Лабораторная работа №8

Создание и работа с внешними библиотеками, подключаемыми к .NET приложению

Цель работы

Познакомиться с порядком подключения внешних библиотек к .NET приложению и их использованием в нем.

Общие сведения

Очень часто при разработке больших приложений приходится решать задачи, которые уже решались кем то ранее и уже представлены в виде готовых решений. Как правило это касается типовых задач, которые в самом общем виде так или иначе однотипно всегда представлены во многих бизнес-приложениях. В этом случае стоит задача разработать требуемый функционал и сделать так, чтобы была возможность подключить его использовать по необходимости.

Рассмотрим наиболее значимый практический пример, когда необходимо вести лог работы приложения. Такой подход позволяет по желанию (необходимости) разработчика вставлять вызовы для записи сообщений в лог и в последствии, в случае возникновения каких либо ошибок, отследить как вело себя приложение. Компонент log4net как раз предназначен для .net-приложений, позволяющий выводить любые сообщения в файл, несколько файлов, в БД или еще куда-нибудь. Кроме самого пользовательского сообщения, может вывести подробную информацию (время, класс, метод, и т.д., где был вызван вывод данного сообщения). В общем случае ту информацию, которую разработчик посчитает необходимой для отслеживания работы приложения и/или локализации ошибок приложения.

Для того, чтобы подключить ведение логов приложения необходимо скачать компонент (проект компонента доступен по ссылке http://logging.apache.org/log4net/) и добавить log4net.dll в References приложения.

В случае, если мы создаем свое Web-приложение или хотим его развернуть на другом сервере, то необходимо скопировать log4net в каталог bin приложения. После этого необходимо сконфигурировать логгер. Для этого необходимо в web.Config в раздел <configSections> (если его нет, то надо создать) добавить следующий фрагмент:

```
<section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler,
log4net">
```

Далее в этом же файле необходимо описать основной раздел log4net, который непосредственно конфигурирует работу логгера:

```
<log4net debug="true">
...
</log4net>
```

Здесь могут быть разделы <**appender**>, <**root**>, описания которых следуют далее.

Appender

Хранит описание типа используемого логгера, которых много (полный список можно посмотреть на сайте разработчика). Например, аппедер, который пишет лог в текстовый файл будет выглядеть, например, так:

```
<appender name="FileAppender" type="log4net.Appender.FileAppender">
  <param name="File" value="application.log" />
  <param name="AppendToFile" value="true" />
  <layout type="log4net.Layout.PatternLayout">
    <param name="ConversionPattern" value="%d [%t] %-5p %c [%x] - %m%n" />
  </layout>
</appender>
```

Имеет ряд ключевых настроека Appender. Некоторые из них описаны нижу (полное описание в документации разработчика).

Filter

Данная настройка позволяет отфильтровывать сообщения по некоторому признаку. Для фильтра обязательно указывается тип. Существует несколько типов фильтров, можно также описать и свои. Наиболее часто используемые фильтры - по типу сообщения (**`LevelRangeFilter`** и **`LevelMatchFilter`**) и по тексту сообщения (**`StringMatchFilter`**). Для начала, остановимся на типах (уровнях) сообщений. Сообщения в log4net делятся на несколько уровней:

- ALL
- DEBUG
- INFO
- WARN
- ERROR
- FATAL
- OFF

Разработчик сам выбирает, какой тип сообщений в каком случае ему использовать. Логгер же можно настроить таким образом, чтобы, к примеру, ошибки выводились отдельно, а отладочные данные - отдельно (все это настраивается фильтрами).

Layout

Здесь задается формат выводимого сообщения. Компонент log4net, помимо самого сообщения, определяемого в приложении, может выводить дополнительную информацию. Например этот раздел может выглядеть следующим образом:

```
<layout type="log4net.Layout.PatternLayout">
   <conversionPattern value="%-5p [%d] [%C.%M] %m%n"/>
</layout>
```

В **conversionPattern** указан шаблон префикса сообщения в логе. Согласно приведенному примеру в логе будут, например, вот так оформленные сообщения:

```
2013-10-14 00:28:45,194 [1] INFO  Deadlock [(null)] - Starting application
```

```
2013-10-14 00:28:45,414 [3] INFO  Deadlock [(null)] - Invoking thread A...
2013-10-14 00:29:41,264 [1] INFO  Deadlock [(null)] - Finishing application
```

Значения ключей conversionPattern:

- **%p** — Тип события лога (DEBUG, ERROR и т.д.)
- **%d** — Дата и время, когда логгирующая функция была вызвана. Можно задавать различные форматы вывода даты и времени. К примеру, `%d{HH:mm:ss,SSS}` отображает только время
- **%C** — Имя класса, в котором была вызвана какая-либо функция лога
- **%M** — Имя метода
- **%m** — Собственно, сообщение в лог
- **%F** — Имя файла
- **%l** — Номер строки
- **%r** — Количество миллисекунд, прошедших с начала работы программы
- **%t** — Имя потока
- **%n** — Переход на новую строку
- **%%** — Символ "%"

Root

Написать Appender-ы недостаточно для того, чтобы log4net начал писать логи. Нужно еще зарегистрировать их в корне лога, вот таким вот образом:

```
<root>
  <level value="DEBUG" />
  <appender-ref ref="FileAppender" />
</root>
```

`level` - уровень, сообщения ниже которого не выводятся ни в один лог. Приведенная запись значит, что в логи не будут выводиться сообщения типа DEBUG и INFO.

Использование log4net в разрабатываемых приложениях

В следующем примере показано, как можно ипользовать компопнет в своем приложении. Значимый код выделен курсивом.

```
using System;
using System.Threading;

using System.Configuration;

using log4net;
using log4net.Config;

class Deadlock {
    //объявляем статическую переменную, через которую работаем логгером и выводим сообщения в лог
    private static readonly ILog log = LogManager.GetLogger(typeof(Deadlock));
    static object lockerX = new object();
    static object lockerY = new object();

    static void Main() {
      //необходимо перед использованием сконфигурировать логгер для работы
      log4net.Config.XmlConfigurator.Configure();

      log.Info("Starting application");
```

```
      Thread tA = new Thread(GoA);
      log.Info("Invoking thread A...");
      tA.Start();

      Thread tB = new Thread(GoB);
      tB.Start();

      log.Info("Finishing application...");
      Console.WriteLine("Done...");
    }


    static void GoA() {
//...
    }

    static void GoB() {
//...
    }


}
```

Согласно документации на log4net интерфейс `ILog` представляет различные методы для вывода сообщений различных типов, представленных в таблице 1.

***Таблица** 1.* Основные функции, заданные в интрефейсе lLog

Функции	Описание	Пример

`Info(String message)`
`Debug(String message)`
`Warn(String message)`
`Error(String message)`
`Fatal(String message)`
	Вывод в лог сообщения `message`	`Logger.Log.Info("Sample log message");`

`InfoFormat(String format, params object[] args)`
 	
`DebugFormat(String format, params object[] args)`
 	
`WarnFormat(String format, params object[] args)`
 	
`ErrorFormat(String format, params object[] args)`
 	
`FatalFormat(String format, params object[] args)`
	Вывод в лог сообщения с параметрами. Аналогично выводу сообщения `String.Format(format, args)`.	`Logger.Log.DebugFormat("Key={0}, Value={1}", key, value);`

`Info(String message, Exception e)`
 	
`Debug(String message,`
	Вывод в лог сообщения с исключением. В лог помещается целиком	`try`
		`{`
		`a = b / c;`
		`}`

Exception e)
 	
Warn(String message, Exception e)
 	
Error(String message, Exception e)
 	
Fatal(String message, Exception e)
	текст исключения и stack trace.	catch (Exception e)
		{
		Logger .Log.Error("Exception catched while calculation.", e);
		}

В самом общем виде разработчик в состоянии и сам для своих собственных нужд разрабатывать модули, вынося в них необходимый функционал, а в дальнейшем подключая его и используя где есть в этом необходимость. Коротко этот процесс может быть описан следующим образом:

1. реализуется код обладающий требуемым функционалом. Обычно на языке высокого уровня он описывается в отдельном пространстве имен (а если надо то в целой иерархии пространств имен). Бизнес логика (функционал) представляется в соответствии с принятыми подходами и концепциями (например ООП).

2. совокупность модулей компилируется и упаковывается в бинарный dll-файл, который будет представлять библиотеку классов. Это можно сделать как средствами Visual Studio (см. рисунок 1), так и через компилятор командной строки, если сборку делать через него. Тогда строка запуска будет выглядеть примерно так:

csc /t:library <**имя файла-модуля(-ей)**>

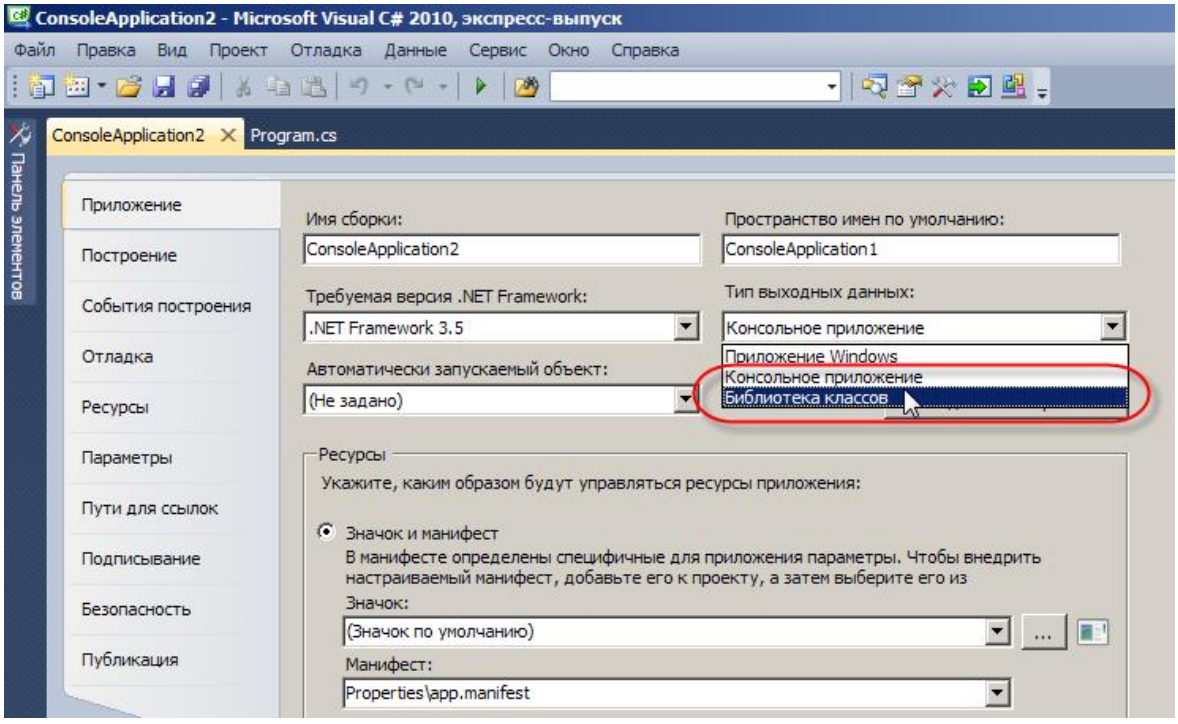

Рисунок 1. **Окно управления свойством проекта в среде** Visual Studio

3. в результате сборки библиотека на выходе получается dll модуль, который также, как и в случае ранее рассмотренного log4net, можно добавлять и в дальнейшем использовать в своих приложениях. К ранее представленному способу сборки приложения с использованием внешнего модуля (через Visual Studio) можно добавить и другой способ с использованием C# компилятора, который штатно всегда присутствует при установленной .NET Framework. Например:

csc /r:log4net.dll /r:ExtraMathLib.dll Deadlock.cs

здесь указано, что для сборки будет использовано два внешних модуля (параметр /r с именем файла модуля через двоеточие) и имя компилируемого модуля, где используется функционал внешних подключаемых модулей. На выходе работы компилятора при отсутствии ошибок получается исполняемый модуль.

Порядок выполнения работы

1. Реализовать программу на C# в соответствии с вариантом задания.
2. При реализации программы дополнить ее работу выводом служебной информации: время вызова; имя вызываемого метода; дополнительное описание. Весь вывод осуществлять с использованием класса логгера .NET-приложений log4net.
3. Опробовать работу программы.
4. Самостоятельно разработать библиотеку функций и скомпоновать ее в виде dll модуля, после чего подключить и оказать работу с ним в тестовом приложении.
5. Сформировать отчет о проделанной работе с выводами по работе.

Содержание отчета

1. Цель работы;
2. Вариант индивидуального задания;
3. Программа на языке C#, реализующая задание к работе;
4. Результаты запуска и выполнения программы;
5. Фрагмент лога работы приложения (не более страницы).
6. Описание самостоятельно разработанного модуля, экспортирующего функции и его исходный код, а также приложения, использующего функции модуля.
7. Выводы по работе.

Варианты индивидуальных заданий

Варианты индивидуальных заданий соответствуют списку заданий из лабораторной работы №2 с учетом дополнений, выполненных в работе №5

Контрольные вопросы:

1 . понятие внешних модулей и необходимость в них

2. назначение log4net