Robert Haken

Novinky .NET 8

Co se událo v .NET 8

- .NET 8 je LTS release (3y support)
- Runtime, Core .NET Libraries, Extension libraries
- ASP.NET Core + Blazor
- C# 12
- Entity Framework Core 8

Core .NET libraries

- JSON Serialization
- Time abstraction
  - `ITimeProvider`, `ITimer`
- UTF8 improvements
- Methods for working with randomness
  - `GetItems<T>()`, `Shuffle<T>()`
- Performance-focused types
  - `FrozenDictionary<TKey, TValue>`, `FrozenSet<T>`
  - `SearchValues<T>`
- System.Numerics and System.Runtime.Intrinsics

Core .NET libraries

- Data validation
  - `[Length(10, 50)]`
  - `[Range(0, 100, MinimumIsExclusive = true, MaximumIsExclusive = true]`
  - `[Base64String]`
  - `[AllowedValues("apple", "banana", "mango")]`, `[DeniedValues(...)]`
- Metrics
- Cryptography - SHA-3 support
- Networking - HTTPS proxy support
- Stream-based ZipFile methods
  - `ZipFile.CreateFromDirectory(string dir, Stream dest)`
  - `ZipFile.ExtractToDirectory(Stream source, string dir)`

Extension libraries

- Keyed DI services

```
services.AddKeyedSingleton<ICacheService, LocalCacheService>("local")

public MyService([FromKeyedServices("local")] ICacheService _cache)

IKeyedServiceProvider.GetRequiredKeyedService<T>(object key)

[Inject(Key = "local")] protected ICacheService CacheService { get; set; }
```

- Hosted lifecycle services

```
interface IHostedLifecycleService : IHostedService
{
        StartingAsync(CancellationToken)
        StartedAsync(CancellationToken)
        StoppingAsync(CancellationToken)
        StoppedAsync(CancellationToken)
}
```

Other

- Options validation - source generators
- Garbage collector - memory limit adjustments on the fly

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

GC.RefreshMemoryLimit();
```

- Configuration-binding source generator

```
<PropertyGroup>
  <EnableConfigurationBindingGenerator>true</EnableConfigurationBindingGenerator>
</PropertyGroup>
```

- Reflection improvements - function-pointers support
- Performance optimizations (e.g. *Dynamic PGO* enabled by default)
- New code analyzers (perf-guidance, ...)

ASP.NET Core

Blazor

- Full-stack web UI
  - Static server rendering
  - Interactive server rendering (Blazor Server)
  - Interactive client rendering (Blazor WebAssembly)
  - Automatic interactive client rendering (Blazor Server first + WASM next time)
  - (+ prerendering)
- New *Blazor Web App* template
- Form handling and model binding
- Enhanced navigation and form handling
- Streaming rendering

Blazor

- Access `HttpContext` as a cascading parameter (static server component)
- Render Razor components outside of ASP.NET Core

```
await using var htmlRenderer = new HtmlRenderer(serviceProvider, loggerFactory);
string html = await htmlRenderer.Dispatcher.InvokeAsync(async () =>
{
    var dictionary = new Dictionary<string, object?>
    {
        { "Message", "Hello from the Render Message component!" }
    };

    var parameters = ParameterView.FromDictionary(dictionary);
    var output = await htmlRenderer.RenderComponentAsync<RenderMessage>(parameters);

    return output.ToHtmlString();
});
```

- Sections support - `<SectionOutlet>`, `<SectionContent>`
- Error page support

Blazor

- QuickGrid no longer "experimental"
- Route to named elements (URL `#fragments`) - automatic scrolling
- Root-level cascading values
- Virtualize empty content
- Close circuits when there are no remaining interactive server components
  - Monitor SignalR circuit activity
- Faster runtime performance with the Jiterpreter (partial JIT support)
- Web-friendly Webcil packaging
- Blazor WebAssembly debugging improvements 🤔
- Support for dialog cancel and close events
  - `<dialog @onclose="..." @oncancel="..." />`

Blazor

- Blazor Identity UI
- Blazor Server with Yarp routing

ASP.NET Core

- Authentication and authorization - Identity API endpoints
  - `.MapIdentityApi<TUser>()` = `/register` + `/login`
- Support for generic attributes, e.g. `[ProducesResponseType<T>]`
- Redis-based output caching `.AddStackExchangeRedisOutputCache(..)`
- SignalR improvements (e.g. stateful reconnect)
- Minimal APIs
  - Binding to forms (explicit `[FromForm]`)
  - Antiforgery, `[AsParameters]`, ...
- Native AOT support (gRPC, Minimal APIs, worker service apps)

C# 12

C# 12 - Primary constructors

```
public class MyService(IDependency _dependency);

public void DoSomething()
{
        _dependency.DoIt();
}
```

C# 12 - Collection expressions

```
// Create an array:
int[] a = [1, 2, 3, 4, 5, 6, 7, 8];

// Create a span
Span<int> b  = ['a', 'b', 'c', 'd', 'e', 'f', 'h', 'i'];

// Create a jagged 2D array:
int[][] twoD = [[1, 2, 3], [4, 5, 6], [7, 8, 9]];

// Create a jagged 2D array from variables:
int[] row0 = [1, 2, 3];
int[] row1 = [4, 5, 6];
int[] row2 = [7, 8, 9];
int[][] twoDFromVariables = [row0, row1, row2];
```

C# 12 - Spread operator

```
int[] row0 = [1, 2, 3];
int[] row1 = [4, 5, 6];
int[] row2 = [7, 8, 9];
int[] single = [..row0, ..row1, ..row2];
foreach (var element in single)
{
    Console.Write($"{element}, ");
}
// output:
// 1, 2, 3, 4, 5, 6, 7, 8, 9,
```

C# 12 - Inline arrays

```
[System.Runtime.CompilerServices.InlineArray(10)]
public struct Buffer
{
    private int _element0;
}

// use as regular array
var buffer = new Buffer();
for (int i = 0; i < 10; i++)
{
    buffer[i] = i;
}

foreach (var i in buffer)
{
    Console.WriteLine(i);
}
```

C# 12 - Other

- `ref readonly` parameters
- Default lambda parameters
- Alias any type
  - `using Point = (int x, int y);`

C# 12 - Interceptors (experimental)

```
var c = new C();
c.InterceptableMethod(1); // (L1,C1): prints "interceptor 1"
c.InterceptableMethod(1); // (L2,C2): prints "other interceptor 1"
c.InterceptableMethod(2); // (L3,C3): prints "other interceptor 2"
c.InterceptableMethod(1); // prints "interceptable 1"

class C
{
    public void InterceptableMethod(int param)
    {
        Console.WriteLine($"interceptable {param}");
    }
}

// generated code
static class D
{
    [InterceptsLocation("Program.cs", line: /*L1*/, character: /*C1*/)] // refers to the call at (L1, C1)
    public static void InterceptorMethod(this C c, int param)
    {
        Console.WriteLine($"interceptor {param}");
    }

    [InterceptsLocation("Program.cs", line: /*L2*/, character: /*C2*/)] // refers to the call at (L2, C2)
    [InterceptsLocation("Program.cs", line: /*L3*/, character: /*C3*/)] // refers to the call at (L3, C3)
    public static void OtherInterceptorMethod(this C c, int param)
    {
        Console.WriteLine($"other interceptor {param}");
    }
}
```

Reference

- What's new in .NET 8 | Microsoft Learn
- What's new in ASP.NET Core 8.0 | Microsoft Learn
- What's new in C# 12 - C# Guide - C# | Microsoft Learn
- What's new with identity in .NET 8 - .NET Blog
- Performance Improvements in .NET 8 - .NET Blog
- Performance Improvements in ASP.NET Core 8 - .NET Blog