

Пояснительная записка

Структура проекта

В союшоне 3 проекта:

LoggerCore - проект с абстракциями и базовыми реализациями, служит ядром для системы

SmartLogger - проект с сильно упрощенным логгированием. Здесь лежит адаптер к NLog, реализующий интерфейс ILogger.

LoggerExample - проект с примером использования на крайне примитивной реализации калькулятора (а также из-за нехватки времени на реализацию юнит-тестов устроил там и тестирование).

Чек-лист

- Замена ядра простая. В LoggerExample.Utils.Loggers.ConfigureLoggers() строки с 32 по 41. Почти все опирается на интерфейсы, так что замена не только ядра, но и инфраструктурных элементов возможна и легка.
- Логгер предоставляет возможности логгирования по уровням, см. SmartLogger
- стектрейс пробрасывается (даже продублировал его отдельным параметром). Пример в LoggerExample.Program строка 33.
- Логирование дебаг сообщений отключается в релизе.
- Добавлен гибкий функционал для вывода источников. Осуществлена поддержка иерархии логгеров. Осуществлена поддержка использования уровней логгирования, что позволяет игнорировать в зависимости от уровня менее важные сообщения.
- Для использования в веб-приложениях был разработан ISmartLogger, SmartLogger - для проброски через DI. В примере LoggerExample, сделана статическая обертка над SmartLogger и вызов логгера выглядит максимально коротко: Log.Exception<T>(ex); Log.Info<T>("сообщение").

Что не успел

- XML-комментарии, к очень большому сожалению, не успел их проставить 😞
- Debug с аргументом exception, а не message и Error с фатал с аргументом message, а не exception.
- Автоматические юнит-тесты, пришлось тестировать на базе LoggerExample ручками (сравнивать лог с ожидаемым).
- Потокбезопасная LoggerServiceCollection на базе ConcurrentDictionary
- Lazy-load LoggerService'ов
- Вспомогательные туллы для логгирования (например разложение объекта на json свойство-значение)

Немного про уровни абстракции

ILogger - интерфейс для коннекта ядер логгирования

LoggerService, ILoggerService - обертка над ILogger, дополнительно хранит себе название источника и уровень логгирования. Упрощает логгирование до уровня SomeServiceLogger.Debug("пример сообщения");

LoggerServiceCollection, ILoggerServiceCollection - хранилище логгерсервисов. Берет на себя ответственность по хранению, удалению и удобной выдаче логгер сервису по запросу вида SomeLoggerService.GetLoggerService<T>(), где T - класс откуда мы вызываем логгер (либа класс от которого хотим позаимствовать логгер).

SmartLogger, ISmartLogger - обертка над LoggerService'ом. Берет на себя ответственность за логгирование без необходимости ручного получения LoggerService для последующей записи в лог, т.е. отвечает за правильный и удобный выбор логгерсервиса для логгирования. На практике нужен для того, чтобы не

писать строчку `var someLoggerService = someLoggerServiceCollection.GetLoggerService<needClass>();`. Позволяет писать логи по формату `SomeSmartLogger.Info<T>("пример сообщения")`, где `T` - класс откуда мы вызываем логгер (либо класс от которого хотим позаимствовать логгер). Подразумевается использование этого уровня на проекте через DI.

Статический класс `Log` - демонстрационная обертка над `SmartLogger`. Писал для демонстрации короткой записи вида `Log.Warn<T>("Слишком коротко")`, где `T` - класс откуда мы вызываем логгер (либо класс от которого хотим позаимствовать логгер).

Что было задумано

Задание подразумевает разработку доп. слоя между ядром логирования и системой. Поэтому хотелось сделать этот слой максимально мощным и гибким, чтобы не сокращать функционал ядер логирования.

Уровни логирования. Тут все просто - хочется иметь возможность отправлять сообщения с разными уровнями логирования и хотелось бы иметь логгерсервисы, которые можно настроить по уровню (например один лог бесконечно спамит `INFO`, а нам хотелось, чтобы сервис обрезал ненужные логи)

Х - видимость	сообщение	FATAL	ERROR	WARN	INFO	DEBUG	TRACE	ALL	
логгер-сервис									
OFF									
FATAL		X							
ERROR		X	X						
WARN		X	X	X					
INFO		X	X	X	X				
DEBUG		X	X	X	X	X			
TRACE		X	X	X	X	X	X		
ALL		X	X	X	X	X	X	X	

Логгер-сервис - это объект, чья ответственность заключается в передаче данных в логгер (адаптер к ядру логирования) и управлением уровнем (детализацией) этого вывода.

`SmartLogger` логирует по сигнатуре `SmartLogger.Info<T>(string message)`, подразумевается что будет передаваться через DI, и передает ответственность по логированию соответствующему логгер-сервису. Определяет нужный сервис исходя из полного имени класса, т.е.

`LoggerExample.Folder1.ExampleClass`

что позволяет определить иерархию выбора логгер-сервиса

```
"" <- LoggerExample <- Folder1 <- ExampleClass
```

Поскольку в подавляющем большинстве числе случаев слои лежат либо в свои проектах, либо в своих папках, такой метод определения логгера считал подходящим. При этом мы не потеряли возможность логировать один и тот же слой в совершенно разных местах. Для этого нужно создать еще один экземпляр логгерсервиса с таким же `"source"`, зарегистрировать его в коллекции сервисов, но уже со своим путем.

Каждому звену в иерархии можно дать свой логгер-сервис со своей реализацией и своим уровнем. Если на текущем уровне нет логгер-сервиса - обращаемся к родителю, и так до логгера по умолчанию ""

При выборе сервиса мы идем по иерархии вниз. Соответственно если у корневого будет уровень ALL, а у дочернего debug - то уровни логирования будут выглядеть так

Логгер-сервис	Назначенный уровень	Уровень, который будет	Кто будет обрабатывать
""	ALL	ALL	Обработчик для ""
LoggerExample	Не назначен	ALL	Обработчик для ""
Folder1	DEBUG	DEBUG	Обработчик для Folder1
Example	Не назначен	DEBUG	Обработчик для Folder1

Как использовать

- 1) Создать экземпляры классов логгер-сервисов для разных слоев. Где нужно назначить название слоя, уровень логирования и адаптер к ядру логирования. Из коробки доступен NLog, для использования другого ядра нужно написать адаптер реализующий интерфейс ILogger
- 2) С помощью SmartLoggerBuilder() (либо руками) задать соответствие полному пути к слою/классу и логгер-сервису. Для корневого логгер-сервиса использовать String.Empty() в качестве пути.
- 3) Используем экземпляр класса SmartLogger для логгирования. Либо пробрасываем через DI, либо делаем статическую обертку для доступа из всех нужных уголков программы.