Armen Melkumyan • 1st
Technical / Solutions Architect
3mo •

From DotNet Technical interviews: Key Differences Between Span<T> and Memory<T> in .NET 🚀

Here's a straightforward breakdown of their differences to help you decide which to use.

Span<T>:

- Purpose: Represents a contiguous memory region, optimized for speed.

- Allocation: Lives on the stack, keeping operations lightweight.

Key Feature: As a ref struct, Span<T>:

Cannot be boxed.

Cannot escape to the heap.

Cannot be used asynchronously or stored in fields.

Use Case: Perfect for short-lived, synchronous operations that demand speed.

Memory<T>: The Asynchronous-Friendly Powerhouse

- Purpose: Designed for scenarios where memory needs to persist or cross boundaries.

- Allocation: Lives on the heap, providing flexibility.

Key Feature: Unlike Span<T>, Memory<T>:

Can be stored in fields.

Supports async/await operations.

Use Case: Ideal for long-lived, asynchronous operations or when passing memory between methods.

When to Choose Which?

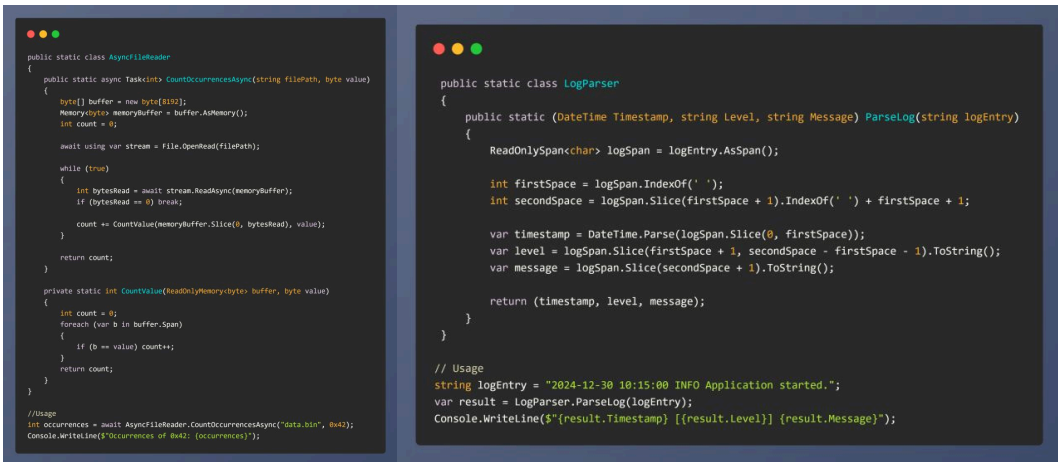
Use Span<T> for performance-critical, synchronous tasks that require speed and efficiency.

Use Memory<T> for asynchronous operations or when you need more flexibility.

Want to dive deeper into these concepts? Check out my book, High Performance .NET for practical examples, detailed explanations, and expert insights into building fast, efficient, and reliable .NET

applications.

#Csharp #DotNet #Interview #HighPerformanceDotNet



```
public static class AsyncFileReader
{
    public static async Task<int> CountOccurrencesAsync(string filePath, byte value)
    {
        byte[] buffer = new byte[8192];
        Memory<byte> memoryBuffer = buffer.AsMemory();
        int count = 0;

        await using var stream = File.OpenRead(filePath);

        while (true)
        {
            int bytesRead = await stream.ReadAsync(memoryBuffer);
            if (bytesRead == 0) break;

            count += CountValue(memoryBuffer.Slice(0, bytesRead), value);
        }

        return count;
    }

    private static int CountValue(ReadOnlyMemory<byte> buffer, byte value)
    {
        int count = 0;
        foreach (var b in buffer.Span)
        {
            if (b == value) count++;
        }
        return count;
    }
}

//Usage
int occurrences = await AsyncFileReader.CountOccurrencesAsync("data.bin", 0x42);
Console.WriteLine($"Occurrences of 0x42: {occurrences}");
```

```
public static class LogParser
{
    public static (DateTime Timestamp, string Level, string Message) ParseLog(string logEntry)
    {
        ReadOnlySpan<char> logSpan = logEntry.AsSpan();

        int firstSpace = logSpan.IndexOf(' ');
        int secondSpace = logSpan.Slice(firstSpace + 1).IndexOf(' ') + firstSpace + 1;

        var timestamp = DateTime.Parse(logSpan.Slice(0, firstSpace));
        var level = logSpan.Slice(firstSpace + 1, secondSpace - firstSpace - 1).ToString();
        var message = logSpan.Slice(secondSpace + 1).ToString();

        return (timestamp, level, message);
    }
}

// Usage
string logEntry = "2024-12-30 10:15:00 INFO Application started.";
var result = LogParser.ParseLog(logEntry);
Console.WriteLine($"{result.Timestamp} [{result.Level}] {result.Message}");
```

125 4 comments 6 reposts