What's new in ASP.NET Core 9.0

Article • 04/18/2024

This article highlights the most significant changes in ASP.NET Core 9.0 with links to relevant documentation.

This article has been updated for .NET 9 Preview 3.

Blazor

This section describes new features for Blazor.

Constructor injection

Razor components support constructor injection.

In the following example, the partial (code-behind) class injects the `NavigationManager` service using a primary constructor:

C#

```
public partial class ConstructorInjection(NavigationManager naviga-
tion)
{
    protected NavigationManager Navigation { get; } = navigation;
}
```

For more information, see ASP.NET Core Blazor dependency injection.

Websocket compression for Interactive Server components

By default, Interactive Server components enable compression for WebSocket connections and set a `frame-ancestors` Content Security Policy (CSP) directive set to `'self'`, which only permits embedding the app in an `<iframe>` of the origin from which the app is served when compression is enabled or when a configuration for the WebSocket context is provided.

Compression can be disabled by setting `ConfigureWebSocketOptions` to `null`, which reduces the vulnerability of the app to attack but may result in reduced performance:

```C#
.AddInteractiveServerRenderMode(o => o.ConfigureWebSocketOptions =
null)
```

Configure a stricter `frame-ancestors` CSP with a value of `'none'` (single quotes required), which allows WebSocket compression but prevents browsers from embedding the app into any `<iframe>`:

```C#
.AddInteractiveServerRenderMode(o => o.ContentSecurityFrameAncestors-
Policy = "'none'")
```

For more information, see the following resources:

- ASP.NET Core Blazor SignalR guidance
- Threat mitigation guidance for ASP.NET Core Blazor interactive server-side rendering

Handle keyboard composition events in Blazor

The new `KeyboardEventArgs.IsComposing` property indicates if the keyboard event is part of a composition session. Tracking the composition state of keyboard events is crucial for handling international character input methods.

SignalR

This section describes new features for SignalR.

Polymorphic type support in SignalR Hubs

Hub methods can now accept a base class instead of the derived class to enable polymorphic scenarios. The base type needs to be annotated to allow polymorphism.

```C#
public class MyHub : Hub
{
    public void Method(JsonPerson person)
    {
        if (person is JsonPersonExtended)
        {
```

```
        }
        else if (person is JsonPersonExtended2)
        {
        }
        else
        {
        }
    }
}

[JsonPolymorphic]
[JsonDerivedType(typeof(JsonPersonExtended), nameof(JsonPersonExtend-
ed))]
[JsonDerivedType(typeof(JsonPersonExtended2), nameof(JsonPersonEx-
tended2))]
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; }
}
```

Minimal APIs

This section describes new features for minimal APIs.

Added `InternalServerError` and `InternalServerError<TValue>` to `TypedResults`

The TypedResults class is a helpful vehicle for returning strongly-typed HTTP status code-based responses from a minimal API. `TypedResults` now includes factory methods and types for returning "500 Internal Server Error" responses from endpoints. Here's an example that returns a 500 response:

C#

```
var app = WebApplication.Create();
```

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

app.Run();
```

Authentication and authorization

This section describes new features for authentication and authorization.

OIDC and OAuth Parameter Customization

The OAuth and OIDC authentication handlers now have an `AdditionalAuthorizationParameters` option to make it easier to customize authorization message parameters that are usually included as part of the redirect query string. In .NET 8 and earlier, this requires a custom OnRedirectToIdentityProvider callback or overridden BuildChallengeUrl method in a custom handler. Here's an example of .NET 8 code:

```C#
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;
    };
});
```

The preceding example can now be simplified to the following code:

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

Configure HTTP.sys extended authentication flags

You can now configure the HTTP_AUTH_EX_FLAG_ENABLE_KERBEROS_CREDENTIAL_CACHING and HTTP_AUTH_EX_FLAG_CAPTURE_CREDENTIAL HTTP.sys flags by using the new `EnableKerberosCredentialCaching` and `CaptureCredentials` properties on the HTTP.sys AuthenticationManager to optimize how Windows authentication is handled. For example:

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

Miscellaneous

The following sections describe miscellaneous new features.

Endpoint metadata on the developer exception page

Attributes added to MVC actions, Minimal APIs, and gRPC methods are examples of endpoint metadata. ASP.NET Core uses endpoint metadata to control endpoint behavior, such as routing, authentication and authorization, response caching, rate limiting, OpenAPI generation, and more.

.NET 9 adds metadata to the developer exception page. The new metadata information appears in the `Routing` section alongside other routing information. This information makes it easier to debug ASP.NET Core errors during development. The following image shows the new metadata information on the developer exception page:

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
RouteNameMetadata	RouteName: GetWeatherForecast
OpenApiOperation	Microsoft.OpenApi.Models.OpenApiOperation
RouteDiagnosticsMetadata	Route: /weatherforecast

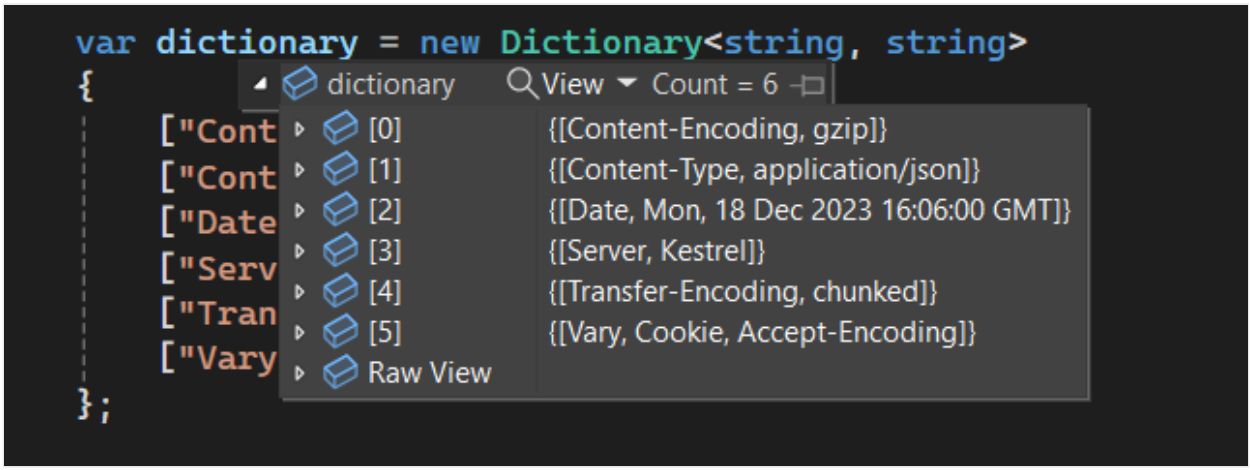
Route Values

No route values.

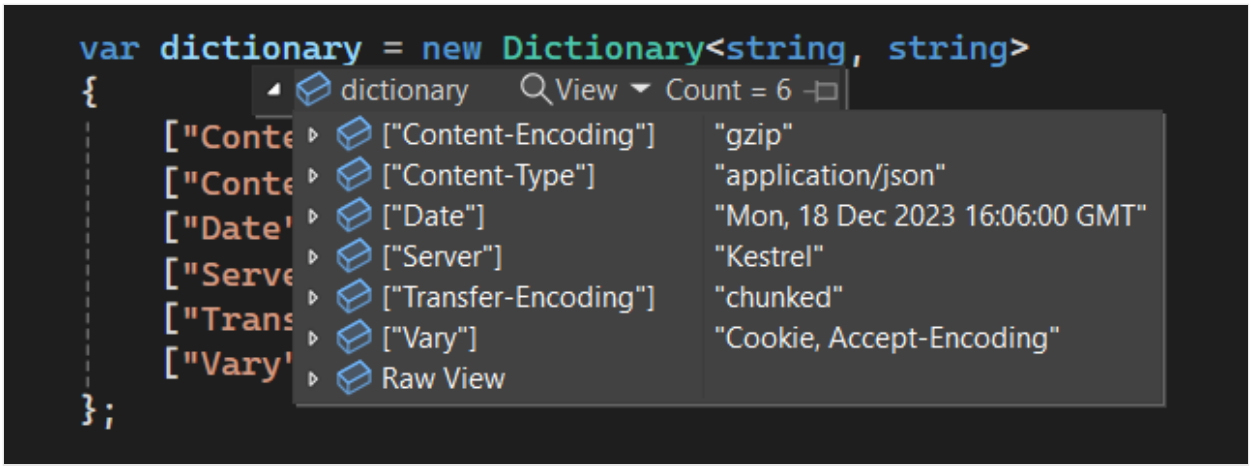
Dictionary debugging improvements

The debugging display of dictionaries and other key-value collections has an improved layout. The key is displayed in the debugger's key column instead of being concatenated with the value. The following images show the old and new display of a dictionary in the debugger.

Before:



After:



ASP.NET Core has many key-value collections. This improved debugging experience applies to:

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

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

ASP.NET Core feedback

ASP.NET Core is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback