در فضای ‌.NET Core، سیستم تزریق وابستگی (Dependency Injection یا DI) از سه نوع ثبت سرویس اصلی یعنی **Transient**، **Scoped** و **Singleton** پشتیبانی می‌کند. استفاده درست از هر کدام از این سبک‌های ثبت می‌تواند از بروز مشکلاتی مثل مصرف بیش از حد منابع، اشتباه در به اشتراک‌گذاری وضعیت (state) و یا ایجاد رفتار نامطلوب در طول دوره حیات درخواست جلوگیری کند. در ادامه به تفصیل هر یک را توضیح می‌دهم:

**۱. AddTransient**

* **تعریف:**  
  زمانی که یک سرویس به عنوان Transient ثبت می‌شود، هر بار که آن سرویس درخواست شود (مثلاً در یک کنترلر یا حین تزریق درون یک کلاس دیگر)، یک نمونه جدید از آن ایجاد می‌شود.
* **کاربرد و مثال‌ها:**  
  اگر سرویسی حالت (state) داخلی نداشته باشد و یا صرفاً منطق‌های سبک محاسباتی یا تبدیل داده است، استفاده از Transient مناسب است.  
  به عنوان مثال سرویس‌هایی که تنها وظیفه‌ی تبدیل ورودی به خروجی ساده را دارند یا برای ماموریت‌های کوتاه‌مدت و بدون نگهداری وضعیت.

مثال:

public interface IOperation

{

Guid OperationId { get; }

}

public class TransientOperation : IOperation

{

public Guid OperationId { get; } = Guid.NewGuid();

}

سپس در Startup یا Program.cs:

services.AddTransient<IOperation, TransientOperation>();

هر بار که IOperation از DI درخواست شود، یک شی جدید با OperationId متفاوت ایجاد خواهد شد.

* **مزایا و مشکلات استفاده نادرست:**
  + **مزایا:**
    - بدون به اشتراک گذاشتن وضعیت، همیشه نمونه‌های تازه.
    - برای مواردی که هیچ داده‌ای بین درخواست‌های متعدد نیاز به به اشتراک‌گذاری ندارد.
  + **مشکلات:**
    - اگر سرویسی هزینه ساخت بالایی داشته باشد یا نیاز به به اشتراک‌گذاری داده‌ها داشته باشد، استفاده از Transient می‌تواند باعث overhead و مصرف بیشترین حافظه یا زمان پردازش شود.
    - در صورتی که اشتباهاً برای مواردی که تغییر وضعیت مداوم لازم است (مثلاً عملیات ذخیره‌سازی در پایگاه داده) این نوع انتخاب شود، ممکن است منجر به عدم نگهداری صحیح تراکنش‌ها یا اتصالات شود.

**۲. AddScoped**

* **تعریف:**  
  با ثبت یک سرویس به صورت Scoped، در طول هر درخواست (به خصوص در فضای وب) یک نمونه از آن سرویس ساخته می‌شود و در همان درخواست (HTTP request) در سراسر پیکربندی DI به اشتراک گذاشته می‌شود. وقتی درخواست تمام شد، آن نمونه منقضی (dispose) می‌شود.
* **کاربرد و مثال‌ها:**  
  برای مواردی که نیاز دارید در طول یک درخواست وب یک شی واحد استفاده شود، مثلاً برای سرویس‌های دسترسی به پایگاه داده مانند DbContext در EF Core، استفاده از Scoped مناسب است.  
  به عنوان مثال:
* public class ApplicationDbContext : DbContext
* {
* public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options) { }
* }

در Startup:

services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

(AddDbContext به صورت پیش‌فرض Scoped است.)

* **مزایا و مشکلات استفاده نادرست:**
  + **مزایا:**
    - در طول همان درخواست، وضعیت مشترک نگه داشته می‌شود (مثلاً تغییرات در Entity Framework فقط یک بار ثبت شوند).
    - مناسب برای ذخیره موقت داده‌های مرتبط با یک تراکنش یا عملیات وب.
  + **مشکلات:**
    - استفاده از Scoped در محیط‌های غیر وب (مانند کنسول اپلیکیشن) یا در مواردی که زنده بودن شیء نیاز به طولانی‌تر از یک درخواست داشته باشد، می‌تواند مشکل‌ساز شود.
    - استفاده نادرست (مانند تزریق یک شی Scoped در یک Singleton) می‌تواند منجر به مشکل "Captive Dependency" شود؛ به این معنی که یک شی دایمی (Singleton) به نمونه‌های دوره‌ای (Scoped) اشاره کند که این کار باعث گیج شدن چرخه حیات (lifetime) می‌شود.

**۳. AddSingleton**

* **تعریف:**  
  با ثبت یک سرویس به صورت Singleton، تنها یک نمونه از آن شی در طول کل عمر برنامه ساخته می‌شود. در اولین درخواست ایجاد شده و سپس برای تمام درخواست‌ها و تزریق‌های بعدی استفاده می‌شود.
* **کاربرد و مثال‌ها:**  
  اگر سرویسی داده‌های مشترک و کم تغییر یا منطق‌های محاسباتی سنگین داشته باشد که نیاز به ایجاد مجدد آن به صورت مکرر نداشته باشید، استفاده از Singleton بسیار مناسب است.  
  به عنوان مثال، سرویس‌های پیکربندی یا cacheهای که داده‌های ثابت را نگهداری می‌کنند:
* public interface ITimeProvider
* {
* DateTime Now { get; }
* }
* public class SystemTimeProvider : ITimeProvider
* {
* public DateTime Now => DateTime.Now;
* }

ثبت سرویس:

services.AddSingleton<ITimeProvider, SystemTimeProvider>();

در این صورت، در سراسر عمر برنامه یک نمونه از SystemTimeProvider ساخته می‌شود و هر جایی که ITimeProvider تزریق شود، همان نمونه به کار خواهد رفت.

* **مزایا و مشکلات استفاده نادرست:**
  + **مزایا:**
    - صرفه‌جویی در حافظه و منابع (به دلیل ایجاد تنها یک نمونه).
    - مناسب برای سرویس‌های بدون وضعیت یا با وضعیت مورد نیاز ثابت در طول عمر برنامه.
  + **مشکلات:**
    - اگر شیء Singleton دارای وضعیت تغییرپذیر باشد، این وضعیت بین تمام درخواست‌ها و کاربران به اشتراک گذاشته می‌شود که ممکن است منجر به مشکلات همزمانی یا اشتباه در داده‌ها شود.
    - استفاده از منابع یا سرویس‌هایی که باید در طول عمر درخواست‌ها مدیریت شوند (مانند پایگاه داده) در یک Singleton می‌تواند باعث "تداخل و نگهداری طولانی مدت منابع" شود.

**نتیجه‌گیری و مشکلات ناشی از عدم استفاده صحیح**

* **عدم استفاده از AddTransient:**
  + اگر یک سرویس بدون حالت مناسب (stateless) را به صورت Singleton ثبت کنید در صورتی که انتظار دارید هر بار یک نمونه جدید داشته باشید، ممکن است تغییرات یا محاسبات تکراری را نداشته باشید یا به اشتراک گذاشتن وضعیت باعث بروز خطا شود.
  + مثلا، اگر یک سرویس محاسباتی که به عنوان Transient باید هر بار یک مقدار جدید تولید کند به صورت Singleton ثبت شود، نتیجه‌ی مورد انتظار رخ نمی‌دهد.
* **عدم استفاده از AddScoped:**
  + ثبت سرویس‌هایی که برای هر درخواست جدید باید مجزا باشند (مانند DbContext) به صورت Singleton می‌تواند باعث بروز مشکلات همزمانی، نگهداری طولانی منابع و نشت حافظه شود.
  + همچنین اگر سرویسی که برای یک تراکنش یا درخواست خاص باید یکسان باشد به صورت Transient ثبت شود، ممکن است در طول یک درخواست چند نمونه جداگانه داشته باشید که باعث ناسازگاری داده‌ها می‌شود.
* **عدم استفاده از AddSingleton:**
  + اگر داده‌های به اشتراک گذاشته شده یا پیکربندی‌هایی را که فقط یک بار باید ساخته شوند به صورت Transient یا Scoped ثبت کنید، هر بار درخواست جدید باعث ایجاد دوباره آن شی خواهد شد که باعث افزایش مصرف حافظه و کاهش کارآیی می‌شود.
  + همچنین اگر منطق سنگین یا منابع مشترک از طریق تولید مجدد نمونه مدیریت شوند، کارآیی و عملکرد برنامه به شدت کاهش پیدا می‌کند.

**مثال جامع:**

فرض کنید یک سرویس مدیریت زمان داریم:

* سرویس بدون وضعیت: **TimeProvider**  
  ما می‌توانیم از آن به صورت Singleton استفاده کنیم چون مقدار آن همیشه یکسان است و تغییر نمی‌کند.
* سرویس عملیات حسابداری یا تبدیل داده (مثلاً تبدیل واحدها) بدون وضعیت و سبک:  
  می‌توان آن را به صورت Transient ثبت کرد تا هر بار یک شی جدید ساخته شود.
* یک سرویس دسترسی به داده (مانند DbContext) که هر درخواست HTTP یک نمونه داشته باشد:  
  از روش Scoped استفاده می‌کنیم.

کد ثبت در DI container به این شکل خواهد بود:

public void ConfigureServices(IServiceCollection services)

{

// Singleton: یک نمونه مشترک در کل عمر برنامه

services.AddSingleton<ITimeProvider, SystemTimeProvider>();

// Transient: هر بار که لازم باشد، یک نمونه جدید ایجاد می‌شود

services.AddTransient<IUnitConverter, UnitConverter>();

// Scoped: یک نمونه به ازای هر درخواست وب

services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

}

**جمع‌بندی:**  
استفاده صحیح از AddTransient، AddScoped و AddSingleton در DI موجب می‌شود که:

* منابع به طور بهینه مدیریت شوند.
* اشتراک وضعیت‌های غیر مناسب (مانند کلاس‌های وضعیت‌دار) اجتناب شود.
* کارآیی برنامه افزایش یابد و از نشت منابع جلوگیری شود.
* رفتار برنامه قابل پیش‌بینی و کنترل باشد.

بدین ترتیب، انتخاب درست سبک ثبت سرویس تأثیر مستقیمی بر عملکرد، مقیاس‌پذیری و ثبات نرم‌افزار شما دارد.