

Introduction à log4net

par Pascal ROZE ([Page perso](#)) ([Blog](#))

Date de publication : 13/12/2008

Dernière mise à jour :

Cet article est une introduction à log4net, un framework aidant le développeur à logger des informations vers diverses cibles.

Avant-Propos.....	3
Introduction.....	3
I - Présentation.....	3
I.1 - En bref.....	3
I.2 - Les loggers.....	3
I.3 - Les appenders.....	4
I.4 - Les filters.....	5
I.5 - Les layouts.....	5
II - Configurer log4net.....	6
II.1 - BasicConfigurator.....	6
II.2 - XmlConfigurator.....	7
II.3 - Les attributs de configuration.....	9
II.4 - Les fichiers de configuration.....	9
II.4.1 - Les fichiers .config.....	10
II.4.2 - Lire un fichier directement.....	10
II.5 - La syntaxe du fichier de configuration.....	11
II.5.1 - Les appenders.....	12
II.5.2 - Les filters.....	12
II.5.3 - Les layouts.....	13
II.5.4 - Le root logger.....	14
II.5.5 - Les loggers.....	14
II.5.6 - Les renderers.....	15
II.5.7 - Les paramètres.....	15
II.5.8 - Les ConversionPatterns.....	16
III - Un exemple d'utilisation de log4net.....	16
III.1 - Quelques bonnes pratiques avant de commencer.....	17
III.2 - Le fichier de configuration.....	17
III.3 - La classe Log.cs.....	22
III.4 - Utilisation des loggers.....	22
IV - Pour aller un peu plus loin.....	23
IV.1 - Si log4net ne logge pas ce que je lui demande.....	23
IV.2 - Si log4net logge trop.....	23
IV.3 - Visualiser les logs.....	23
IV.4 - Personnalisation.....	24
Conclusion.....	24
Liens utiles.....	24
Remerciements.....	24

Avant-Propos

Les sources de l'application web utilisée comme exemple dans le chapitre III sont disponibles [ici](#)

Introduction

Les développeurs écrivent très souvent des applications qui nécessitent une fonctionnalité de log. Ces applications formatent et loggent de l'information en réponse à des événements. Par exemple, les développeurs loggent généralement une information en cas d'exception ou d'échec de connexion à une base de données. Ils peuvent aussi logger une information afin de tracer en continu le fonctionnement de leur application.

Les logs se présentent en général sous la forme de fichiers texte, d'enregistrements en base ou de mails.

Mettre en place un tel système de log peut être long et contraignant. Log4net est là pour vous faciliter la tâche.

I - Présentation

I.1 - En bref

Log4net est un outil pour aider le développeur à logger des informations vers diverses cibles. Log4net est un port de l'excellent framework log4j vers .NET.

Le site officiel est  <http://logging.apache.org/log4net/>

Log4net a trois composants principaux : les loggers, appenders et layouts.

- **Les loggers:** Définissent le type de message ainsi que son niveau
- **Les appenders:** Destination du message
- **Les layouts:** Formatage du message

Ces trois types de composants travaillent ensemble pour permettre le log des messages selon leur type et niveau, contrôler leur format et leur journalisation. Ces composants sont aidés par les **filters** qui commandent les actions de l'append et les **object renderers** qui transforment des objets en chaînes de caractères.

I.2 - Les loggers

L'avantage principal de toute API de log face à un trivial `System.Console.WriteLine` réside dans la possibilité de désactiver/activer le log de certains messages en fonction de leur état et niveau. Cette fonctionnalité implique une hiérarchisation selon des critères choisis par le développeur.

Des niveaux peuvent être assignés aux loggers. Les niveaux sont des instances de la classe `log4net.Core.Level`. Les niveaux suivants sont définis par ordre de priorité croissante :




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

I.3 - Les appenders

Log4net permet de logger vers de multiples destinations via l'utilisation d'appenders.

La liste des appenders disponibles est la suivante :

Type	Description
 AdoNetAppender	Ecriture des évènements dans une base de données à l'aide de requêtes ou de procédures stockées.
 AnsiColorTerminalAppender	Ecrit en couleur les évènements vers un terminal window ANSI.
 AspNetTraceAppender	Ecriture des évènements vers l'ASP trace context.
 BufferingForwardingAppender	Mise en cache avant transmission aux appenders enfants.
 ColoredConsoleAppender	Ecriture des évènements dans la console de l'application. Possibilité de configurer le style, la police et la couleur du message pour chaque niveau. Les évènements sont dirigés vers le flux de sortie ou d'erreur standard.
 ConsoleAppender	Identique au précédent sans la personnalisation du message.
 EventLogAppender	Ecriture des évènements dans l'Event Log Windows.
 FileAppender	Ecriture des évènements dans un fichier.
 ForwardingAppender	Transmet les évènements aux appenders enfants.
 LocalSyslogAppender	Ecriture des évènements dans le syslog service (UNIX seulement).
 MemoryAppender	Stocke les évènements dans un buffer mémoire.
 NetSendAppender	Ecriture des évènements dans Windows Messenger service.
 OutputDebugStringAppender	Ecriture des évènements dans le debugger. Si l'application ne possède pas de debugger, le debugger système affiche les chaînes de caractères. Si le debugger système est inactif les messages sont ignorés.
 RemoteSyslogAppender	Ecriture des évènements vers un remote syslog service en utilisant UDP.
 RemotingAppender	Ecriture des évènements vers un remoting sink en utilisant .NET remoting.
 RollingFileAppender	Ecriture des évènements dans un fichier. Possibilité de log sur plusieurs fichiers en fonction de la date et de l'heure.
 SmtpAppender	Envoi des évènements sur une adresse mail.
 SmtpPickupDirAppender	Ecriture des messages SMTP dans un pickup directory. Ces fichiers peuvent être







	lus et envoyés par un agent SMTP (IIS SMTP agent).
 TelnetAppender	Etablir une connexion via Telnet pour recevoir les évènements.
 TraceAppender	Ecriture des évènements dans .NET trace system.
 UdpAppender	Ecriture des évènements en tant que datagrammes UDP vers un remote host ou un multicast group en utilisant un client UDP.

Il est possible d'associer plusieurs appenders à un logger afin d'écrire la même information en plusieurs endroits différents. On peut par exemple souhaiter que lorsqu'une exception est levée dans notre application, l'information soit écrite dans un fichier texte mais également envoyer par mail.

I.4 - Les filters

Les appenders peuvent filtrer les évènements qui leurs sont fournis. Les filtres peuvent être indiqués dans la configuration pour permettre un contrôle précis des évènements qui sont inscrits par les différents appenders. La façon la plus simple est d'indiquer un seuil sur l'appender. Seuls les évènements qui ont un niveau supérieur ou égal au seuil seront inscrits.








Les filtres suivants sont inclus dans le package de log4net :

Type	Description
 DenyAllFilter	Désactive tous les évènements.
 LevelMatchFilter	Correspondance exacte avec le niveau de l'évènement.
 LevelRangeFilter	Au moins une correspondance avec un des niveaux de l'évènement.
 LoggerMatchFilter	Correspondance avec le nom du logger.
 PropertyFilter	Correspondance avec une propriété.
 StringMatchFilter	Correspondance avec une partie du message.

I.5 - Les layouts

Le plus souvent, les utilisateurs souhaitent adapter non seulement la destination des informations à logger mais également leur format de sortie. Ceci est accompli en associant un layout à un appender. Le layout est en charge du formatage du message selon les souhaits de l'utilisateur, tandis qu'un appender prend soin d'envoyer le message composé à sa destination. Le PatternLayout, permet à l'utilisateur d'indiquer le format de message parmi des modèles de conversion semblables à ceux de la fonction printf du langage C.

Les layouts suivants sont inclus dans le package de log4net :

Type	Description
 ExceptionLayout	Rendu équivalent à celle d'une exception.
 PatternLayout	Formate le message selon les flags précisés.
 RawTimeStampLayout	Extrait le timestamp de l'événement.
 RawUtcTimeStampLayout	Extrait le timestamp de l'événement formaté en Universal Time.
 SimpleLayout	Format simple : [level] - [message]
 XmlLayout	Format équivalent à un élément XML.
 XmlLayoutSchemaLog4j	Format équivalent à un élément XML qui applique le dtd log4j.

II - Configurer log4net

Mettre en place un système de log dans une application est une action couteuse en temps. Les observations montrent qu'environ 4% du code est dédié au logging.

On comprend facilement l'importance de pouvoir gérer ces logs de manière dynamique, sans avoir à modifier le code existant. Cela peut s'avérer particulièrement utile lorsque l'application est en production. Si l'on détecte un problème de fonctionnement, il suffit d'augmenter le niveau des logs afin de le situer, et ce, juste en modifiant la configuration.

Log4net est entièrement configurable via le code. Cependant, il est bien plus flexible d'utiliser des fichiers de configuration. A l'heure actuelle, ces derniers se présentent sous la forme de fichiers XML.

La configuration de log4net étant, pour moi, la partie la plus importante de sa mise en place, je vais y consacrer un chapitre complet.

Log4net possède 2 configurateurs : le BasicConfigurator et le XmlConfigurator.

II.1 - BasicConfigurator

Voyons l'utilisation du BasicConfigurator via une application console très simple.

```
using System;
using log4net;
using log4net.Config;

public class Program
{
    // On définit une variable logger static qui référence l'instance du logger nommé Program
    private static readonly ILog log = LogManager.GetLogger(typeof (Program));

    static void Main(string[] args)
    {
        // On charge la configuration de base qui log dans la console
        BasicConfigurator.Configure();

        log.Info("Démarrage de l'application.");
        var task = new Task();
        task.Run();
        log.Info("Fin de l'application.");
        Console.ReadLine();
    }
}

public class Task
{
    private static readonly ILog log = LogManager.GetLogger(typeof (Task));
```


```
public void Run()  
{  
    log.Debug("Lancement de la tâche");  
}  
}
```

On commence par importer les classes nécessaires à l'utilisation de log4net. On définit ensuite une variable logger static qui porte le même nom que la classe où elle se trouve.

L'appel de la méthode `BasicConfigurator.Configure()` charge une configuration par défaut de log4net. Cette méthode ajoute un `ConsoleAppender` au logger root. L'information sera formaté suivant le pattern suivant (plus de détails sur les `ConversionPatters`) : « %-4timestamp [%thread] %-5level %logger %ndc - %message%newline ». Par défaut, le niveau du logger root est `Level.DEBUG`.

La sortie de Program est :


`BasicConfigurator` ne permet que des configurations très simples de log4net. Seulement un appender peut être configuré en utilisant ce configurateur et toutes les informations seront loggées via cet appender.

Pour plus d'informations,  [BasicConfigurator Class](#).

L'exemple précédent produit toujours la même sortie. Heureusement, il est facile de modifier la classe `Program` afin de contrôler le format des logs pendant le runtime via le configurateur `XmlConfigurator`.

II.2 - XmlConfigurator

Le `XmlConfigurator` permet de charger une configuration définie dans un fichier séparé. Cela permet de modifier la façon dont les informations sont loggées sans avoir à recompilier l'application.

Pour plus d'informations,  [XmlConfigurator Class](#).

Reprenons l'application précédente et intégrons-y le `XmlConfigurator`.

La class `Program` ressemble maintenant à :

```
using System;  
using log4net;  
using log4net.Config;  
  
public class Program  
{  
    // On définit une variable logger static qui référence l'instance du logger nommé Program  
    private static readonly ILog log = LogManager.GetLogger(typeof (Program));  
  
    static void Main(string[] args)  
    {  
        // On récupère le chemin du fichier de config  
        var configFile = Directory.GetCurrentDirectory() + @"\log4net.config";  
  
        // On remplace le BasicConfigurator par le XmlConfigurator  
        // et on charge la configuration définie dans le fichier log4net.config  
        XmlConfigurator.Configure(new FileInfo(configFile));  
  
        log.Info("Démarrage de l'application.");  
        var task = new Task();  
        task.Run();  
        log.Info("Fin de l'application.");  
    }  
}
```

```
        Console.ReadLine();  
    }  
}
```

Voilà un exemple de fichier de config qui donne exactement le même résultat que l'exemple précédent avec le BasicConfigurator.

```
<log4net>  
  <!-- A1 est un ConsoleAppender -->  
  <appender name="A1" type="log4net.Appender.ConsoleAppender">  
  
    <!-- A1 utilise un PatternLayout -->  
    <layout type="log4net.Layout.PatternLayout">  
      <conversionPattern value="%-4timestamp [%thread] %-5level %logger %ndc - %message%newline" />  
    </layout>  
  </appender>  
  
  <!-- On définit le logger root au niveau DEBUG et on l'associe à l'appender A1 -->  
  <root>  
    <level value="DEBUG" />  
    <appender-ref ref="A1" />  
  </root>  
</log4net>
```

Supposons maintenant que nous souhaitons que le logger « Task » (utilisé dans la classe Task) ne logge plus que des informations dont le niveau est supérieur ou égal au niveau WARN.

Le fichier de configuration suivant permet cela :

```
<log4net>  
  <!-- A1 est un ConsoleAppender -->  
  <appender name="A1" type="log4net.Appender.ConsoleAppender">  
  
    <!-- A1 utilise un PatternLayout -->  
    <layout type="log4net.Layout.PatternLayout">  
      <conversionPattern value="%-4timestamp [%thread] %-5level %logger %ndc - %message%newline" />  
    </layout>  
  </appender>  
  
  <!-- On définit le logger root au niveau DEBUG et on l'associe à l'appender A1 -->  
  <root>  
    <level value="DEBUG" />  
    <appender-ref ref="A1" />  
  </root>  
  <!-- On ne loggue que les informations dont le niveau est supérieur à WARN pour le logger Task -->  
  <logger name="Task">  
    <level value="WARN"/>  
  </logger>  
</log4net>
```

La sortie de l'application est maintenant :

Voici un autre exemple de fichier de configuration qui utilise plusieurs appenders :

```
<log4net>  
  <appender name="A1" type="log4net.Appender.ConsoleAppender">  
    <layout type="log4net.Layout.PatternLayout">  
      <conversionPattern value="%-4timestamp [%thread] %-5level %logger %ndc - %message%newline" />  
    </layout>  
  </appender>  
  <appender name="RollingFile" type="log4net.Appender.RollingFileAppender">  
    <file value="example.log" />  
    <appendToFile value="true" />  
    <maximumFileSize value="100KB" />  
  </appender>  
</log4net>
```



```
<maxSizeRollBackups value="2" />
<layout type="log4net.Layout.PatternLayout">
  <conversionPattern value="%level %thread %logger - %message%newline" />
</layout>
</appender>
<root>
  <level value="DEBUG" />
  <appender-ref ref="A1" />
  <appender-ref ref="RollingFile" />
</root>
</log4net>
```

On remarque qu'un second appender a été associé au logger root.

Les informations seront maintenant également loggées dans un fichier exemple.log.

Dès que le fichier atteindra 100KB, un nouveau fichier exemple.log sera créé et l'ancien sera automatiquement renommé en exemple.log.1.

Comme vous avez pu le voir, on a modifié dynamiquement la façon dont les informations sont loggées.

II.3 - Les attributs de configuration

Log4net peut être configuré en utilisant des attributs au niveau de l'assembly plutôt qu'au niveau du code.

La classe  **XmlConfiguratorAttribute** permet au XmlConfigurator d'être configuré en utilisant les propriétés suivantes :

- **ConfigFile** : Si spécifié, correspond au nom du fichier de configuration utilisé par le XmlConfigurator. Le chemin est relatif au dossier de base de l'application (AppDomain.CurrentDomain.BaseDirectory). Cette propriété ne peut pas être utilisée conjointement avec la propriété ConfigFileExtension.
- **ConfigFileExtension** : Si spécifié, correspond à l'extension du fichier de configuration. Le nom du fichier de l'assembly est utilisé comme base à laquelle on ajoute l'extension. Par exemple, si l'assembly est chargée depuis un fichier MyApp.exe et que l'extension est log4net, alors le nom du fichier de configuration sera MyApp.exe.log4net. Cette propriété ne peut pas être utilisée conjointement avec la propriété ConfigFile.
- **Watch** : Si ce flag est à true, le framework « surveillera » le fichier de configuration et rechargera la configuration à chaque modification du fichier.

Cet attribut ne peut être utilisé qu'au niveau de l'assembly et ne peut être utilisé qu'une fois par assembly.

Exemple d'utilisation :

```
[assembly: log4net.Config.XmlConfigurator(ConfigFile = "Config/Log4Net.config", Watch = true)]
public static class Log
{
}
```

II.4 - Les fichiers de configuration

Généralement, le paramétrage de log4net est spécifié via un fichier de configuration. Ce fichier peut être lu de 2 façons :

- En utilisant l'API .NET System.Configuration
- En lisant directement le fichier.

II.4.1 - Les fichiers .config

L'API System.Configuration est disponible seulement si le paramétrage se trouve dans le fichier de configuration de l'application (app.config ou web.config). Le principal avantage de cette méthode est que cela nécessite moins de permissions pour lire le fichier de configuration que de lire un fichier directement (pas besoin de donner des droits au user ASPNET par exemple). La seule façon de configurer une application en utilisant System.Configuration est d'appeler la méthode log4net.Config.XmlConfigurator.Configure() ou la méthode log4net.Config.XmlConfigurator.Configure(ILoggerRepository).

Afin d'intégrer la configuration dans le fichier .config, il faut identifier le nom de la section en ajoutant un élément à la section configSections. La section doit spécifier le log4net.Config.Log4NetConfigurationSectionHandler qui sera utilisé lors du parsing de la section de configuration. Ci-dessous un exemple :

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

<log4net>
  <appender name="SmtpAppender" type="log4net.Appender.SmtpAppender,log4net">
    <to value="destinataire@mail.com" />
    <from value="expediteur@mail.com" />
    <subject value="Alerte, tout a pété" />
    <smtpHost value="smtp.com" />
    <bufferSize value="0" />
    <threshold value="ERROR" />
    <layout type="log4net.Layout.PatternLayout,log4net">
      <conversionPattern value="LEVEL: %level %newlineDATE: %date  LOGGER: %logger %newline%newline
%message" />
    </layout>
  </appender>
  <root>
    <level value="ALL" />
    <appender-ref ref="SmtpAppender" />
  </root>
</log4net>
```

Lorsque vous utilisez le fichier .config pour spécifier la configuration, le nom de la section et le nom de l'élément XML doivent être log4net.

II.4.2 - Lire un fichier directement

Le XmlConfigurator peut lire directement n'importe quel fichier XML et l'utiliser pour configurer log4net, y compris les fichiers .config. Le fichier à lire pour charger la configuration peut être spécifié en utilisant n'importe laquelle des méthodes de log4net.Config.XmlConfigurator qui accepte un objet System.IO.FileInfo en paramètre. Les méthodes ConfigureAndWatch peuvent être utilisées pour surveiller les modifications apportées au fichier de configuration et automatiquement reconfigurer log4net.

De plus, il est possible de spécifier le fichier de configuration en utilisant log4net.Config.XmlConfiguratorAttribute.

La configuration est lue depuis le noeud log4net présent dans le fichier. Un seul élément log4net est autorisé par fichier mais il peut se trouver n'importe où dans la hiérarchie XML. Il peut par exemple être le nœud racine :

```
<?xml version="1.0" encoding="utf-8" ?>
<log4net>
  <appender name="Console" type="log4net.Appender.ConsoleAppender">
    <layout type="log4net.Layout.PatternLayout">
      <conversionPattern value="%-4timestamp [%thread] %-5level %logger %ndc - %message%newline" />
    </layout>
  </appender>
  <root>
    <level value="DEBUG" />
  </root>
</log4net>
```

```
<appender-ref ref="Console" />
</root>
</log4net>
```

II.5 - La syntaxe du fichier de configuration

Ce paragraphe a pour but de définir la syntaxe acceptée par log4net.Config.XmlConfigurator.

Reprenons l'exemple précédent.

```
<?xml version="1.0" encoding="utf-8" ?>
<log4net>
  <appender name="Console" type="log4net.Appender.ConsoleAppender">
    <layout type="log4net.Layout.PatternLayout">
      <conversionPattern value="%-4timestamp [%thread] %-5level %logger %ndc - %message%newline" />
    </layout>
  </appender>
</root>
  <level value="DEBUG" />
  <appender-ref ref="Console" />
</root>
</log4net>
```

L'élément racine doit être <log4net>. Cela ne signifie pas que cet élément ne peut pas être contenu dans un autre élément XML.

L'élément <log4net> accepte les attributs suivants:

Attribut	Description
debug	Optionnel. La valeur peut être true ou false. La valeur par défaut est false. Mettre la valeur à true pour activer le debug interne de log4net.
update	Optionnel. La valeur peut être Merge ou Overwrite. La valeur par défaut est Merge. Mettre la valeur à Overwrite pour réinitialiser la configuration avec la config actuelle.
threshold	Optionnel. La valeur doit être l'un des niveaux de log. La valeur par défaut est ALL. Modifier la valeur pour limiter les messages qui sont loggés dans l'application sans tenir compte du logger qui log le message.

L'élément <log4net> accepte les éléments enfants suivant :

Élément	Description
appender	0 élément ou plus. Définit un appender
logger	0 élément ou plus. Définit la configuration d'un logger
renderer	0 élément ou plus. Définit un objet renderer
root	Optionnel. 1 élément au maximum. Définit la configuration du logger racine
param	0 élément ou plus. Paramètres spécifiques au référentiel

II.5.1 - Les appenders

Les appenders peuvent être uniquement définis en tant que noud enfant de l'élément <log4net>. Chaque appender doit être nommé de façon unique. Le type de l'appender doit être spécifié.

L'exemple suivant porte sur un appender de type log4net.Appender.ConsoleAppender.

```
<appender name="Console" type="log4net.Appender.ConsoleAppender">
  <layout type="log4net.Layout.PatternLayout">
    <conversionPattern value="%-4timestamp [%thread] %-5level %logger %ndc - %message%newline" />
  </layout>
</appender>
```

L'élément <appender> accepte les attributs suivants :

Attribut	Description
name	Attribut obligatoire. La valeur doit être une chaîne de caractères. Le nom doit être unique parmi tous les appenders définis dans le fichier de configuration. Le nom est utilisé par l'élément <appender-ref> d'un logger pour référencer un appender.
type	Attribut obligatoire. La valeur est le type de l'appender.

L'élément <appender> accepte les éléments enfants suivant :

Élément	Description
appender-ref	0 élément ou plus. Autorise l'appender à référencer un autre appender. Non supporté pour tous les appenders.
filter	0 élément ou plus. Définit les filtres utilisés par cet appender.
layout	Optionnel. 1 élément au maximum. Définit le layout utilisé par cet appender.
param	0 élément ou plus. Paramètres spécifiques à l'appender.

Des exemples de configuration d'appenders sont disponibles  [ici](#).

II.5.2 - Les filters.

Les éléments filters peuvent être uniquement définis en tant que noud enfant des éléments <appender>.

L'élément <filter> accepte l'attribut suivant :

Attribute	Description
type	Attribut obligatoire. La valeur doit être le nom de ce filter.

L'élément <filter> accepte l'élément enfant suivant :

Elément	Description
param	0 élément ou plus. Paramètres spécifiques au filter.

Les filters forment une chaîne à travers laquelle l'événement (de log) doit passer. Chaque filter de la chaîne peut accepter l'événement et stopper le traitement, rejeter l'événement et stopper le traitement, ou autoriser l'événement à passer au filter suivant. Si l'événement atteint la fin de la chaîne sans être rejeté, il est simplement accepté et il sera loggé.

Exemple :

```
<filter type="log4net.Filter.LevelRangeFilter">
  <levelMin value="INFO" />
  <levelMax value="FATAL" />
</filter>
```

Ce filter va rejeter tout événement qui aura un niveau inférieur à INFO ou supérieur à FATAL. Tous les événements compris entre INFO et FATAL seront loggés.

Si vous voulez seulement autoriser les messages qui contiennent la chaîne de caractères 'database' alors vous devrez spécifier les filters suivants :

```
<filter type="log4net.Filter.StringMatchFilter">
  <stringToMatch value="database" />
</filter>
<filter type="log4net.Filter.DenyAllFilter" />
```

Le premier filter va rechercher la chaîne de caractères 'database' dans le message de l'événement. Si le texte est trouvé, le filter va accepter le message et le traitement va s'arrêter, le message sera loggé. Si le texte n'est pas trouvé, l'événement sera passé au filter suivant. S'il n'y a pas de filter suivant, l'événement sera implicitement accepté et le message sera loggé. Etant donné que nous voulons pas loggés les messages ne contenant pas la chaîne 'database', nous devons utiliser le filter log4net.Filter.DenyAllFilter qui va rejeter tous les événements qui l'atteindront. Ce filter est uniquement utile à la fin de la chaîne de filters.

Si nous voulons autoriser les messages qui contiennent les chaînes de caractères 'database' ou 'ldap', nous pouvons utiliser les filters suivants :

```
<filter type="log4net.Filter.StringMatchFilter">
  <stringToMatch value="database" />
</filter>
<filter type="log4net.Filter.StringMatchFilter">
  <stringToMatch value="ldap" />
</filter>
<filter type="log4net.Filter.DenyAllFilter" />
```

II.5.3 - Les layouts

Les éléments layout peuvent être uniquement définis en tant que noud enfant des éléments <appender>.

L'élément <layout> accepte l'attribut suivant :

Attribut	Description
type	Attribut obligatoire. La valeur doit être le nom de ce filter.

L'élément <layout> accepte l'élément enfant suivant :

Elément	Description
param	0 élément ou plus. Paramètres spécifiques au layout.

L'exemple suivant montre comment configurer un layout qui utilise log4net.Layout.PatternLayout.

```
<layout type="log4net.Layout.PatternLayout">
  <conversionPattern value="%date [%thread] %-5level %logger [%ndc] - %message%newline" />
</layout>
```

II.5.4 - Le root logger

Seulement un élément root logger peut être défini et il doit être un élément enfant du nœud <log4net>. Le root logger est la racine de la hiérarchie des loggers. Tous les loggers héritent de ce logger.

Un exemple de root logger :

```
<root>
  <level value="INFO" />
  <appender-ref ref="ConsoleAppender" />
</root>
```

L'élément <root> n'accepte pas d'attributs.

L'élément <root> accepte les éléments enfants suivant :

Elément	Description
appender-ref	0 élément ou plus. Permet au logger de référencer des appenders via leur nom.
level	Optionnel. 1 élément au maximum. Définit le niveau de log pour ce logger. Ce logger acceptera seulement les événements qui auront un niveau égal ou supérieur.
param	0 élément ou plus. Paramètres spécifiques au logger.

II.5.5 - Les loggers

Les éléments loggers peuvent être uniquement définis en tant que nœud enfant de l'élément <log4net>.

Un exemple :

```
<logger name="LoggerName">
  <level value="DEBUG" />
  <appender-ref ref="ConsoleAppender" />
</logger>
```

L'élément <logger> accepte les attributs suivants :

Attribut	Description
name	Attribut obligatoire. Représente le nom du logger.
additivity	Optionnel. La valeur peut être true ou false. La valeur par défaut est true. Mettre la valeur à false pour empêcher ce logger d'hériter

	des appenders définis dans les loggers parents.
--	---

L'élément <logger> accepte les éléments enfants suivant :

Élément	Description
appender-ref	0 élément ou plus. Permet au logger de référencer des appenders via leur nom.
level	Optionnel. 1 élément au maximum. Définit le niveau de log pour ce logger. Ce logger acceptera seulement les événements qui auront un niveau égal ou supérieur
param	0 élément ou plus. Paramètres spécifiques au logger.

II.5.6 - Les renderers

Les éléments renderers peuvent être uniquement définis en tant que noud enfant de l'élément <log4net>.

Un exemple :

```
<renderer renderingClass="MyClass.MyRenderer" renderedClass="MyClass.MyFunkyObject" />
```

L'élément <renderer> accepte les attributs suivant :

Attribut	Description
renderingClass	Obligatoire. La valeur doit être le nom du type de ce renderer.
renderedClass	Obligatoire. La valeur doit être le nom du type de la cible de ce renderer.

L'élément <renderer> n'accepte pas d'élément enfant.

II.5.7 - Les paramètres

Les éléments paramètres peuvent être enfant de nombreux éléments.

Un exemple :

```
<param name="ConversionPattern" value="%date [%thread] %-5level %logger [%ndc] - %message%newline" />
```

L'élément <param> accepte les attributs suivant :

Attribut	Description
name	Obligatoire. Doit être le nom du paramètre à définir dans l'objet parent.
value	Optionnel. L'un des attributs value ou type doit être spécifié. La valeur est une chaîne de caractères qui peut être convertie en la valeur du paramètre.
type	Optionnel. L'un des attributs value ou type doit être spécifié. La valeur de cet attribut

	est le nom d'un type pour créer et définir la valeur du paramètre.
--	--

L'élément <param> accepte l'élément enfant suivant :

Élément	Description
param	0 élément ou plus. Paramètres spécifiques au paramètre.

Un exemple de param qui utilise des éléments param imbriqués :

```
<param name="evaluator" type="log4net.spi.LevelEvaluator">
  <param name="Threshold" value="WARN"/>
</param>
```

II.5.8 - Les ConversionPatterns

L'élément conversionPattern définit le template du contenu loggé. Un conversionPattern est composé de texte littéral et d'expressions de contrôle de format appelées conversion specifiers. Chaque conversion specifier commence par le signe % et est suivi par un format modifier optionnel puis par un nom de conversion pattern. Le nom du conversionPattern spécifie le type de données, par exemple, le nom du logger, la date, le nom du thread. Le format modifier permet de spécifier une longueur de texte, un padding ou une justification. Voyons un exemple simple en prenant le conversionPattern suivant :


```
%-5level %date %message%newline.
```

Un appel aux méthodes Debug("message 1") puis Info("message 2") générera la sortie :

```
DEBUG 2008-12-04 23:58:23,484 message1
```

```
INFO 2008-12-04 23:58:23,486 message 2
```

Dans l'exemple ci-dessus, le conversion specifier %-5level signifie que le niveau de l'événement loggé sera justifié à gauche d'une longueur de 5 caractères.

Vous trouverez ici la liste complète des conversionPatterns et des format modifiers :  <http://logging.apache.org/log4net/release/sdk/log4net.Layout.PatternLayout.html>

Remarque importante : Que se passe-t-il si le fichier de configuration est invalide ou mal formaté ?

Un tel fichier de configuration n'empêchera pas l'application de fonctionner correctement. Par contre, log4net ne sera pas opérationnel et aucun log ne pourra être effectué. Un fichier de configuration invalide est facilement repérable. Pour cela, il suffit d'activer les logs internes de log4net (voir la marche à suivre). On verra alors apparaître l'erreur suivante :

log4net:ERROR XmlConfigurator : Error while loading XML configuration , avec des détails complémentaires concernant l'erreur tels que le numéro de ligne

III - Un exemple d'utilisation de log4net

Comme un exemple vaut mieux qu'un long discours, dans ce chapitre, nous allons voir un exemple d'utilisation de log4net au sein d'un projet web. Nous verrons comment associer différents appenders à un même logger, comment configurer les informations à logger, comment jouer avec les niveaux de logs .

Les sources du projet sont téléchargeables via le lien se trouvant à la fin de ce document.

III.1 - Quelques bonnes pratiques avant de commencer.

- La construction de chaîne de caractères est très consommatrice en ressources. Afin de ne pas en construire inutilement, vérifier que le logger est bien configuré avec un niveau de log suffisant :

```
if (Log.MainLogger.IsDebugEnabled)
{
    //construction de la chaîne de caractères consommatrice en ressources
    // TODO
    Log.MainLogger.Debug (msg) ;
}
```

- Faciliter la visualisation des logs : mettre ce qui est fixe au début (comme le niveau du log ou le nom du logger), ce qui est variable à la fin (comme le message loggé).

Exemple : [DEBUG] - TaskLogger - 09/12/2008 00:12:54 - Construction de l'objet Task

III.2 - Le fichier de configuration

Attaquons dans le vif du sujet avec le fichier de configuration de log4net. Comme vu précédemment, ce fichier permet principalement de définir quelles informations seront loggées, de quelle manière, vers quel support.

Voilà le fichier de configuration dans son intégralité :

```
<log4net debug="true" >
  <appender name="AspNetTraceAppender" type="log4net.Appender.AspNetTraceAppender" >
    <threshold value="ALL"/>
    <layout type="log4net.Layout.PatternLayout">
      <conversionPattern value="[%level] %date %logger - %message" />
    </layout>
  </appender>
  <appender name="EventLogAppender" type="log4net.Appender.EventLogAppender" >
    <applicationName value="Log4net.WebApp" />
    <filter type="log4net.Filter.LevelRangeFilter">
      <levelMin value="DEBUG" />
      <levelMax value="INFO" />
    </filter>
    <layout type="log4net.Layout.PatternLayout">
      <conversionPattern value="[%level] %date %logger - %message" />
    </layout>
  </appender>
  <appender name="DatabaseAppender" type="log4net.Appender.AdoNetAppender">
    <bufferSize value="0" />
    <threshold value="ERROR" />
    <connectionString value="Provider=SQLOLEDB;Data Source=PASCAL-PC
\SQLEXPRESS;Database=Logs;Trusted_Connection=yes" />
    <commandText value="INSERT INTO Logs(Date, Level, Who, Message) VALUES(?, ?, ?, ?)" />
    <parameter>
      <parameterName value="@Date" />
      <dbType value="DateTime" />
      <size value="255" />
      <layout type="log4net.Layout.PatternLayout">
        <conversionPattern value="%date{yyyy}'-'MM'-'dd HH':'mm}" />
      </layout>
    </parameter>
    <parameter>
      <parameterName value="@Level" />
      <dbType value="AnsiString" />
      <size value="10" />
      <layout type="log4net.Layout.PatternLayout">
        <conversionPattern value="%level" />
      </layout>
    </parameter>
  </appender>
</log4net>
```

```

</parameter>
<parameter>
  <parameterName value="@Who" />
  <dbType value="AnsiString" />
  <size value="100" />
  <layout type="log4net.Layout.PatternLayout">
    <conversionPattern value="%property{log_who}" />
  </layout>
</parameter>
<parameter>
  <parameterName value="@Message" />
  <dbType value="AnsiString" />
  <size value="500" />
  <layout type="log4net.Layout.PatternLayout">
    <conversionPattern value="%message" />
  </layout>
</parameter>
</appender>
<appender name="SmtpAppender" type="log4net.Appender.SmtpAppender, log4net">
  <to value="votre adresse mail" />
  <from value="logger@webapp.com" />
  <subject value="Alerte dans AppWeb" />
  <smtpHost value="l'adresse du smtp" />
  <bufferSize value="0" />
  <threshold value="DEBUG" />
  <layout type="log4net.Layout.PatternLayout, log4net">
    <conversionPattern value="LEVEL: %level %newlineDATE: %date %newlineLOGGER: %logger %newline%newline%message" />
  </layout>
</appender>
<appender name="RollingFile" type="log4net.Appender.RollingFileAppender">
  <file value="Log/Log.txt"/>
  <threshold value="INFO"/>
  <appendToFile value="true"/>
  <rollingStyle value="Date"/>
  <datePattern value="yyyyMMdd"/>
  <layout type="log4net.Layout.PatternLayout">
    <conversionPattern value="*%-10level %-30date %-25logger %message %newline"/>
  </layout>
</appender>
<root>
  <level value="ALL"/>
</root>
<logger name="MonitoringLogger">
  <level value="ALL"/>
  <appender-ref ref="RollingFile"/>
  <appender-ref ref="DatabaseAppender"/>
  <appender-ref ref="AspNetTraceAppender"/>
  <appender-ref ref="EventLogAppender"/>
</logger>
<logger name="ExceptionLogger">
  <level value="ERROR"/>
  <appender-ref ref="SmtpAppender"/>
  <appender-ref ref="RollingFile"/>
</logger>
</log4net>

```

Détaillons plus précisément le contenu de ce fichier en commençant par les appenders.

On peut voir 5 appenders différents, c'est-à-dire qu'on pourra logger vers 5 destinations différentes :

- **SmtpAppender**

```

<appender name="SmtpAppender" type="log4net.Appender.SmtpAppender, log4net">
  <to value="votre adresse mail" />
  <from value="logger@webapp.com" />
  <subject value="Alerte dans AppWeb" />
  <smtpHost value="l'adresse du smtp" />
  <bufferSize value="0" />
  <threshold value="DEBUG" />

```

```
<layout type="log4net.Layout.PatternLayout,log4net">
  <conversionPattern value="LEVEL: %level %newlineDATE: %date %newlineLOGGER: %logger %newline%newline%message" />
</layout>
</appender>
```

Ici, on a créé un appender de type `SmtpAppender` dont le but est d'envoyer un mail.

Les éléments **to**, **from**, **smtpHost** et **subject** se passent de commentaires.

L'élément **threshold** définit le niveau minimum auquel doit être associé l'événement pour être logué. Ici, tout événement d'un niveau égal ou supérieur à `DEBUG` sera loggé.

L'élément **bufferSize** définit la taille du buffer cyclique utilisé. Par défaut, la valeur est 512K, autrement dit, un mail sera envoyé dès que le buffer atteindra une taille de 512K. Ici, une valeur nulle implique un envoi de mail pour chaque événement.

L'élément **conversionPattern** définit le template de l'information loggée. Ici, on loggera le niveau de l'événement, la date/heure actuelle, le nom du logger et le message.

- **DatabaseAppender**

```
<appender name="DatabaseAppender" type="log4net.Appender.AdoNetAppender">
  <bufferSize value="0" />
  <threshold value="ERROR" />
  <connectionString value="Provider=SQLOLEDB;Data Source=PASCAL-PC
\SQLEXPRESS;Database=Logs;Trusted_Connection=yes" />
  <commandText value="INSERT INTO Logs(Date, Level, Who, Message) VALUES(?, ?, ?, ?)" />
  <parameter>
    <parameterName value="@Date" />
    <dbType value="DateTime" />
    <size value="255" />
    <layout type="log4net.Layout.PatternLayout">
      <conversionPattern value="%date" />
    </layout>
  </parameter>
  <parameter>
    <parameterName value="@Level" />
    <dbType value="AnsiString" />
    <size value="10" />
    <layout type="log4net.Layout.PatternLayout">
      <conversionPattern value="%level" />
    </layout>
  </parameter>
  <parameter>
    <parameterName value="@Who" />
    <dbType value="AnsiString" />
    <size value="100" />
    <layout type="log4net.Layout.PatternLayout">
      <conversionPattern value="%property{log_who}" />
    </layout>
  </parameter>
  <parameter>
    <parameterName value="@Message" />
    <dbType value="AnsiString" />
    <size value="500" />
    <layout type="log4net.Layout.PatternLayout">
      <conversionPattern value="%message" />
    </layout>
  </parameter>
</appender>
```

Cet appender sert à logger en base.

bufferSize = 0 indique que l'on logge à chaque événement.

threshold à la valeur ERROR donc tout événement ayant un niveau strictement inférieur à ERROR ne sera pas loggé.

connectionString correspond à la chaîne de connexion à la base.

commandText correspond à la requête paramétrée exécutée lors du log.

Les éléments **parameters** correspondent aux paramètres de la requête.

 **Illes doivent apparaître dans le même ordre que dans la requête.**

Remarque : pour le paramètre log_who, on utilise la syntaxe :

```
<conversionPattern value="%property{log_who}" />
```

Cela signifie que l'on utilise une property de log4net que l'on définit à la main dans le code ()

- **AspNetTraceAppender**

```
<appender name="AspNetTraceAppender" type="log4net.Appender.AspNetTraceAppender" >
  <threshold value="ALL"/>
  <layout type="log4net.Layout.PatternLayout">
    <conversionPattern value="[%level] %date %logger - %message" />
  </layout>
</appender>
```

Cet appender permet d'écrire dans la trace d'ASP.NET.

- **EventLogAppender**

```
<appender name="EventLogAppender"
type="log4net.Appender.EventLogAppender" >
  <applicationName value="Log4net.WebApp" />
  <filter type="log4net.Filter.LevelRangeFilter">
    <levelMin value="DEBUG" />
    <levelMax value="INFO" />
  </filter>
  <layout type="log4net.Layout.PatternLayout">
    <conversionPattern value="[%level] %date %logger - %message" />
  </layout>
</appender>
```

Cet appender permet d'écrire dans l'observateur d'événements de Windows, dans la section Application.

applicationName permet de spécifier la source associée à l'événement.

On utilise un filter de type LevelRangeFilter. Cela signifie que l'événement sera loggé si son niveau est compris entre levelMin et levelMax. Autrement dit, ici, seuls les événements ayant un niveau DEBUG ou INFO seront loggés.

- **RollingFileAppender**

```
<appender name="RollingFile" type="log4net.Appender.RollingFileAppender">
  <file value="Log/Log.txt"/>
  <threshold value="INFO"/>
  <appendToFile value="true"/>
  <rollingStyle value="Date"/>
  <datePattern value="yyyyMMdd"/>
  <layout type="log4net.Layout.PatternLayout">
```

```
<conversionPattern value="*%-10level %-30date %-25logger %message %newline"/>
</layout>
</appender>
```

Le `RollingFileAppender`, tout comme le `FileAppender`, permet de logger dans un fichier. Mais il permet en plus de renommer automatiquement les fichiers de logs suivant certains critères (date, taille du fichier .)

file indique le chemin du fichier où seront écrits les logs.

appendToFile indique si le fichier sera écrasé (false) ou si les logs seront écrits à la suite (true).

rollingStyle définit le critère suivant lequel sera renommé le fichier.

datePattern définit le pattern utilisé pour renommer le fichier quand le `rollingStyle` a la valeur `Date`.

Ici, on a `datePattern = "yyyyMMdd"`. Cela signifie que le 1^{er} jour, les logs seront écrits dans `log.txt`. Au deuxième jour, le fichier `log.txt` de la veille sera renommé en `log.txt.20001205` (pour une date du jour étant 06/12/2008).

`rollingStyle` peut également avoir la valeur `Size`. Cela signifie que le fichier sera automatiquement renommé en `log.txt.1` dès qu'il aura atteint une taille limite. Cette taille est définie avec le paramètre `maximumFileSize`. Exemple :

```
<rollingStyle value="Size"/>
<maximumFileSize value="100KB"/>
```

On peut également mixer les 2 `rollingStyles` de la façon suivante:

```
<rollingStyle value="Composite"/>
<maximumFileSize value="1MB"/>
<datePattern value="yyyyMMdd"/>
```

Après les `appenders`, on définit le logger root, le logger par défaut :

```
<root>
  <level value="ALL"/>
</root>
```

On se contente d'y spécifier le niveau de log souhaité.

La suite consiste en créer les loggers que nous souhaitons utilisés dans l'application. Dans l'exemple présent, j'ai choisi de créer 2 loggers :

- un logger général, appelé `MonitoringLogger`
- un logger spécialement dédié au log des erreurs et exceptions, appelé `ExceptionLogger`

Pour créer un logger, il suffit d'ajouter un nœud logger dont l'attribut `name` correspond au nom que l'on souhaite donner au logger. On spécifie ensuite le niveau associé au logger, ainsi que les `appenders` associés.

```
<logger name="MonitoringLogger">
  <level value="ALL"/>
  <appender-ref ref="RollingFile"/>
  <appender-ref ref="DatabaseAppender"/>
  <appender-ref ref="AspNetTraceAppender"/>
  <appender-ref ref="EventLogAppender"/>
</logger>
<logger name="ExceptionLogger">
  <level value="ERROR"/>
  <appender-ref ref="SmtpAppender"/>
  <appender-ref ref="RollingFile"/>
</logger>
```

Ici, on peut voir que le logger `MonitoringLogger` a le niveau `ALL`, c'est-à-dire que tous les événements seront loggés. Les événements seront écrits dans un fichier, en base, dans la trace `asp.net` et dans le journal des événements comme le montre la liste des `appenders` associés.

Quant à lui, le logger `ExceptionHandler` a le niveau `ERROR` étant donné que l'on souhaite ne logger que les erreurs et exceptions. Les événements seront écrits dans un fichier et envoyés par mail.

III.3 - La classe `Log.cs`

Cette classe static permet de centraliser l'instanciation de `log4net` et la création des loggers.

```
using log4net;
using log4net.Config;

[assembly: XmlConfigurator(ConfigFile = "Config/Log4Net.config", Watch = true)]
namespace WebApp
{
    public static class Log
    {
        public static ILog MonitoringLogger
        {
            get { return LogManager.GetLogger("MonitoringLogger"); }
        }

        public static ILog ExceptionLogger
        {
            get { return LogManager.GetLogger("ExceptionHandler"); }
        }
    }
}
```

La ligne suivante:

```
[assembly: XmlConfigurator(ConfigFile = "Config/Log4Net.config", Watch = true)]
```

permet de spécifier le fichier de config de `log4net`. L'attribut `Watch = true` indique que la configuration de `log4net` sera automatiquement rechargée lors de toute modification du fichier de config.

Pour créer un logger, il faut appeler la méthode `LogManager.GetLogger` en lui passant comme paramètre l'attribut `name` de l'élément logger défini dans le fichier de config. Ici, on encapsule cet appel pour chacun des loggers en créant une propriété `MonitoringLogger` et une propriété `ExceptionHandler`. Il suffira d'appeler simplement ces propriétés pour logger les événements souhaités.

La méthode `GetLogger` agit à la manière d'un singleton. Si le logger existe déjà, alors l'instance existante sera retournée. Sinon, une nouvelle instance est créée.

III.4 - Utilisation des loggers.

Pour logger un message, il suffit d'appeler le logger de son choix puis la fonction correspondant au niveau de l'événement.

Par exemple, pour logger une erreur avec le logger `ExceptionHandler`, il suffit d'appeler :

`Log.ExceptionLogger.Error(msg)` où `msg` est le message à logger.

Pour logger une info de debug avec le logger `MonitoringLogger`, il faut appeler :

Log.MonitoringLogger.Debug(msg) où msg est le message à logger.

Pour ajouter une propriété à celles de log4net (comme on a pu le voir pour l'append , on utilise la syntaxe suivante:

```
log4net.ThreadContext.Properties["log_who"] = user;  
Log.MonitoringLogger.Debug(msg);
```

Afin de mettre en pratique tout ça, je vous conseille d'exécuter le site web et d'observer comment les informations sont loggées. Une fois cette étape passée, vous pourrez modifier le fichier de configuration de log4net afin de vous familiariser avec la façon dont fonctionne log4net.

IV - Pour aller un peu plus loin

IV.1 - Si log4net ne logge pas ce que je lui demande

Si jamais log4net ne logge pas correctement, il est possible d'activer ses logs internes.

Pour cela, il suffit d'ajouter dans le Web.Config ou App.Config, les lignes suivantes :

```
<system.diagnostics>  
<trace autoflush="true">  
<listeners>  
<add name="textWriterTraceListener" type="System.Diagnostics.TextWriterTraceListener" initializeData="Log  
\\log4net_InternalDebug.txt"/>  
</listeners>  
</trace>  
</system.diagnostics>
```

Il faut également rajouter la ligne suivante dans la section appSettings :

```
<add key="log4net.Internal.Debug" value="true"/>
```

Log4net écrira alors ses propres logs dans le fichier Log\\log4net_InternalDebug.txt.

Il est également possible de spécifier un chemin absolu comme par exemple C:\\Temp\\log4net_InternalDebug.txt.


IV.2 - Si log4net logge trop

Il est possible de désactiver tous les logs très simplement. Pour cela il suffit d'ajouter threshold="OFF" dans le nœud log4net.

```
<log4net threshold="OFF">  
...  
</log4net>
```

IV.3 - Visualiser les logs

Il existe quelques logiciels permettant de visualiser les logs de façon un peu plus conviviale que notepad.

Baretail est l'un de ces logiciels (disponible  [ici](#)). Il permet notamment de visualiser les logs en surlignant les lignes qui contiennent des mots clés paramétrables et actualise automatiquement l'affichage lors de la modification du fichier.

IV.4 - Personnalisation

Bien que log4net propose de très nombreux appenders, il se peut que vous ne trouviez pas celui qui répond à votre besoin. Pour cela, il est possible de créer son propre appender. Cela sera abordé dans un prochain article.

Conclusion

Log4net est un puissant framework de log entièrement configurable, paramétrable et personnalisable. Toutes ses qualités vous permettront de mettre en place dans vos applications un système de log répondant à vos besoins d'une façon simple et rapide.

Liens utiles



Le site officiel de log4net



Le SDK de log4net



Un fichier d'aide chm sur log4net (très utile car il n'y a pas de fonction de recherche sur le SDK officiel)

Remerciements

Je tiens à remercier Louis-Guillaume Morand et Laurent Dardenne pour leur relecture et conseils.