

ASP.NET Core 9.0 全新功能探索

多奇數位創意有限公司

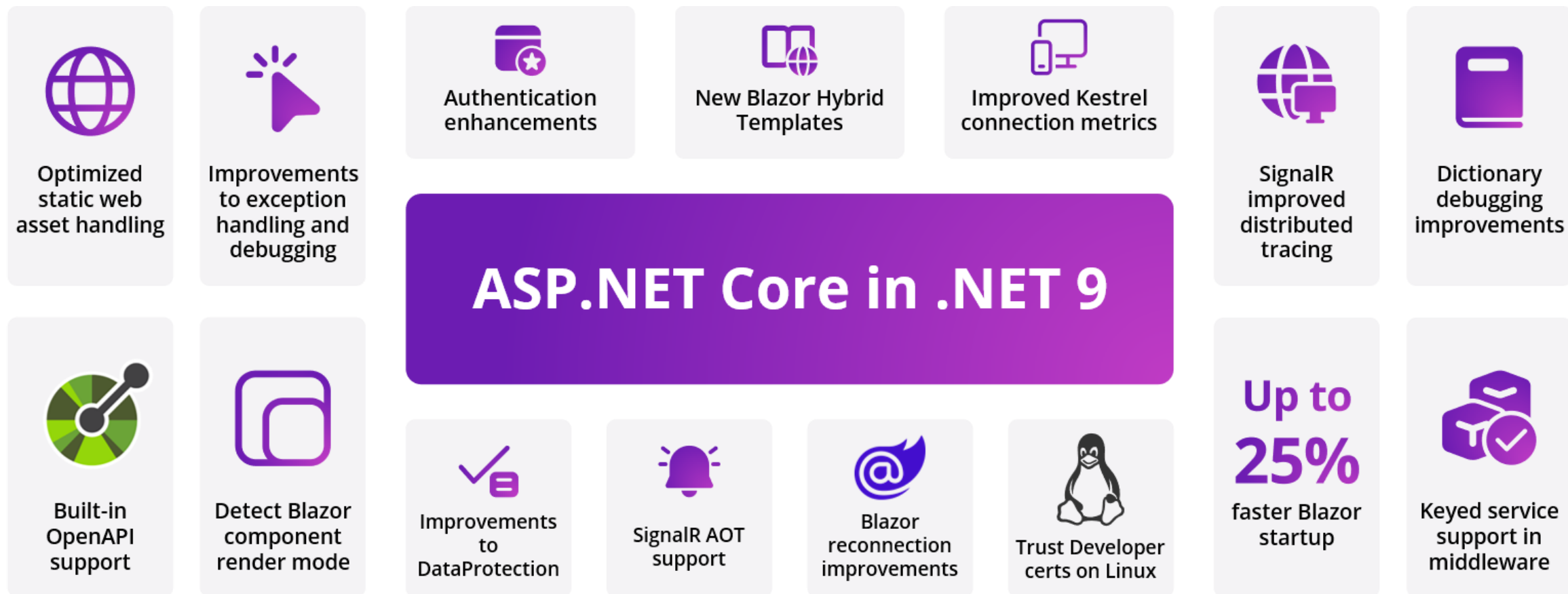
技術總監 黃保翕 (Will 保哥)

<https://blog.miniasp.com>



.NET Conf
2024

.NET 9 中 ASP.NET Core 的一些改進





Breaking changes in .NET 9

.NET 9 的破壞性變更

開發用憑證匯出時不再自動建立資料夾

- [Dev cert export no longer creates folder](#)
 - 當您匯出 ASP.NET Core 開發憑證時，如果該目錄不存在，它將不再建立用來匯出憑證的目錄。
- 先前的行為
 - `dotnet dev-certs https -ep C:\NonExistent\cert.pfx` （不存在的資料夾會自動建立目錄）
- 現在的行為（.NET 9）
 - `dotnet dev-certs https -ep C:\NonExistent\cert.pfx` （不存在的資料夾會報錯）
- 變更的理由
 - 開發憑證會隨其**私密金鑰**一起匯出，因此未經授權的存取可能會造成問題。
 - 然而，如果 Client 應用程式不會以當前使用者身份執行，可能有必要讓多個帳戶都能讀取它。
 - 與其嘗試確定並安全地建立目標目錄的權限，**dotnet dev-certs** 要求該目錄必須已經存在！

貼心提醒 你的開發用測試憑證該更新了！

- 先清除舊的根憑證

```
dotnet dev-certs https --clean
```

- 重新安裝並信任 .NET SDK 所附的根憑證

```
dotnet dev-certs https --trust
```

開發環境中預設啟用 ValidateOnBuild

- [HostBuilder enables ValidateOnBuild/ValidateScopes in development environment](#)
- 先前的行為
 - 之前 [ValidateOnBuild](#) 和 [ValidateScopes](#) 在「任何環境」預設皆為 **false**
 - 只有在明確呼叫 [UseDefaultServiceProvider](#) 時才會啟用
- 現在的行為 (.NET 9)
 - 現在 [ValidateOnBuild](#) 和 [ValidateScopes](#) 在「開發環境」預設為 **true**
 - 您可以通過呼叫 [UseDefaultServiceProvider](#) 來停用這個預設值
- 變更的理由
 - 驗證有助於在應用程式**啟動初期**捕捉問題，而不是在應用程式與其服務提供者互動時才發現(或根本沒發現)

認識 ValidateOnBuild 和 ValidateScopes

- **ValidateOnBuild**

- 此選項決定在呼叫 **BuildServiceProvider** 時，是否驗證**所有服務**能否成功建立。
- **注意**：並不是在服務「注入」的時候檢查，而是在**建立服務提供者**時檢查。
- 啟用此驗證可在應用程式啟動時及早發現**服務註冊**中的問題，例如**未註冊的相依性**或**無法解析的服務**。
- 需要注意的是，此驗證不會檢查**開放式泛型服務**。
- 在**開發環境**中，啟用這些驗證有助於及早發現配置錯誤，提升應用程式的穩定性。
- 在**生產環境**中，則可視需求決定是否啟用，因為這些驗證可能會**對效能產生影響**。

- **ValidateScopes**

- 此選項確保**服務的生命週期**配置正確，特別是**範圍 (Scoped)** 服務不會被誤用為**單體 (Singleton)**。
- 啟用後，應用程式會在解析服務時檢查範圍服務是否被單例服務或其他不正確的生命週期範圍引用，防止潛在的生命週期錯誤。
- [ASP.Net Core 2 ServiceProviderOptions.ValidateScopes Property](#)

示範：ValidateOnBuild

```
public class MyService
{
    private readonly IDependency _dependency;

    public MyService(IDependency dependency)
    {
        _dependency = dependency;
    }
}

public interface IDependency { }
```

```
using Microsoft.EntityFrameworkCore;
using WebApplication1.Models;

var builder = WebApplication.CreateBuilder(args);

// 如果沒有註冊相依的服務，應用程式啟動時會拋出 InvalidOperationException
//builder.Services.AddTransient<IDependency, Dependency>();
builder.Services.AddTransient<MyService>();
```


啟動時的 ValidateOnBuild 錯誤訊息

Unhandled exception. **System.AggregateException:** Some services are not able to be constructed (Error while validating the service descriptor 'ServiceType: MyService Lifetime: Transient ImplementationType: MyService': Unable to resolve service for type 'IDependency' while attempting to activate 'MyService'.)

---> **System.InvalidOperationException:** Error while validating the service descriptor 'ServiceType: MyService Lifetime: Transient ImplementationType: MyService': Unable to resolve service for type 'IDependency' while attempting to activate 'MyService'.

---> **System.InvalidOperationException:** Unable to resolve service for type 'IDependency' while attempting to activate 'MyService'.

BuildServiceProvider 發生錯誤

```
1 Unhandled exception. System.AggregateException: Some services are not able to be constructed (Error while validating the service descriptor 'ServiceType: MyService Lifetime: 
2 ---> System.InvalidOperationException: Error while validating the service descriptor 'ServiceType: MyService Lifetime: Transient ImplementationType: MyService': Unable to re
3 ---> System.InvalidOperationException: Unable to resolve service for type 'IDependency' while attempting to activate 'MyService'.
4 at Microsoft.Extensions.DependencyInjection.ServiceLookup.CallSiteFactory.CreateArgumentCallSites(ServiceIdentifier serviceIdentifier, Type implementationType, CallSiteChain
5 at Microsoft.Extensions.DependencyInjection.ServiceLookup.CallSiteFactory.CreateConstructorCallSite(ResultCache lifetime, ServiceIdentifier serviceIdentifier, Type implement
6 at Microsoft.Extensions.DependencyInjection.ServiceLookup.CallSiteFactory.TryCreateExact(ServiceDescriptor descriptor, ServiceIdentifier serviceIdentifier, CallSiteChain ca
7 at Microsoft.Extensions.DependencyInjection.ServiceLookup.CallSiteFactory.GetCallSite(ServiceDescriptor serviceDescriptor, CallSiteChain callSiteChain)
8 at Microsoft.Extensions.DependencyInjection.ServiceProvider.ValidateService(ServiceDescriptor descriptor)
9 --- End of inner exception stack trace ---
10 at Microsoft.Extensions.DependencyInjection.ServiceProvider.ValidateService(ServiceDescriptor descriptor)
11 at Microsoft.Extensions.DependencyInjection.ServiceProvider..ctor(ICollection`1 serviceDescriptors, ServiceProviderOptions options)
12 --- End of inner exception stack trace ---
13 at Microsoft.Extensions.DependencyInjection.ServiceProvider..ctor(ICollection`1 serviceDescriptors, ServiceProviderOptions options)
14 at Microsoft.Extensions.DependencyInjection.ServiceCollectionContainerBuilderExtensions.BuildServiceProvider(IServiceCollection services, ServiceProviderOptions options)
15 at Microsoft.Extensions.Hosting.HostApplicationBuilder.Build()
16 at Microsoft.AspNetCore.Builder.WebApplicationBuilder.Build()
17 at Program.<Main>$(String[] args) in G:\Projects\WebApplication1\WebApplication1\Program.cs:line 37
18
```

示範：ValidateScopes

```
public class MyService
{
    private readonly IDependency _dependency;

    public MyService(IDependency dependency)
    {
        _dependency = dependency;
    }
}

public interface IDependency { string Title { get; set; } }
public class Dependency : IDependency { public string Title { get; set; } }
```

```
// 如果 MyService 註冊為 Singleton, 但 IDependency 註冊為 Scoped 時,
// 啟動時會拋出 InvalidOperationException
builder.Services.AddScoped<IDependency, Dependency>();
builder.Services.AddSingleton<MyService>();
```

啟動時的 ValidateOnBuild 錯誤訊息

Unhandled exception. **System.AggregateException**: Some services are not able to be constructed (Error while validating the service descriptor 'ServiceType: MyService Lifetime: Singleton ImplementationType: MyService': Cannot consume scoped service 'IDependency' from singleton 'MyService'.)

---> **System.InvalidOperationException**: Error while validating the service descriptor 'ServiceType: MyService Lifetime: Singleton ImplementationType: MyService': Cannot consume scoped service 'IDependency' from singleton 'MyService'.

---> **System.InvalidOperationException**: Cannot consume scoped service 'IDependency' from singleton 'MyService'.

正式環境開啟驗證的方法

- 正式環境通常不會開啟驗證，因為這會導致影響啟動應用程式的速度

```
builder.Host.UseDefaultServiceProvider((context, options) =>
{
    options.ValidateScopes = true;
    options.ValidateOnBuild = true;
});
```

其他冷門的 ASP.NET Core 破壞性變更

- [DefaultKeyResolution.ShouldGenerateNewKey has altered meaning](#)
 - DefaultKeyResolution.ShouldGenerateNewKey 不再反映預設鍵是否接近其到期時間。
 - Microsoft.AspNetCore.DataProtection.KeyManagement.Internal.DefaultKeyResolution.ShouldGenerateNewKey
- [Legacy Mono and Emscripten JavaScript APIs not exported to global namespace](#)
 - Blazor WebAssembly 不再將舊版的 Mono 和 Emscripten API 匯出到全域命名空間。
 - 這些 API 現在可以透過 Blazor.runtime 物件存取。
- [Middleware types with multiple constructors](#)
 - 以前，當從相依性注入容器實例化具有多個可滿足建構函數的中介軟體類型時，會使用參數最多的建構函數。
 - 現在，只有當相依性注入容器實作了 [IServiceProviderIsService](#) 時才會這樣。如果沒有，在執行時會拋出異常。



Visual Studio 2022

Visual Studio 2022 v17.12 GA

- 完整支援 **.NET 9** 開發！
 - [Visual Studio 2022 release notes](#)
- 新增專案範本
 - [Teams Toolkit](#) 新增 AI 功能的專案範本
- 好用的 Web 功能
- 更強大的 Git 版控功能
- 支援在多個 Visual Studio 2022 實例中複製檔案
- 大量的 GitHub Copilot 更新
 - 記得來報名我在 2024/12/20 (五) 的 [GitHub Copilot 進階開發實戰](#) 課程！🧐

Teams Toolkit 新增 AI 功能的專案範本

×

建立新的 Teams 應用程式

選取應用程式類型以開始使用

所有應用程式類型

所有平台

索引標籤

Outlook

Microsoft 365 應用程式

回應機器人

回應機器人的簡易實作，其已準備好進行自訂。

機器人

基本 AI 聊天機器人

Teams 中使用 Teams AI 程式庫的基本 AI 聊天機器人。

機器人

Chat With Your Data

Expand AI bot's knowledge with your content to get accurate answers to your questions

機器人

架構

.NET 8.0 (長期支援)

上一步

下一頁

專案範本支援 OpenAI 與 Azure OpenAI

×

服務設定

選取必要的 LLM 服務之後，您可以在稍後的階段設定相關聯的索引鍵值。

LLM Service

選取要存取 LLM 的服務

選取要存取 LLM 的服務

OpenAI

Azure OpenAI

上一步

建立

更簡便的 Teams App 服務設定

×

服務設定

選取必要的 LLM 服務之後，您可以在稍後的階段設定相關聯的索引鍵值。

LLM Service

Azure OpenAI

▼

Azure OpenAI 服務金鑰

Azure OpenAI 端點

Azure OpenAI 部署名稱

上一步

建立

HTTP file (.http) 支援請求變數

- [HTTP file](#) (.http) 現在支援請求變數！👍
- 這意味著您可以發送請求，然後在未來的請求中使用來自回應或請求的資料。

```
# @name login
POST {{TodoApi_HostAddress}}/users/token
Content-Type: application/json

{
  "username": "{{myusername}}",
  "password": "{{mypassword}}"
}

###

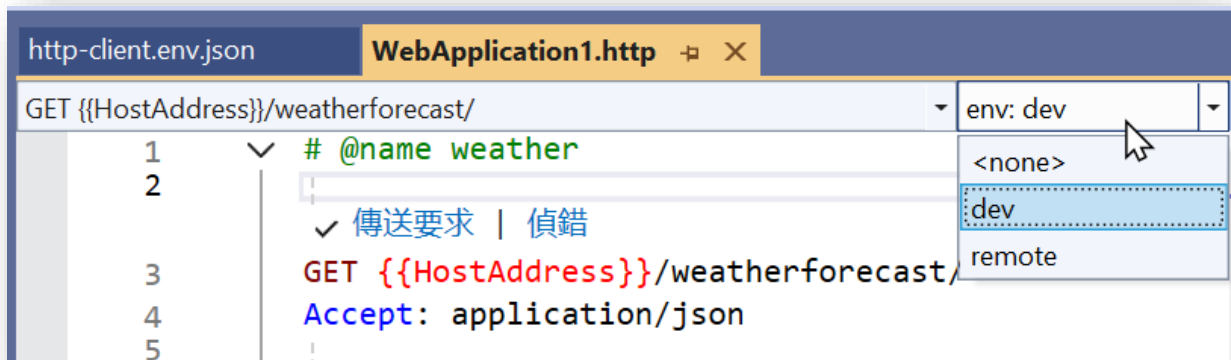
GET {{TodoApi_HostAddress}}/todos
Authorization: Bearer {{login.response.body.$.token}}
```

HTTP file (.http) 支援環境設定檔

- [HTTP file](#) (.http) 現在支援 **http-client.env.json** [環境設定檔](#) ! 👍
- 環境設定檔跟 .http 檔案放在一起，或放到任何一個上層資料夾皆可！



```
WebApplication1.http  http-client.env.json
結構描述: http://json-schema.org/draft-04/schema
1  {
2    "dev": {
3      "HostAddress": "https://localhost:44320"
4    },
5    "remote": {
6      "HostAddress": "https://contoso.com"
7    }
8  }
```



```
http-client.env.json  WebApplication1.http
GET {{HostAddress}}/weatherforecast/  env: dev
1  # @name weather
2
3  GET {{HostAddress}}/weatherforecast/
4  Accept: application/json
5
```

支援更多內嵌提示 (Inlay Hint)

- 新增的程式語言

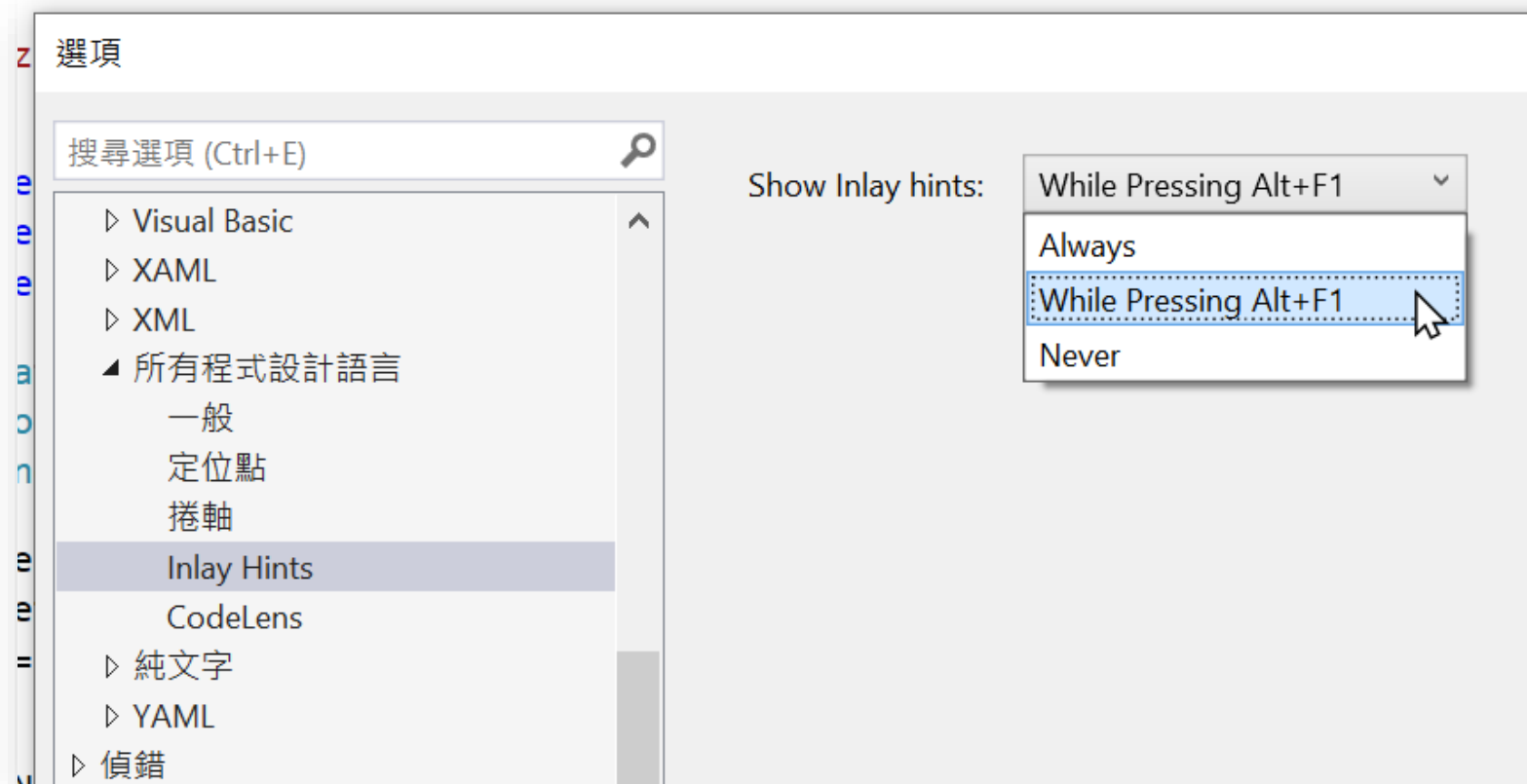
- JavaScript
- TypeScript
- Python
- Razor

- 使用方式

- 長按 **Alt+F1**

- 更多選項設定

- **Tools > Options > Text Editor > All Languages > Inlay Hints**



增強的 IEnumerable 視覺化工具

```
[HttpGet]
public async Task<ActionResult<IEnumerable<Course>>> GetCourses()
{
    var data = await _context.Courses.ToListAsync();
    return data;
}
```

運算式:

data.Where(p => p.CourseId > 10)

篩選

值	Course	Course.CourseId	Course.Title	Course.Cre
{Web	WebApplication1.Models.Course	12	十項超實用 Excel 技能 — 節省 50% 的工作效率	2
Coun	WebApplication1.Models.Course	13	Windows Containers	
	WebApplication1.Models.Course	15	1	

匯出至 CSV...

匯出至 Excel...

運算式:

data.Wh

Where<> (擴充功能) IEnumerable<Course> IEnumerable<Course>.Where<Course> (Fu
Filters a sequence of values based on a predicate.

SkipWhile<>

篩選

TakeWhile<>

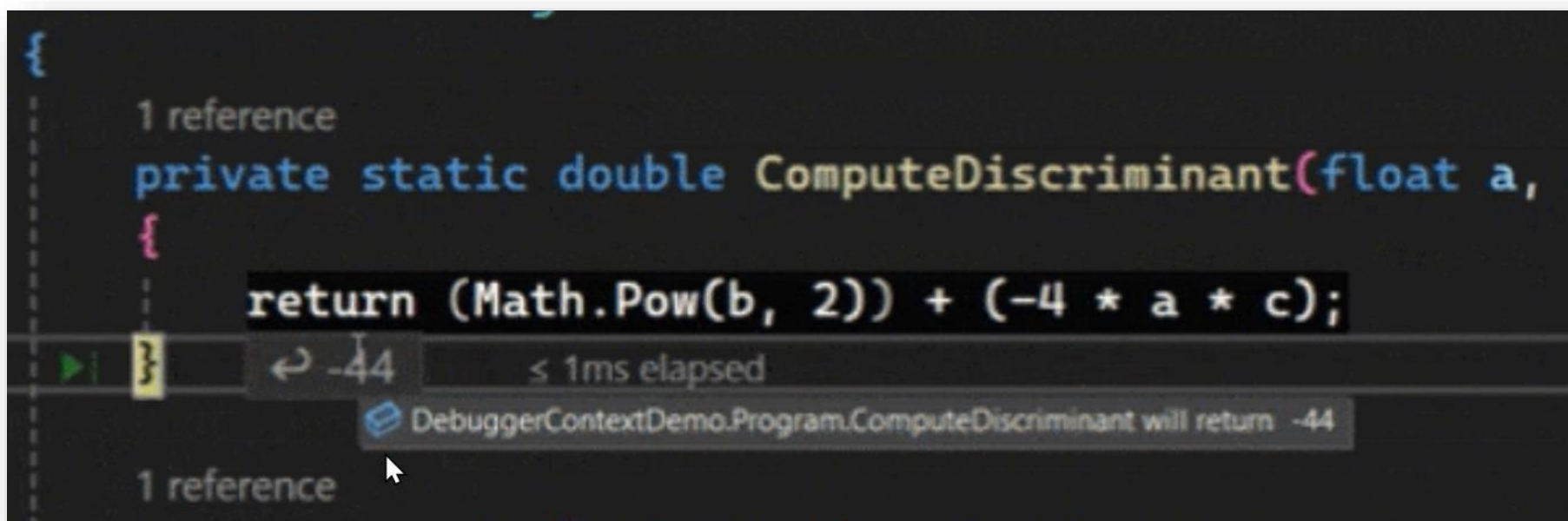
Co	Course.CourseId	Course.Title	Course.Cre
	WebApplication1.Models.Course	12	十項超實用 Excel 技能 — 節省 50% 的工作效率
	WebApplication1.Models.Course	13	Windows Containers 微服務架構實戰

在 JavaScript 和 TypeScript 中支援 Vitest

- 在使用 JavaScript 和 TypeScript 專案時，現在可以使用 [Vitest](#) 撰寫測試案例。
- 在 JavaScript 和 TypeScript 專案 (JSTS)，擁有 **.esproj** 副檔名的專案中，已新增支援發現並執行使用 [Vitest](#) 撰寫的測試。
- 要開始使用 [Vitest](#) 測試
 - 先在 JSTS 專案中使用解決方案總管的 npm 安裝 **vitest** 套件
npm i -D vitest
 - 然後修改專案檔案以宣告以下屬性：
<JavaScriptTestRoot>test\</JavaScriptTestRoot>
<JavaScriptTestFramework>Vitest</JavaScriptTestFramework>
 - 確保 JavaScriptTestRoot 的值具有正確的相對路徑，指向測試檔案所在的位置。
 - 將測試加入正確的資料夾並建構專案/解決方案之後，應該會在**測試總管**中看到測試案例。

更好的偵錯體驗

- 輕鬆匯入和匯出斷點群組
- 使用診斷工具視窗選擇並比較兩個記憶體快照
- 偵錯器 (Debugger) 現在使用 AI 輔助顯示 Inline 的回傳值，以提升效率。

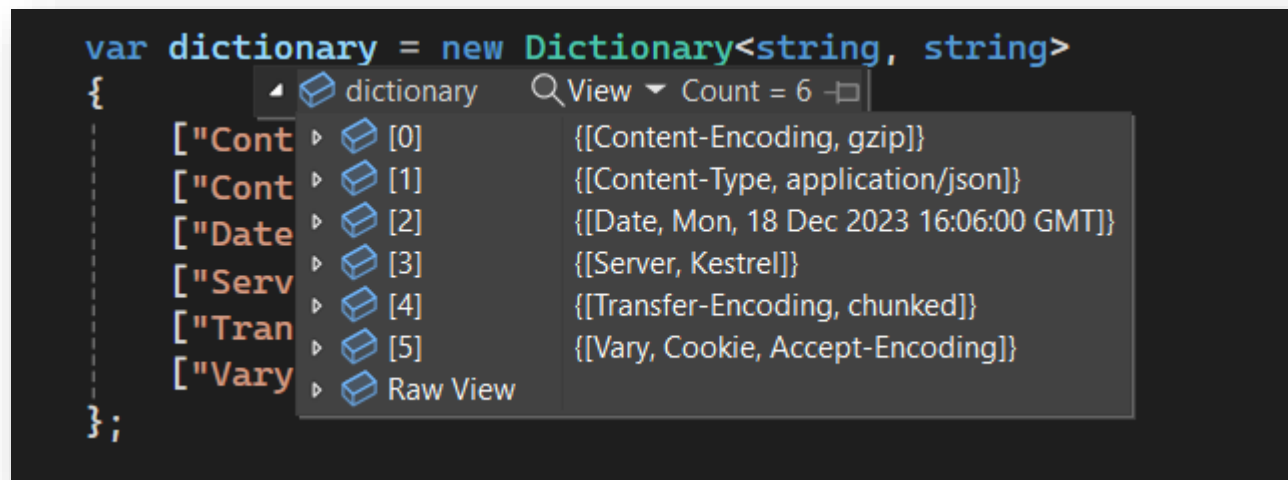


The screenshot shows a code editor with a C# method `ComputeDiscriminant` and its execution in a debugger. The code is as follows:

```
{  
    1 reference  
    private static double ComputeDiscriminant(float a,  
    {  
        return (Math.Pow(b, 2)) + (-4 * a * c);  
    }  
    1 reference
```

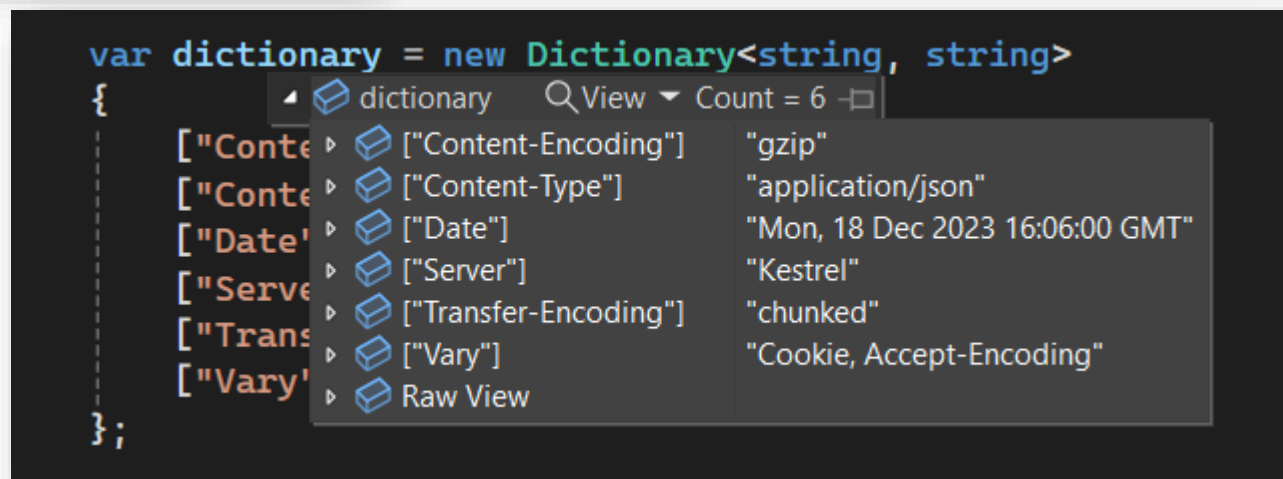
The debugger is paused at the closing brace of the method. A tooltip displays the return value `-44` and the elapsed time `< 1ms elapsed`. Below the tooltip, a message states: `DebuggerContextDemo.Program.ComputeDiscriminant will return -44`. A mouse cursor is pointing at this message.

針對字典與鍵值集合的偵錯畫面 UX 改善



以前

現在



ASP.NET Core 有許多鍵值集合

- HTTP headers
- Query strings
- Forms
- Cookie
- View data
- Route data
- Features

Visual Studio 2022 相關連結

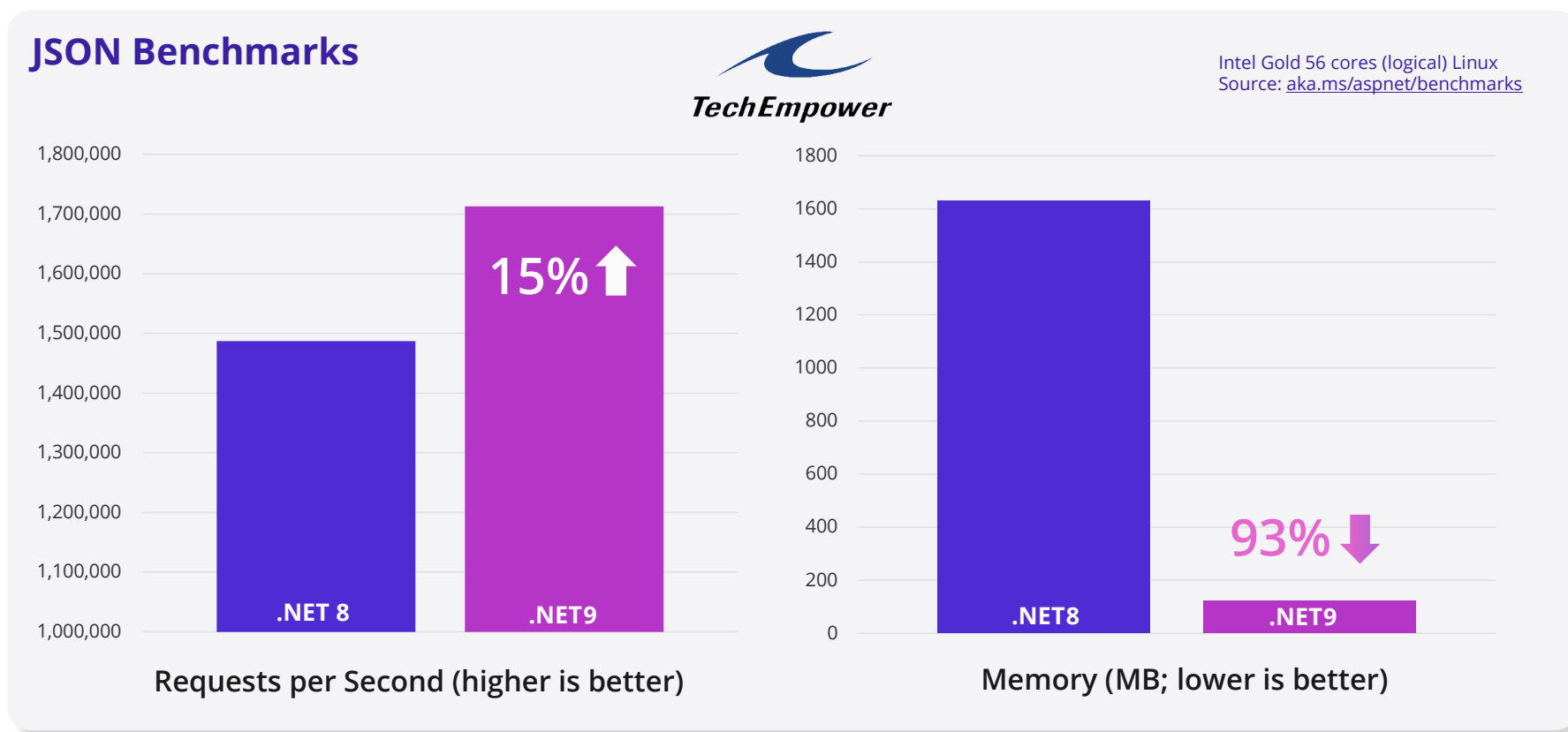
- [Visual Studio 2022 v17.12 with .NET 9](#)
- [Visual Studio 2022 release notes](#)
- [Teams Toolkit Visual Studio Overview](#)
- [.NET default templates for dotnet new](#)
- [.http seems to have an incorrect understanding of JSONPath.](#)



Minimal APIs

.NET 9 Minimal API 效能提升

- TechEmpower 基準測試顯示，.NET 9 在處理量和記憶體使用上表現更好。
- 記憶體使用量的下降是由於伺服器垃圾回收 (Server GC) 的改變所致。



TypedResults 新增兩個型別

- InternalServerError
- InternalServerError<TValue>

```
var app = WebApplication.Create();  
  
app.MapGet("/", () => TypedResults.InternalServerError("Something went wrong!"));  
  
app.Run();
```

路由群組更新兩個擴充方法

- 可在路由群組上呼叫以下兩個擴充方法
 - ProducesProblem
 - ProducesValidationProblem
- 這些方法表示路由群組中的所有端點都可以回傳 **ProblemDetails** 或 **ValidationProblemDetails** 回應，用於 **OpenAPI Metadata** 的目的！

```
var app = WebApplication.Create();  
var todos = app.MapGroup("/todos").ProducesProblem();  
todos.MapGet("/", () => new Todo(1, "Create sample app", false));  
todos.MapPost("/", (Todo todo) => Results.Ok(todo));  
app.Run();  
record Todo(int Id, string Title, boolean IsCompleted);
```


Problem 與 ValidationProblem 支援新型別

- 在 .NET 9 之前，在 Minimal API 中建構 [Problem](#) 和 [ValidationProblem](#) 結果類型，需要將 **errors** 和 **extensions** 屬性初始化為 **IDictionary<string, object?>** 的實作。
- 在 .NET 9 中，這些建構 API 支援接受 **IEnumerable<KeyValuePair<string, object?>>** 的多載！👍

```
var app = WebApplication.Create();
app.MapGet("/", () =>
{
    var extensions = new List<KeyValuePair<string, object?>> { new("test", "value") };
    return TypedResults.Problem("This is an error with extensions",
                                extensions: extensions);
});
```



OpenAPI

內建支援 OpenAPI 文件產生

- [OpenAPI 規格](#)是描述 HTTP API 的標準。
- 這個標準允許開發者定義 API 的形狀，這些 API 可以注入到客戶端產生器、伺服器產生器、測試工具、文件等中。
- 在 .NET 9 中，ASP.NET Core 透過 [Microsoft.AspNetCore.OpenApi](#) 套件提供了內建的支援來產生代表 Controller-based 或 Minimal API 的 OpenAPI 文件！👍
- 只要兩行 Code：
 - 將 **AddOpenApi** 加入到應用程式的 DI 容器中，以註冊所需的相依性。
 - 將 **MapOpenApi** 註冊到應用程式的路由中，以建立所需的 OpenAPI 端點。
- 預設端點
`/openapi/v1.json`
- 詳見 [Microsoft.AspNetCore.OpenApi 文件](#)！

OpenAPI 的範例程式

```
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

// Learn more about configuring OpenAPI at https://aka.ms/aspnet/openapi
builder.Services.AddOpenApi();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.MapOpenApi();
}

app.UseHttpsRedirection();
```

OpenAPI 文件可以在建置時期自動產生

- 安裝 Microsoft.Extensions.ApiDescription.Server 套件
 - dotnet add package Microsoft.Extensions.ApiDescription.Server
- 若要修改發出的 OpenAPI 文件的位置，請在應用程式的專案檔中設定 OpenApiDocumentsDirectory 屬性的目標路徑：

```
<PropertyGroup>
```

```
  <OpenApiDocumentsDirectory>$(MSBuildProjectDirectory)</OpenApiDocumentsDirectory>
```

```
</PropertyGroup>
```

- 預設輸出目錄為專案根目錄，輸出的檔名為 **{專案名稱}.json**
 - 如果專案名稱為 WebApplication1.csproj
 - 那麼輸出檔名為 **WebApplication1.json**

Microsoft.AspNetCore.OpenApi 支援修剪與AOT


- OpenAPI 在 ASP.NET Core 中支援修剪和原生 AOT。
- 以下步驟將建立並發布一個具有修剪和原生 AOT 的 OpenAPI 應用程式：
 - `dotnet new webapiaot`
 - `dotnet add package Microsoft.AspNetCore.OpenApi`
 - 更新 Program.cs
 - + `builder.Services.AddOpenApi();`
 - `var app = builder.Build();`
 - + `app.MapOpenApi();`
 - `dotnet publish`



Static asset delivery optimization

靜態資產傳遞最佳化

全新的 MapStaticAssets 端點

- ASP.NET Core 應用程式中優化靜態資產傳遞的新功能！
 - 關於 Blazor 應用程式的靜態資產傳遞資訊，請參閱 [ASP.NET Core Blazor 靜態檔案](#)。
- 它設計用於所有 UI 框架，包括 **Blazor**、**Razor Pages** 和 **MVC** 等等。
- 可以直接視為 **UseStaticFiles** 的直接替代品！
 - +app.MapStaticAssets();
 - app.UseStaticFiles();
- 使用 **MapStaticAssets** 時，優化會自動進行。當新增或更新函式庫時，例如新增新的 JavaScript 或 CSS，資產會在**建置過程中**進行優化。優化對於低速網路特別有效益！
-  別誤會！這個端點並不是在「**執行時期**」才進行動態壓縮！

MapStaticAssets 的優點

- MapStaticAssets 提供了以下 UseStaticFiles 所沒有的優點：
 - 建置時壓縮應用程式中的所有資產：
 - 在開發時期使用 **gzip**，在發布時使用 **gzip + brotli**。
 - 所有資產都經過壓縮，目標是將資產的大小減少到最小。
 - 基於內容的 **ETags**：每個資源的 **Etags** 是內容的 [SHA-256](#) 雜湊的 [Base64](#) 編碼字串。這確保瀏覽器只有在內容改變時才會重新下載檔案。

檔案	原始	壓縮	減少百分比
bootstrap.min.css	163	17.5	89.26%
jquery.js	89.6	28	68.75%
bootstrap.min.js	78.5	20	74.52%
總計	331.1	65.5	80.20%

使用 MapStaticAssets 端點的注意事項

- **MapStaticAssets** 通過結合建置和發佈時的過程來收集應用程式中所有靜態資源的資訊。這些資訊隨後由 Runtime 的函式庫使用，以有效地將這些檔案提供給瀏覽器。
- **MapStaticAssets** 在大多數情況下可以替代 **UseStaticFiles**
- 使用 **MapStaticAssets** 的時機
 - 當靜態資產可以在**建置**和**發佈**時**已知的資產**就可以用！🔥
- 使用 **UseStaticFiles** 的時機
 - 如果應用程式從**其他位置**提供資產，如**磁碟**或**嵌入式資源**等等。
 - 如果你有些工作要在**建置時期**才會整合其他靜態檔案，就不太適用！
 - 你想在 MSBuild 處理一些檔案操作，就不太適合用 **MapStaticAssets**！



SignalR

SignalR Hubs 中的多型支援

- Hub 方法現在可以接受**基底類別而非衍生類別**，以啟用**多型**情境。
- 基底類型需要透過 [JsonDerivedType] 屬性標註以允許多型。

```
[JsonPolymorphic]
[JsonDerivedType(typeof(JsonPersonExtended), nameof(JsonPersonExtended))]
[JsonDerivedType(typeof(JsonPersonExtended2), nameof(JsonPersonExtended2))]
private class JsonPerson
{
    public string Name { get; set; }
    public Person Child { get; set; }
    public Person Parent { get; set; }
}

private class JsonPersonExtended : JsonPerson
{
    public int Age { get; set; }
}

private class JsonPersonExtended2 : JsonPerson
{
    public string Location { get; set; }
}
```

SignalR 支援 ActivitySource 以提高可觀測性

- SignalR 現在針對 Hub 伺服器端和客戶端都有一個 ActivitySource
 - .NET SignalR 伺服器端 ActivitySource
 - 名為 **Microsoft.AspNetCore.SignalR.Server** 的 SignalR ActivitySource 會發出 Hub 方法呼叫的事件：
- ```
<PackageReference Include="OpenTelemetry.Exporter.OpenTelemetryProtocol" Version="1.9.0" />
<PackageReference Include="OpenTelemetry.Extensions.Hosting" Version="1.9.0" />
<PackageReference Include="OpenTelemetry.Instrumentation.AspNetCore" Version="1.9.0" />
```
- .NET SignalR 客戶端 ActivitySource
    - .NET SignalR 客戶端有一個名為 **Microsoft.AspNetCore.SignalR.Client** 的 ActivitySource。
    - 現在 Hub 呼叫會建立一個客戶端範圍。請注意，其他 SignalR 客戶端，例如 JavaScript 客戶端，不支援追蹤。此功能將在未來的版本中添加到更多客戶端。
    - 客戶端和伺服器上的 Hub 呼叫支援[上下文傳播](#)(context propagation)。傳播追蹤上下文可實現真正的分散式追蹤。現在可以看到呼叫從客戶端流向伺服器並返回。

# 伺服器端要加入以下程式到 Program.cs 文件

```
using OpenTelemetry.Trace;

builder.Services.AddOpenTelemetry()
 .WithTracing(tracing =>
 {
 if (builder.Environment.IsDevelopment())
 {
 // View all traces only in development environment.
 tracing.SetSampler(new AlwaysOnSampler());
 }

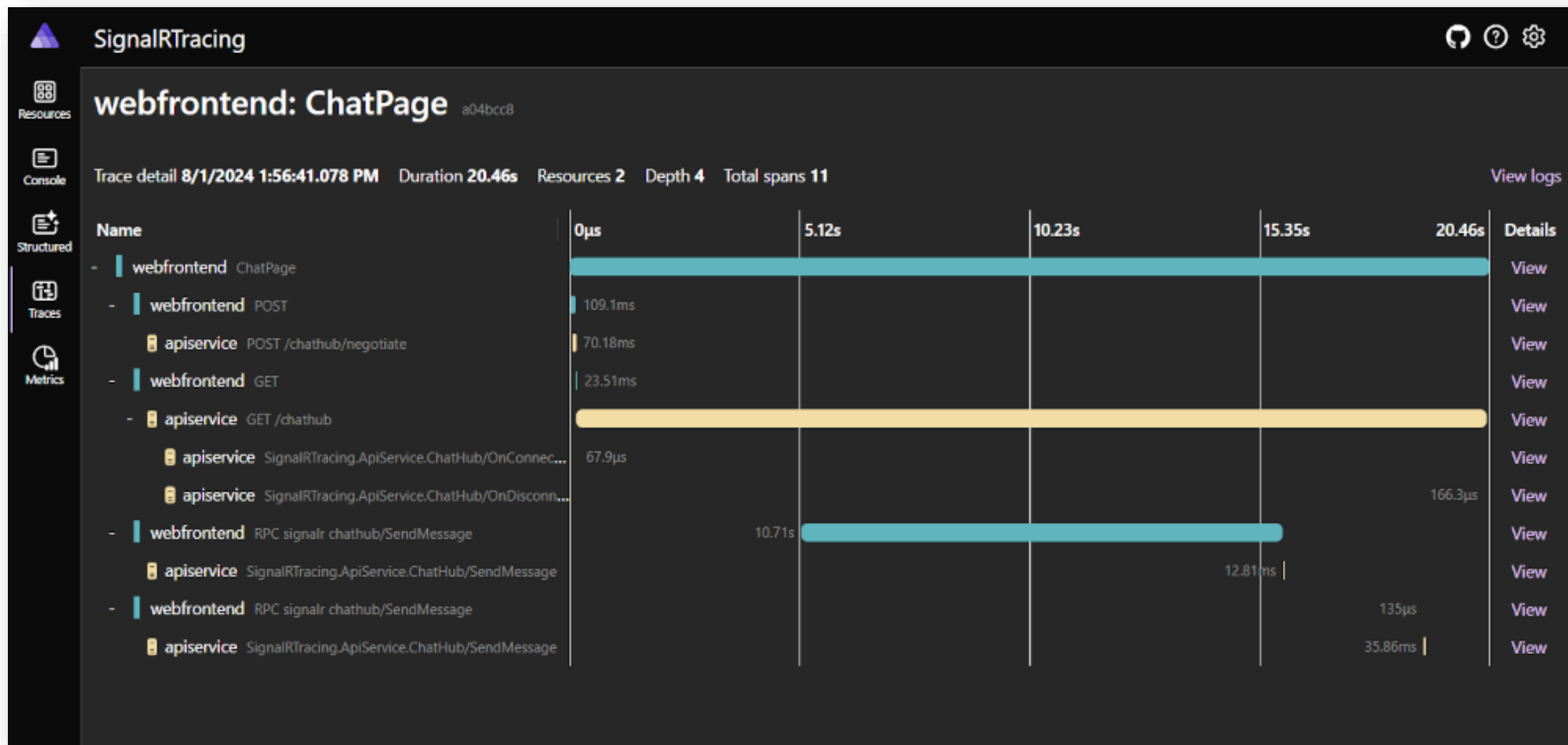
 tracing.AddAspNetCoreInstrumentation();
 tracing.AddSource("Microsoft.AspNetCore.SignalR.Server");
 });

builder.Services.ConfigureOpenTelemetryTracerProvider(tracing => tracing.AddOtlpExporter());
```

# 來自 Aspire 儀表板的範例輸出 (Server)

Timestamp	Name	Spans	Duration
7/5/2024 8:46:56.51...	unknown_service:webApplication27: C/OnDisconnectedAsync e5206a4	unknown_service:webApplication27 (1)	19.0µs
7/5/2024 8:47:18.28...	unknown_service:WebApplication27: GET d3b7fd9	unknown_service:WebApplication27 (1)	80.64ms
7/5/2024 8:47:18.35...	unknown_service:WebApplication27: POST /default/negotiate 6d3ef74	unknown_service:WebApplication27 (1)	10.57ms
7/5/2024 8:47:18.39...	unknown_service:WebApplication27: GET /default 31dd5ac	unknown_service:WebApplication27 (1)	22.18s
7/5/2024 8:47:18.41...	unknown_service:WebApplication27: MyHub/OnConnectedAsync 5ec2ebf	unknown_service:WebApplication27 (1)	25.5µs
7/5/2024 8:47:22.47...	unknown_service:WebApplication27: MyHub/SendMessage e63a6ed	unknown_service:WebApplication27 (1)	7.83ms
7/5/2024 8:47:22.96...	unknown_service:WebApplication27: MyHub/SendMessage c80a8e6	unknown_service:WebApplication27 (1)	155.7µs
7/5/2024 8:47:23.13...	unknown_service:WebApplication27: MyHub/SendMessage 2ebf38b	unknown_service:WebApplication27 (1)	107.1µs
7/5/2024 8:47:40.55...	unknown_service:WebApplication27: GET 0b106a2	unknown_service:WebApplication27 (1)	4.82ms
7/5/2024 8:47:40.56...	unknown_service:WebApplication27: MyHub/OnDisconnectedAsync 7fa3a43	unknown_service:WebApplication27 (1)	16.5µs

# 來自 Aspire 儀表板的範例輸出 (Client)





# SignalR 支援修剪和原生 AOT

- 繼續在 .NET 8 中開始的[原生 AOT 旅程](#)，.NET 9 已經為 SignalR 客戶端和伺服器場景啟用了修剪和原生提前編譯（AOT）支援。
- 您現在可以利用使用 SignalR 進行即時網路通訊的應用程式中的 Native AOT 性能優勢。
- 詳見 [SignalR 支援修剪和原生 AOT](#) 文件。
- 總之，發行 AOT 的可執行檔僅 10MB 左右！👍

# SignalR 在原生 AOT 依然有許多限制

- 目前僅支援 JSON 協議：
  - 如前述程式碼所示，使用 JSON 序列化和 Native AOT 的應用程式必須使用 System.Text.Json 原始碼產生器 (Source Generator)。
  - 這遵循與 Minimal API 相同的做法。
- 在 SignalR 伺服器上，類型為 `IAsyncEnumerable<T>` 和 `ChannelReader<T>` 的 Hub 方法參數，其中 T 是 `ValueType` (`struct`)，不受支援。使用這些類型會在開發過程中啟動應用程式或在發佈的應用程式後導致執行時期異常。欲了解更多資訊，請參閱 [SignalR：在原生 AOT 中使用 IAsyncEnumerable<T> 和 ChannelReader<T> 與 ValueType \(dotnet/aspnetcore #56179\)](#)。
- [強型別的 Hub](#) 與 Native AOT ( `PublishAot` ) 不相容。使用強類型的中樞與 Native AOT 會在建置和發佈時產生警告，並導致運行時異常。使用強類型的中樞與修剪 ( `PublishedTrimmed` ) 是支援的。
- 僅支援 `Task`、`Task<T>`、`ValueTask` 或 `ValueTask<T>` 作為非同步回傳類型。



Authentication and authorization

驗證與授權

# OpenIdConnectHandler 新增 PAR 支援

- **OpenIdConnectHandler** 新增了對 OAuth 2.0 推送授權請求 ( **PAR** ) 的支援
- **PAR** (Pushed Authorization Requests) (推送授權請求) 是 OAuth 2.0 的一項擴充功能 ([RFC 9126](#))，它通過將**授權參數**從**前端通道**移至**後端通道**來提高 OAuth 和 OIDC 流程的安全性。
- 也就是說，將授權參數從瀏覽器中的重定向 URL 移至後端的直接機器對機器的 HTTP 呼叫。它允許客戶端直接將授權請求的內容推送至授權伺服器，換取一個 request\_uri，在隨後的授權請求中引用該 URI。
- 這可以防止瀏覽器中的網路攻擊者：
  - 看到授權參數，這可能會洩露個人身份資訊 ( PII )。
  - 篡改這些參數。例如，網路攻擊者可以更改所請求的訪問範圍。

詳見 [OpenIdConnectHandler adds support for Pushed Authorization Requests \(PAR\)](#)

# 啟用 PAR 的範例

```
builder.Services
 .AddAuthentication(options =>
 {
 options.DefaultScheme = CookieAuthenticationDefaults.AuthenticationScheme;
 options.DefaultChallengeScheme = OpenIdConnectDefaults.AuthenticationScheme;
 })
 .AddCookie()
 .AddOpenIdConnect("oidc", oidcOptions =>
 {
 // Other provider-specific configuration goes here.

 // The default value is PushedAuthorizationBehavior.UseIfAvailable.

 // 'OpenIdConnectOptions' does not contain a definition for 'PushedAuthorizationBehavior'
 // and no accessible extension method 'PushedAuthorizationBehavior' accepting a first argument
 // of type 'OpenIdConnectOptions' could be found
 oidcOptions.PushedAuthorizationBehavior = PushedAuthorizationBehavior.Disable;
 });
```

# OIDC 和 OAuth 參數自定義

- OAuth 和 OIDC 認證處理器現在有一個 `AdditionalAuthorizationParameters` 選項，使得自定義通常作為重定向查詢字串一部分的授權訊息參數變得更加容易。在 .NET 8 及更早版本中，這需要一個自定義的 `OnRedirectToIdentityProvider` 回呼或在自定義處理器中覆寫 `BuildChallengeUrl` 方法。

## .NET 8

```
builder.Services.AddAuthentication().AddOpenIdConnect(options =>
{
 options.Events.OnRedirectToIdentityProvider = context =>
 {
 context.ProtocolMessage.SetParameter("prompt", "login");
 context.ProtocolMessage.SetParameter("audience", "https://api.example.com");
 return Task.CompletedTask;
 };
});
```

## .NET 9

```
builder.Services.AddAuthentication().AddOpenIdConnect(options =>
{
 options.AdditionalAuthorizationParameters.Add("prompt", "login");
 options.AdditionalAuthorizationParameters.Add("audience", "https://api.example.com");
});
```

# 設定 HTTP.sys 擴展驗證旗標

- 您現在可以透過在 HTTP.sys 的 [AuthenticationManager](#) 上使用新的 `EnableKerberosCredentialCaching` 和 `CaptureCredentials` 屬性來配置 [HTTP\\_AUTH\\_EX\\_FLAG\\_ENABLE\\_KERBEROS\\_CREDENTIAL\\_CACHING](#) 和 [HTTP\\_AUTH\\_EX\\_FLAG\\_CAPTURE\\_CREDENTIAL](#) HTTP.sys 旗標，以優化 Windows 驗證的處理方式。例如：

```
webBuilder.UseHttpSys(options =>
{
 options.Authentication.Schemes = AuthenticationSchemes.Negotiate;
 options.Authentication.EnableKerberosCredentialCaching = true;
 options.Authentication.CaptureCredentials = true;
});
```



Miscellaneous

雜項



# 新的 HybridCache 函式庫

- [HybridCache](#) 被設計為現有 [IDistributedCache](#) 和 [IMemoryCache](#) 使用的直接替代品，並提供簡單的 API 來添加新的快取程式碼。它為 **in-process** 和 **out-of-process** 的快取提供了一個統一的 API。
- HybridCache 增加了以下新功能，例如：
  - 雪崩效應保護 (Stampede protection) 以防止同時獲取相同工作
  - 可配置的序列化 (Configurable serialization)
- 注意事項
  - HybridCache 目前仍處於預覽階段，但將在 .NET 9.0 之後的 .NET Extensions 未來小版本中完全發布。
  - 詳見 [New HybridCache library](#) 與 [HybridCache library in ASP.NET Core](#)

# 關於 HybridCache 物件重用的注意事項

- 在使用 **IDistributedCache** 的典型現有程式碼中，每次從快取中取出物件都會導致反序列化。
- 這種行為意味著**每個平行呼叫者**都會得到物件的獨立實例，這些實例無法與其他實例互動。結果是執行緒安全，因為沒有同時修改同一物件實例的風險。
- 因為許多 **HybridCache** 的使用將會從現有的 **IDistributedCache** 程式碼適應過來，**HybridCache** 預設保留此行為以避免引入平行處理錯誤。
- 然而，以下使用案例本質上是執行緒安全的：
  - 如果被快取的類型是不可變的。
  - 如果程式碼不會修改它們。
- 在這種情況下，告知 **HybridCache** 可以安全地重複使用實例的方法是：
  - 將類型標記為 **sealed**。在 C# 中，**sealed** 關鍵字表示該類別**不能被繼承**。
  - 套用 [**ImmutableObject(true)**] 屬性，該屬性表示物件的狀態在建立後不能被改變。
- 通過重複使用實例，HybridCache 可以減少與每次呼叫反序列化相關的 CPU 和物件分配的額外負擔。這在快取的物件很大或經常被存取的情況下，可以帶來效能提升。

# 開發者例外頁面改進

- 當應用程式在開發期間拋出未處理的異常時，會顯示 ASP.NET Core 開發者例外頁面。開發者例外頁面提供關於異常和請求的詳細資訊。
- 開發者例外頁面中新增了 Endpoint Metadata
  - ASP.NET Core 使用端點 Metadata 來控制端點行為，例如路由、回應快取、速率限制、OpenAPI 產生等。
  - 下圖顯示開發者異常頁面中 Routing 區段的新 Metadata 資訊：
- 開發者例外頁面也有一些微小的 UX 提升
  - 更好的文字換行。長的 Cookies、查詢字串值和方法名稱不再會增加水平捲軸
  - 現代設計中發現的更大文字
  - 更一致的表格大小

# 開發者例外頁面改進

**An unhandled exception occurred while processing the request.**

InvalidOperationException: A test error

Program+<>c\_\_DisplayClass0\_0.<<Main>\$>b\_\_0() in Program.cs, line 37

Stack Query Cookies Headers **Routing**

## Endpoint

Name	Value
Display Name	HTTP: GET /weatherforecast
Route Pattern	/weatherforecast
Route Order	0
Route HTTP Method	GET

## Endpoint Metadata

Type	Detail
RuntimeMethodInfo	WeatherForecast[] <<Main>\$>b__0()
HttpMethodMetadata	HttpMethods: GET, Cors: False
ProducesResponseTypeMetadata	Produces StatusCode: 200, ContentTypes: application/json, Type: WeatherForecast[]
EndpointNameMetadata	EndpointName: GetWeatherForecast

## An unhandled exception occurred while processing the request.

Exception: Demonstration exception. The list:  
New Line 1  
New Line 2  
New Line 3

DeveloperExceptionPageSample.Startup+<>c.<Configure>b\_1\_1(HttpContext context) in **Startup.cs**, line 40

**Stack** Query Cookies Headers Routing

### Exception: Demonstration exception. The list: New Line 1 New Line 2 New Line 3

DeveloperExceptionPageSample.Startup+<>c.<Configure>b\_1\_1(HttpContext context) in **Startup.cs**

+ | 40. throw new Exception(string.Concat(

Microsoft.AspNetCore.Diagnostics.DeveloperExceptionPageMiddlewareImpl.Invoke(HttpContext context) in **DeveloperExceptionPageMiddlewareImpl.cs**

+ | 109. await \_next(context);

Show raw exception details

# 修復 IIS 應用程式回收期間的 503 錯誤

- 預設情況下，現在在 IIS 通知回收或關閉後與 ANCM 告知受控伺服器開始關閉之間有 1 秒的延遲。
- 此延遲可透過環境變數 ANCM\_shutdownDelay 或設定 shutdownDelay 處理程序設定來調整。兩者的值均以毫秒為單位。
- 此延遲主要是為了減少競爭條件的可能性，其中：
  - IIS 尚未開始將請求排隊到新應用程式。
  - ANCM 開始拒絕進入舊應用程式的新請求。
- 較慢的機器或 CPU 使用率較高的機器可能需要調整此值以減少 503 的可能性。

# 設定 shutdownDelay 的範例

```
<aspNetCore processPath="dotnet" arguments="myapp.dll" stdoutLogEnabled="false" stdoutLogFile=".logsstdout">
 <handlerSettings>
 <!-- Milliseconds to delay shutdown by.
 this doesn't mean incoming requests will be delayed by this amount,
 but the old app instance will start shutting down after this timeout occurs -->
 <handlerSetting name="shutdownDelay" value="5000" />
 </handlerSettings>
</aspNetCore>
```

# 新增 ASP0026 程式碼分析器規則

- ASP0026 : 當 [Authorize] 被來自「更遠處」的 [AllowAnonymous] 覆蓋時發出警告的分析器
- 看似直覺地將 [Authorize] 屬性放置在比 [AllowAnonymous] 屬性更「靠近」MVC Action 的地方，會覆蓋 [AllowAnonymous] 屬性並強制執行授權。然而，這不一定是這樣。
- 以下圖示為示範最接近 Action 的 [Authorize] 被**較遠的** [AllowAnonymous] 給覆蓋！

```
[AllowAnonymous]
public class MyControllerAnon : ControllerBase
{
}

[Authorize] // Overridden by the [AllowAnonymous] attribute on MyControllerAnon
public class MyControllerInherited : MyControllerAnon
{
}

public class MyControllerInherited2 : MyControllerAnon
{
 [Authorize] // Overridden by the [AllowAnonymous] attribute on MyControllerAnon
 public IActionResult Private() => null;
}
```



# 改善 Kestrel 連線度量 (connection metrics)

- ASP.NET Core 9 對 Kestrel 的連線度量進行了重大改進，現在包含了連線失敗的原因 Metadata。
- **kestrel.connection.duration** 度量現在在 **error.type** 屬性中包含了連線關閉的原因。
- 以下是 error.type 值的一小部分範例：
  - **tls\_handshake\_failed** - 連線需要 TLS，且 TLS 握手失敗。
  - **connection\_reset** - 連線在請求進行時被客戶端意外關閉。
  - **request\_headers\_timeout** - Kestrel 因為沒有及時收到請求標頭而關閉了連線。
  - **max\_request\_body\_size\_exceeded** - Kestrel 因為上傳的資料超過最大尺寸而關閉了連線。
- 以前，診斷 Kestrel 連線問題需要伺服器記錄詳細的低層級日誌。
- 然而，日誌的產生和儲存可能很昂貴，且在雜訊中找到正確的資訊可能很困難。
- 詳見 [Improved Kestrel connection metrics](#) 與 [ASP.NET Core metrics](#)

# 自定義 Kestrel 命名管道端點

- Kestrel 的命名管道支援已透過進階自定義選項得到改善。
- 新的 **CreateNamedPipeServerStream** 方法在命名管道選項上允許每個端點自定義管道。
- 一個這種做法有用的範例是需要兩個不同存取安全性的管道端點的 Kestrel 應用程式。可以使用 **CreateNamedPipeServerStream** 選項來建立具有自定義安全設置的管道，這取決於管道的名稱。

```
var builder = WebApplication.CreateBuilder();

builder.WebHost.ConfigureKestrel(options =>
{
 options.ListenNamedPipe("pipe1");
 options.ListenNamedPipe("pipe2");
});

builder.WebHost.UseNamedPipes(options =>
{
 options.CreateNamedPipeServerStream = (context) =>
 {
 var pipeSecurity = CreatePipeSecurity(context.NamedPipeEndpoint.PipeName);

 return NamedPipeServerStreamAcl.Create(context.NamedPipeEndPoint.PipeName, PipeDirection.InOut,
 NamedPipeServerStream.MaxAllowedServerInstances, PipeTransmissionMode.Byte,
 context.PipeOptions, inBufferSize: 0, outBufferSize: 0, pipeSecurity);
 };
});
```

# ExceptionHandlerMiddleware 強化

- [ExceptionHandlerMiddleware](#) 選項可以根據異常類型選擇狀態碼
- 配置 ExceptionHandlerMiddleware 時新增的選項，允許應用程式開發者在請求處理期間發生異常時選擇返回的狀態碼。
- 新選項會改變 ProblemDetails 回應中由 ExceptionHandlerMiddleware 設置的狀態碼。

```
app.UseExceptionHandler(new ExceptionHandlerOptions
{
 StatusCodeSelector = ex => ex is TimeoutException
 ? StatusCodes.Status503ServiceUnavailable
 : StatusCodes.Status500InternalServerError,
});
```

# HTTP metrics 可以在特定端點退出

- [Opt-out of HTTP metrics on certain endpoints and requests](#)
- .NET 9 引入了在特定端點和請求上選擇退出 HTTP 指標的能力。對於經常被自動化系統呼叫的端點，如健康檢查，退出記錄指標是有益的。對這些請求記錄指標通常是不必要的。
  - 將 [\[DisableHttpMetrics\]](#) 屬性新增到 Web API 控制器、SignalR Hub 或 gRPC 服務
  - 在應用程式啟動時 Map 端點時呼叫 DisableHttpMetrics 擴充方法

```
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddHealthChecks();

var app = builder.Build();
app.MapHealthChecks("/healthz").DisableHttpMetrics();
app.Run();
```

# 刪除金鑰的資料保護支援

- 在 .NET 9 之前，為了防止資料遺失，資料保護金鑰的設計上是不能刪除的。
- 刪除一個金鑰會使其保護的資料無法取回。由於這些金鑰的體積小，其累積通常對系統影響不大。
- 然而，為了適應極長時間運行的服務，我們引入了刪除金鑰的選項。一般來說，應該只刪除舊的金鑰。只有在您能接受為了儲存空間而冒著資料遺失的風險時，才應該刪除金鑰。我們建議不要刪除資料保護金鑰。
- 詳見 [Data Protection support for deleting keys](#)

# Middleware 支援 Keyed DI

- Middleware 現在在建構式和 Invoke/InvokeAsync 方法中都支援 [Keyed DI](#)

```
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddKeyedSingleton<MySingletonClass>("test");

builder.Services.AddKeyedScoped<MyScopedClass>("test2");

var app = builder.Build();
app.UseMiddleware<MyMiddleware>();
app.Run();
```

```
internal class MyMiddleware
{
 private readonly RequestDelegate _next;

 public MyMiddleware(RequestDelegate next,
 [FromKeyedServices("test")] MySingletonClass service)
 {
 _next = next;
 }

 public Task Invoke(HttpContext context,
 [FromKeyedServices("test2")]
 MyScopedClass scopedService) => _next(context);
}
```

# 在 Linux 上信任 ASP.NET Core 的 HTTPS 開發憑證

- 在基於 Ubuntu 和 Fedora 的 Linux 發行版上，**dotnet dev-certs https --trust** 現在將 ASP.NET Core HTTPS 開發憑證配置為受信任的憑證，用於：
  - Chromium 瀏覽器，例如 Google Chrome、Microsoft Edge 和 Chromium。
  - Mozilla Firefox 和 Mozilla 衍生瀏覽器。
  - .NET API，例如，HttpClient
- 之前，**--trust** 僅在 **Windows** 和 **macOS** 上有效。憑證信任是依據使用者套用的。
- 為了在 OpenSSL 中建立信任，dev-certs 工具：
  - 將證書放置在 `~/.aspnet/dev-certs/trust`
  - 在目錄上執行 OpenSSL 的簡化版本 [c\\_rehash](#) 工具，並要求使用者更新 `SSL_CERT_DIR` 環境變數
- 為了在 **dotnet** 中建立信任，工具將憑證放置在 My/Root 證書存儲中。
- 為了在 [NSS 資料庫](#) 中建立信任，如果有，工具會在家目錄中搜尋 Firefox 設定檔，`~/.pki/nssdb`，以及 `~/snap/chromium/current/.pki/nssdb`。對於找到的每個目錄，工具會在 **nssdb** 中新增一個條目。

# 專案範本已更新至最新的前端框架與函式庫

- 範本已更新至最新的 Bootstrap、jQuery 和 jQuery Validation 版本
  - Bootstrap 5.3.3
  - jQuery 3.7.1
  - jQuery Validation 1.21.0





Links

相關連結

# 相關連結

- [What's new in ASP.NET Core 9.0](#)
- [What's New in EF Core 9](#)
- [What's new in .NET 9](#)
- [Breaking changes in .NET 9](#)
- [Announcing .NET 9](#)



# 聯絡資訊

The Will Will Web

網路世界的學習心得與技術分享

<http://blog.miniasp.com>

Facebook

Will 保哥的技術交流中心

<http://www.facebook.com/will.fans>

Twitter

[https://twitter.com/Will\\_Huang](https://twitter.com/Will_Huang)



多奇·教育訓練

**THANK YOU!**

Q&A