



МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет Информационных технологий Кафедра  
Информатики и информационных технологий

направление подготовки 09.03.02 «Информационные  
системы и технологии»

## ЛАБОРАТОРНАЯ РАБОТА № 17

Дисциплина: BackEnd-разработка

---

Тема: *Изучение кеширования в веб-приложении на основе ASP.NET  
Core*

---

Выполнил(а): студент(ка) группы 221-3711

Костоваров А.С.

(Фамилия И.О.)

Дата, подпись

(Дата)

(Подпись)

Проверил:

(Фамилия И.О., степень, звание)

(Оценка)

Дата, подпись

(Дата)

(Подпись)

Замечания:

---

---

Москва

2024

В данной работе мы реализуем механизм кэширования для оценки производительности нашего веб приложения ASP Net Core. Создаем новое приложение и базу данных для тестов получения данных, для нее создаем контекст и подключение.

```
// Настройка подключения к базе данных MySQL
var connectionString = "Server=127.0.0.1;Port=3306;Database=exampleschema;User=root;Password=pugpug12;";
builder.Services.AddDbContext<LABAcontext>(options =>
    options.UseMySQL(connectionString, new MySQLServerVersion(new Version(8, 0, 36))));
```

```
using Laba17.Data;
using Microsoft.EntityFrameworkCore;

namespace Laba17
{
    // Контекст данных для базы данных MySQL
    Ссылка: 6
    public class LABAcontext : DbContext
    {
        Ссылка: 2
        public DbSet<Example> Examples { get; set; }

        Ссылка: 0
        public LABAcontext(DbContextOptions<LABAcontext> options) : base(options) { }
    }
}
```

Далее добавим поддержку кэширования.

```
// Настройка кэширования
builder.Services.AddMemoryCache(); // Внутренний кэш и кэш памяти
builder.Services.AddDistributedMemoryCache(); // Использование распределенного кэша в памяти

builder.Services.AddControllersWithViews(); // Регистрация контроллеров и представлений

app.UseResponseCaching(); // Использование кэширования ответов
```

Теперь создадим класс для обработки кэширования и получения данных из базы данных.

В нем определим 3 метода, для распределенного кэша, кэша памяти и простого получения данных из базы данных.

```

// Получение всех примеров из базы данных или кэша памяти
Ссылка: 1
public Example[] GetExamplesMemoryCache()
{
    // Проверка, есть ли данные в кэше памяти
    if (!_cache.TryGetValue("examples", out Example[] examples))
    {
        // Если данных нет в кэше, извлекаем их из базы данных
        examples = _context.Examples.ToArray();

        // Кэшируем данные в памяти с абсолютным временем жизни 5 минут
        _cache.Set("examples", examples, new MemoryCacheEntryOptions
        {
            AbsoluteExpirationRelativeToNow = TimeSpan.FromMinutes(5)
        });
    }

    // Возвращаем данные (из кэша или базы данных)
    return examples;
}

```

Данный метод реализует кэш памяти. Кэш памяти хранится в оперативной памяти на том же сервере, где запущено приложение.

Доступ к данным происходит очень быстро, так как данные находятся в оперативной памяти.

Недостаток: данные теряются при перезапуске приложения, и кэш не может быть разделен между несколькими серверами.

```

// Получение всех примеров из базы данных или распределенного кэша
Ссылка: 1
public async Task<Example[]> GetExamplesDistributedCacheAsync()
{
    // Ключ для кэширования данных
    var cacheKey = "examples_all";

    // Получение данных из распределенного кэша
    var examplesJson = await _distributedCache.GetStringAsync(cacheKey);
    if (examplesJson != null)
    {
        // Если данные найдены в кэше, десериализуем их
        return JsonSerializer.Deserialize<Example[]>(examplesJson);
    }
    else
    {
        // Если данные не найдены в кэше, извлекаем их из базы данных
        var examples = _context.Examples.ToArray();

        // Кэшируем данные в распределенном кэше с абсолютным временем жизни 5 минут
        await _distributedCache.SetStringAsync(cacheKey, JsonSerializer.Serialize(examples), new DistributedCacheEntryOptions
        {
            AbsoluteExpirationRelativeToNow = TimeSpan.FromMinutes(5)
        });
    }
    return examples;
}

```

Следующий метод реализует использование распределенного кэша.

Распределенный кэш может храниться в различных внешних хранилищах, таких как Redis, SQL Server (в нашем случае кэш хранится в базе данных).

Данные кэша могут быть доступны из нескольких экземпляров приложения, что делает этот тип кэша идеальным для масштабируемых приложений. Доступ к данным может быть медленнее по сравнению с кэшем в памяти, так как данные хранятся вне оперативной памяти.

```
// Получение всех примеров непосредственно из базы данных
Ссылка: 1
public Example[] GetExamplesDatabase()
{
    // Извлечение данных из базы данных без кэширования
    return _context.Examples.ToArray();
}
```

Данный метод получает данные непосредственно из базы данных. Такой метод имеет большой недостаток: получение данных может быть медленным, особенно если бд находится под большой нагрузкой.

В контроллере мы определили методы для получения данных используя все эти методы.

```
// Получение всех примеров из базы данных
[HttpGet("Database")]
Ссылка: 0
public IActionResult GetExamplesDatabase()
{
    // Измерение времени выполнения запроса
    Stopwatch sw = new Stopwatch();

    sw.Start();
    var examples = exservice.GetExamplesDatabase();
    sw.Stop();

    // Передача данных во View
    ViewData["AllTable"] = examples;
    ViewData["AllTime"] = sw.ElapsedMilliseconds;

    return View("Index");
}
```

```
// Получение всех примеров из кэша памяти
[HttpGet("MemoryCache")]
Ссылка: 0
public IActionResult GetExamplesMemoryCache()
{
    // Измерение времени выполнения запроса
    Stopwatch sw = new Stopwatch();

    sw.Start();
    var examples = exservice.GetExamplesMemoryCache();
    sw.Stop();

    // Передача данных во View
    ViewData["AllTablecache"] = examples;
    ViewData["AllTimecache"] = sw.ElapsedMilliseconds;

    return View("Index");
}
```

```
// Получение всех примеров из распределенного кэша
[HttpGet("DistributedCache")]
Ссылка: 0
public async Task<IActionResult> GetExamplesDistributedCache()
{
    // Измерение времени выполнения запроса
    Stopwatch sw = new Stopwatch();

    sw.Start();
    var examples = await exservice.GetExamplesDistributedCacheAsync();
    sw.Stop();

    // Передача данных во View
    ViewData["AllTablecache"] = examples;
    ViewData["AllTimecache"] = sw.ElapsedMilliseconds;

    return View("Index");
}
```

Теперь протестируем приложения и получим время выполнения запросов, чтобы оценить, какой запрос быстрее.

## Home Page

### Database Results:

```
[{"Id":1,"Name":"2","Description":"3343525"}, {"Id":2,"Name":"4535","Description":"535"}]
```

Request Time: 468 ms



Делаем прямой запрос к базе данных и видим результат в 468 миллисекунд.

## Home Page

### Cache Results:

```
[{"Id":1,"Name":"2","Description":"3343525"}, {"Id":2,"Name":"4535","Description":"535"}, {"Id":4,"Name":"432fd","Description":"rhtrey"}]
```

Cache Time: 45 ms

## Home Page

### Cache Results:

```
[{"Id":1,"Name":"2","Description":"3343525"}, {"Id":2,"Name":"4535","Description":"535"}, {"Id":4,"Name":"432fd","Description":"rhtrey"}]
```

Cache Time: 11 ms

Теперь сделаем запрос к кэшу памяти, видим, что результат гораздо быстрее.

А теперь запрос к распределенному кэшу, запрос выполняется быстро, но все же медленнее чем запрос к данным в оперативной памяти.

Таким образом заметна разница при использовании того или иного типа кэша, что полезно знать, при оптимизации приложения.

## Код приложения

```
using Laba17;
using Microsoft.EntityFrameworkCore; using
Microsoft.Extensions.Caching.Distributed; using
Microsoft.Extensions.DependencyInjection;

var builder = WebApplication.CreateBuilder(args);

// Настройка подключения к базе данных MySQL var connectionString
=
"Server=127.0.0.1;Port=3306;Database=exampleschema;User=root;Password=pugpug12;";
builder.Services.AddDbContext<LABAcontext>(options =>
    options.UseMySQL(connectionString, new MySqlServerVersion(new Version(8, 0, 36))));

// Настройка кэширования
builder.Services.AddMemoryCache(); // Внутренний кэш и кэш памяти
builder.Services.AddDistributedMemoryCache(); // Использование распределенного кэша в
памяти
builder.Services.AddControllersWithViews(); // Регистрация контроллеров и
представлений
builder.Services.AddCors(options =>
{
    options.AddDefaultPolicy(builder =>
    {
```

```

        builder.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader();
    }); }); var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseDeveloperExceptionPage();
} app.UseRouting();
app.UseCors();
app.UseAuthentication(); app.UseAuthorization();
app.UseResponseCaching(); // Использование кэширования ответов

// Настройка маршрутизации app.UseRouting();

// Определение маршрута по умолчанию для контроллеров app.MapControllerRoute(
name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

app.UseStaticFiles();

app.Run();

using Laba17.Data; using
Microsoft.EntityFrameworkCore;

namespace Laba17
{
    // Контекст данных для базы данных MySQL
    public class LABAcontext : DbContext
    {
        public DbSet<Example> Examples { get; set; }

        public LABAcontext(DbContextOptions<LABAcontext> options) : base(options) { }
    }
}

using Laba17.Data;
using Microsoft.Extensions.Caching.Distributed;
using Microsoft.Extensions.Caching.Memory; using System.Text.Json;

namespace Laba17
{
    public class ExampleService
    {
        // кэш памяти
        public IMemoryCache _cache;

        private LABAcontext _context;

        // распределенного кэша private readonly
        IDistributedCache _distributedCache;

        // Конструктор сервиса, принимающий зависимости через DI
        public ExampleService(IMemoryCache memoryCache, LABAcontext context,
            IDistributedCache distributedCache)
        {

```

```

        _cache = memoryCache; // Инициализация кэша памяти
        _context = context; // Инициализация контекста базы данных
        _distributedCache = distributedCache; // Инициализация распределенного кэша
    }

    // Получение всех примеров из базы данных или кэша памяти
    public Example[] GetExamplesMemoryCache()
    {
        // Проверка, есть ли данные в кэше памяти
        if (!_cache.TryGetValue("examples", out Example[] examples))
        {
            // Если данных нет в кэше, извлекаем их из базы данных
            examples = _context.Examples.ToArray();

            // Кэшируем данные в памяти с абсолютным временем жизни 5 минут
            _cache.Set("examples", examples, new MemoryCacheEntryOptions
            {
                AbsoluteExpirationRelativeToNow = TimeSpan.FromMinutes(5)
            });
        }
        // Возвращаем данные (из кэша или базы данных)
        return examples;
    }

    // Получение всех примеров из базы данных или распределенного кэша
    public async Task<Example[]> GetExamplesDistributedCacheAsync()
    {
        // Ключ для кэширования данных
        var cacheKey = "examples_all";

        // Получение данных из распределенного кэша
        var examplesJson = await _distributedCache.GetStringAsync(cacheKey);
        if (examplesJson != null)
        {
            // Если данные найдены в кэше, десериализуем их
            return JsonSerializer.Deserialize<Example[]>(examplesJson);
        }
        else
        {
            // Если данные не найдены в кэше, извлекаем их из базы данных
            var examples = _context.Examples.ToArray();

            // Кэшируем данные в распределенном кэше с абсолютным временем жизни 5 минут
            await _distributedCache.SetStringAsync(cacheKey,
            JsonSerializer.Serialize(examples), new DistributedCacheEntryOptions
            {
                AbsoluteExpirationRelativeToNow = TimeSpan.FromMinutes(5)
            });
            return examples;
        }
    }

    // Получение всех примеров непосредственно из базы данных
    public Example[] GetExamplesDatabase()
    {
        // Извлечение данных из базы данных без кэширования
        return _context.Examples.ToArray();
    }

```



```

    }
}

namespace Laba17.Data
{
    // Модель данных для примера      public
    class Example
    {
        public int Id { get; set; }
        public string Name { get; set; }      public string
        Description { get; set; }
    }
}

using Microsoft.AspNetCore.Mvc; using
Microsoft.Extensions.Caching.Memory; using
Microsoft.Extensions.Caching.Distributed; using
System.Diagnostics;

namespace Laba17.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class HomeController : Controller
    {
        // Сервис для работы с кэшем
        ExampleService exservice;

        // Конструктор контроллера
        public HomeController(IMemoryCache cache, LABAcontext _context,
        IDistributedCache distributedCache)
        {
            // Инициализация сервиса с кэшем памяти и распределенным кэшем
            exservice = new ExampleService(cache, _context, distributedCache);
        }

        // Получение всех примеров из базы данных
        [HttpGet("Database")]
        public IActionResult GetExamplesDatabase()
        {
            // Измерение времени выполнения запроса
            Stopwatch sw = new Stopwatch();

            sw.Start();      var examples =
            exservice.GetExamplesDatabase();
            sw.Stop();

            // Передача данных во View
            ViewData["AllTable"] = examples;
            ViewData["AllTime"] = sw.ElapsedMilliseconds;

            return View("Index");
        }

        // Получение всех примеров из кэша памяти
        [HttpGet("MemoryCache")]
        public IActionResult GetExamplesMemoryCache()
    }
}

```

```

    {
        // Измерение времени выполнения запроса
        Stopwatch sw = new Stopwatch();

        sw.Start();
        var examples =
exservice.GetExamplesMemoryCache();
sw.Stop();

        // Передача данных во View
        ViewData["AllTablecache"] = examples;
        ViewData["AllTimecache"] = sw.ElapsedMilliseconds;

        return View("Index");
    }

    // Получение всех примеров из распределенного кэша
    [HttpGet("DistributedCache")]
    public async Task<IActionResult> GetExamplesDistributedCache()
    {
        // Измерение времени выполнения запроса
        Stopwatch sw = new Stopwatch();

        sw.Start();
        var examples = await exservice.GetExamplesDistributedCacheAsync();
sw.Stop();

        // Передача данных во View
        ViewData["AllTablecache"] = examples;
        ViewData["AllTimecache"] = sw.ElapsedMilliseconds;

        return View("Index");
    }
}

@using System.Text.Json
@{
    ViewData["Title"] = "Данные";
}

<h2>Home Page</h2>

@if (ViewData["AllTable"] != null)
{
    <h3>Database Results:</h3>
    <pre>@Html.Raw(JsonSerializer.Serialize(ViewData["AllTable"]))</pre>
    <p>Request Time: @ViewData["AllTime"] ms</p> }

@if (ViewData["AllTablecache"] != null)
{
    <h3>Cache Results:</h3>
    <pre>@Html.Raw(JsonSerializer.Serialize(ViewData["AllTablecache"]))</pre>
    <p>Cache Time: @ViewData["AllTimecache"] ms</p> }

```