4.48 Оценка 253.59 Рейтинг

Газпромбанк

Очень большой банк

Подписаться

eugene_naryshkin
24 янв 2023 в 09:00

Структурное логирование в .NET на примере Serilog

4 мин 16K

Блог компании Газпромбанк, .NET*, Проектирование и рефакторинг*, C#*

Все мы знаем, что логирование - вещь очень полезная для современного проекта. С помощью него можно быстро локализовать и устранить ошибку в продукте, восстановить кейс, который к ней привёл, посмотреть историю действий пользователя.

Существует несколько видов логирования, такие как:

1. **Классическое** - когда весь лог это набор строк, в котором порой сложно разобраться и что-то в нём проанализировать.

```
_logger.LogInformation($"The magic number is {number}");
```

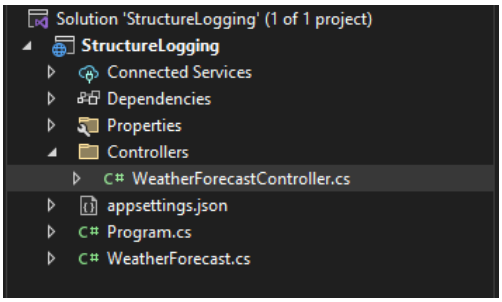
2. **Структурное** - когда на одно событие будет создаваться две записи лога, одна запись это шаблон вывода сообщения, вторая запись - объект, который будет подставлен в шаблон.

```
_logger.LogInformation("The magic number is {number}", number);
```

Структурное логирование открывает возможности хранить и анализировать события в различного рода хранилищах, таких как NoSql, Sql базах данных. Для .NET существует множество сторонних библиотек для такого логирования, например Serilog или NLog.
Давайте рассмотрим, как воспользоваться одной из этих библиотек и начать вести лог правильно?

Классическое логирование

1. Создадим новый проект по шаблону *ASP.NET Core Web API*. По умолчанию после создания мы будем иметь такую структуру решения:

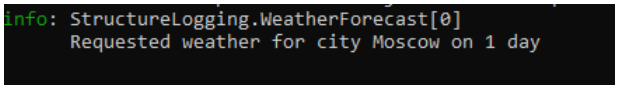


Структура решения StructureLogging

2. Изменим метод *Get* контроллера *WeatherForecast*, заменив его на следующий код:

```
public IEnumerable<WeatherForecast> Get(string city, int day)
{
    _logger.LogInformation($"Requested weather for city {city} on {day} day");

    return Enumerable.Range(1, 5).Select(index => new WeatherForecast
    {
        Date = DateTime.Now.AddDays(index),
        TemperatureC = Random.Shared.Next(-20, 55),
        Summary = Summaries[Random.Shared.Next(Summaries.Length)]
    })
    .ToArray();
}
```

3. Запустим приложение и выполним GET запрос к *http://localhost:7005/weatherforecast?city=Moscow&day=1* и посмотрим результат выполнения в консоли:



Результат логирования

Видно, что запись события это просто строка, и куда бы она ни попала (файл, СУБД, журнал событий) - она так и останется строкой, из которой что-то значимое для анализа можно будет вытащить только с помощью регулярных выражений, как пример. Такой вариант мало кого устраивает и на помощь приходит **структурное логирование**.

Структурное логирование

Теперь изменим немного проект, который создали выше. Для его модификации будем использовать библиотеку Serilog.

1. Установим NuGet-пакеты Serilog, Serilog.Sinks.Console и Serilog.Extensions.Hosting.
2. В файле Program.cs добавим настройку и добавление Serilog в качестве логгера:

```
using Microsoft.AspNetCore.Builder;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Serilog;

var builder = WebApplication.CreateBuilder(args);
/*
.................................
*/

Log.Logger = new LoggerConfiguration()
```

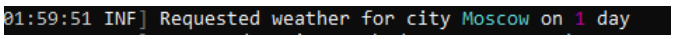
```
    .Enrich.FromLogContext()
    .WriteTo.Console()
    .CreateLogger();

builder.Host.ConfigureLogging(logging =>
{
    logging.AddSerilog();
    logging.SetMinimumLevel(LogLevel.Information);
})
.UseSerilog();

/*
.....................................
*/
```

3. В методе Get контроллера WeatherForecast изменим запись в лог:

```
_logger.LogInformation("Requested weather for city {City} on {Day} day", city, day);
```

Мы убрали интерполяцию строки и теперь значения city и day по порядку будут подставляться вместо якорей {City} и {Day} в шаблоне.

4. Снова запустим приложение и выполним GET запрос к *http://localhost:7005/weatherforecast?city=Moscow&day=1* и посмотрим результат выполнения в консоли:



Результат логирования

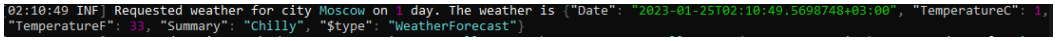
Видно, что наши значения, передаваемые в логгер окрасились в разные цвета. Всё потому, что они больше не считаются простой строкой, а являются объектами, подставляемыми в шаблон.

Также можно передавать и более сложные объекты, например пользовательские типы данных. Единственное, что **нужно помнить для сложных объектов** - при указании якоря впереди ставится @ *{@Weather}* для обозначения Serilog'у, что объект пользовательский, либо в пользовательском объекте нужно переопределить метод *ToString*, дополнив его собственной реализацией.

Снова изменим код метода *Get* контроллера *WeatherForecast*:

```
public IEnumerable<WeatherForecast> Get(string city, int day)
{
    var result = Enumerable.Range(1, 5).Select(index => new WeatherForecast
    {
        Date = DateTime.Now.AddDays(index),
        TemperatureC = Random.Shared.Next(-20, 55),
        Summary = Summaries[Random.Shared.Next(Summaries.Length)]
    })
    .ToArray();

    _logger.LogInformation("Requested weather for city {City} on {Day} day. The weather is {@We

    return result;
}
```

Затем запустим приложение и выполним GET запрос к *http://localhost:7005/weatherforecast?city=Moscow&day=1* и посмотрим результат выполнения в консоли:



Результат логирования

Сложный объект "разложился" по своим свойствам и в будущем готов для анализа логов(если мы сейчас говорим о чём-то более сложном, чем консоль).
В следующий раз мы разберемся, как воспользоваться структурным логированием в связке с *ElasticSearch* и увидим, как оно позволяет анализировать логи нашего приложения.

Вывод

У структурного логирования есть плюсы, такие как возможность отделять данные от события, с последующим анализом, простота настройки и использования.

Теги: .net, c#, serilog, logging, asp.net core, asp.net core webapi

Хабы: Блог компании Газпромбанк, .NET, Проектирование и рефакторинг, C#

Газпромбанк
Очень большой банк

Сайт

9 Карма 0 Рейтинг

Евгений Нарышкин @eugene_naryshkin
Senior .NET Developer

Подписаться

Telegram

Комментарии 16

НЛО прилетело и опубликовало эту надпись здесь

YegorP
24 янв 2023 в 09:30

> добавить логирование в middleware всех входящих/исходящих реквестов

... вместе с паролями, токенами и файлами на пару гигабайт.

> а бизнес логику покрывать юнит тестами

... и кусать локти пытаясь диагностировать любые непредвиденные кейсы.

> уберёт ILogger из всех мест приложения

Aspect-oriented programming вам в помощь, но только не в виде мидлвари на самом входе и на самом выходе.

Ответить

НЛО прилетело и опубликовало эту надпись здесь