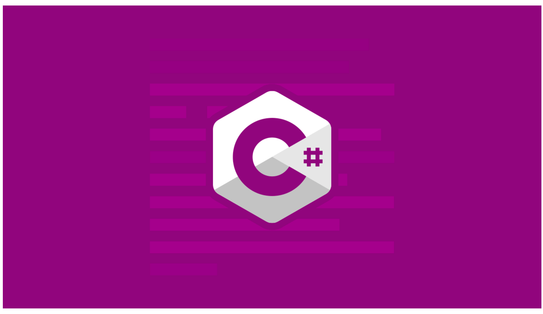
.NET Core Web API 中的异常处理

作者：微信公众号：【架构师老卢】 1-31 10:14 131



概述：有效处理异常对于构建健壮且用户友好的应用程序至关重要。本文深入探讨 .NET Core Web API 中的异常处理。演示如何创建自定义异常并全局管理它们，以实现更简洁、更高效的代码库。为什么异常处理很重要正确处理的异常可确保应用程序在所有情况下的行为都具有可预测性。它可以防止向最终用户公开敏感的错误详细信息。它为用户提供有意义的反馈，增强他们的体验。它使代码库更易于维护和调试。构建自定义异常自定义异常允许您更清楚地表达特定的错误条件。让我们为核心 Web API 创建四个自定义异常。NotFoundException当找不到请求的资源时，将引发此异常。public class NotFoun

有效处理异常对于构建健壮且用户友好的应用程序至关重要。本文深入探讨 .NET Core Web API 中的异常处理。演示如何创建自定义异常并全局管理它们，以实现更简洁、更高效的代码库。

为什么异常处理很重要

- 正确处理的异常可确保应用程序在所有情况下的行为都具有可预测性。
- 它可以防止向最终用户公开敏感的错误详细信息。
- 它为用户提供有意义的反馈，增强他们的体验。
- 它使代码库更易于维护和调试。

构建自定义异常

自定义异常允许您更清楚地表达特定的错误条件。让我们为核心 Web API 创建四个自定义异常。

NotFoundException

当找不到请求的资源时，将引发此异常。

```
public class NotFoundException : Exception
{
    public NotFoundException(string message) : base(message) { }
}
```

ValidationException

当数据验证失败时，请使用此选项。

```
public class ValidationException : Exception
{
    public ValidationException(string message) : base(message) { }
}
```

UnauthorizedAccessException

指示对资源的未经授权的访问。

```
public class UnauthorizedAccessException : Exception
{
    public UnauthorizedAccessException(string message) : base(message) { }
}
```

InternalServerErrorException

表示应用程序中的意外故障。

```
public class InternalServerErrorException : Exception
{
    public InternalServerErrorException(string message) : base(message) { }
}
```

全局异常处理

.NET Core 允许全局处理异常，而不是将 try-catch 块分散在整个代码中。这是使用中间件或过滤器完成的。我们将重点介绍如何使用全局筛选器。

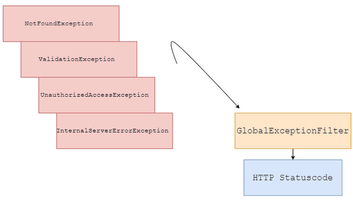
```
public class GlobalExceptionFilter : IExceptionFilter
{
    public void OnException(ExceptionContext context)
    {
        var statusCode = context.Exception switch
        {
            NotFoundException => StatusCodes.Status404NotFound,

            ValidationException => StatusCodes.Status400BadRequest,

            UnauthorizedAccessException => StatusCodes.Status401Unauthorized,

            _ => StatusCodes.Status500InternalServerError
        };

        context.Result = new ObjectResult(new
        {
            error = context.Exception.Message,
            stackTrace = context.Exception.StackTrace
        })
        {
            StatusCode = statusCode
        };
    }
}
```

此筛选器将截获异常并将其转换为适当的 HTTP 响应。

注册全局筛选器

在 中，注册全局筛选器。Startup.cs

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllers(options =>
    {
        options.Filters.Add(new GlobalExceptionFilter());
    });
}
```



引发自定义异常，由 .Net Core Web API 中的全局筛选器捕获 - 图片来源：由作者创建

自定义异常的用法

为了演示 .NET Core Web API 中自定义异常和全局异常筛选器的用法，让我们创建一个简单的示例。这将涉及一个引发这些异常的控制器，演示全局异常筛选器如何捕获和处理这些异常。

```
[ApiController]
[Route("[controller]")]
public class SampleController : ControllerBase
{
    [HttpGet("not-found")]
    public ActionResult GetNotFound()
    {
        // Simulate a situation where a resource is not found
        throw new NotFoundException("The requested resource was not found.");
    }

    [HttpGet("invalid")]
    public ActionResult GetInvalid()
    {
        // Simulate a validation error
        throw new ValidationException("Validation failed for the request.");
    }

    [HttpGet("unauthorized")]
    public ActionResult GetUnauthorized()
    {
        // Simulate unauthorized access
        throw new UnauthorizedAccessException("You do not have permission to access this resource.");
    }

    [HttpGet("internal-error")]
    public ActionResult GetInternalError()
    {
        // Simulate an internal server error
        throw new InternalServerErrorException("An unexpected error occurred.");
    }
}
```

运行 Web API 并向这些终结点发出请求时，将引发相应的自定义异常。

A GET request to /sample/not-found will throw the NotFoundException.

A GET request to /sample/invalid will throw the ValidationException.

当引发这些异常时，全局异常筛选器会捕获它们。然后，它将每个异常映射到相应的 HTTP 状态代码，并向客户端返回结构化响应。

例如，如果引发，筛选器将返回 404 Not Found 状态，其中包含包含错误消息和堆栈跟踪的 JSON 正文。NotFoundException

对全局异常筛选器进行单元测试

测试全局异常筛选器包括检查它是否正确地将异常转换为相应的 HTTP 响应。

若要对此进行测试，可以模拟引发异常的操作方法，然后断言筛选器的响应。使用 Moq 和 xUnit 的示例测试。

```
public class GlobalExceptionFilterTests
{
    [Fact]
    public void OnException_ShouldSetCorrectStatusCodeForNotFoundException()
    {
        // Arrange
        var exceptionContextMock = new Mock<ExceptionContext>();
        exceptionContextMock.SetupGet(x => x.Exception).Returns(new NotFoundException("Not found"));
        var filter = new GlobalExceptionFilter();

        // Act
        filter.OnException(exceptionContextMock.Object);

        // Assert
        var objectResult = exceptionContextMock.Object.Result as ObjectResult;
        Assert.NotNull(objectResult);
        Assert.Equal(StatusCodes.Status404NotFound, objectResult.StatusCode);
    }
```

```
    }
```

良好的异常处理可以显著提高软件的质量。