Navigating Dependency Lifetimes: A Practical Comparison of AddTransient, AddScoped, and AddSingleton in .NET

Shyamprasad Narapareddy · Follow
2 min read · Dec 11, 2023

21

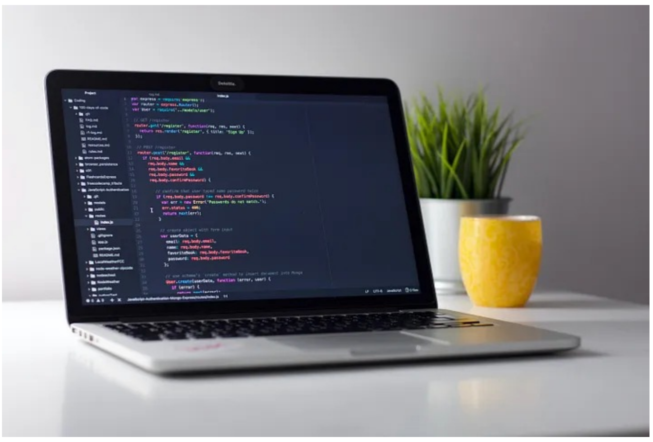
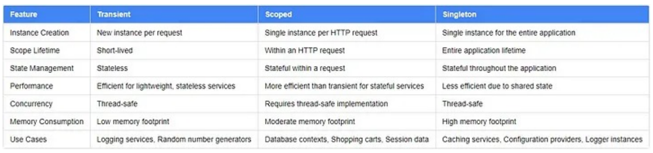


Photo by Clément Hélardot on Unsplash

Dependency Injection (DI) is a pivotal element in constructing scalable and maintainable .NET applications, especially in the realm of Web APIs.

Choosing the appropriate scope depends on the nature of your service and its relationship with other dependencies.

Detailed Comparison of Scopes



Feature	Transient	Scoped	Singleton
Instance Creation	New instance per request	Single instance per HTTP request	Single instance for the entire application
Scope Lifetime	Short-lived	Within an HTTP request	Entire application lifetime
State Management	Stateless	Stateful within a request	Stateful throughout the application
Performance	Efficient for lightweight, stateless services	More efficient than transient for stateful services	Less efficient due to shared state
Concurrency	Thread-safe	Requires thread-safe implementation	Thread-safe
Memory Consumption	Low memory footprint	Moderate memory footprint	High memory footprint
Use Cases	Logging services, Random number generators	Database contexts, Shopping carts, Session data	Caching services, Configuration providers, Logger instances

Comparision of AddTransient vs AddScoped vs AddSingleton

C# Example: Demonstrating Different Scopes

Here’s an example illustrating the behavior of each scope:

```
public interface IOperationService
{
        Guid GetOperationId();
    }

    public class TransientService : IOperationService
    {
        public Guid GetOperationId()
        {
            return Guid.NewGuid();
        }
    }

    public class ScopedService : IOperationService
    {
        private readonly Guid _operationId;

        public ScopedService()
        {
            _operationId = Guid.NewGuid();
        }

        public Guid GetOperationId()
        {
            return _operationId;
        }
    }

    public class SingletonService : IOperationService
    {
        private readonly Guid _operationId;

        public SingletonService()
        {
            _operationId = Guid.NewGuid();
        }

        public Guid GetOperationId()
        {
            return _operationId;
        }
    }
```

```
    public class Startup
    {
        public void ConfigureServices(IServiceCollection services)
        {
            // Transient service
            services.AddTransient<IOperationService, TransientService>();

            // Scoped service
            services.AddScoped<IOperationService, ScopedService>();

            // Singleton service
            services.AddSingleton<IOperationService, SingletonService>();
        }
    }
```

Expected Behavior:
TransientService: Each call to *GetOperationId* will return a new GUID.
ScopedService: The same GUID will be returned within a single HTTP request, but different requests will receive different GUIDs.
SingletonService: The same GUID will be returned throughout the entire application lifetime.

Optimal selection of the dependency injection scope is essential in crafting efficient and resilient .NET Web APIs. Grasp the distinctions among `AddTransient`, `AddScoped`, and `AddSingleton`, and make a thoughtful choice of scope tailored to your service's requirements and behavior. Employing the right scope enables you to attain peak performance, effective state management, and efficient memory utilization.

Dependency Injection C Sharp Programming Developer Programming
Web Development

21

Written by Shyamprasad Narapareddy
Follow
6 Followers · 21 Following

Passionate C# dev with a keen interest in tech, continuously exploring new advancements. Devoted to both programming excellence and prioritizing family values.

No responses yet

What are your thoughts?

Respond

More from Shyamprasad Narapareddy

In Learningbowl by Shyamprasad Narapareddy

Produce HTML code from SQL query

Introduction

Oct 16, 2018 44

Shyamprasad Narapareddy

Singleton Pattern: Exploring the Best Use Cases for C# Developers

1. Database Connection: Use Case: When you have a need to manage a single database...

Nov 24, 2023 2

Shyamprasad Narapareddy

From Sheets to Mobile Apps: Mobile App Development with...

Recently, while searching the internet for free tools to build a user interface for interacting...

Dec 15, 2023 3

Shyamprasad Narapareddy

RESTful API Design: Navigating Responses with Common HTTP...

REST APIs use HTTP status codes to indicate the result of a client's request. Here are som...

Nov 28, 2023 2

See all from Shyamprasad Narapareddy

Recommended from Medium

Harendra

How I Am Using a Lifetime 100% Free Server

Get a server with 24 GB RAM + 4 CPU + 200 GB Storage + Always Free

Oct 26 6.9K 102

Ravi Patel

Understanding the Factory Method Design Pattern in C#

In the world of software development, design patterns are essential for writing clean,...

Jul 2 59

Lists

General Coding Knowledge
20 stories · 1803 saves

Coding & Development
11 stories · 933 saves

Stories to Help You Grow as a Software Developer
19 stories · 1512 saves

ChatGPT
21 stories · 904 saves

Kamlesh Singh

Run, Use, and Map Methods in .NET Core Pipeline

In .NET Core, “Run”, “Map”, and “Use ”are all methods that help you configure the...

Oct 5 · 15

Cesar I. Egoavil

Parallel.ForEach vs. Parallel.ForEachAsync in C#

I needed to improve an application to process and transform multiple records received fro...

Jun 13 · 49 · 1

Raza Sherazi

Custom File Extension Validation in ASP.NET Core

Introduction: In modern web applications, handling file uploads securely is crucial. One...

Aug 11

In Stackademic by Abdur Rahman

Python is No More The King of Data Science

5 Reasons Why Python is Losing Its Crown

Oct 23 · 9.2K · 35

See more recommendations