

Les Framework Java

Log4j

Claude Duvallet

Université du Havre
UFR Sciences et Techniques
25 rue Philippe Lebon - BP 540
76058 LE HAVRE CEDEX
Claude.Duvallet@gmail.com
<http://litis.univ-lehavre.fr/~duvallet/>

Log4j

- 1 Introduction
- 2 Les composants

Log4j : présentation (1/3)

- ▶ Les bonnes pratiques de développement déconseillent l'utilisation des méthodes `System.out.print*` et `System.err.print*` pour afficher des messages et recommandent plutôt l'utilisation d'un logger tel Log4J apportant plus de souplesse.
- ▶ Log4J simplifie les gestions des logs et le débogage des applications Java en fournissant des classes et des méthodes pour l'enregistrement de ces informations.
- ▶ Les fichiers journaux d'une application représentent la mémoire d'une application, un historique permanent de la vie de celle-ci, il est donc important de correctement enregistrer ces messages.
- ▶ Le développeur préférera envoyer le message qu'il souhaite afficher ou enregistrer au logger en lui assignant un certain niveau de criticité (DEBUG, INFO, WARNING, ERROR, CRITICAL).

Log4j : présentation (2/3)

- ▶ On indiquera aussi la classe/la méthode à l'origine de ce message, la ligne dans le code source, ou toute autre information utile.
- ▶ On peut facilement demander à une application d'afficher tous les messages de niveau DEBUG et supérieur à l'écran lors de la phase de développement.
- ▶ Puis on peut lui demander de n'afficher que les messages de niveau WARNING et supérieur dans un fichier de log en phase de production.
- ▶ Ces différents types d'affichage des messages peuvent être configurés facilement au runtime de log4j par fichier XML ou par fichier de propriétés, donc de façon totalement externe au code.

Log4j : présentation (3/3)

- ▶ Log4J est constitué de 3 composants principaux qui permettent de configurer le dispositif de journalisation :
 - les Loggers pour écrire les messages,
 - les Appenders pour sélectionner la destination des messages
 - et les Layouts pour la mise en forme des messages.
- ▶ Log4J permet donc non seulement de gagner en flexibilité sur la gestion des messages d'une application mais également de faciliter la recherche et la détection d'erreurs.

Le framework Log4j : origines

- ▶ C'est une partie du projet Jakarta, sous-projet d'Apache.
- ▶ Il est distribué en Open Source sous Licence BSD.
- ▶ Les premières versions sont apparues en 1996.
- ▶ La documentation officielle est disponible à cette adresse : <http://jakarta.apache.org/log4j/docs/index.html>.
- ▶ La version actuelle est 1.2.

La classe Logger

- ▶ Le Logger est l'entité de base pour effectuer la journalisation, il est mis en œuvre 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, il est préférable d'utiliser le nom de la classe pour des raisons de facilité.

Les niveaux de journalisation (1/3)

- ▶ La notion de niveau de journalisation ou de priorité d'un message représente 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 : journaliser une erreur grave pouvant mener à l'arrêt prématuré de l'application.
 - ERROR : journaliser une erreur qui n'empêche cependant pas l'application de fonctionner.
 - WARN : journaliser un avertissement, il peut s'agir par exemple d'une incohérence dans la configuration.
 - INFO : journaliser des messages à caractère informatif.
 - DEBUG : générer des messages pouvant être utiles au débogage.

Les niveaux de journalisation (2/3)

- ▶ 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).
- ▶ Plus 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`.
- ▶ 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.

Les niveaux de journalisation (3/3)

- ▶ 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);
}
```
- ▶ Il est possible d'effectuer une journalisation avec des messages localisés au moyen des méthodes `l7dlog(Priority, String cle, [Object[],] Throwable)`.
 - `cle` correspond à l'identifiant du message dans le `ResourceBundle` positionné via la méthode `setResourceBundle`.
 - Notez que pour ces méthodes, il n'existe pas de raccourci.

L'interface Appender (1/3)

- ▶ Bien que vous ne devriez pas avoir à manipuler les Appenders directement en Java, il est nécessaire de connaître leur fonctionnement afin de configurer correctement Log4j.
- ▶ Les Appenders, 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'Appenders 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) ;

L'interface Appender (2/3)

- ▶ Suite des Appenders :
 - `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 ;

L'interface Appender (3/3)

- ▶ Fin des Appenders :
 - `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.
- ▶ Les paramètres nécessaires à certains de ces Appenders sont détaillés dans la partie configuration.
- ▶ Notez cependant qu'il est possible d'affecter un niveau seuil (threshold) à tous les Appenders étendant la classe `org.apache.log4j.AppenderSkeleton`.

Cibles des messages (2/3)

- ▶ En affectant par exemple une cible console au logger racine et une cible fichier au logger `org.test` aura les conséquences suivantes :
 - les messages du logger `org` seront affichés en console.
 - et les messages de `org.progx` (et de tous ses enfants) seront affichés en console et enregistrés dans un fichier.
- ▶ Vous pouvez néanmoins prévenir ce fonctionnement en exécutant `setAdditivity(false)` sur le logger concerné.
- ▶ Attention, toutefois car ceci brisera la chaîne de délégation des appenders :

```
Logger.getRootLogger().addAppender(new ConsoleAppender());
Logger log1 = Logger.getLogger("org");
log1.setAdditivity(false);
log1.addAppender(new FileAppender(new SimpleLayout(), "test.log"));
Logger log2 = Logger.getLogger("org.test");
```

Cibles des messages (1/3)

- ▶ Un appender représente donc la cible d'un message, c'est-à-dire l'endroit où celui-ci sera physiquement affiché ou stocké.
- ▶ `log4j` vous propose ainsi des appenders pour la console, les fichiers, les sockets, le gestionnaire d'événements Windows NT, le démon Unix `syslog` ou encore les composants graphiques.
- ▶ Chaque logger dispose de la méthode `addAppender()` permettant de lui affecter une nouvelle cible.
- ▶ La hiérarchie des loggers joue un rôle très important.
- ▶ En effet, chaque message de journalisation sera transmis aux cibles du logger courant ainsi qu'aux cibles de tous ses parents.

Cibles des messages (3/3)

- ▶ Dans l'exemple précédent, les loggers `org` et `org.test` utilisent une cible de type fichier.
- ▶ Aucun d'entre eux ne pourra bénéficier de la cible console affectée à la racine.
- ▶ Et si les différentes cibles offertes par `log4j` ne vous suffisent pas, vous pourrez en créer de nouvelles très facilement.
- ▶ Pouvoir personnaliser la destination des messages ne donne absolument aucune indication sur leur format.

Les Layouts

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

Format des messages

- ▶ Le plus intéressant est indubitablement `PatternLayout` dont la souplesse saura combler toutes vos exigences :

```
Logger log = Logger.getLogger("org.test");
PatternLayout layout = new PatternLayout("%d %-5p %c - %F:%%L - %m%n");
ConsoleAppender stdout = new ConsoleAppender(layout);
log.addAppender(stdout);
```

- ▶ Le format défini dans cet exemple affiche l'heure et la date, le niveau d'erreur (aligné à gauche), le nom du logger, le nom du fichier, le numéro de la ligne de code correspondante et enfin le message lui-même.
- ▶ Le résultat apparaîtra ainsi dans votre console :

```
2003-44-29 04:44:32,211 DEBUG org.test - exemple3.java:18 - Starting
2003-44-29 04:44:32,221 DEBUG org.test - exemple3.java:20 - Exiting
```

Les Layouts

- ▶ Les Layouts fournis par `log4j` sont les suivants, l'existence du `PatternLayout` permet de formater 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.
 - `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 `FileAppenders`) ;
 - `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).