MICROSOFT ARCHITECT
By Joydip Kanjilal, Columnist, InfoWorld
SEP 23, 2019 3:00 AM PDT

Using advanced NLog features in ASP.NET Core

Take advantage of the free and flexible NLog to log your application data with the features and efficiency you need

NLog is an open source and lightweight logging platform with rich log routing and management capabilities. NLog is also easy to configure and extend. I've discussed using NLog in .NET and using NLog in ASP.NET Core in earlier posts. In this article, I will discuss how we can perform more advanced operations using NLog.

Specifically, we will look at how to configure NLog using a .config file as well as code-based configuration, how to automatically rotate logs, how to use a database as a log target, and how to improve performance by logging data asynchronously. I will also share some best practices for using NLog.

[Using Visual Studio Code? Don't miss these 10 Visual Studio Code extensions for every developer. • Or these 7 Visual Studio Code extensions you didn't know you needed. | Keep up with hot topics in programming with InfoWorld's App Dev Report newsletter.]

Create an ASP.NET Core MVC project in Visual Studio

First off, let's create an ASP.NET Core project in Visual Studio 2019. Assuming Visual Studio 2019 is installed in your system, follow the steps outlined below to create a new ASP.NET Core MVC project in Visual Studio.

1. Launch the Visual Studio IDE.
2. Click on "Create new project."

3. In the “Create new project” window, select “ASP.NET Core Web Application” from the list of templates displayed.
4. Click Next.
5. In the “Configure your new project” window, specify the name and location for the new project.
6. Optionally, select the “Place solution and project in the same directory” check box.
7. Click Create.
8. In the “Create a New ASP.NET Core Web Application” window shown next, select .NET Core as the runtime and ASP.NET Core 2.2 (or later) from the drop-down list at the top.
9. Select “Web Application (Model-View-Controller)” as the project template to create a new ASP.NET Core MVC application.
10. Ensure that the check boxes “Enable Docker Support” and “Configure for HTTPS” are unchecked as we won’t be using those features here.
11. Ensure that Authentication is set to “No Authentication” as we won’t be using authentication either.
12. Click Create.

We’ll use this project in the sections below to illustrate how we can work with NLog in ASP.NET Core MVC.

Install the NuGet packages for NLog

Next install the following packages via the NuGet Package Manager or the NuGet Package Manager Console.

- NLog.Web.AspNetCore
- NLog.Extensions.Logging
- NLog.Config

When you install the NLog.Config package, a file called NLog.config and its dependencies will be added to your project. Note that the NLog.Config package is not required to work with NLog. It is needed only if you want to work with NLog using a configuration file, rather than code-based configuration.

Configure NLog using a .config file

NLog provides support for both .config file-based configuration as well as code-based configuration. Once you have installed the NLog.Config package in your project, a file named NLog.config is created in your project with the following contents:

```
<?xml version="1.0" encoding="utf-8" ?>
<nlog xmlns="http://www.nlog-project.org/schemas/NLog.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
autoReload="true">
  <extensions>
    <add assembly="NLog.Web.AspNetCore"/>
  </extensions>
  <targets>
    <target name="logfile" xsi:type="File" fileName="D:\logs\LogMessages-${shortdate}.lo
  </targets>
  <rules>
    <logger name="*" minlevel="Trace" writeTo="logfile" />
  </rules>
</nlog>
```

The following code snippet illustrates how you can use NLog to log data in your action methods.

```
public class HomeController : Controller
 {
     Logger _logger =
     (Logger)LogManager.GetCurrentClassLogger(typeof(Logger));
     public IActionResult Index()
      {
         _logger.Info("Application started");
          return View();
      }
    //Other action methods
  }
```

If you would like to find a target programmatically, you can use the following code.

```
var fileTarget = (FileTarget)LogManager.Configuration.FindTargetByName("logfile");
```

Configure NLog using code-based configuration

You can configure NLog programmatically as well. To do this, you'll need to take advantage of the NLog API. The following method illustrates how we can configure NLog programmatically.

```
private static void ConfigureNLog()
        {
            var logConfiguration = new LoggingConfiguration();
            var dbTarget = new DatabaseTarget();
            dbTarget.ConnectionString = "Data Source=JOYDIP;initial
            catalog=NLogDemo;User Id=sa;Password=sa1@3#.;";
            dbTarget.CommandText = "INSERT INTO DbLog
           (level, callsite, message, logdatetime)" +
                " Values(@level, @callsite, @message, @logdatetime)";
            dbTarget.Parameters.Add
            (new DatabaseParameterInfo("@level", "${level}"));
            dbTarget.Parameters.Add
            (new DatabaseParameterInfo("@callSite", "${callSite}"));
            dbTarget.Parameters.Add
            (new DatabaseParameterInfo("@message", "${message}"));
            dbTarget.Parameters.Add
            (new DatabaseParameterInfo("@logdatetime","${date:s}"));
            var rule = new LoggingRule("*", LogLevel.Debug, dbTarget);
            logConfiguration.LoggingRules.Add(rule);
            LogManager.Configuration = logConfiguration;
        }
```

Automatically rotate logs using NLog

You can configure NLog to automatically rotate your logs. As an example, you can configure it to preserve the logs for the last *n* days and delete the remaining logs automatically. This feature is extremely useful in production, where you would otherwise need to delete old log files often. The following code snippet illustrates how you can implement automatic log rotation in NLog using .config file-based configuration.

```
<?xml version="1.0" encoding="utf-8" ?>
<nlog xmlns="http://www.nlog-project.org/schemas/NLog.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
autoReload="true">
      <targets>
    <target name="logfile"
            xsi:type="File"
            fileName="${basedir}/logs/App.log"
            layout="${longdate}  ${message}"
            archiveFileName="${basedir}/logs/archive.{#}.log"
            archiveEvery="Day"
            archiveNumbering="Rolling"
            maxArchiveFiles="7"
            concurrentWrites="true"
            keepFileOpen="true" />
  </targets>
  <rules>
    <logger name="*" minlevel="Info" writeTo="logfile" />
  </rules>
</nlog>
```

Log to the database using NLog

Create the database table

You can use NLog to log data to the database as well. Here is the script for creating a table in the database that we'll use for logging data.

```
CREATE TABLE [dbo].[DbLog](
      [Id] [int] IDENTITY(1,1) NOT NULL,
      [Level] [varchar](max) NULL,
      [CallSite] [varchar](max) NULL,
      [Message] [varchar](max) NULL,
      [AdditionalInfo] [varchar](max) NULL,
      [LogDateTime] [datetime] NOT NULL,
 CONSTRAINT [PK_DbLogs] PRIMARY KEY CLUSTERED
(
      [Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_L
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
```

Specify the database connection string and command text properties

You need to specify the database connection details as shown in the code snippet given below. Note how the connectionString and the commandText properties have been used.

```
<target name="database" xsi:type="Database" keepConnection="true"
 useTransactions="true"
 dbProvider="System.Data.SqlClient"
 connectionString="data source=localhost;initial
 catalog=NLogDemo;integrated security=false;
 persist security info=True;User ID=sa;Password=sa1@3#."
 commandText="INSERT INTO DbLog (level, callsite, message, additionalInfo,
 logdatetime) Values (@level, @callsite, @message, @additionalInfo,
 @logdatetime)">
```

Specify the log target in NLog.config

Next, you should specify the target in the NLog.config file so that you can log data to the database.

```
<target name="database" xsi:type="Database" keepConnection="true" useTransactions="true
connectionString="data source=localhost;initial catalog=NLogDemo;integrated security=fal
commandText="INSERT INTO DbLog (level, callsite, message, additionalInfo, logdatetime) V
</target>
```

Map the parameters from the NLog layout to the database

Lastly, you should map the parameters using the parameter tag as shown in the code snippet given below.

```
<parameter name="@level" layout="${level}" />
<parameter name="@callSite" layout="${callsite}" />
<parameter name="@message" layout="${message}" />
<parameter name="@additionalInfo" layout="${var:AdditionalInfo}" />
<parameter name="@logdatetime" layout="${date:s}" />
```

Log to the database using NLog — complete NLog.config

The complete code of the NLog.config file is given below for your reference.

```
<?xml version="1.0" encoding="utf-8" ?>
<nlog xmlns="http://www.nlog-project.org/schemas/NLog.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
autoReload="true">
  <extensions>
    <add assembly="NLog.Web.AspNetCore"/>
  </extensions>
  <variable name="AdditionalInfo" value=""/>
  <targets>
    <target name="database" xsi:type="Database" keepConnection="true"
    useTransactions="true"
    dbProvider="System.Data.SqlClient"
    connectionString="data source=localhost;initial
    catalog=NLogDemo;integrated security=false;persist security
    info=True;User ID=sa;Password=sa1@3#."
    commandText="INSERT INTO DbLog
    (level, callsite, message, additionalInfo, logdatetime)
    Values (@level, @callsite, @message, @additionalInfo, @logdatetime)">
      <parameter name="@level" layout="${level}" />
      <parameter name="@callSite" layout="${callsite}" />
      <parameter name="@message" layout="${message}" />
      <parameter name="@additionalInfo" layout="${var:AdditionalInfo}" />
      <parameter name="@logdatetime" layout="${date:s}" />
    </target>
  </targets>
  <rules>
    <logger levels="Debug,Info,Error,Warn,Fatal" name="databaseLogger" writeTo="databas
  </rules>
</nlog>
```

In addition to Microsoft SQL Server, you can use NLog to log data to other databases such as MySQL, Oracle Database, and SQLite as well.

Improve NLog performance with AsyncWrapper

NLog support several kinds of targets, such as AsyncWrapper, BufferingWrapper, FallbackGroup, and RetryingWrapper. Asynchronous target wrappers enable you to queue messages and process them in a separate thread for improved performance. The following code snippet illustrates the syntax for using AsyncWrapper.

```
<targets>
  <target xsi:type="AsyncWrapper"
          name="String"
          queueLimit="Integer"
          timeToSleepBetweenBatches="Integer"
          batchSize="Integer"
          overflowAction="Enum">
    <target ... />
  </target>
</targets>
```

You can take advantage of an asynchronous wrapper and wrap a file target inside it to log data to the file asynchronously as shown in the code snippet given below.

```
  <targets>
    <target name="asyncFile" xsi:type="AsyncWrapper">
      <target xsi:type="File" name="fileLog"
         fileName="${basedir}/Logs/${shortdate}.log"
            layout="${longdate} ${uppercase:${level}} ${message}"/>
    </target>
  </targets>
  <rules>
    <logger levels="Debug,Info,Error,Warn,Fatal" writeTo="asyncFile"/>
  </rules>
```

Alternatively, you can use the following code to wrap all targets with AsyncWrapper.

```
<targets async="true">
  ... Write your targets here ...
</targets>
```

Best practices when using NLog

This section lists some of the best practices that should be followed when working with NLog.

1. A logger instance should be static. This is to avoid the processing overhead that will be incurred if the logger instance needs to be created multiple times.

2. Take advantage of NLog's support for formatting (when you need structured logging using NLog) to avoid string allocation and concatenation.

3. Specify `throwConfigExceptions="true"` to ensure that NLog will provide the details if there is an issue with the NLog configuration. For example:

```
<nlog xmlns="http://www.nlog-project.org/schemas/NLog.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
autoReload="true" throwConfigExceptions="true">
```

4. Make a call to the Shutdown () method of the LogManager class to flush and close all internal threads and timers when you are done logging data using NLog:

```
NLog.LogManager.Shutdown();
```

5. Make sure that you haven't combined the Async attribute with AsyncWrapper. If you combine them, processing will be slow.

6. You should use the log level Trace infrequently, as Trace implies that you are logging everything. Consider using log levels Debug or Info instead.

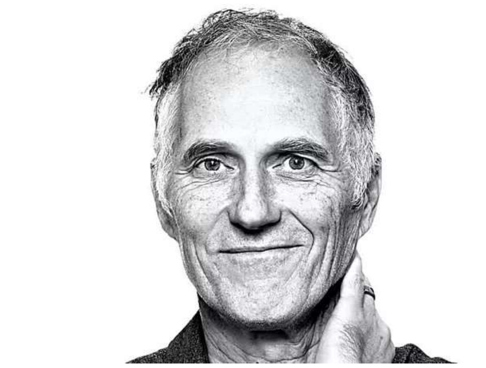
7. While NLog is lightweight and fast, you can improve performance even further by using asynchronous wrappers.

There is much more to say about NLog. As noted above, NLog provides support for structured logging as well. Further, you can now filter and analyse large volumes of log data easily. I will discuss these and more features of NLog in future posts here.

Joydip Kanjilal is a Microsoft MVP in ASP.Net, as well as a speaker and author of several books and articles. He has more than 20 years of experience in IT including more than 16 years in Microsoft .Net and related technologies.

Follow

YOU MAY ALSO LIKE


Tim O'Reilly: the golden age of the programmer is over

6 ways Alibaba Cloud challenges AWS, Azure, and GCP

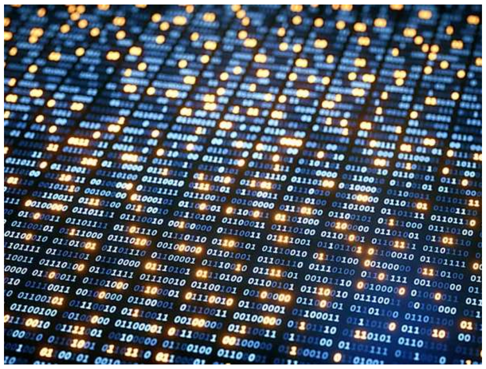
Pulling devops and multicloud together

How to use SortedDictionary, SortedList, and SortedSet in C#

Prisma ORM for Node.js is ready for production

LLVM 12 arrives with x86, AArch optimizations

Edge computing archetypes are emerging,


What is a computational storage drive? Much-

6 reasons to switch to managed Kubernetes

The 24 highest paying developer roles in 2020

The shifting market for PostgreSQL

SPONSORED LINKS

dtSearch® instantly searches terabytes of files, emails, databases, web data. See site for hundreds of reviews; enterprise & developer evaluations

Truly modern web app and API security thinking. It's a thing. See how.

Want lightning fast analytics? See why the Incorta data analytics platform is changing enterprise data forever.

2020 was a year of rapid progression of digital transformation for businesses. The following is a snapshot of the digital transformation advancements made across all facets of business.

DDoS extortion attacks are real. Don’t Negotiate. Mitigate with NETSCOUT. Learn more.