуры
- обрабатывает запросы пользователей

ASP.NET request pipeline

ASP.NET request pipeline



конфигурация хоста

конфигурация хоста

пустой шаблон

```
WebApplicationBuilder builder = WebApplication.CreateBuilder(args);
WebApplication app = builder.Build();

app.MapGet("/", () ⇒ "Hello World!");

app.Run();
```

конфигурация хоста

DI и конфигурации

```
builder.Services.AddScoped<IApplicationService, ApplicationService>();

IConfigurationBuilder configurationBuilder = builder.Configuration;
configurationBuilder.Add(new MyConfigurationSource());
```

конфигурация хоста

конфигурации по умолчанию

- аргументы консоли
- переменные окружения
- `appsettings.json`
- `appsettings.{ASPNETCORE_ENVIRONMENT}.json`
  - значение среды берётся из переменных окружения

конфигурация хоста

Host

```
builder.Host.ConfigureServices(x ⇒ x.AddScoped<IApplicationService, ApplicationService>());
builder.Host.ConfigureAppConfiguration(x ⇒ x.Add(new MyConfigurationSource()));
```

конфигурация хоста

WebHost

```
builder.WebHost.UseKestrel();
```

конфигурация хоста

виды билдеров хоста

- WebApplication.CreateBuilder
- WebApplication.CreateSlimBuilder
  - минимально необходимые для запуска конфигурации
- WebApplication.CreateEmptyBuilder
  - никаких конфигураций

конфигурация хоста

хост без веб сервера

```
public class EmptyServer : IServer
{
    public IFeatureCollection Features { get; } = new FeatureCollection();

    public Task StartAsync<TContext>(
        IHttpApplication<TContext> application,
        CancellationToken cancellationToken)
        where TContext : notnull
    {
        return Task.CompletedTask;
    }

    public Task StopAsync(CancellationToken cancellationToken)
    {
        return Task.CompletedTask;
    }

    public void Dispose() { }
}
```

конфигурация хоста

хост без веб сервера

```
WebApplicationBuilder builder = WebApplication.CreateEmptyBuilder(new());

builder.Services.AddLogging(x ⇒ x.AddConsole());
builder.WebHost.UseServer(new EmptyServer());

WebApplication app = builder.Build();
app.Run();
```

```
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /Users/george/Documents/dotnet/playground/asp/Playground.AspNet.Empty
```

hosted services

hosted services

IHostedServices

```
public interface IHostedService
{
    Task StartAsync(CancellationToken cancellationToken);

    Task StopAsync(CancellationToken cancellationToken);
}
```

hosted services

BackgroundService

```
public class MyBackgroundService : BackgroundService
{
    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
    {
        using var timer = new PeriodicTimer(TimeSpan.FromSeconds(2));

        while (await timer.WaitForNextTickAsync(stoppingToken))
        {
            Console.WriteLine($"Hello, it's {DateTimeOffset.UtcNow:h:mm:ss}!");
        }
    }
}
```

hosted services

BackgroundService

```
builder.Services.AddHostedService<MyBackgroundService>();
```

```
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /Users/george/Documents/dotnet/playground/asp/Playground.AspNet.Empty
Hello, it's 6:57:57!
Hello, it's 6:57:59!
Hello, it's 6:58:01!
```

hosted services

DI

```
public class MyBackgroundService : BackgroundService
{
    private readonly IServiceScopeFactory _scopeFactory;
    private readonly ILogger<MyBackgroundService> _logger;

   ...

    private async Task ExecuteSingleAsync(CancellationToken cancellationToken)
    {
        await using AsyncServiceScope scope = _scopeFactory.CreateAsyncScope();
        using var timer = new PeriodicTimer(TimeSpan.FromSeconds(2));

        IApplicationService service = scope.ServiceProvider.GetRequiredService<IApplicationService>();

        while (await timer.WaitForNextTickAsync(cancellationToken))
        {
            await service.DoSomeBackgroundOperationAsync(cancellationToken);
        }
    }
}
```

hosted services

управление временем жизни

```
public class MyBackgroundService : BackgroundService
{
   ...

    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
    {
        while (stoppingToken.IsCancellationRequested is false)
        {
            try
            {
                await ExecuteSingleAsync(stoppingToken);
            }
            catch (Exception e)
            {
                _logger.LogError(e, "Error while executing MyBackgroundService");
            }
        }
    }

   ...
}
```

hosted services

особенности реализации BackgroundService

- при вызове метода `StartAsync` таска вашей операции не авейтится, а складывается в поле
- стоит помнить про то, как выполняется асинхронный код до `await`

особенности реализации BackgroundService

обязательная логика перед выполнением операции

```
public class MyBackgroundService : BackgroundService
{
    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
    {
        await DoSomeInitializationRequiredBeforeApplicationStart();

        // Some background service logic
        // ...
    }

    private async Task DoSomeInitializationRequiredBeforeApplicationStart()
    {
        // Some initialization logic
        // ...
    }
}
```

особенности реализации BackgroundService

обязательная логика перед выполнением операции

```
public class MyBackgroundService : BackgroundService
{
    public override async Task StartAsync(CancellationToken cancellationToken)
    {
        await DoSomeInitializationRequiredBeforeApplicationStart();
        await base.StartAsync(cancellationToken);
    }

    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
    {
        // Some background service logic
        // ...
    }

    private async Task DoSomeInitializationRequiredBeforeApplicationStart()
    {
        // Some initialization logic
        // ...
    }
}
```

особенности реализации BackgroundService

блокировка запуска приложения

```
public class MyBackgroundService : BackgroundService
{
    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
    {
        Thread.Sleep(TimeSpan.FromHours(24));
        await ExecuteServiceOperationAsync(stoppingToken);
    }

    private async Task ExecuteServiceOperationAsync(CancellationToken cancellationToken)
    {
        // Some background service logic
        _ = cancellationToken;
    }
}
```

особенности реализации BackgroundService

блокировка запуска приложения

```
public class MyBackgroundService : BackgroundService
{
    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
    {
        await Task.Yield();

        Thread.Sleep(TimeSpan.FromHours(24));
        await ExecuteServiceOperationAsync(stoppingToken);
    }

    private async Task ExecuteServiceOperationAsync(CancellationToken cancellationToken)
    {
        // Some background service logic
        _ = cancellationToken;
    }
}
```

HTTP эндпоинты

HTTP эндпоинты

Minimal API

```
app.MapGet(pattern: "/", () ⇒ "Hello World!");
app.MapPost(pattern: "/users", () ⇒ "User created!");
```

HTTP эндпоинты

Minimal API

```
app.MapGet(
    pattern: "/api/users/{userId}/posts",
    (
        [FromRoute] long userId,
        [FromQuery] int pageSize,
        [FromQuery] string pageToken) => { });
```

```
http://localhost:8080/api/users/1/posts?pageSize=30&pageToken=123
```

HTTP эндпоинты

Minimal API

```
app.MapGet("/api/users/{userId}/posts",
    async (
        [FromRoute] long userId,
        [FromQuery] int pageSize,
        [FromQuery] string pageToken,
        [FromServices] IUserService userService,
        CancellationToken cancellationToken) ⇒
    {
        Post[] posts = await userService
            .SearchUserPostsAsync(userId, pageSize, pageToken, cancellationToken)
            .ToArrayAsync();

        return Results.Ok(posts);
    });
```

HTTP эндпоинты

Controllers

```
[ApiController]
[Route("api/[controller]")]
public class UsersController : ControllerBase { }
```

HTTP эндпоинты

Controllers

```
[ApiController]
[Route("api/[controller]")]
public class UsersController : ControllerBase
{
    private readonly IUserService _userService;

    public UsersController(IUserService userService)
    {
        _userService = userService;
    }

    [HttpGet("{userId}/posts")]
    public async Task<ActionResult<Post[]>> SearchUserPostsAsync(
        [FromRoute] long userId,
        [FromQuery] int pageSize,
        [FromQuery] string pageToken,
        CancellationToken cancellationToken)
    {
        Post[] posts = await _userService
            .SearchUserPostsAsync(userId, pageSize, pageToken, cancellationToken)
            .ToArrayAsync(cancellationToken);

        return Ok(posts);
    }
}
```

HTTP эндпоинты

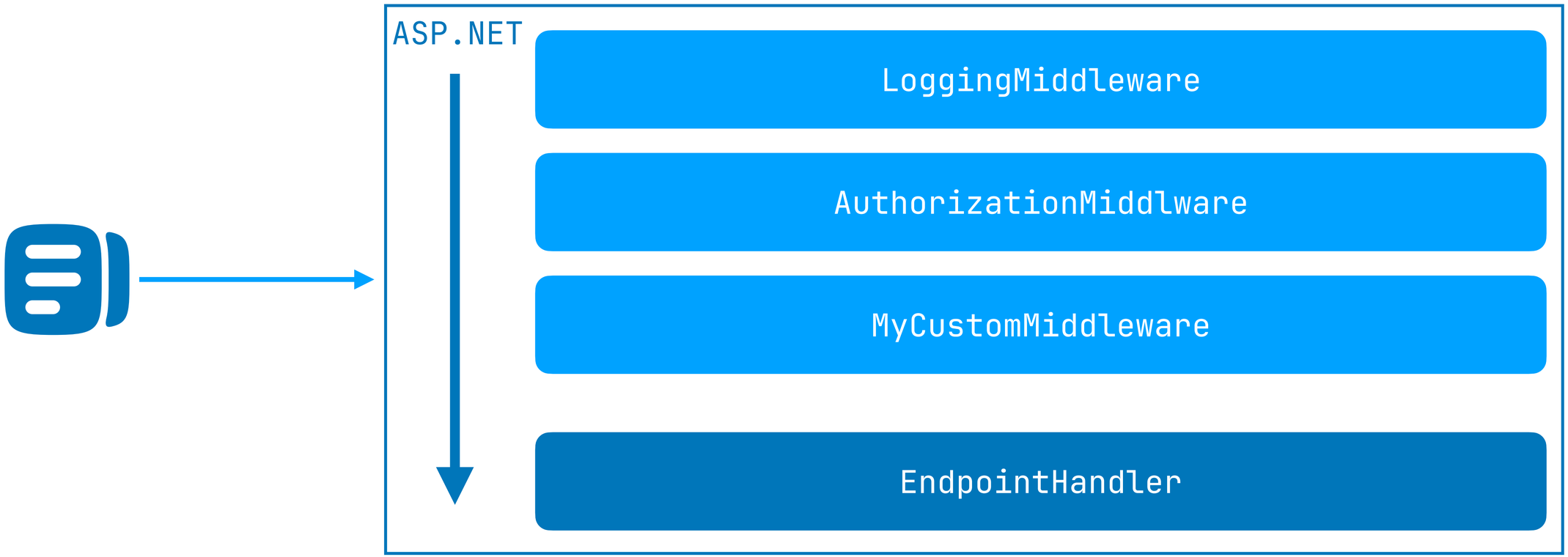
регистрация контроллеров

- контроллеры являются частью AspNet.Mvc
- они не регистрируются в шаблонах по умолчанию

```
// ...

builder.Services.AddControllers();

// ...

var app = builder.Build();

// ...

app.MapControllers();

app.Run();
```

middleware

middleware


ASP.NET
LoggingMiddleware
AuthorizationMiddlware
MyCustomMiddleware
EndpointHandler

middleware

IMiddleware

```
public interface IMiddleware
{
    Task InvokeAsync(HttpContext context, RequestDelegate next);
}
```

middleware

IMiddleware

```
public class ExceptionFormattingMiddleware : IMiddleware
{
    public async Task InvokeAsync(HttpContext context, RequestDelegate next)
    {
        try
        {
            await next(context);
        }
        catch (Exception e)
        {
            var message = $"""
            Exception occured while processing request, type = {e.GetType().Name}, message = {e.Message}";
            """;

            context.Response.StatusCode = StatusCodes.Status500InternalServerError;
            await context.Response.WriteAsJsonAsync(new { message = message });
        }
    }
}
```

middleware

регистрация middleware

```
// ...

builder.Services.AddScoped<ExceptionFormattingMiddleware>();

WebApplication app = builder.Build();

app.UseMiddleware<ExceptionFormattingMiddleware>();

app.MapControllers();

// ...
```

authentication & authorization

authentication & authorisation

- аутентификация – процесс проверки того, что identity пользователя корректна
- авторизация – процесс проверки того, что пользователь имеет доступ каким-либо ресурсам/операциям

authentication & authorisation

в ASP.NET

- аутентификация – реализована через middleware, выполняется до обработки запросов ASP.NET MVC
- авторизация – реализована через filters, выполняется во время обработки запросов в ASP.NET MVC

authorization

AuthorizationAttribute

- при помощи `AuthorizationAttribute` можно задать роли, которые имеют доступ к эндпоинту
- можно реализовывать кастомные правила авторизации через `IAuthorizationFilter`

```
[HttpGet("{userId}/posts")]
[Authorize(Roles = "user,moderator,admin")]
public async Task<ActionResult<Post[]>> SearchUserPostsAsync(
```