Log4j

Introduction

Log4j est une API de journalisation très répandue dans le monde Java. Il s'agit d'un système hautement configurable, que ce soit au niveau de ce qui doit être enregistré ou de la destination des enregistrements (serveur de logging, fichiers tournants, *etc.*).

Logger

Comme son nom l'indique, le *Logger* est l'entité de base pour effectuer la journalisation, il est mis en oeuvre par le biais de la classe **org.apache.log4j.Logger**. L'obtention d'une instance de *Logger* se fait en appelant la méthode statique **Logger.getLogger** :

```
import org.apache.log4j.Logger;
public class MaClasse {
	private static final Logger logger = Logger.getLogger(MaClasse.class);
	// suite
}
```

Il est possible de donner un nom arbitraire au Logger, cependant, comme nous le verrons lorsque nous parlerons des hiérarchies de Loggers et de la configuration, il est préférable d'utiliser le nom de la classe pour des raisons de facilité.

Niveaux de journalisation

Il s'agit d'une entité représentant l'importance du message à journaliser, elle est représentée par la classe **org.apache.log4j.Level**. Un message n'est journalisé que si sa priorité est supérieure ou égale à la priorité du *Logger* effectuant la journalisation. L'API Log4j définit 5 niveaux de logging présentés ici par gravité décroissante :

- **FATAL** : utilisé pour journaliser une erreur grave pouvant mener à l'arrêt prématuré de l'application ;
- **ERROR** : utilisé pour journaliser une erreur qui n'empêche cependant pas l'application de fonctionner ;
- **WARN** : utilisé pour journaliser un avertissement, il peut s'agir par exemple d'une incohérence dans la configuration, l'application peut continuer à fonctionner mais pas forcément de la façon attendue ;
- **INFO** : utilisé pour journaliser des messages à caractère informatif (nom des fichiers, *etc.*) ;
- **DEBUG** : utilisé pour générer des messages pouvant être utiles au débogage.

Deux niveaux particuliers, **OFF** et **ALL** sont utilisés à des fins de configuration. La version 1.3 introduira le niveau **TRACE** qui représente le niveau le plus fin (utilisé par exemple pour journaliser l'entrée ou la sortie d'une méthode). Il va de soi que plus l'on descend dans les niveaux, plus les messages sont nombreux. Si vous avez besoin de niveaux supplémentaires, vous pouvez créer les vôtres en sous-classant **org.apache.log4j.Level**, néanmoins, ceux qui sont proposés devraient être suffisants. La journalisation d'un message à un niveau donné se fait au moyen de la méthode **log(Priority, String)**, il existe diverses variantes permettant par exemple de passer un **Throwable** dont la trace sera enregistrée. Pour les niveaux de base, des méthodes de raccourcis sont fournies, elle portent le nom du niveau :

```
try {
// équivaut à logger.info("Message d'information");
logger.log(Level.INFO, "Message d'information");
// Code pouvant soulever une Exception
//...
} catch(UneException e) {
// équivaut à logger.log(Level.FATAL, "Une exception est survenue", e);
logger.fatal("Une exception est survenue", e);
}
```

Appenders

Bien que vous ne devriez pas avoir à manipuler les *Appenders* directement en Java, à moins que vous ne créiez les votres, il est nécessaire de connaître leur fonctionnement afin de configurer correctement Log4j.

Les *Appender*s, représentés par l'interface **org.apache.log4j.Appender**, sont le moyen utilisé par log4j pour enregistrer les événements de journalisation. Chaque *Appender* a une façon spécifique d'enregistrer ces événements. Log4j vient avec une série d'*Appender*s qu'il est utile de décrire, puisqu'ils seront repris dans la configuration :

- **org.apache.log4j.jdbc.JDBCAppender** : Effectue la journalisation vers une base de données ;
- **org.apache.log4j.net.JMSAppender** : Utilise JMS pour journaliser les événements ;
- **org.apache.log4j.nt.NTEventLogAppender** : Journalise *via* le journal des événements de Windows (NT/2000/XP) ;
- **org.apache.log4j.lf5.LF5Appender** : Journalise les événements vers une console basée sur Swing, celle-ci permet de trier ou de filtrer les événements ;
- **org.apache.log4j.varia.NullAppender** : N'effectue aucune journalisation ;
- **org.apache.log4j.net.SMTPAppender** : Envoie un email lorsque certains événements surviennent (à ne pas activer avec un niveau de journalisation **DEBUG**...) ;
- **org.apache.log4j.net.SocketAppender** : Envoie les événements de journalisation vers un serveur de journalisation ;
- **org.apache.log4j.net.SyslogAppender** : Journalise les événements vers un daemon *Syslog* (distant ou non) ;
- **org.apache.log4j.net.TelnetAppender** : Journalise les événements vers un socket auquel on peut se connecter *via* telnet ;
- **org.apache.log4j.ConsoleAppender** : Effectue la journalisation vers la console ;
- **org.apache.log4j.FileAppender** : Journalise dans un fichier ;
- **org.apache.log4j.DailyRollingFileAppender** : Journalise dans un fichier qui tourne régulièrement (contrairement à ce que son nom suggère, ce n'est pas forcément tous les jours) ;
- **org.apache.log4j.RollingFileAppender** : Journalise dans un fichier, celui-ci est renommé lorsqu'il atteint une certaine taille et la journalisation reprend dans un nouveau fichier (par exemple, on aura le fichier logfile dans lequel s'effectue la journalisation et logfile.1 qui contient les événements antérieurs).

Les paramètres nécessaires à certains de ces *Appender*s sont détaillés dans la partie configuration.

Layouts

Les *Layout*s sont utilisés pour mettre en forme les différents événements de journalisation avant qu'ils ne soient enregistrés. Ils sont utilisés en conjugaison avec les *Appender*s. Bien que tous les *Appender*s acceptent un *Layout*, ils ne sont pas forcés de l'utiliser (les *Appender*s utilisant un *Layout* sont repérables au fait que leur méthode **requiresLayout** renvoie true).

Les *Layout*s fournis par log4j sont les suivants, l'existance du **PatternLayout** permet de formatter les événements d'à peu près n'importe quelle façon :

- **org.apache.log4j.SimpleLayout** : Comme son nom l'indique, il s'agit du *Layout*le plus simple, les événements journalisés ont le format **Niveau - Message[Retour à la ligne]** ;
- **org.apache.log4j.PatternLayout** : *Layout* le plus flexible, le format du message est spécifié par un motif (pattern) composé de texte et de séquences d'échappement indiquant les informations à afficher. Reportez vous à la JavaDoc pour une description complète des séquences d'échappement existantes. Par défaut, les événements sont journalisés au format **Message[Retour à la ligne]** ;
- **org.apache.log4j.XMLLayout** : Comme son nom l'indique, formate les données de l'événement de journalisation en XML (à utiliser en conjugaison avec un *Appender* de la famille des *FileAppender*s) ;
- **org.apache.log4j.HTMLLayout** : Les événements sont journalisés au format HTML. Chaque nouvelle session de journalisation (réinitialisation de Log4j) donne lieu à un document HTML complet (*ie*. préambule DOCTYPE, <html>, *etc*).

Fichier Properties

La configuration de log4j peut se faire *via* un fichier properties classique (clé=valeur(s)) ou un fichier XML. La seconde solution offre plus de possibilités et, de par son format, est plus structurée ce qui permet de s'y retrouver plus facilement.

La configuration *via* des fichiers properties est la première à avoir été implémentée, sa structuration est basique et ses possibilités sont plus restreintes que celles des fichiers XML (certains *Appender*s ne peuvent pas être configurés *via* un fichier properties). Néanmoins, il est utile de connaître son organisation puisqu'elle reste très répandue.

L'ordre de configuration présenté ici (*Appender*s puis *Logger*s) est purement arbitraire (elle suit en fait l'ordre utilisé dans la DTD des fichiers XML), vous pouvez faire l'inverse ou même mélanger les deux, log4j s'y retrouve parfaitement.

Vous pouvez définir un niveau minimal pour tous les *Logger*s au moyen de la clé **log4j.threshold**. Cela signifie que tous les messages en dessous de ce niveau seront ignorés par tous les *Logger*s **quel que soit leur niveau** (mais les messages au dessus de ce niveau sont traités normalement, en fonction du niveau de chaque *Logger*).

Si vous voulez que log4j affiche des messages de débogage indiquant les opération effectuées en interne, vous pouvez positionner à **true** la clé **log4j.debug**. Cela peut être utile lors de la mise au point d'une configuration ou d'un Appender personnalisé.

Le fichier properties doit se trouver dans la racine du 'src'.

Configuration des Appenders

Il faut savoir qu'au niveau du fichier de configuration, chaque *Appender* doit avoir un nom afin de pouvoir y faire référence lors de la configuration des *Logger*s. Les paramètres de configuration concernant les *Appender*s sont préfixés par **log4j.appender**. La déclaration d'un *Appender* d'un nom donné se fait de la façon suivante :

```
log4j.appender.NomAppender=ClasseAppender
```

Où *NomAppender* est remplacé par le nom que l'on souhaite attribuer à l'*Appender* et *ClasseAppender* la classe d'implémentation de l'*Appender*. Si l'*Appender* n'a besoin d'aucun autre paramètre pour fonctionner (pas de *Layout* ni de paramètre particulier), il peut être utilisé directement par un *Logger*.

Les options disponibles pour un *Appender* sont parfois décrites dans sa documentation mais vous pouvez toujours les retrouver en regardant les setters de cet *Appender* (méthodes **setXXX**). Le nom du paramètre est alors le nom du setter sans **set** et avec la première lettre en minuscule (par exemple **setLayout** correspond à la propriété *Layout*). Par exemple, si l'on veut configurer le layout pour un **ConsoleAppender**, on procèdera de la façon suivante :

```
log4j.appender.Console=org.apache.log4j.ConsoleAppender
log4j.appender.Console.layout=org.apache.log4j.SimpleLayout
```

Une fois les *Appender*s configurés, il vous reste à configurer les *Logger*s pour qu'ils les utilisent.

Configuration des loggers

La forme de la configuration des *Logger*s est similaire à celle des *Appender*s, néanmoins, les seules choses à configurer sont les *Appender*s et, éventuellement, le niveau et l'additivité des *Appender*s. Aussi la configuration d'un *Logger* a l'aspect suivant :

```
log4j.logger.nom.du.logger=[niveau], appender1, appender2
```

Où :
• **nom.du.logger** est le nom du *Logger* lorsqu'il est demandé *via* **Logger.getLogger** (vous pouvez ne spécifier qu'une partie du nom du *Logger* et vous reposer sur les mécanismes d'héritage de propriétés) ;
• **niveau** est le nom du niveau à attribuer au *Logger*, s'il n'est pas précisé, le niveau est hérité du parent (ou positionné à **DEBUG** pour le *Logger* racine) ;
• **appender*n*** est le nom d'un *Appender* tel qu'il a été déclaré par **log4j.appender.nomAppender**.

*Attention! Le paramètre **niveau** est facultatif mais **pas** la virgule qui le suit !*

La seule particularité concerne la configuration du *Logger* racine. En effet, celui-ci n'a pas de nom attribué puisqu'il n'est accessible que par la méthode **Logger.getRootLogger()**. Aussi, il existe une clé particulière servant à sa configuration qui est **log4j.rootLogger**

```
log4j.rootLogger=DEBUG, file, ConsoleAppender
```

Exemple de fichier Properties

```
log4j.rootLogger=DEBUG,stdout, fichier

log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%-4r %-5p [%t] %c %3x - %m%n

log4j.appender.fichier=org.apache.log4j.RollingFileAppender
log4j.appender.fichier.File=c:\\logs\\exemple.log
log4j.appender.file.MaxFileSize=100KB
log4j.appender.fichier.layout=org.apache.log4j.PatternLayout
log4j.appender.fichier.layout.ConversionPattern=%d %-5p %c - %F:%-4L - %m%n
```

Substitute symbol

%c Logger, %c{2 } last 2 partial names
%C Class name (full agony), %C{2 } last 2 partial names
%d{dd MMM yyyy HH:MM:ss } Date, format see java.text.SimpleDateFormat
%F File name
%l Location (caution: compiler-option-dependently)
%L Line number
%m user-defined message
%M Method name
%p Level
%r Milliseconds since program start
%t Threadname
%x, %X see Doku
%% individual percentage sign

Caution: %C, %F, %l, %L, %M slow down program run!

%-5p Remplissage par des espaces à droite si le nom < à 5 caractères
%5p Remplissage par des espaces à gauche si le nom < à 5 caractères
%.10c Tronquer le début du message si > 10 caractères

La documentation de Log4j:

http://logging.apache.org/log4j/1.2/apidocs/index.html