

управление данными в микросервисах на C#

ASP.NET

работа с
базами данных в .NET

работа с базами данных в .NET

System.Data

- набор общих абстракций для работы с базами данных
 - подключения
 - запросы
 - параметры
 - транзакции

работа с базами данных в .NET

DbConnection

```
async Task DoSomething(DbConnection connectionParameter)
{
    await using DbConnection connection = connectionParameter;

    await connection.OpenAsync();
    await connection.CloseAsync();
}
```

работа с базами данных в .NET

DbCommand

- определяет запрос к базе данных
- хранит текст запроса
- хранит параметры запроса
- хранит коннекшен над которым будет выполнен запрос
- можно выполнить через `ExecuteReaderAsync` и `ExecuteNonQueryAsync`

работа с базами данных в .NET

DbCommand

```
async Task HandleCommandNonQueryAsync(  
    DbCommand command,  
    CancellationToken cancellationToken)  
{  
    int rows = await command.ExecuteNonQueryAsync(cancellationToken);  
    Console.WriteLine($"Command affected {rows} rows");  
}
```

работа с базами данных в .NET

DbDataReader

- однонаправленный итератор по строкам запроса
- переключается на следующую строку вызовом `ReadAsync`
- имеет методы получения данных текущей строки
 - `GetInt32`
 - `GetString`
 - ...
- реализует `IAsyncDisposable`

работа с базами данных в .NET

DbDataReader

```
async IEnumerable<Model> HandleCommandAsync(  
    DbCommand command,  
    [EnumeratorCancellation] CancellationToken cancellationToken)  
{  
    await using DbDataReader reader = await command.ExecuteReaderAsync(cancellationToken);  
  
    while (await reader.ReadAsync(cancellationToken))  
    {  
        yield return new Model(  
            Id: reader.GetInt32("id"),  
            Name: reader.GetString("name"));  
    }  
}
```


работа с базами данных в .NET

транзакции

```
async Task ExecuteTransactionalOperationAsync(DbConnection connection)
{
    await using DbTransaction transaction = connection.BeginTransaction(IsolationLevel.ReadCommitted);

    // ...

    await transaction.CommitAsync();
}
```

работа с базами данных в .NET

транзакции

```
async Task ExecuteTransactionalOperationAsync(DbConnection connection)
{
    using var transaction = new TransactionScope(
        TransactionScopeOption.Required,
        new TransactionOptions { IsolationLevel = IsolationLevel.ReadCommitted },
        TransactionScopeAsyncFlowOption.Enabled);

    // ...

    transaction.Complete();
}
```

работа с базами данных в .NET

npqsql

- Postgres драйвер для .NET
- реализует абстракции из System.Data для работы с Postgres

работа с базами данных в .NET

NpgsqlDataSource

- фабрика для соединений
- создаётся через NpgsqlDataSourceBuilder
 - конфигурирует маппинги
 - конфигурирует логгирование
 - ...

```
var connectionString = "Host=localhost;Username=postgres;Password=postgres;Database=postgres";  
var dataSourceBuilder = new NpgsqlDataSourceBuilder(connectionString);  
  
await using NpgsqlDataSource dataSource = dataSourceBuilder.Build();
```

работа с базами данных в .NET

NpgsqlConnection и пулинг соединений

- подключение к Postgres базе
- привязано к физическому соединению, но не наоборот
- Npgsql реализует пулинг физических соединений, несколько объектов `NpgsqlConnection`, созданных в разное время, могут использовать одно физическое соединение

работа с базами данных в .NET

NpgsqlCommand

```
const string sql = """
select *
from users
where age > :age
""";

await using NpgsqlCommand command = new NpgsqlCommand(sql, connection)
{
    Parameters =
    {
        new NpgsqlParameter("age", 18),
    },
};
```

работа с базами данных в .NET

NpgsqlCommand

```
const string sql = """
select *
from users
where age > :age
""";
```

```
await using NpgsqlCommand command = connection.CreateCommand();
command.CommandText = sql;
command.Parameters.Add(new NpgsqlParameter("age", 18));
```

работа с базами данных в .NET

МАППИНГ ТИПОВ

```
create type status as enum
(
    'pending',
    'processing',
    'succeeded',
    'failed'
);

create type work_item as
(
    name    text,
    status  status
)
```

```
public enum Status
{
    Pending,
    Processing,
    Succeeded,
    Failed,
}

public record WorkItem(
    string Name,
    Status Status);
```


работа с базами данных в .NET

МАППИНГ ТИПОВ

```
dataSourceBuilder.MapEnum<Status>(pgName: "status");  
dataSourceBuilder.MapComposite<WorkItem>(pgName: "work_item");
```

работа с базами данных в .NET

миграции

работа с базами данных в .NET

миграции

- обеспечивают соответствие схемы базы данных вашему коду
- представляют собой набор SQL скриптов отражающие историческое изменение схемы базы данных
- миграции не должны изменяться, любые изменения схемы данных требуют создания новых миграций

FluentMigrator

определение миграций

```
public class InitialMigration : Migration
{
    public override void Up() { }

    public override void Down() { }
}
```

FluentMigrator

определение миграций

```
public class InitialMigration : Migration
{
    public override void Up()
    {
        Create.Table("users")
            .WithColumn("user_id").AsInt64().PrimaryKey().Identity()
            .WithColumn("user_name").AsString().NotNullable();
    }

    public override void Down()
    {
        Delete.Table("users");
    }
}
```

FluentMigrator

определение миграций

```
public class InitialMigration : IMigration
{
    public void GetUpExpressions(IMigrationContext context)
    {
        context.Expressions.Add(new ExecuteSqlStatementExpression
        {
            SqlStatement = """
create table users
(
    user_id bigint primary key generated always as identity,
    user_name text not null
);
""",
        });
    }

    public void GetDownExpressions(IMigrationContext context)
    {
        context.Expressions.Add(new ExecuteSqlStatementExpression { SqlStatement = """drop table users;""" });
    }

    public string ConnectionString => throw new NotSupportedException();
}
```

FluentMigrator

определение миграций

```
[Migration(version: 1727972936, description: "Initial migration")]  
public class InitialMigration : IMigration
```

FluentMigrator

определение миграций

```
#pragma warning disable SA1649
```

```
[Migration(version: 1727972936, description: "Initial migration")]  
public class InitialMigration : IMigration
```


FluentMigrator

хранение миграций

```
create table "VersionInfo"  
(  
    "Version"      bigint not null,  
    "AppliedOn"    timestamp,  
    "Description"  varchar(1024)  
);
```

FluentMigrator

выполнение миграций

```
serviceCollection
    .AddFluentMigratorCore()
    .ConfigureRunner(runner => runner
        .AddPostgres()
        .WithGlobalConnectionString("Host=localhost;Username=postgres;Password=postgres")
        .WithMigrationsIn(typeof(IMigrationAssemblyMarker).Assembly));
```

FluentMigrator

выполнение миграций

```
await using (AsyncServiceScope scope = serviceProvider.CreateAsyncScope())
{
    IMigrationRunner runner = scope.ServiceProvider.GetRequiredService<IMigrationRunner>();
    runner.MigrateUp();
}
```

FluentMigrator

выполнение миграций

```
await using (AsyncServiceScope scope = serviceProvider.CreateAsyncScope())
{
    IMigrationRunner runner = scope.ServiceProvider.GetRequiredService<IMigrationRunner>();
    runner.MigrateDown(1727972936);
}
```

работа с базами данных в .NET

реализация стандартных операций репозитория

репозитории

- тип, ответственный за реализацию операций с базами данных
- реализует
 - вставку
 - обновление
 - удаление
 - поиск данных

репозитории

множественная вставка/обновление данных

- множественная обработка данных помогает в оптимизации
- помогает снизить затраты на
 - сериализацию
 - обращения к сетевым ресурсам

множественная вставка/обновление данных

кастомные типы данных

```
create type user_insert_model as (name text, age int);
```

```
public readonly record struct UserInsertModel(string Name, int Age);
```


множественная вставка/обновление данных

кастомные типы данных

```
async Task InsertUsersAsync(IEnumerable<User> users, CancellationToken cancellationToken)
{
    UserInsertModel[] models = users.Select(x => new UserInsertModel(x.Name, x.Age)).ToArray();

    const string sql = """
insert into users (user_name, user_age)
select name, age from unnest(:users::user_insert_model[]);
""";







    await using NpgsqlConnection connection = await dataSource.OpenConnectionAsync(cancellationToken);

    await using var command = new NpgsqlCommand(sql, connection)
    {
        Parameters = { new NpgsqlParameter("users", models) }
    };


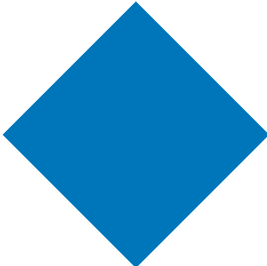



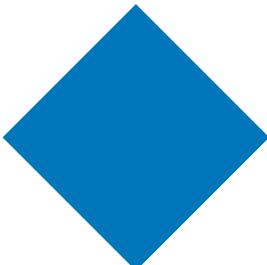
    await command.ExecuteNonQueryAsync(cancellationToken);
}
```

множественная вставка/обновление данных

unnest

unnest([,
[)



col1	col2
	
	
	

множественная вставка/обновление данных

кастомные типы данных

- требуют миграций для создания
- требуют миграций для изменения
- не гибкое решение

множественная вставка/обновление данных

пересборка объектов



множественная вставка/обновление данных

пересборка объектов

```
async Task InsertUsersAsync(IReadOnlyCollection<User> users, CancellationToken cancellationToken)
{
    const string sql = """
insert into users (user_name, user_age)
select name, age from unnest(:names, :ages) as source(name, age);
""";

    await using NpgsqlConnection connection = await dataSource.OpenConnectionAsync(cancellationToken);

    await using var command = new NpgsqlCommand(sql, connection)
    {
        Parameters =
        {
            new NpgsqlParameter("names", users.Select(x => x.Name).ToArray()),
            new NpgsqlParameter("ages", users.Select(x => x.Age).ToArray()),
        },
    };

    await command.ExecuteNonQueryAsync(cancellationToken);
}
```

множественная вставка/обновление данных

обновление данных

```
async Task UpdateUsersAsync(IReadOnlyCollection<User> users, CancellationToken cancellationToken)
{
    const string sql = """
update users
set user_name = source.name,
    user_age = source.age
from (select * from unnest(:ids, :names, :ages)) as source(id, name, age)
where user_id = source.id
""";

    await using NpgsqlConnection connection = await dataSource.OpenConnectionAsync(cancellationToken);

    await using var command = new NpgsqlCommand(sql, connection)
    {
        Parameters =
        {
            new NpgsqlParameter("ids", users.Select(x => x.Id).ToArray()),
            new NpgsqlParameter("names", users.Select(x => x.Name).ToArray()),
            new NpgsqlParameter("ages", users.Select(x => x.Age).ToArray()),
        },
    };

    await command.ExecuteNonQueryAsync(cancellationToken);
}
```

ПОИСК ДАННЫХ

модели запросов

```
public record UserQuery(  
    long[] Ids,  
    string? NamePattern,  
    int? MinAge,  
    int Cursor,  
    int PageSize);
```

```
public interface IUserRepository  
{  
    IEnumerable<User> QueryAsync(UserQuery query, CancellationToken cancellationToken);  
}
```

ПОИСК ДАННЫХ

фильтрации по модели запроса

```
select user_id, user_name, user_age
from users
where
    (user_id > :cursor)
    and (cardinality(:ids) = 0 or user_id = any (:ids))
    and (:name_pattern is null or user_name like :name_pattern)
    and (:min_age is null or user_age > :min_age)
order by user_id
limit :page_size;
```


ПОИСК ДАННЫХ

фильтрации по модели запроса

```
async IEnumerable<User> QueryUsersAsync(
    UserQuery query,
    [EnumeratorCancellation] CancellationToken cancellationToken)
{
    ...

    await using NpgsqlConnection connection = await dataSource.OpenConnectionAsync(cancellationToken);

    await using var command = new NpgsqlCommand(sql, connection)
    {
        Parameters =
        {
            new NpgsqlParameter("ids", query.Ids),
            new NpgsqlParameter("name_pattern", query.NamePattern),
            new NpgsqlParameter("min_age", query.MinAge),
            new NpgsqlParameter("cursor", query.Cursor),
            new NpgsqlParameter("page_size", query.PageSize),
        },
    };

    ...
}
```

ПОИСК ДАННЫХ

фильтрации по модели запроса

```
async IEnumerable<User> QueryUsersAsync(
    UserQuery query,
    [EnumeratorCancellation] CancellationToken cancellationToken)
{
    ...

    await using NpgsqlDataReader reader = await command.ExecuteReaderAsync(cancellationToken);

    while (await reader.ReadAsync(cancellationToken))
    {
        yield return new User(
            reader.GetInt64("user_id"),
            reader.GetString("user_name"),
            reader.GetInt32("user_age"));
    }
}
```

web vs application server

web vs application sever

web server

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

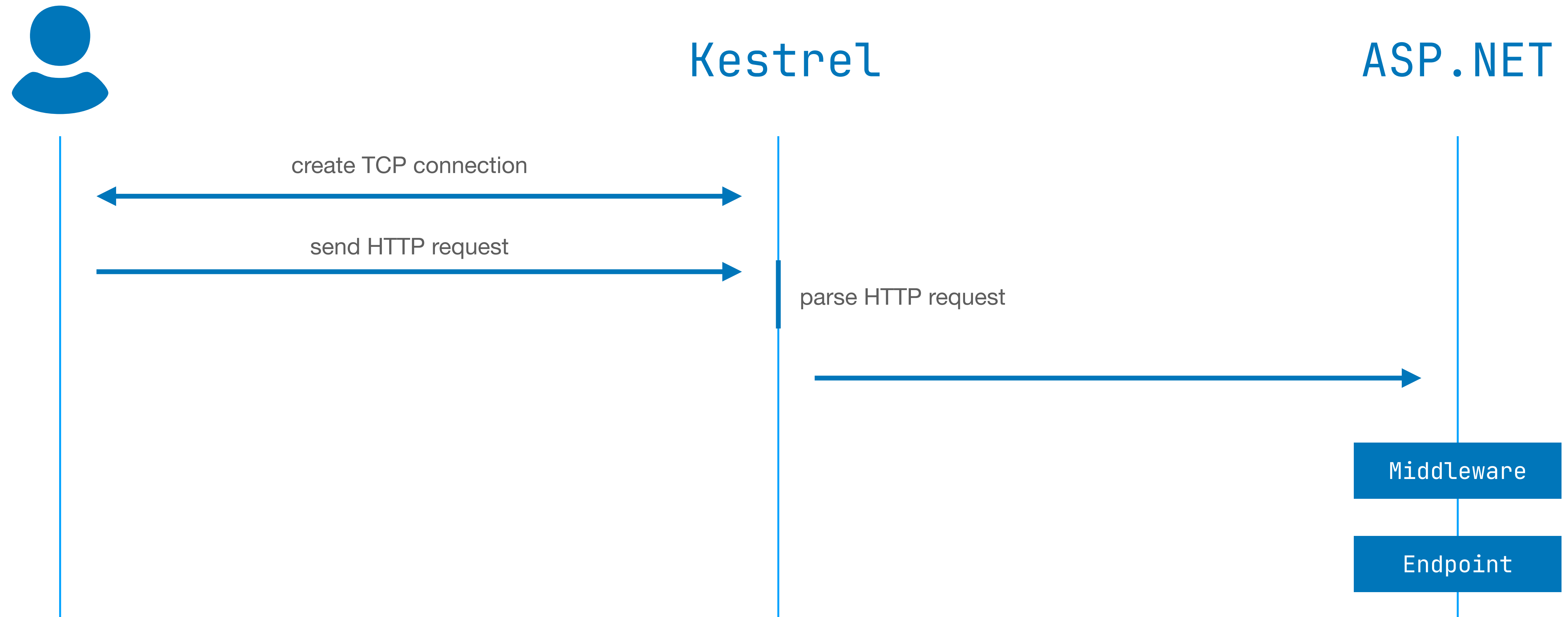
web vs application sever

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` задача вашей операции не авертится, 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 cancellation_token)
    {
        // Some background service logic
        _ = cancellation_token;
    }
}
```

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

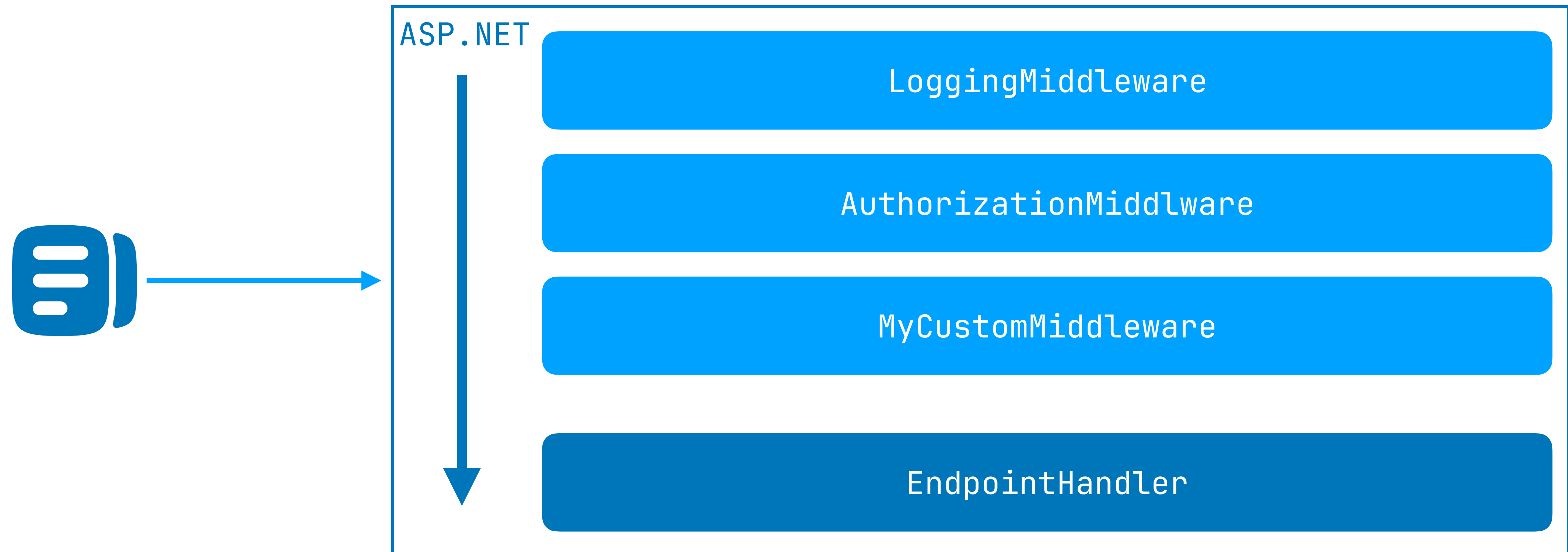
регистрация контроллеров

- 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



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<ExceptionHandlerMiddleware>();  
  
WebApplication app = builder.Build();  
  
app.UseMiddleware<ExceptionHandlerMiddleware>();  
  
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(  

```