Asp.Net Core 全局设计 针对 StatusCode 的捕获：

public class ErrorHandlingFilter : ExceptionFilterAttribute

{

public override void OnException(ExceptionContext context)

{

HandleExceptionAsync(context);

context.ExceptionHandled = true;

}

private static void HandleExceptionAsync(ExceptionContext context)

{

var exception = context.Exception;

if (exception is DivideByZeroException)

SetExceptionResult(context, exception, HttpStatusCode.NotFound);

else if (exception is FileNotFoundException)

SetExceptionResult(context, exception, HttpStatusCode.Unauthorized);

else if (exception is InvalidDataException)

SetExceptionResult(context, exception, HttpStatusCode.BadRequest);

else

SetExceptionResult(context, exception, HttpStatusCode.InternalServerError);

}

private static void SetExceptionResult(

ExceptionContext context,

Exception,

HttpStatusCode code)

{

context.Result = new JsonResult(new { id=200, Code = code})

{

StatusCode = (int)code

};

}

}

public void ConfigureServices(IServiceCollection services)

{

services.AddMvc(

options =>

{

options.Filters.Add(new ErrorHandlingFilter());

}

).SetCompatibilityVersion(CompatibilityVersion.Version\_2\_1);

}

以上是对 Action 内部抛出的异常进行捕获：

也就是捕获发生在Action方法里的异常。 如果是发生在Action方法外的异常则捕获不了，例如

[HttpPost("Register")]

[Authorize]

public IActionResult PostRegister(GTable gtb)

{

this.Init("");

return Ok(this.DB.SaveTable(gtb));

}

这个是不会被捕获的

状态码的捕获机制：

app.UseStatusCodePages(async context =>

{

context.HttpContext.Response.ContentType = "text/plain";

await context.HttpContext.Response.WriteAsync(

"Status code page, status code: " +

context.HttpContext.Response.StatusCode);

});

string OriginalURL = "";

var statusCodeReExecuteFeature = context.Features.Get<IStatusCodeReExecuteFeature>();

if (statusCodeReExecuteFeature != null)

{

OriginalURL =

statusCodeReExecuteFeature.OriginalPathBase

+ statusCodeReExecuteFeature.OriginalPath

+ statusCodeReExecuteFeature.OriginalQueryString;

}

这是对不同的状态吗进行捕获， 注意不会对抛出的异常进行捕获

全局错误的捕获机制：

app.UseExceptionHandler(errorApp =>

{

errorApp.Run(async context =>

{

context.Response.StatusCode = 568;

context.Response.ContentType = "text/html";

await context.Response.WriteAsync("<html lang='en'><body>");

await context.Response.WriteAsync("ERROR!<br><br>");

var exceptionHandlerPathFeature = context.Features.Get<IExceptionHandlerPathFeature>();

// Use exceptionHandlerPathFeature to process the exception (for example,

// logging), but do NOT expose sensitive error information directly to

// the client.

if (exceptionHandlerPathFeature?.Error is Exception)

{

await context.Response.WriteAsync($"File error {exceptionHandlerPathFeature.Error.Message} !<br><br>");

await context.Response.WriteAsync($"Stack Trace: {exceptionHandlerPathFeature.Error.StackTrace} !<br><br>");

await context.Response.WriteAsync($"Path: {exceptionHandlerPathFeature.Path} !<br><br>");

}

await context.Response.WriteAsync("<a href=''>Home</a><br>");

await context.Response.WriteAsync("</body></html>");

await context.Response.WriteAsync(new string(' ', 512)); // IE padding

});

});

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

else

{

app.UseExceptionHandler(errorApp =>

{

errorApp.Run(async context =>

{

context.Response.StatusCode = 500;

context.Response.ContentType = "text/html";

await context.Response.WriteAsync("<html lang=\"en\"><body>\r\n");

await context.Response.WriteAsync("ERROR!<br><br>\r\n");

var exceptionHandlerPathFeature =

context.Features.Get<IExceptionHandlerPathFeature>();

// Use exceptionHandlerPathFeature to process the exception (for example,

// logging), but do NOT expose sensitive error information directly to

// the client.

if (exceptionHandlerPathFeature?.Error is FileNotFoundException)

{

await context.Response.WriteAsync("File error thrown!<br><br>\r\n");

}

await context.Response.WriteAsync("<a href=\"/\">Home</a><br>\r\n");

await context.Response.WriteAsync("</body></html>\r\n");

await context.Response.WriteAsync(new string(' ', 512)); // IE padding

});

});

app.UseHsts();

}

# <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/error-handling?view=aspnetcore-2.2>

# Handle errors in ASP.NET Core

This article covers common approaches to handling errors in ASP.NET Core apps.

[View or download sample code](https://github.com/aspnet/AspNetCore.Docs/tree/master/aspnetcore/fundamentals/error-handling/samples). ([How to download](https://docs.microsoft.com/en-us/aspnet/core/index?view=aspnetcore-2.2#how-to-download-a-sample).) The article includes instructions about how to set preprocessor directives (#if, #endif, #define) in the sample app to enable different scenarios.

## Developer Exception Page

The Developer Exception Page displays detailed information about request exceptions. The page is made available by the [Microsoft.AspNetCore.Diagnostics](https://www.nuget.org/packages/Microsoft.AspNetCore.Diagnostics/) package, which is in the [Microsoft.AspNetCore.App metapackage](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/metapackage-app?view=aspnetcore-2.2). Add code to the Startup.Configure method to enable the page when the app is running in the Development [environment](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/environments?view=aspnetcore-2.2):

C#

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

else

{

app.UseExceptionHandler("/Error");

app.UseHsts();

}

Place the call to [UseDeveloperExceptionPage](https://docs.microsoft.com/dotnet/api/microsoft.aspnetcore.builder.developerexceptionpageextensions.usedeveloperexceptionpage) before any middleware that you want to catch exceptions.

Warning

Enable the Developer Exception Page **only when the app is running in the Development environment**. You don't want to share detailed exception information publicly when the app runs in production. For more information on configuring environments, see [Use multiple environments in ASP.NET Core](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/environments?view=aspnetcore-2.2).

The page includes the following information about the exception and the request:

* Stack trace
* Query string parameters (if any)
* Cookies (if any)
* Headers

To see the Developer Exception Page in the [sample app](https://github.com/aspnet/AspNetCore.Docs/tree/master/aspnetcore/fundamentals/error-handling/samples), use the DevEnvironment preprocessor directive and select **Trigger an exception** on the home page.

## Exception handler page

To configure a custom error handling page for the Production environment, use the Exception Handling Middleware. The middleware:

* Catches and logs exceptions.
* Re-executes the request in an alternate pipeline for the page or controller indicated. The request isn't re-executed if the response has started.

In the following example, [UseExceptionHandler](https://docs.microsoft.com/dotnet/api/microsoft.aspnetcore.builder.exceptionhandlerextensions.useexceptionhandler) adds the Exception Handling Middleware in non-Development environments:

C#

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

else

{

app.UseExceptionHandler("/Error");

app.UseHsts();

}

The Razor Pages app template provides an Error page (.cshtml) and [PageModel](https://docs.microsoft.com/dotnet/api/microsoft.aspnetcore.mvc.razorpages.pagemodel) class (ErrorModel) in the Pages folder. For an MVC app, the project template includes an Error action method and an Error view. Here's the action method:

C#

[AllowAnonymous]

public IActionResult Error()

{

return View(new ErrorViewModel

{ RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });

}

Don't decorate the error handler action method with HTTP method attributes, such as HttpGet. Explicit verbs prevent some requests from reaching the method. Allow anonymous access to the method so that unauthenticated users are able to receive the error view.

### Access the exception

Use [IExceptionHandlerPathFeature](https://docs.microsoft.com/dotnet/api/microsoft.aspnetcore.diagnostics.iexceptionhandlerpathfeature) to access the exception and the original request path in an error handler controller or page:

C#

var exceptionHandlerPathFeature =

HttpContext.Features.Get<IExceptionHandlerPathFeature>();

if (exceptionHandlerPathFeature?.Error is FileNotFoundException)

{

ExceptionMessage = "File error thrown";

}

if (exceptionHandlerPathFeature?.Path == "/index")

{

ExceptionMessage += " from home page";

}

Warning

Do **not** serve sensitive error information to clients. Serving errors is a security risk.

To see the exception handling page in the [sample app](https://github.com/aspnet/AspNetCore.Docs/tree/master/aspnetcore/fundamentals/error-handling/samples), use the ProdEnvironment and ErrorHandlerPage preprocessor directives, and select **Trigger an exception** on the home page.

## Exception handler lambda

An alternative to a [custom exception handler page](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/error-handling?view=aspnetcore-2.2#exception-handler-page) is to provide a lambda to [UseExceptionHandler](https://docs.microsoft.com/dotnet/api/microsoft.aspnetcore.builder.exceptionhandlerextensions.useexceptionhandler). Using a lambda allows access to the error before returning the response.

Here's an example of using a lambda for exception handling:

C#

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

else

{

app.UseExceptionHandler(errorApp =>

{

errorApp.Run(async context =>

{

context.Response.StatusCode = 500;

context.Response.ContentType = "text/html";

await context.Response.WriteAsync("<html lang=\"en\"><body>\r\n");

await context.Response.WriteAsync("ERROR!<br><br>\r\n");

var exceptionHandlerPathFeature =

context.Features.Get<IExceptionHandlerPathFeature>();

// Use exceptionHandlerPathFeature to process the exception (for example,

// logging), but do NOT expose sensitive error information directly to

// the client.

if (exceptionHandlerPathFeature?.Error is FileNotFoundException)

{

await context.Response.WriteAsync("File error thrown!<br><br>\r\n");

}

await context.Response.WriteAsync("<a href=\"/\">Home</a><br>\r\n");

await context.Response.WriteAsync("</body></html>\r\n");

await context.Response.WriteAsync(new string(' ', 512)); // IE padding

});

});

app.UseHsts();

}

Warning

Do **not** serve sensitive error information from [IExceptionHandlerFeature](https://docs.microsoft.com/dotnet/api/microsoft.aspnetcore.diagnostics.iexceptionhandlerfeature) or [IExceptionHandlerPathFeature](https://docs.microsoft.com/dotnet/api/microsoft.aspnetcore.diagnostics.iexceptionhandlerpathfeature) to clients. Serving errors is a security risk.

To see the result of the exception handling lambda in the [sample app](https://github.com/aspnet/AspNetCore.Docs/tree/master/aspnetcore/fundamentals/error-handling/samples), use the ProdEnvironment and ErrorHandlerLambda preprocessor directives, and select **Trigger an exception** on the home page.

## UseStatusCodePages

By default, an ASP.NET Core app doesn't provide a status code page for HTTP status codes, such as 404 - Not Found. The app returns a status code and an empty response body. To provide status code pages, use Status Code Pages middleware.

The middleware is made available by the [Microsoft.AspNetCore.Diagnostics](https://www.nuget.org/packages/Microsoft.AspNetCore.Diagnostics/) package, which is in the [Microsoft.AspNetCore.App metapackage](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/metapackage-app?view=aspnetcore-2.2).

To enable default text-only handlers for common error status codes, call [UseStatusCodePages](https://docs.microsoft.com/dotnet/api/microsoft.aspnetcore.builder.statuscodepagesextensions.usestatuscodepages) in the Startup.Configure method:

C#

app.UseStatusCodePages();

Call UseStatusCodePages before request handling middleware (for example, Static File Middleware and MVC Middleware).

Here's an example of text displayed by the default handlers:

Status Code: 404; Not Found

To see one of the various status code page formats in the [sample app](https://github.com/aspnet/AspNetCore.Docs/tree/master/aspnetcore/fundamentals/error-handling/samples), use one of the preprocessor directives that begin with StatusCodePages, and select **Trigger a 404** on the home page.

## UseStatusCodePages with format string

To customize the response content type and text, use the overload of [UseStatusCodePages](https://docs.microsoft.com/dotnet/api/microsoft.aspnetcore.builder.statuscodepagesextensions.usestatuscodepages) that takes a content type and format string:

C#

app.UseStatusCodePages(

"text/plain", "Status code page, status code: {0}");

## UseStatusCodePages with lambda

To specify custom error-handling and response-writing code, use the overload of [UseStatusCodePages](https://docs.microsoft.com/dotnet/api/microsoft.aspnetcore.builder.statuscodepagesextensions.usestatuscodepages) that takes a lambda expression:

C#

app.UseStatusCodePages(async context =>

{

context.HttpContext.Response.ContentType = "text/plain";

await context.HttpContext.Response.WriteAsync(

"Status code page, status code: " +

context.HttpContext.Response.StatusCode);

});

## UseStatusCodePagesWithRedirect

The [UseStatusCodePagesWithRedirects](https://docs.microsoft.com/dotnet/api/microsoft.aspnetcore.builder.statuscodepagesextensions.usestatuscodepageswithredirects) extension method:

* Sends a 302 - Found status code to the client.
* Redirects the client to the location provided in the URL template.

C#

app.UseStatusCodePagesWithRedirects("/StatusCode?code={0}");

The URL template can include a {0} placeholder for the status code, as shown in the example. If the URL template starts with a tilde (~), the tilde is replaced by the app's PathBase. If you point to an endpoint within the app, create an MVC view or Razor page for the endpoint. For a Razor Pages example, see [StatusCode.cshtml](https://github.com/aspnet/AspNetCore.Docs/tree/master/aspnetcore/fundamentals/error-handling/samples/2.x/Pages/StatusCode.cshtml) in the [sample app](https://github.com/aspnet/AspNetCore.Docs/tree/master/aspnetcore/fundamentals/error-handling/samples).

This method is commonly used when the app:

* Should redirect the client to a different endpoint, usually in cases where a different app processes the error. For web apps, the client's browser address bar reflects the redirected endpoint.
* Shouldn't preserve and return the original status code with the initial redirect response.

## UseStatusCodePagesWithReExecute

The [UseStatusCodePagesWithReExecute](https://docs.microsoft.com/dotnet/api/microsoft.aspnetcore.builder.statuscodepagesextensions.usestatuscodepageswithreexecute) extension method:

* Returns the original status code to the client.
* Generates the response body by re-executing the request pipeline using an alternate path.

C#

app.UseStatusCodePagesWithReExecute("/StatusCode","?code={0}");

If you point to an endpoint within the app, create an MVC view or Razor page for the endpoint. For a Razor Pages example, see [StatusCode.cshtml](https://github.com/aspnet/AspNetCore.Docs/tree/master/aspnetcore/fundamentals/error-handling/samples/2.x/Pages/StatusCode.cshtml) in the [sample app](https://github.com/aspnet/AspNetCore.Docs/tree/master/aspnetcore/fundamentals/error-handling/samples).

This method is commonly used when the app should:

* Process the request without redirecting to a different endpoint. For web apps, the client's browser address bar reflects the originally requested endpoint.
* Preserve and return the original status code with the response.

The URL and query string templates may include a placeholder ({0}) for the status code. The URL template must start with a slash (/). When using a placeholder in the path, confirm that the endpoint (page or controller) can process the path segment. For example, a Razor Page for errors should accept the optional path segment value with the @page directive:

CSHTML

@page "{code?}"

The endpoint that processes the error can get the original URL that generated the error, as shown in the following example:

C#

var statusCodeReExecuteFeature = HttpContext.Features.Get<IStatusCodeReExecuteFeature>();

if (statusCodeReExecuteFeature != null)

{

OriginalURL =

statusCodeReExecuteFeature.OriginalPathBase

+ statusCodeReExecuteFeature.OriginalPath

+ statusCodeReExecuteFeature.OriginalQueryString;

}

## Disable status code pages

Status code pages can be disabled for specific requests in a Razor Pages handler method or in an MVC controller. To disable status code pages, use the [IStatusCodePagesFeature](https://docs.microsoft.com/dotnet/api/microsoft.aspnetcore.diagnostics.istatuscodepagesfeature):

C#

var statusCodePagesFeature = HttpContext.Features.Get<IStatusCodePagesFeature>();

if (statusCodePagesFeature != null)

{

statusCodePagesFeature.Enabled = false;

}

## Exception-handling code

Code in exception handling pages can throw exceptions. It's often a good idea for production error pages to consist of purely static content.

### Response headers

Once the headers for a response are sent:

* The app can't change the response's status code.
* Any exception pages or handlers can't run. The response must be completed or the connection aborted.

## Server exception handling

In addition to the exception handling logic in your app, the [HTTP server implementation](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/servers/index?view=aspnetcore-2.2) can handle some exceptions. If the server catches an exception before response headers are sent, the server sends a 500 - Internal Server Error response without a response body. If the server catches an exception after response headers are sent, the server closes the connection. Requests that aren't handled by your app are handled by the server. Any exception that occurs when the server is handling the request is handled by the server's exception handling. The app's custom error pages, exception handling middleware, and filters don't affect this behavior.

## Startup exception handling

Only the hosting layer can handle exceptions that take place during app startup. The host can be configured to [capture startup errors](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/host/web-host?view=aspnetcore-2.2#capture-startup-errors) and [capture detailed errors](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/host/web-host?view=aspnetcore-2.2#detailed-errors).

The hosting layer can show an error page for a captured startup error only if the error occurs after host address/port binding. If binding fails:

* The hosting layer logs a critical exception.
* The dotnet process crashes.
* No error page is displayed when the HTTP server is [Kestrel](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/servers/kestrel?view=aspnetcore-2.2).

When running on [IIS](https://docs.microsoft.com/en-us/iis) or [IIS Express](https://docs.microsoft.com/en-us/iis/extensions/introduction-to-iis-express/iis-express-overview), a 502.5 - Process Failure is returned by the [ASP.NET Core Module](https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/aspnet-core-module?view=aspnetcore-2.2) if the process can't start. For more information, see [Troubleshoot ASP.NET Core on IIS](https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/iis/troubleshoot?view=aspnetcore-2.2). For information on troubleshooting startup issues with Azure App Service, see [Troubleshoot ASP.NET Core on Azure App Service](https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/azure-apps/troubleshoot?view=aspnetcore-2.2).

## Database error page

The [Database Error Page](https://docs.microsoft.com/dotnet/api/microsoft.aspnetcore.builder.databaseerrorpageextensions.usedatabaseerrorpage) middleware captures database-related exceptions that can be resolved by using Entity Framework migrations. When these exceptions occur, an HTML response with details of possible actions to resolve the issue is generated. This page should be enabled only in the Development environment. Enable the page by adding code to Startup.Configure:

C#

if (env.IsDevelopment())

{

app.UseDatabaseErrorPage();

}

## Exception filters

In MVC apps, exception filters can be configured globally or on a per-controller or per-action basis. In Razor Pages apps, they can be configured globally or per page model. These filters handle any unhandled exception that occurs during the execution of a controller action or another filter. For more information, see [Filters in ASP.NET Core](https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/filters?view=aspnetcore-2.2#exception-filters).

Tip

Exception filters are useful for trapping exceptions that occur within MVC actions, but they're not as flexible as the Exception Handling Middleware. We recommend using the middleware. Use filters only where you need to perform error handling differently based on which MVC action is chosen.

## Model state errors

For information about how to handle model state errors, see [Model binding](https://docs.microsoft.com/en-us/aspnet/core/mvc/models/model-binding?view=aspnetcore-2.2) and [Model validation](https://docs.microsoft.com/en-us/aspnet/core/mvc/models/validation?view=aspnetcore-2.2).

## Additional resources

* [Common errors reference for Azure App Service and IIS with ASP.NET Core](https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/azure-iis-errors-reference?view=aspnetcore-2.2)
* [Troubleshoot ASP.NET Core on IIS](https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/iis/troubleshoot?view=aspnetcore-2.2)
* [Troubleshoot ASP.NET Core on Azure App Service](https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/azure-apps/troubleshoot?view=aspnetcore-2.2)

<https://stackoverflow.com/questions/38630076/asp-net-core-web-api-exception-handling>

I started using ASP.NET Core for my new REST API project after using regular ASP.NET Web API for many years. I don't see a good way to handle exceptions in ASP.NET Core Web API. I tried to implement exception handling filter/attribute:

public class ErrorHandlingFilter : ExceptionFilterAttribute

{

public override void OnException(ExceptionContext context)

{

HandleExceptionAsync(context);

context.ExceptionHandled = true;

}

private static void HandleExceptionAsync(ExceptionContext context)

{

var exception = context.Exception;

if (exception is MyNotFoundException)

SetExceptionResult(context, exception, HttpStatusCode.NotFound);

else if (exception is MyUnauthorizedException)

SetExceptionResult(context, exception, HttpStatusCode.Unauthorized);

else if (exception is MyException)

SetExceptionResult(context, exception, HttpStatusCode.BadRequest);

else

SetExceptionResult(context, exception, HttpStatusCode.InternalServerError);

}

private static void SetExceptionResult(

ExceptionContext context,

Exception exception,

HttpStatusCode code)

{

context.Result = new JsonResult(new ApiResponse(exception))

{

StatusCode = (int)code

};

}

}

And here is my Startup filter registration:

services.AddMvc(options =>

{

options.Filters.Add(new AuthorizationFilter());

options.Filters.Add(new ErrorHandlingFilter());

});

The issue I was having is that when exception occurres in my AuthorizationFilter it's not being handled by ErrorHandlingFilter. I was expecting it to be caught there just like it worked with old ASP.NET Web API.

So how can I catch all application exceptions as well as any exceptions from Action Filters?

[c#](https://stackoverflow.com/questions/tagged/c%23) [exception](https://stackoverflow.com/questions/tagged/exception) [asp.net-core](https://stackoverflow.com/questions/tagged/asp.net-core)

[share](https://stackoverflow.com/q/38630076)[improve this question](https://stackoverflow.com/posts/38630076/edit)

[edited Mar 14 at 23:09](https://stackoverflow.com/posts/38630076/revisions)

[[https://i.stack.imgur.com/csomC.jpg?s=32&g=1](https://stackoverflow.com/users/2141621/camilo-terevinto)](https://stackoverflow.com/users/2141621/camilo-terevinto)

[Camilo Terevinto](https://stackoverflow.com/users/2141621/camilo-terevinto)

**20.3k**64070

asked Jul 28 '16 at 7:44

[[https://i.stack.imgur.com/rHVf5.jpg?s=32&g=1](https://stackoverflow.com/users/2061604/andrei)](https://stackoverflow.com/users/2061604/andrei)

[Andrei](https://stackoverflow.com/users/2061604/andrei)

**22.8k**27108160

* 1

Have you tried UseExceptionHandler middleware? – [Pawel](https://stackoverflow.com/users/1168070/pawel) [Aug 3 '16 at 9:54](https://stackoverflow.com/questions/38630076/asp-net-core-web-api-exception-handling#comment64854633_38630076)

* I have an example [here](https://stackoverflow.com/a/55166404/1671558) on how to use UseExceptionHandler middleware – [Ilya Chernomordik](https://stackoverflow.com/users/1671558/ilya-chernomordik) [Mar 24 at 10:54](https://stackoverflow.com/questions/38630076/asp-net-core-web-api-exception-handling#comment97372261_38630076)

add a comment

7 Answers

[active](https://stackoverflow.com/questions/38630076/asp-net-core-web-api-exception-handling?answertab=active#tab-top)[oldest](https://stackoverflow.com/questions/38630076/asp-net-core-web-api-exception-handling?answertab=oldest#tab-top)[votes](https://stackoverflow.com/questions/38630076/asp-net-core-web-api-exception-handling?answertab=votes#tab-top)

367

Exception Handling Middleware

After many experiments with different exception handling approaches I ended up using middleware. It worked the best for my ASP.NET Core Web API application. It handles application exceptions as well as exceptions from filters and I have full control over exception handling and creating a response json. Here is my exception handling middleware:

public class ErrorHandlingMiddleware

{

private readonly RequestDelegate next;

public ErrorHandlingMiddleware(RequestDelegate next)

{

this.next = next;

}

public async Task Invoke(HttpContext context /\* other dependencies \*/)

{

try

{

await next(context);

}

catch (Exception ex)

{

await HandleExceptionAsync(context, ex);

}

}

private static Task HandleExceptionAsync(HttpContext context, Exception ex)

{

var code = HttpStatusCode.InternalServerError; // 500 if unexpected

if (ex is MyNotFoundException) code = HttpStatusCode.NotFound;

else if (ex is MyUnauthorizedException) code = HttpStatusCode.Unauthorized;

else if (ex is MyException) code = HttpStatusCode.BadRequest;

var result = JsonConvert.SerializeObject(new { error = ex.Message });

context.Response.ContentType = "application/json";

context.Response.StatusCode = (int)code;

return context.Response.WriteAsync(result);

}

}

Register it **before MVC** in Startup class:

app.UseMiddleware(typeof(ErrorHandlingMiddleware));

app.UseMvc();

You can add stack trace, exception type name, error codes or anything you want to it. Very flexible. Here is an example of exception response:

{ "error": "Authentication token is not valid." }

Consider injecting IOptions<MvcJsonOptions> to the Invoke method to then use it when you serialize the response object to utilize ASP.NET MVC's serialization settings in JsonConvert.SerializeObject(errorObj, opts.Value.SerializerSettings) for better serialization consistency across all endpoints.

Approach 2

There is another API called UseExceptionHandler that makes life a little easier:

app.UseExceptionHandler(a => a.Run(async context =>

{

var feature = context.Features.Get<IExceptionHandlerPathFeature>();

var exception = feature.Error;

var result = JsonConvert.SerializeObject(new { error = exception.Message });

context.Response.ContentType = "application/json";

await context.Response.WriteAsync(result);

}));

This is not a very obvious but easy way to set up exception handling. However I still prefer the middleware approach over it as I get more control with ability to inject necessary dependencies.

[share](https://stackoverflow.com/a/38935583)[improve this answer](https://stackoverflow.com/posts/38935583/edit)

[edited Mar 22 at 19:46](https://stackoverflow.com/posts/38935583/revisions)

answered Aug 13 '16 at 18:08

[[https://i.stack.imgur.com/rHVf5.jpg?s=32&g=1](https://stackoverflow.com/users/2061604/andrei)](https://stackoverflow.com/users/2061604/andrei)

[Andrei](https://stackoverflow.com/users/2061604/andrei)

**22.8k**27108160

* 3

I have been beating my head against the desk trying to get a custom middleware to work today, and it works basically the same way (I'm using it to manage unit of work/transaction for a request). The problem I'm facing is that raised exceptions in 'next' are not caught in the middleware. As you can imagine, this is problematic. What am I doing wrong/missing? Any pointers or suggestions? – [brappleye3](https://stackoverflow.com/users/680876/brappleye3) [Feb 17 '17 at 2:43](https://stackoverflow.com/questions/38630076/asp-net-core-web-api-exception-handling#comment71733320_38935583)

* 2

I typically do a mix of both middleware and IExceptionFilter. The filter handles the controller errors directly, and I use the middleware for a more "low level" handling. As a hint, if someone needs to execute code by exception type in the global handler, to make it more "readable" feel free to give a look at a small library I made just for that: [medium.com/@nogravity00/…](https://medium.com/@nogravity00/asp-net-core-mvc-and-exception-handling-f0da1c820d4a) – [João Simões](https://stackoverflow.com/users/1841558/jo%c3%a3o-sim%c3%b5es) [Feb 17 '17 at 19:35](https://stackoverflow.com/questions/38630076/asp-net-core-web-api-exception-handling#comment71766884_38935583)

* 4

@brappleye3 - I figured out what the problem was. I was just registering the middleware in the wrong place in the Startup.cs class. I moved app.UseMiddleware<ErrorHandlingMiddleware>(); to just before app.UseStaticFiles();. The exception seems to be caught correctly now. This leads me to believe app.UseDeveloperExceptionPage(); app.UseDatabaseErrorPage(); app.UseBrowserLink(); Do some internal magic middleware hackery to get the middleware ordering right. – [DarthJam](https://stackoverflow.com/users/2018591/darthjam" \o "716 reputation) [Mar 28 '17 at 10:36](https://stackoverflow.com/questions/38630076/asp-net-core-web-api-exception-handling#comment73217920_38935583)

* 4

I agree that custom middleware can be very useful but would question using exceptions for NotFound, Unauthorised and BadRequest situations. Why not simply set the status code (using NotFound() etc.) and then handle it in your custom middleware or via UseStatusCodePagesWithReExecute? See [devtrends.co.uk/blog/handling-errors-in-asp.net-core-web-api](https://www.devtrends.co.uk/blog/handling-errors-in-asp.net-core-web-api) for more info – [Paul Hiles](https://stackoverflow.com/users/246396/paul-hiles) [May 25 '17 at 19:35](https://stackoverflow.com/questions/38630076/asp-net-core-web-api-exception-handling#comment75390772_38935583)

* 3

It's bad because it's always serializing to JSON, completely ignoring content negotiation. – [Konrad](https://stackoverflow.com/users/2828480/konrad) [Sep 18 '18 at 10:14](https://stackoverflow.com/questions/38630076/asp-net-core-web-api-exception-handling#comment91712450_38935583)

[show **31** more comments](https://stackoverflow.com/questions/38630076/asp-net-core-web-api-exception-handling)

24

Your best bet is to use Middleware to achieve logging you're looking for. You want to put your exception logging in one middleware and then handle your error pages displayed to the user in a different middleware. That allows separation of logic and follows the design Microsoft has laid out with the 2 middleware componenets. Here's a good link to Microsoft's documentation: [Error Handling in ASP.Net Core](https://docs.asp.net/en/latest/fundamentals/error-handling.html)

For your specific example, you may want to use one of the extensions in the [StatusCodePage middleware](https://github.com/aspnet/Diagnostics/blob/dev/src/Microsoft.AspNetCore.Diagnostics/StatusCodePage/StatusCodePagesMiddleware.cs) or roll your own like [this](https://stackoverflow.com/a/38810819/4023606).

You can find an example here for logging exceptions: [ExceptionHandlerMiddleware.cs](https://github.com/aspnet/Diagnostics/blob/dev/src/Microsoft.AspNetCore.Diagnostics/ExceptionHandler/ExceptionHandlerMiddleware.cs)

public void Configure(IApplicationBuilder app)

{

// app.UseErrorPage(ErrorPageOptions.ShowAll);

// app.UseStatusCodePages();

// app.UseStatusCodePages(context => context.HttpContext.Response.SendAsync("Handler, status code: " + context.HttpContext.Response.StatusCode, "text/plain"));

// app.UseStatusCodePages("text/plain", "Response, status code: {0}");

// app.UseStatusCodePagesWithRedirects("~/errors/{0}");

// app.UseStatusCodePagesWithRedirects("/base/errors/{0}");

// app.UseStatusCodePages(builder => builder.UseWelcomePage());

app.UseStatusCodePagesWithReExecute("/Errors/{0}"); // I use this version

// Exception handling logging below

app.UseExceptionHandler();

}

If you don't like that specific implementation, then you can also use [ELM Middleware](https://github.com/aspnet/Diagnostics/blob/dev/src/Microsoft.AspNetCore.Diagnostics.Elm/ElmCaptureMiddleware.cs), and here are some examples: [Elm Exception Middleware](http://www.talkingdotnet.com/aspnet-core-diagnostics-middleware-error-handling/)

public void Configure(IApplicationBuilder app)

{

app.UseStatusCodePagesWithReExecute("/Errors/{0}");

// Exception handling logging below

app.UseElmCapture();

app.UseElmPage();

}

If that doesn't work for your needs, you can always roll your own Middleware component by looking at their implementations of the ExceptionHandlerMiddleware and the ElmMiddleware to grasp the concepts for building your own.

It's important to add the exception handling middleware below the StatusCodePages middleware but above all your other middleware components. That way your Exception middleware will capture the exception, log it, then allow the request to proceed to the StatusCodePage middleware which will display the friendly error page to the user.

[share](https://stackoverflow.com/a/38918481)[improve this answer](https://stackoverflow.com/posts/38918481/edit)

[edited May 23 '17 at 11:47](https://stackoverflow.com/posts/38918481/revisions)

[[https://www.gravatar.com/avatar/a007be5a61f6aa8f3e85ae2fc18dd66e?s=32&d=identicon&r=PG](https://stackoverflow.com/users/-1/community)](https://stackoverflow.com/users/-1/community)

[Community](https://stackoverflow.com/users/-1/community)**♦**

**1**1

answered Aug 12 '16 at 12:53

[[https://i.stack.imgur.com/qVsEJ.jpg?s=32&g=1](https://stackoverflow.com/users/4023606/ashley-lee)](https://stackoverflow.com/users/4023606/ashley-lee)

[Ashley Lee](https://stackoverflow.com/users/4023606/ashley-lee)

**2,755**11326

* You're welcome. I also provided a link to an example for overriding the default UseStatusPages on edge cases that may better meet your request. – [Ashley Lee](https://stackoverflow.com/users/4023606/ashley-lee) [Aug 12 '16 at 13:19](https://stackoverflow.com/questions/38630076/asp-net-core-web-api-exception-handling#comment65194795_38918481)
* 1

Note that Elm doesn't persist the logs, and it's recommended to use Serilog or NLog to provide the serialization. See [ELM logs disappears. Can we persist it to a file or DB?](https://github.com/aspnet/Diagnostics/issues/328) – [Michael Freidgeim](https://stackoverflow.com/users/52277/michael-freidgeim) [Feb 21 '17 at 12:59](https://stackoverflow.com/questions/38630076/asp-net-core-web-api-exception-handling#comment71885719_38918481)

add a comment

14

To Configure exception handling behavior per exception type you can use Middleware from NuGet packages:

* [Community.AspNetCore.ExceptionHandling.NewtonsoftJson](https://www.nuget.org/packages/Community.AspNetCore.ExceptionHandling.NewtonsoftJson)for ASP.NET Core 2.0
* [Community.AspNetCore.ExceptionHandling.Mvc](https://www.nuget.org/packages/Community.AspNetCore.ExceptionHandling.Mvc) for ASP.NET Core 2.1+.

Code sample:

public void ConfigureServices(IServiceCollection services)

{

services.AddMvc();

services.AddExceptionHandlingPolicies(options =>

{

options.For<InitializationException>().Rethrow();

options.For<SomeTransientException>().Retry(ro => ro.MaxRetryCount = 2).NextPolicy();

options.For<SomeBadRequestException>()

.Response(e => 400)

.Headers((h, e) => h["X-MyCustomHeader"] = e.Message)

.WithBody((req,sw, exception) =>

{

byte[] array = Encoding.UTF8.GetBytes(exception.ToString());

return sw.WriteAsync(array, 0, array.Length);

})

.NextPolicy();

// Ensure that all exception types are handled by adding handler for generic exception at the end.

options.For<Exception>()

.Log(lo =>

{

lo.EventIdFactory = (c, e) => new EventId(123, "UnhandlerException");

lo.Category = (context, exception) => "MyCategory";

})

.Response(null, ResponseAlreadyStartedBehaviour.GoToNextHandler)

.ClearCacheHeaders()

.WithObjectResult((r, e) => new { msg = e.Message, path = r.Path })

.Handled();

});

}

public void Configure(IApplicationBuilder app, IHostingEnvironment env)

{

app.UseExceptionHandlingPolicies();

app.UseMvc();

}

[share](https://stackoverflow.com/a/51430291)[improve this answer](https://stackoverflow.com/posts/51430291/edit)

answered Jul 19 '18 at 19:15

[[https://www.gravatar.com/avatar/d099e21f9756006ee9d5d6fb62223ea3?s=32&d=identicon&r=PG&f=1](https://stackoverflow.com/users/9252793/ihar-yakimush)](https://stackoverflow.com/users/9252793/ihar-yakimush)

[Ihar Yakimush](https://stackoverflow.com/users/9252793/ihar-yakimush)

**352**25

add a comment

12

Firstly, thanks to Andrei as I've based my solution on his example.

I'm including mine as it's a more complete sample and might save readers some time.

The limitation of Andrei's approach is that doesn't handle logging, capturing potentially useful request variables and content negotiation (it will always return JSON no matter what the client has requested - XML / plain text etc).

My approach is to use an ObjectResult which allows us to use the functionality baked into MVC.

This code also prevents caching of the response.

The error response has been decorated in such a way that it can be serialized by the XML serializer.

public class ExceptionHandlerMiddleware

{

private readonly RequestDelegate next;

private readonly IActionResultExecutor<ObjectResult> executor;

private readonly ILogger logger;

private static readonly ActionDescriptor EmptyActionDescriptor = new ActionDescriptor();

public ExceptionHandlerMiddleware(RequestDelegate next, IActionResultExecutor<ObjectResult> executor, ILoggerFactory loggerFactory)

{

this.next = next;

this.executor = executor;

logger = loggerFactory.CreateLogger<ExceptionHandlerMiddleware>();

}

public async Task Invoke(HttpContext context)

{

try

{

await next(context);

}

catch (Exception ex)

{

logger.LogError(ex, $"An unhandled exception has occurred while executing the request. Url: {context.Request.GetDisplayUrl()}. Request Data: " + GetRequestData(context));

if (context.Response.HasStarted)

{

throw;

}

var routeData = context.GetRouteData() ?? new RouteData();

ClearCacheHeaders(context.Response);

var actionContext = new ActionContext(context, routeData, EmptyActionDescriptor);

var result = new ObjectResult(new ErrorResponse("Error processing request. Server error."))

{

StatusCode = (int) HttpStatusCode.InternalServerError,

};

await executor.ExecuteAsync(actionContext, result);

}

}

private static string GetRequestData(HttpContext context)

{

var sb = new StringBuilder();

if (context.Request.HasFormContentType && context.Request.Form.Any())

{

sb.Append("Form variables:");

foreach (var x in context.Request.Form)

{

sb.AppendFormat("Key={0}, Value={1}<br/>", x.Key, x.Value);

}

}

sb.AppendLine("Method: " + context.Request.Method);

return sb.ToString();

}

private static void ClearCacheHeaders(HttpResponse response)

{

response.Headers[HeaderNames.CacheControl] = "no-cache";

response.Headers[HeaderNames.Pragma] = "no-cache";

response.Headers[HeaderNames.Expires] = "-1";

response.Headers.Remove(HeaderNames.ETag);

}

[DataContract(Name= "ErrorResponse")]

public class ErrorResponse

{

[DataMember(Name = "Message")]

public string Message { get; set; }

public ErrorResponse(string message)

{

Message = message;

}

}

}

[share](https://stackoverflow.com/a/51847739)[improve this answer](https://stackoverflow.com/posts/51847739/edit)

[edited Sep 11 '18 at 15:18](https://stackoverflow.com/posts/51847739/revisions)

answered Aug 14 '18 at 18:30

[[https://i.stack.imgur.com/AF7O0.jpg?s=32&g=1](https://stackoverflow.com/users/74178/countzero)](https://stackoverflow.com/users/74178/countzero)

[CountZero](https://stackoverflow.com/users/74178/countzero)

**3,585**33649

* 100% agree, no content negotiation is limited and not fully working approach – [Konrad](https://stackoverflow.com/users/2828480/konrad) [Sep 18 '18 at 10:35](https://stackoverflow.com/questions/38630076/asp-net-core-web-api-exception-handling#comment91713203_51847739)

add a comment

12

Well accepted answer helped me a lot but i wanted to pass HttpStatusCode in my middleware to manage error status code at runtime.

According to [this link](https://blogs.msdn.microsoft.com/brandonh/2017/07/31/using-middleware-to-trap-exceptions-in-asp-net-core/) i got some idea to do the same. So i merged the Andrei Answer with this. So my final code is below:   
1. Base class

public class ErrorDetails

{

public int StatusCode { get; set; }

public string Message { get; set; }

public override string ToString()

{

return JsonConvert.SerializeObject(this);

}

}

2. Custom Exception Class Type

public class HttpStatusCodeException : Exception

{

public HttpStatusCode StatusCode { get; set; }

public string ContentType { get; set; } = @"text/plain";

public HttpStatusCodeException(HttpStatusCode statusCode)

{

this.StatusCode = statusCode;

}

public HttpStatusCodeException(HttpStatusCode statusCode, string message) : base(message)

{

this.StatusCode = statusCode;

}

public HttpStatusCodeException(HttpStatusCode statusCode, Exception inner) : this(statusCode, inner.ToString()) { }

public HttpStatusCodeException(HttpStatusCode statusCode, JObject errorObject) : this(statusCode, errorObject.ToString())

{

this.ContentType = @"application/json";

}

}

3. Custom Exception Middleware

public class CustomExceptionMiddleware

{

private readonly RequestDelegate next;

public CustomExceptionMiddleware(RequestDelegate next)

{

this.next = next;

}

public async Task Invoke(HttpContext context /\* other dependencies \*/)

{

try

{

await next(context);

}

catch (HttpStatusCodeException ex)

{

await HandleExceptionAsync(context, ex);

}

catch (Exception exceptionObj)

{

await HandleExceptionAsync(context, exceptionObj);

}

}

private Task HandleExceptionAsync(HttpContext context, HttpStatusCodeException exception)

{

string result = null;

context.Response.ContentType = "application/json";

if (exception is HttpStatusCodeException)

{

result = new ErrorDetails() { Message = exception.Message, StatusCode = (int)exception.StatusCode }.ToString();

context.Response.StatusCode = (int)exception.StatusCode;

}

else

{

result = new ErrorDetails() { Message = "Runtime Error", StatusCode = (int)HttpStatusCode.BadRequest }.ToString();

context.Response.StatusCode = (int)HttpStatusCode.BadRequest;

}

return context.Response.WriteAsync(result);

}

private Task HandleExceptionAsync(HttpContext context, Exception exception)

{

string result = new ErrorDetails() { Message = exception.Message, StatusCode = (int)HttpStatusCode.InternalServerError }.ToString();

context.Response.StatusCode = (int)HttpStatusCode.BadRequest;

return context.Response.WriteAsync(result);

}

}

4. Extension Method

public static void ConfigureCustomExceptionMiddleware(this IApplicationBuilder app)

{

app.UseMiddleware<CustomExceptionMiddleware>();

}

5. Configure Method in startup.cs

app.ConfigureCustomExceptionMiddleware();

app.UseMvc();

**Now my login method in Account controller :**

try

{

IRepository<UserMaster> obj = new Repository<UserMaster>(\_objHeaderCapture, Constants.Tables.UserMaster);

var Result = obj.Get().AsQueryable().Where(sb => sb.EmailId.ToLower() == objData.UserName.ToLower() && sb.Password == objData.Password.ToEncrypt() && sb.Status == (int)StatusType.Active).FirstOrDefault();

if (Result != null)//User Found

return Result;

else// Not Found

throw new HttpStatusCodeException(HttpStatusCode.NotFound, "Please check username or password");

}

catch (Exception ex)

{

throw ex;

}

Above you can see if i have not found the user then raising the HttpStatusCodeException in which i have passed HttpStatusCode.NotFound status and a custom message   
In middleware

catch (HttpStatusCodeException ex)

blocked will be called which will pass control to

private Task HandleExceptionAsync(HttpContext context, HttpStatusCodeException exception) method

.  
  
  
But what if i got runtime error before? For that i have used try catch block which throw exception and will be catched in catch (Exception exceptionObj) block and will pass control to

Task HandleExceptionAsync(HttpContext context, Exception exception)

method.  
  
I have used a single ErrorDetails class for uniformity.

[share](https://stackoverflow.com/a/52717826)[improve this answer](https://stackoverflow.com/posts/52717826/edit)

answered Oct 9 '18 at 9:38

[[https://www.gravatar.com/avatar/dc73f1dc307f1f3943d69547d524adde?s=32&d=identicon&r=PG&f=1](https://stackoverflow.com/users/1363880/arjun)](https://stackoverflow.com/users/1363880/arjun)

[Arjun](https://stackoverflow.com/users/1363880/arjun)

**256**1417

* 2

Answers should be like this. – [Hassan Tareq](https://stackoverflow.com/users/4802664/hassan-tareq) [Jan 14 at 3:33](https://stackoverflow.com/questions/38630076/asp-net-core-web-api-exception-handling#comment95179667_52717826)

* Great Answer. Appreciate knowledge sharing – [Mustafa Mohammadi](https://stackoverflow.com/users/2313991/mustafa-mohammadi) [Mar 1 at 3:22](https://stackoverflow.com/questions/38630076/asp-net-core-web-api-exception-handling#comment96638820_52717826)

add a comment

11

Latest Asp.Net Core (at least from 2.2, probably earlier) has a built-in middleware that makes it a bit easier compared to the implementation in the accepted answer:

app.UseExceptionHandler(a => a.Run(async context =>

{

var exceptionHandlerPathFeature = context.Features.Get<IExceptionHandlerPathFeature>();

var exception = exceptionHandlerPathFeature.Error;

var result = JsonConvert.SerializeObject(new { error = exception.Message });

context.Response.ContentType = "application/json";

await context.Response.WriteAsync(result);

}));

It should do pretty much the same, just a bit less code to write. Remember to add it before UseMvc as order is important.

[share](https://stackoverflow.com/a/55166404)[improve this answer](https://stackoverflow.com/posts/55166404/edit)

[edited Mar 14 at 15:47](https://stackoverflow.com/posts/55166404/revisions)

answered Mar 14 at 15:30

[[https://i.stack.imgur.com/cNgN2.png?s=32&g=1](https://stackoverflow.com/users/1671558/ilya-chernomordik)](https://stackoverflow.com/users/1671558/ilya-chernomordik)

[Ilya Chernomordik](https://stackoverflow.com/users/1671558/ilya-chernomordik)

**9,594**74893

add a comment

7

First, configure ASP.NET Core 2 Startup to re-execute to an error page for any errors from the web server and any unhandled exceptions.

public void Configure(IApplicationBuilder app, IHostingEnvironment env)

{

if (env.IsDevelopment()) {

// Debug config here...

} else {

app.UseStatusCodePagesWithReExecute("/Error");

app.UseExceptionHandler("/Error");

}

// More config...

}

Next, define an exception type that will let you throw errors with HTTP status codes.

public class HttpException : Exception

{

public HttpException(HttpStatusCode statusCode) { StatusCode = statusCode; }

public HttpStatusCode StatusCode { get; private set; }

}

Finally, in your controller for the error page, customize the response based on the reason for the error and whether the response will be seen directly by an end user. This code assumes all API URLs start with /api/.

[AllowAnonymous]

public IActionResult Error()

{

// Gets the status code from the exception or web server.

var statusCode = HttpContext.Features.Get<IExceptionHandlerFeature>()?.Error is HttpException httpEx ?

httpEx.StatusCode : (HttpStatusCode)Response.StatusCode;

// For API errors, responds with just the status code (no page).

if (HttpContext.Features.Get<IHttpRequestFeature>().RawTarget.StartsWith("/api/", StringComparison.Ordinal))

return StatusCode((int)statusCode);

// Creates a view model for a user-friendly error page.

string text = null;

switch (statusCode) {

case HttpStatusCode.NotFound: text = "Page not found."; break;

// Add more as desired.

}

return View("Error", new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier, ErrorText = text });

}

ASP.NET Core will log the error detail for you to debug with, so a status code may be all you want to provide to a (potentially untrusted) requester. If you want to show more info, you can enhance HttpException to provide it. For API errors, you can put JSON-encoded error info in the message body by replacing return StatusCode... with return Json...