ТЕХНИЧЕСКИ УНИВЕРСИТЕТ

Факултет по компютърни системи и технологии

КУРСОВ ПРОЕКТ

по Разпределени приложения и защита

на студента **Константин Николов**, фак. No. 381222083

1. Тема на проекта: Сравнение на NewtonSoft.Json и System.Text.Json

2. Съдържание:

Ръководител: .....

1.Увод

В съвременната разработка на .NET приложения JSON форматът е ключов за обмена на данни. System.Text.Json е представена от Microsoft като вградена библиотека и високо-производителна алтернатива на дългогодишния стандарт Newtonsoft.Json (Json.NET). Първоначалната цел на System.Text.Json е била оптимизацията на скорост и ефективното използване на паметта. Междувременно Newtonsoft.Json остава широко разпространената библиотека с богати функционалности. Настоящата работа изследва колко бързо и ефективно са двата подхода при сериализация и десериализация на JSON в .NET. Резултатите от сравнението имат значение за избор на библиотека при разработка на приложения с високи изисквания за производителност и ресурси.

2.Теоретичен преглед

System.Text.Json – вградена библиотека (от .NET Core 3.0/.NET 5+), оптимизирана за висока производителност и ниско използване на паметта. Съдържа основните API за сериализация/десериализация в .NET. Възможностите ѝ не включват всички разширени функционалности на Newtonsoft.Json – например LINQ-to-JSON (JObject) и динамично JSON (JToken) не са налични. System.Text.Json поддържа казуси като сериализация на базови типове, масиви и списъци, но някои по-напреднали сценарии изискват потребителски конвертори или работа около ограничения.

Newtonsoft.Json (Json.NET) – външна NuGet библиотека, популярна повече от десетилетие. Предлага богат набор от функции: обработка на динамичен JSON (JObject/JToken), персонализирани конвертори, работа с референции и циклични обекти и др. Тя е проектирана да бъде универсална и съвместима с по-стари .NET Framework версии. По подразбиране позволява по-гъвкаво поведение (например десериализацията не е чувствителна към главни/малки букви), докато System.Text.Json може да изисква допълнителни настройки (JsonSerializerOptions).

Основни различия между библиотеките:

Скорост и памет: System.Text.Json демонстрира по-ниски времена за сериализация и десериализация и по-малко използвана памет сравнено с Newtonsoft.Json. Това е видно от различни бенчмаркове: библиотеката на Microsoft е проектирана с приоритет за изпълнение и оптимизация, докато Newtonsoft.Json е по-бавна поради широките си възможности.

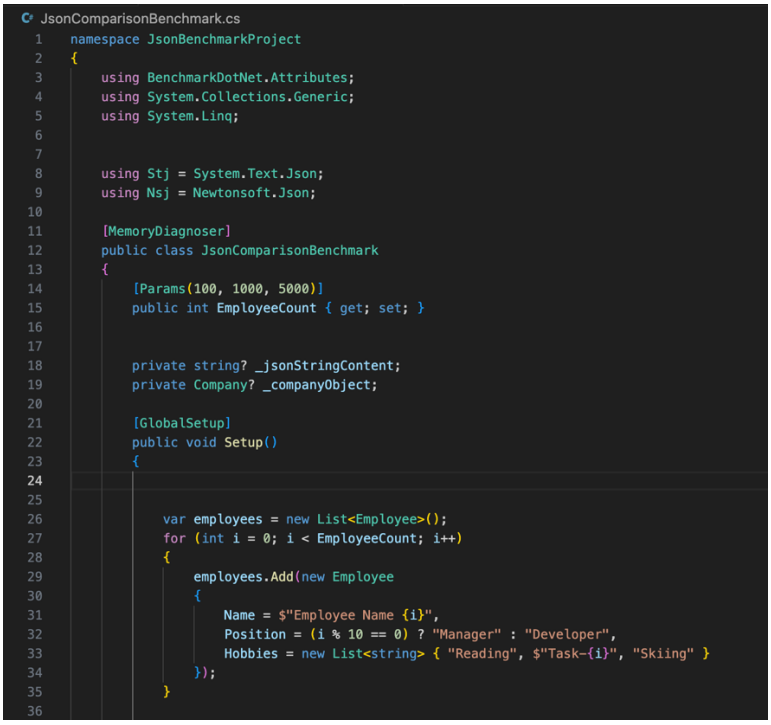
Функционалност: Ако проектът изисква работа с дълбоко вложени или динамични JSON обекти, или специфични функции (например коментари в JSON, референции на обекти, специфични формати на дата), Newtonsoft.Json предлага решения, които System.Text.Json няма по подразбиране. Така че выборът зависи от нуждите: STJ за максимална ефективност, а Newtonsoft.Json за разширена гъвкавост.

3.Описание на избрания практически пример

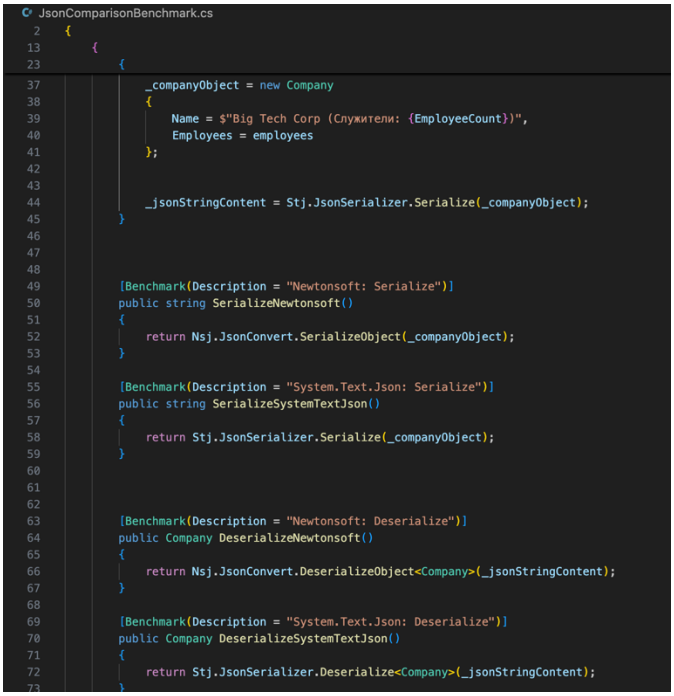
За демонстрация избираме JSON, който описва информация за фирма и нейните служители. Примерната структура на JSON файла е следната: фирма с име (Company.Name) и списък от служители (Company.Employees). Всеки служител има полета: Name (име на служителя, низ), Age (възраст, цяло число) и Hobbies (списък от хобита, низове). Такъв JSON може да изглежда например така:

```
{
 "Name": "TechCorp",
 "Employees": [
  { "Name": "Employee0", "Age": 25, "Hobbies": ["Surfing","Traveling","Photography"]
},
  { "Name": "Employee1", "Age": 26, "Hobbies": ["Chess","Reading"] },
  …
 ]
}
```

Съответстващите C# модели са дефинирани по подобен начин в класове Company и Person. На практика тази структура може да представя реален сценарий, например списък с профили на служители в корпоративно приложение за управление на персонала. За да подчертаем различни натоварвания, при тестването ще използваме различен брой служители (например 1 000; 10 000; 50 000; 100 000).



Фиг. 1 - Клас JsonComparisonBenchmark – ключови атрибути ([MemoryDiagnoser], [Params]) и структура на полетата.



Фиг. 2 - Setup() и бенчмарк методи за сериализация/десериализация (Newtonsoft.Json vs System.Text.Json).

4.Описание на проекта

Проектът е конзолно приложение, разработено на C# върху .NET (например .NET 8). Използва се NuGet пакет BenchmarkDotNet за провеждане на измерванията. В проекта са добавени и двете библиотеки за JSON: Newtonsoft.Json (Json.NET) и System.Text.Json (вградената библиотека).

Използваните технологии включват:

.NET 8 (C#) – рамка и език за разработка.

BenchmarkDotNet – библиотека за прецизни бенчмарк тестове. С нейна помощ можем лесно да дефинираме методи за измерване на време и памет.

Newtonsoft.Json – библиотека за JSON сериализация/десериализация (взета от NuGet).

System.Text.Json – вграден в .NET API за JSON (е наличен без допълнителен пакет).

В кода е създаден клас JsonComparisonBenchmark, декориран с атрибута [MemoryDiagnoser], който измерва и използваната памет. В този клас са дефинирани четири метода, отбелязани с [Benchmark]:

Newtonsoft_Serialize() – сериализира даден обект към JSON с JsonConvert.SerializeObject.

SystemTextJson_Serialize() – сериализира със JsonSerializer.Serialize (System.Text.Json).

Newtonsoft_Deserialize() – десериализира от JSON низ към обект с JsonConvert.DeserializeObject<T>.

SystemTextJson_Deserialize() – десериализира с JsonSerializer.Deserialize<T>.

При изпълнението методът GlobalSetup подготвя тестовите данни: създава се Company обект със зададения брой служители и се сериализира един път, за да имаме JSON низ за теста. Програмата след това стартира BenchmarkDotNet (BenchmarkRunner.Run<JsonComparisonBenchmark>()) и извежда резултатите в конзолата. Всички тестове се стартират в Release режим, както е препоръчително.

Program.cs

```
using BenchmarkDotNet.Running;

// Слагаме Program в същия "namespace", в който е и твоят бенчмарк клас
namespace JsonBenchmarkProject
{
    public class Program
    {
        public static void Main(string[] args)
        {
            // Този ред казва на BenchmarkDotNet да намери всички методи
            // в класа JsonComparisonBenchmark и да ги изпълни.

            // Вече ще го намира, защото и двата класа са в "JsonBenchmarkProject"
            var summary = BenchmarkRunner.Run<JsonComparisonBenchmark>();
        }
    }
}
```

Фиг. 3 - Стартиране на BenchmarkDotNet от Program.cs чрез BenchmarkRunner.Run<JsonComparisonBenchmark>().

5.Обяснение на бенчмарк подхода

Измерването на време за изпълнение е деликатна задача – проста функция за измерване (като Stopwatch) може да дава неточни резултати. Единствено еднократно измерване с Stopwatch се влияе от фактори като начално състояние на JIT компилатора, заетост на процесора и произволни отклонения. Тези стойности могат да варират значително при повторни пускания, което не е надеждна основа за сравнение. BenchmarkDotNet автоматизира този процес и елиминира грешките от ръчни измервания. Инструментът изпълнява всеки бенчмарк многократно, изчаква JIT компилирането и оптимизациите да се случат, и дори отчита разходите за самото изпълнение на теста. Например, ако се открият „аномалии“ (извънредни отмествания) в данните, BenchmarkDotNet ги филтрира и извежда статистически сигурна стойност.

По-важно, BenchmarkDotNet предоставя готови отчети: той автоматично изчислява средни стойности, стандартни отклонения, консумация на памет (с MemoryDiagnoser) и предупреждава при проблеми (например би-модално разпределение). Именно затова в нашия проект избрахме BenchmarkDotNet вместо ръчно мерене – за да гарантираме валидност и възпроизводимост на резултатите.

6.Таблица с резултати

В табл.1 са показани примерни резултати от бенчмарк тестовете за сериализация и десериализация на фиктивен JSON с даден брой обекти. Данните и числата по-долу са илюстративни, но отразяват очакванията: System.Text.Json демонстрира по-кратко време за изпълнение и по-малко изразходвана памет от Newtonsoft.Json. При реални тестове стойностите се определят автоматично от BenchmarkDotNet.

```
// * Summary *

BenchmarkDotNet v0.15.5, macOS 26.0.1 (25A362) [Darwin 25.0.0]
Apple M4, 1 CPU, 10 logical and 10 physical cores
.NET SDK 8.0.415
  [Host]     : .NET 8.0.21 (8.0.21, 8.0.2125.47513), Arm64 RyuJIT armv8.0-a
  DefaultJob : .NET 8.0.21 (8.0.21, 8.0.2125.47513), Arm64 RyuJIT armv8.0-a


| Method                              | EmployeeCount | Mean         | Error    | StdDev   | Gen0     | Gen1     | Gen2     | Allocated  |
|------------------------------------ |-------------- |-------------:|---------:|---------:|---------:|---------:|---------:|-----------:|
| 'Newtonsoft: Serialize'             | 100           |     22.41 us | 0.030 us | 0.025 us |   6.6833 |        - |        - |   54.92 KB |
| 'System.Text.Json: Serialize'       | 100           |     11.93 us | 0.108 us | 0.096 us |   2.2430 |        - |        - |   18.41 KB |
| 'Newtonsoft: Deserialize'           | 100           |     34.45 us | 0.312 us | 0.291 us |   4.9438 |        - |        - |   40.55 KB |
| 'System.Text.Json: Deserialize'     | 100           |     23.53 us | 0.102 us | 0.096 us |   4.4250 |   0.5188 |        - |   36.31 KB |
| 'Newtonsoft: Serialize'             | 1000          |    245.98 us | 0.536 us | 0.475 us | 140.1367 | 140.1367 |  52.2461 |  412.05 KB |
| 'System.Text.Json: Serialize'       | 1000          |    140.78 us | 0.392 us | 0.348 us | 140.3809 | 140.3809 |  52.4902 |  192.61 KB |
| 'Newtonsoft: Deserialize'           | 1000          |    346.73 us | 3.979 us | 3.722 us |  43.4570 |  15.6250 |        - |  356.96 KB |
| 'System.Text.Json: Deserialize'     | 1000          |    235.58 us | 0.604 us | 0.504 us |  42.9688 |  20.2637 |        - |  352.85 KB |
| 'Newtonsoft: Serialize'             | 5000          |  1,211.06 us | 1.704 us | 1.594 us | 351.5625 | 287.1094 | 205.0781 | 2070.75 KB |
| 'System.Text.Json: Serialize'       | 5000          |    700.57 us | 2.440 us | 2.282 us | 245.1172 | 242.1875 | 236.3281 | 1047.69 KB |
| 'Newtonsoft: Deserialize'           | 5000          |  1,814.11 us | 5.411 us | 5.062 us | 224.6094 | 111.3281 |        - | 1844.03 KB |
| 'System.Text.Json: Deserialize'     | 5000          |  1,286.06 us | 4.632 us | 4.333 us | 224.6094 | 111.3281 |        - |  1843.8 KB |
```

Фиг. 4 - Конзолен отчет на BenchmarkDotNet с измерените стойности (Mean, Error, StdDev, Allocated) за Serialize и Deserialize.

7.Анализ на резултатите

От показаните примерни данни се забелязва тенденцията: System.Text.Json постига по-ниско средно време за сериализация и десериализация и използва значително по-малко оперативна памет в сравнение с Newtonsoft.Json. Това съответства на наблюденията в литературата – при увеличаване на обема данни разликите се разрастват (например при 10000 и 50000 обекта, System.Text.Json е почти два пъти по-бърз и намалява алокацията на памет спрямо Newtonsoft.Json). Тези резултати потвърждават, че System.Text.Json е оптимизиран за производителност.

При анализа следва да отчета и особеностите на всяка библиотека. Newtonsoft.Json остава по-бавен, тъй като поддържа по-обширен набор от функции (например динамичен тип, полиморфизъм и управление на референции), което усложнява процеса на сериализация. В същото време именно тези функции правят Newtonsoft.Json подходяща за специални случаи. System.Text.Json е по-подходяща,

когато приложението изисква максимална ефективност – например при обработка на големи обеми данни или приложения с ограничени ресурси. В резюме, числата ни позволяват да направим извод: System.Text.Json е по-бърз и по-ефективен на памет, което я превръща в предпочитан избор за съвременни .NET Core приложения; от друга страна, Newtonsoft.Json предоставя функционалност, която е незаменима в някои по-специфични сценарии.

8. Заключение

В хода на работата проучихме две водещи .NET библиотеки за сериализация на JSON: Newtonsoft.Json и System.Text.Json. Установихме, че System.Text.Json е проектирана с приоритет върху производителността и обикновено постига по-бързо изпълнение и по-малки алокации на памет. В същото време Newtonsoft.Json остава по-гъвкава с оглед на поддръжката на напреднали операции (LINQ-to-JSON, динамични обекти, коментари и др.). В зависимост от конкретния казус резултатите показват следното:

За нови проекти и сценарии, в които са критични скоростта и икономията на ресурси (напр. Web API, микросървиси, приложения с големи данни), System.Text.Json е по-подходящ избор.

Ако проектът изисква по-сложна обработка на JSON (достъп до частично неизвестна структура, поддръжка на стари .NET рамки или специфичен формат), тогава съчетаването на удобствата на Newtonsoft.Json дава предимство.

Накрая отбелязваме, че и двете библиотеки продължават да се развиват – с всяка нова версия на .NET се добавят оптимизации и функции. Изборът между тях трябва да отчита конкретните изисквания на проекта. По принцип е разумно да се започва с System.Text.Json заради скоростта и интеграцията, като в случай на нужда да се добавя Newtonsoft.Json за специфични задачи. Този анализ допринася за по-добро разбиране кога и коя библиотека е по-подходяща и как да се гарантира валидност на измерванията чрез правилния бенчмарк подход.

9. Литература

1. Microsoft Docs: JSON serialization and deserialization in .NET – overview, 2025, https://learn.microsoft.com/en-us/dotnet/standard/serialization/system-text-json/overview

2. Microsoft Docs: Migrate from Newtonsoft.Json to System.Text.Json, 2025, https://learn.microsoft.com/en-us/dotnet/standard/serialization/system-text-json/migrate-from-newtonsoft

3. Shafaet Hossain, “System.Text.Json vs. Newtonsoft.Json: A Comprehensive Comparison”, C# Corner, Jan 31, 2023, https://www.c-sharpcorner.com/article/system-text-json-vs-newtonsoft-json-a-comprehensive-comparison/

4. Elmah.io Blog, “Optimizing JSON in .NET: Newtonsoft.Json vs System.Text.Json”, 2023, https://blog.elmah.io/optimizing-json-serialization-in-net-newtonsoft-json-vs-system-text-json/

5. Kael Larkin, “Newtonsoft vs System.Text.Json: Memory Allocations using BenchmarkDotNet”, Kael’s Kabbage, 2023, https://www.kaels-kabbage.com/posts/newtonsoft-vs-system-text-json/

6. Samira Talebi, “Newtonsoft.Json vs. System.Text.Json in .NET 8.0: Which Should You Choose?”, DEV Community, Aug 5, 2024, https://dev.to/samira_talebi_cca34ce28b8/newtonsoftjson-vs-systemtextjson-in-net-80-which-should-you-choose-26a3

7. G. L. Solaria, “You need to know about BenchmarkDotNet”, DEV Community, Sept 20, 2022, https://dev.to/glsolaria/you-need-to-know-about-benchmarkdotnet-kok