عنوان مقاله: بررسی ۱۱ ویژگی جدید ۸ .NET

نویسنده مقاله: تیم فنی نیک‌آموز

تاریخ انتشار: ۲۱ آبان ۱۴۰۲

منبع: https://nikamooz.com/net-۸-new-features/

ویژگی های .NET 8 به گونه‌ای طراحی شده‌اند که جامعه برنامه نویسان دات نت ، مشتاق عرضه این ورژن هستند. طراحی نسخه LTS این تضمین را می‌دهد که پشتیبانی، به‌روزرسانی و رفع مشکلات به‌صورت مداوم در .NET 8 انجام می‌شود و این موضوع آن را به یک انتخاب مطلوب برای پروژه‌های درازمدت تبدیل می‌کند. در این مطلب، قصد داریم به Feature ها و بهبودات ارائه‌شده در .NET 8 بپردازیم تا شما با مزیت‌های آن به‌خوبی آشنا شوید.

ویژگی های اضافه شده در دات نت ۸

Feature های جدید ارائه شده در .NET 8 به گونه‌ای هستند که کارایی و خلاقیت شما به سطح جدیدی ارتقا می‌یابند. در این بخش، ویژگی های دات نت ۸ را مورد بررسی قرار می‌دهیم. توجه شود که تمرکز این مطلب غالباً برروی بهبودها و ویژگی های .NET 8 مرتبط با .NET libraries Core است. در مقالات آتی، سایر موارد مهم همچون نحوه بهبود .NET 8 از نظر کارایی (Performance) مورد بررسی قرار می‌گیرد.

بهبود Garbage Collector در 8 .NET

یکی از ویژگی های جدید 8 .NET. این است که از این پس، می‌توانید ازطریق Garbage Collector (گاربیج کالکتور | GC)، محدودیت حافظه را برای اپلیکیشن مورد استفاده تغییر دهید. به واسطه این مشخصه جذاب دات نت ۸ ، این امکان را خواهید داشت که میزان منابع مصرفی اپلیکیشن را براساس نیاز آن افزایش یا کاهش دهید. در شرایطی که تقاضا برای Resource پایین باشد، می‌توان اپلیکیشن را به مقدار حداقلی، Scale کرد تا بدین طریق مطمئن شد که برای Resource های بلااستفاده، هزینه‌ای متحمل نمی‌شوید. با اعمال این تغییرات در 8 .NET. ، شما امکان فراخوانی API مربوطه، یعنی ()RefreshMemoryLimit را دارید و می‌توانید یک محدودیت حافظه جدید در نظر بگیرید.

با اجرای Snippet Code زیر، API مذکور را فراخوانی کنید:

```
GC.RefreshMemoryLimit();
```

البته شما امکان به‌روزرسانی برخی تنظیمات پیکربندی GC مرتبط با محدودیت حافظه را نیز خواهید داشت.

تکه کد زیر، Hard Limit مربوط به Heap را روی ۱۰۰ MiB تنظیم می‌کند:

```
AppContext.SetData("GCHeapHardLimit", (ulong)100 * 1024 * 1024);
GC.RefreshMemoryLimit();
```

به روز رسانی JSON

در نسخه 8 .NET. ، نحوه مدیریت Serialization و Deserialization مربوط به JSON بهبود پیدا کرده است. به این ترتیب، دات نت ۸ از نوع‌های عددی جدیدی همچون Half ، Int128 و UInt128 و همچنین مقدارهای Memory<T> و ReadOnlyMemory<T> پشتیبانی می‌کند. حال به قطعه کد زیر توجه کنید:

```
Console.WriteLine(JsonSerializer.Serialize(new object[] { Half.MaxValue,
Int128.MaxValue, UInt128.MaxValue }));
//
[۶۵۵۰۰,۱۷۰۱۴۱۱۸۳۴۶۰۴۶۹۲۳۱۷۳۱۶۸۷۳۰۳۷۱۵۸۸۴۱۰۵۷۲۷,۳۴۰۲۸۲۳۶۶۹۲۰۹۳۸۴۶۳۴۶۳۳۷۴۶۰۷
۴۳۱۷۶۸۲۱۱۴۵۵]
```

در مثال زیر، مقادیر Byte به رشته‌های Base64 و سایر نوع‌ها به آرایه‌های JSON سریال‌سازی شده‌اند:

```
JsonSerializer.Serialize<ReadOnlyMemory<byte>>(new byte[] { 1, 2, 3 }); // "AQID"
JsonSerializer.Serialize<Memory<int>>(new int[] { 1, 2, 3 }); // [1,2,3]
```

در واقع، بهبودهای متعددی در Serialization و Deserialization مربوط به System.Text.Json Namespace ایجاد شده که در ادامه لیست شده‌اند:

- پشتیبانی توکار (Built-in) از نوع‌های جدید
- بهبود Source generator
- امکان غیرفعال‌سازی Reflection-Based Serializer
- پشتیبانی از امکان سریال‌سازی در سلسله مراتب Interface

مثال زیر نشان می‌دهد که Property های اینترفیسی که به‌تازگی پیاده‌سازی شده و Base Interface آن سریال‌سازی شده‌اند:

```
IDerived value = new DerivedImplement { Base = 0, Derived = 1 };
JsonSerializer.Serialize(value); // {"Base":0,"Derived":1}
public interface IBase
{
    public int Base { get; set; }
}
public interface IDerived : IBase
{
    public int Derived { get; set; }
}
public class DerivedImplement : IDerived
{
    public int Base { get; set; }
    public int Derived { get; set; }
}
```

- ارائه سیاست‌های نامگذاری جدید

می‌توان مشابه سیاست‌های مربوط به JsonNamingPolicy.CamelCase ، از سیاست‌های جدید نامگذاری در NET 8. استفاده کرد:

```
var options = new JsonSerializerOptions { PropertyNamingPolicy =
JsonNamingPolicy.SnakeCaseLower };
JsonSerializer.Serialize(new { PropertyName = "value" }, options); // {
"property_name" : "value" }
```

- امکان Deserialization به فیلدهای و ویژگی‌های «فقط خواندنی»

در مثال زیر، Deserialization به نوع cutomerInfo انجام می‌شود، به طوری که دو Property فقط خواندنی دارا است:

```
using System.Text.Json;

CustomerInfo customer =
    JsonSerializer.Deserialize<CustomerInfo>("""{"Names":["John
Doe"],"Company":{"Name":"Contoso"}}""")!;

Console.WriteLine(JsonSerializer.Serialize(customer));

class CompanyInfo
{
    public required string Name { get; set; }
    public string? PhoneNumber { get; set; }
}

[JsonObjectCreationHandling(JsonObjectCreationHandling.Populate)]
class CustomerInfo
{
    // Both of these properties are read-only.
    public List<string> Names { get; } = new();
    public CompanyInfo Company { get; } = new() { Name = "N/A",
PhoneNumber = "N/A" };
}
```

پیش از .NET 8 ، از مقادیر ورودی چشم‌پوشی می‌شد و مقدار پیش‌فرض برای Property های Names و Company نگهداری می‌شدند:

```
{"Names":[],"Company":{"Name":"N/A","PhoneNumber":"N/A"}}
```

در حالی که در دات نت ۸ ، مقادیر ورودی به منظور پر کردن Property های Read-Only در طول Deserialization مورد استفاده قرار می‌گیرند. به منظور درک بهتر، به خروجی زیر توجه کنید:

```
{"Names":["John Doe"],"Company":{"Name":"Contoso","PhoneNumber":null}}
```

- ارائه متدهای (Methods) جدید برای JsonNode API

```
public partial class JsonNode
{
    // Creates a deep clone of the current node and all its descendants.
    public JsonNode DeepClone();

    // Returns true if the two nodes are equivalent JSON representations.
    public static bool DeepEquals(JsonNode? node1, JsonNode? node2);

    // Determines the JsonValueKind of the current node.
    public JsonValueKind GetValueKind(JsonSerializerOptions options =
null);

    // If node is the value of a property in the parent
    // object, returns its name.
    // Throws InvalidOperationException otherwise.
    public string GetPropertyName();

    // If node is the element of a parent JsonArray,
    // returns its index.
    // Throws InvalidOperationException otherwise.
    public int GetElementIndex();

    // Replaces this instance with a new value,
    // updating the parent object/array accordingly.
    public void ReplaceWith<T>(T value);

    // Asynchronously parses a stream as UTF-8 encoded data
    // representing a single JSON value into a JsonNode.
    public static Task<JsonNode?> ParseAsync(
        Stream utf8Json,
        JsonNodeOptions? nodeOptions = null,
        JsonDocumentOptions documentOptions = default,
        CancellationToken cancellationToken = default);
}

public partial class JsonArray
{
    // Returns an IEnumerable<T> view of the current array.
    public IEnumerable<T> GetValues<T>();
}
```

- امکان انتخاب Member های غیرعمومی در Serialization Contract برای Type داده شده (ازطریق Json Include و Json Constructor)

```
string json = JsonSerializer.Serialize(new MyPoco(42)); // {"X":42}
JsonSerializer.Deserialize<MyPoco>(json);

public class MyPoco
{
    [JsonConstructor]
    internal MyPoco(int x) => X = x;

    [JsonInclude]
    internal int X { get; }
}
```

- ارائه Extension Method های سریال‌سازی جریانی (Streaming Deserialization)

قطعه کد زیر، چگونگی استفاده از این Extension Method را نشان می‌دهد:

```
const string RequestUri = "https://api.contoso.com/books";
using var client = new HttpClient();
IAsyncEnumerable<Book>                          books                          =
client.GetFromJsonAsAsyncEnumerable<Book>(RequestUri);

await foreach (Book book in books)
{
    Console.WriteLine($"Read book '{book.title}'");
}

public record Book(int id, string title, string author, int publishedYear);
```

- ارائه اکستنشن متد WithAddedModifier

شما می‌توانید با استفاده از اکستنشن متد WithAddedModifier(IJsonTypeInfoResolver, Action<JsonTypeInfo>)، تغییرات مختلفی را به Serialization Contract های اینستنس‌های IJsonTypeInfoResolver اعمال کنید.

```
var options = new JsonSerializerOptions
{
    TypeInfoResolver = MyContext.Default
        .WithAddedModifier(static typeInfo =>
        {
            foreach (JsonPropertyInfo prop in typeInfo.Properties)
                prop.Name = prop.Name.ToUpperInvariant();
        })
};
```

- قابلیت ساخت اینستنس‌های JsonContent (ازطریق کانترکت‌های Trim-Safe و Source-Generated)

```
var book = new Book(id: 42, "Title", "Author", publishedYear: 2023);
HttpContent content = JsonContent.Create(book, MyContext.Default.Book);

public record Book(int id, string title, string author, int
publishedYear);

[JsonSerializable(typeof(Book))]
public partial class MyContext : JsonSerializerContext
{ }
```

- امکان کنترل JsonSerializerOptions Instance های متوقف شده (Freeze شده)

1. JsonSerializerOptions.MakeReadOnly(): این Overload برای Trim-safe طراحی شده و در شرایطی که انتخاب‌های مربوط به Instance با یک Resolver کانفیگ نشده باشند، Exception خواهد داد.
2. JsonSerializerOptions.MakeReadOnly(Boolean): اگر به این Overload مقدار True پاس داده شود، option های Instance با Reflection Resolver پیش‌فرض پر می‌شوند.

قابلیت Time Abstraction

قابلیت Time Abstraction، یکی از ویژگی های NET 8. محسوب می‌شود که می‌تواند تمام مشکلات مربوط به زمان را حل کند. در حقیقت در NET 8. ، کلاس TimeProvider و اینترفیس ITimer به شما کارایی انتزاع زمانی را ارائه می‌دهد و با کمک آن، شبیه‌سازی زمان در سناریوهای تست (Test) تسهیل می‌یابد. TimeProvider یک کلاس انتزاعی است که کارایی‌های مختلفی دارد و می‌توان آن را به‌عنوان یک انتخاب ایده‌آل برای ادغام با فریمورک‌های شبیه‌سازی (Mocking Framework) استفاده کرد. اساساً منظور از Mocking فرآیندی است که به شما امکان ساخت یک Mocking Object و استفاده از آن برای شبیه‌سازی رفتار شی اصلی را می‌دهد.

عملیات زمانی زیر در Time Abstraction پشتیبانی می‌شوند:

- استخراج ساعت زمانی محلی (Local) و سراسری (UTC)
- دریافت Timestamp برای بررسی کارایی
- ساخت تایمر

به منظور درک بهتر، به قطعه کد زیر توجه کنید:

```
// Get system time.
DateTimeOffset utcNow = TimeProvider.System.GetUtcNow();
DateTimeOffset localNow = TimeProvider.System.GetLocalNow();

// Create a time provider that works with a
// time zone that's different than the local time zone.
private class ZonedTimeProvider : TimeProvider
{
    private TimeZoneInfo _zoneInfo;

    public ZonedTimeProvider(TimeZoneInfo zoneInfo) : base()
    {
        _zoneInfo = zoneInfo ?? TimeZoneInfo.Local;
    }

    public override TimeZoneInfo LocalTimeZone => _zoneInfo;

    public static TimeProvider FromLocalTimeZone(TimeZoneInfo zoneInfo) =>
        new ZonedTimeProvider(zoneInfo);
}

// Create a timer using a time provider.
ITimer timer = timeProvider.CreateTimer(callBack, state, delay,
Timeout.InfiniteTimeSpan);

// Measure a period using the system time provider.
long providerTimestamp1 = TimeProvider.System.GetTimestamp();
long providerTimestamp2 = TimeProvider.System.GetTimestamp();

var period = GetElapsedTime(providerTimestamp1, providerTimestamp2);
```

بهبود UTF8

شما می‌توانید در .NET 8 امکان نمایش یک شبه‌رشته برای Type را فعال کنید. تنها کافی است از Interface جدید .NET 8 تحت عنوان IUtf8SpanFormattable برای Type موردنظر استفاده کنید. این ویژگی دات نت ۸ به‌صورت خاص برای فرمت UTF-8 طراحی شده است و برای همه نوع‌های داده اصلی (Primitive Data Type) پیاده‌سازی شده‌اند. مثال زیر به شما کمک می‌کند تا با این ویژگی .NET 8 آشنایی کافی داشته باشید.

```
static bool FormatHexVersion(
    short major,
    short minor,
    short build,
    short revision,
    Span<byte> utf8Bytes,
    out int bytesWritten) =>
    Utf8.TryWrite(
        utf8Bytes,
        CultureInfo.InvariantCulture,
        $"{major:X4}.{minor:X4}.{build:X4}.{revision:X4}",
        out bytesWritten);
```

ارائه متدهایی برای Randomness

در نوع‌های System.Random و System.Security.Cryptography.RandomNumberGenerator ، متدهایی ارائه شده‌اند که به واسطه آن‌ها امکان کار با تصادفی بودن (Randomness) موارد فراهم شده است. با به‌کارگیری این مشخصه از میان انواع ویژگی های NET 8. ، اجازه دارید مستقیماً Randomness را به‌عنوان Selector استفاده کنید. این مشخصه به‌طور خاص برای اپلیکیشن‌های یادگیری ماشین (Machine Learning) کاربردی است.

- **<GetItems<T>()**: متدهای جدید System.Random.GetItems و System.Security.Cryptography.RandomNumberGenerator.GetItems به شما این امکان را می‌دهند که به‌صورت تصادفی یا همان Random، تعداد خاصی از آیتم‌ها را به‌عنوان مجموعه ورودی انتخاب کنید. می‌توان مثال زیر را در بازی Simon استفاده کرد که در آن بازیکنان باید رنگ‌های مجموعه‌ای از دکمه‌ها را به‌خاطر بسپارند.

```
private static ReadOnlySpan<Button> s_allButtons = new[]
{
    Button.Red,
    Button.Green,
    Button.Blue,
    Button.Yellow,
};

// ...

Button[] thisRound = Random.Shared.GetItems(s_allButtons, 31);
...// Rest of game goes here
```

- **<Shuffle<T>()**: متدهای جدید Random.Shuffle و RandomNumberGenerator.Shuffle<T>(Span<T>) به شما اجازه می‌دهد که ترتیب یک مجموعه را به‌صورت تصادفی استفاده کنید. این ویژگی NET 8. در کاهش سوگیری (Bias) مرحله Training مؤثر واقع می‌شود.

به مثال زیر از Shuffle داده‌های آموزشی دقت کنید:

```
YourType[] trainingData = LoadTrainingData();
Random.Shared.Shuffle(trainingData);

IDataView sourceData = mlContext.Data.LoadFromEnumerable(trainingData);

DataOperationsCatalog.TrainTestData split =
mlContext.Data.TrainTestSplit(sourceData);
model = chain.Fit(split.TrainSet);

IDataView predictions = model.Transform(split.TestSet);
... //
```

بهبود رمزنگاری و پشتیبانی از SHA-3

در شرایطی که همه‌روزه شاهد تکامل انواع Thread در فضای اینترنت هستیم، هنوز یکی از ویژگی های NET 8. به‌عنوان راه نجات اپلیکیشن ارائه شده است. به‌واسطه پشتیبانی از SHA-3 در دات نت هشت، این اطمینان حاصل می‌شود که اپلیکیشن های NET. امن باقی می‌مانند. مثال پایین، به شما نحوه استفاده از API ها، شامل مشخصه SHA3_256.IsSupported را نمایش می‌دهد و شما با توجه به آن، چگونگی پشتیبانی از SHA-3 را درک می‌کنید.

```
// Hashing example
if (SHA3_256.IsSupported)
{
    byte[] hash = SHA3_256.HashData(dataToHash);
}
else
{
    // ...
}

// Signing example
if (SHA3_256.IsSupported)
{
     using ECDsa ec = ECDsa.Create(ECCurve.NamedCurves.nistP256);
     byte[] signature = ec.SignData(dataToBeSigned,
HashAlgorithmName.SHA3_256);
}
else
{
    // ...
{
```

بهبود شبکه (Networking)

در نسخه NET 8. ، پروکسی HTTPS توسط HttpClient پشتیبانی می‌شود. به این ترتیب، یک کانال رمزگذاری شده بین کلاینت و Proxy ایجاد می‌شود تا بدین طریق، به تمام Request ها با حفظ حریم خصوصی رسیدگی شود. به منظور فعال‌سازی HTTPS Proxy، باید Environment Variable خاصی که all_proxy نام دارد را تنظیم کنید. البته می‌توان از کلاس WebProxy نیز استفاده کرد تا پروکسی ازطریق برنامه‌نویسی تحت نظارت شما باشد.

در سیستم عامل Unit:

```
export all_proxy=https://x.x.x.x:3218
```

در Windows:

```
set all_proxy=https://x.x.x.x:3218
```

اعتبارسنجی داده ها (Data Validation)

یک از ویژگی های NET 8. ، ارائه Attribute های جدیدی برای اعتبارسنجی داده‌ها در فضای نام System.ComponentModel.DataAnnotations است. این صفت‌ها با هدف صحت‌سنجی سناریوهای مربوط به سرویس‌های ابری بومی (Cloud-Native) در دات نت ۸ قرار گرفته و به منظور تعیین اعتبار داده‌های ورودی غیرکاربری، مانند configuration options، طراحی شده است. علاوه‌بر این، در نسخه NET 8. ، تعدادی Property جدید به Type های RangeAttribute و RequiredAttribute اضافه شده است.

معیار های اصلی (Metrics)

API های جدید ارائه شده در NET 8. ، به شما اجازه می‌دهند که بتوانید در زمان ساخت، برای اشیای Meter و Instrument ، زوج Tag های کلید – مقدار ضمیمه کنید.

```
MeterOptions options = new MeterOptions("name")
{
    Version = "version",
    // Attach these tags to the created meter
    Tags = new TagList() { { "MeterKey1", "MeterValue1" }, { "MeterKey2",
"MeterValue2" } }
};


Meter meter = meterFactory.Create(options);


Instrument instrument = meter.CreateCounter<int>("counter", null, null,
new TagList() { { "counterKey1", "counterValue1" } });
 instrument.Add (1) ;
```

این API های ارائه‌شده عبارتند از:

- MeterOptions
- Meter(MeterOptions)
- CreateCounter<T>(String, String, String, IEnumerable<KeyValuePair<String,Object>>)

بهبود پشتیبانی از SIMD

در نسخه 8 .NET ، معرفی Vector512<T> و همچنین پشتیبانی از دستورالعمل‌های Intel Advanced Vector Extensions 512 (AVX-512) به پشتوانه قوی‌تری برای SIMD منجر می‌شود. در حقیقت، بهره‌وری از قدرتِ سخت‌افزارهای مدرن به‌همراه مجموعه دستورالعمل‌های AVX-512، بهبود قابل توجهی در کارایی به‌همراه دارد. این موضوع به‌طور خاص برای اپلیکیشن‌هایی صادق است که به‌شدت روی پردازش داده‌ها (Data Processing) اکتفا می‌کنند.

متد های مبتنی بر جریان ZipFile

شما می‌توانید به کمک Overload های جدید ZipFile.ExtractToDirectory ، یک جریان (Stream) فراهم کنید که حاوی فایل زیپ شده باشد و بتوانید محتوای آن را به یک سیستم فایل (File System) استخراج کنید. به‌عنوان یکی از ویژگی های 8 .NET ، این موضوع در مواقعی که فضای دیسک محدود باشد، مفید واقع می‌شود. در ادامه Overload های جدید قابل مشاهده هستند:

```
namespace System.IO.Compression;

public static partial class ZipFile

{

    public static void CreateFromDirectory(string sourceDirectoryName,
Stream destination);

    public static void CreateFromDirectory(string sourceDirectoryName,
Stream destination, CompressionLevel compressionLevel, bool
includeBaseDirectory);

    public static void CreateFromDirectory(string sourceDirectoryName,
Stream destination, CompressionLevel compressionLevel, bool
includeBaseDirectory, Encoding? entryNameEncoding);


    public static void ExtractToDirectory(Stream source, string
destinationDirectoryName) { }

    public static void ExtractToDirectory(Stream source, string
destinationDirectoryName, bool overwriteFiles) { }

    public static void ExtractToDirectory(Stream source, string
destinationDirectoryName, Encoding? entryNameEncoding) { }

    public static void ExtractToDirectory(Stream source, string
destinationDirectoryName, Encoding? entryNameEncoding, bool
overwriteFiles) { }

{
```

مروری بر ویژگی های 8 .NET

نسخه جدید 8 .NET. در روزهای پیش رو عرضه خواهد شد؛ این نسخه به گونه‌ای اهمیت دارد که برنامه نویسان دات نت بی‌صبرانه منتظر انتشار آن هستند تا از مزایای آن در کنار خلاقیت و سازندگی خودشان بهره‌وری کنند. این مطلب، با هدف آشنایی با ویژگی های 8 .NET. ارائه شده است و به‌عنوان راهنما می‌تواند کمک کننده باشد. شایان ذکر است که در مقاله‌های آتی، به بررسی قابلیت های 8 .NET. از نقطه نظر کارایی می‌پردازیم.