

Jakarta Commons Logging

par [Sébastien Le Ray](#)

Date de publication : 22/06/2005

Dernière mise à jour : 04/07/2005

Ce tutorial est le premier de la série sur les API Commons de Jakarta. Nous étudierons ici l'API Commons Logging qui est une couche d'abstraction pour la plupart des systèmes de journalisation existants.

- I - Introduction
 - I.A - Utilité de la journalisation
 - I.B - Avantages de Commons Logging
- II - Exemple d'utilisation
- III - Configuration
 - III.A - LogFactory
 - III.B - Système de journalisation
- IV - Références
- V - Remerciements
- VI - Téléchargements

I - Introduction

Dans ce tutoriel, nous verrons comment configurer et utiliser l'API Jakarta Commons Logging. Nous n'aborderons pas la configuration du système sous-jacent puisque ce n'est pas le but de commons logging. Un tutoriel sur Log4J sera disponible prochainement.

I.A - Utilité de la journalisation

La journalisation permet d'enregistrer les événements qui surviennent au niveau d'une application. Elle est plus pratique que l'utilisation de **System.out** ou **System.err**, d'une part parce que ceux-ci ne sont pas toujours disponibles (comme dans le cas d'applications serveur par exemple), d'autre part parce qu'ils ne permettent pas de filtrer ce qui doit être affiché.

Les systèmes de journalisation permettent d'attribuer des niveaux à chaque message (débogage, information, etc.), il est ensuite possible d'indiquer quel est le niveau minimum qu'un message doit avoir pour être enregistré. Ils offrent également la possibilité de définir l'endroit où vont être envoyés les messages : console, fichier ou même serveur de journalisation. Ce système est donc beaucoup plus souple que les classiques **System.out** et **System.err**.

On trouve une multitude de systèmes de journalisation, Sun en a même intégré un au JDK 1.4, Apache Jakarta a également créé le sien (Log4J). Dans certain cas, il peut être important de ne pas dépendre d'un système particulier, c'est alors que commons logging peut être utile.

I.B - Avantages de Commons Logging

Commons logging est une API fournie par le projet Apache Jakarta afin d'utiliser la journalisation de façon transparente, sans se préoccuper du système qui la réalisera effectivement. Ainsi, on utilise commons logging pour toutes les instructions de journalisation et c'est au moment de l'exécution du programme que l'API déterminera quel est le système à utiliser. Cela permet de ne pas dépendre d'un système, la seule dépendance est vis-à-vis du package commons logging qui ne pèse qu'une quarantaine de kilo-octets et peut donc être embarqué dans n'importe quelle application.

Comme les commons logging doivent être compatibles avec une grande variété de systèmes de journalisation (Log4J, l'API de logging du JDK 1.4, Avalon, etc.), ils ne reprennent que les caractéristiques communes à tous ces systèmes et ne sont donc pas très compliqués à prendre en main.

Nous commencerons par présenter un exemple simple d'utilisation des commons logging, puis nous expliquerons la façon de les configurer.

II - Exemple d'utilisation

Nous allons commencer par un exemple très simple d'utilisation. Le développeur n'a à se servir que de deux entités, définies dans le package **org.apache.commons.logging** : **Log** et **LogFactory**.

Comme son nom l'indique, la **LogFactory** permet de créer des entités **Log**. Il existe deux façons d'obtenir un **Log** de la part de la **LogFactory**, une version longue :

```
Log log = LogFactory.getFactory().getInstance(MaClasse.class);
```

et une version courte :

```
Log log = LogFactory.getLog(MaClasse.class);
```

Les deux sont strictement équivalents, la deuxième forme est un raccourci pour la première. Voici un exemple d'utilisation des commons logging :

```
package org.phpopensoft.java.tutoriels.commons.logging;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

/**
 * Classe d'exemple pour l'utilisation des commons logging.
 * Enregistre des messages à différents niveaux de gravité.
 * @author Sébastien Le Ray
 */
public class CommonsLoggingSample {
    public static void main(String[] args) {
        Log log = LogFactory.getLog(CommonsLoggingSample.class);
        log.trace("Message de trace");
        log.info("Message d'information");
        log.warn("Message d'avertissement");
        log.error("Message d'erreur");
        log.fatal("Message d'erreur fatale");
    }
}
```

Pour pouvoir compiler ce code, vous devez récupérer l'API commons logging sur le site <http://www.apache.org/dist/jakarta/commons/logging/> et l'inclure dans le chemin de compilation. Voici un exemple d'exécution :

```
4 juin 2005 18:07:08 org.phpopensoft.java.tutoriels.commons.logging.CommonsLoggingSample main
INFO: Message d'information
4 juin 2005 18:07:08 org.phpopensoft.java.tutoriels.commons.logging.CommonsLoggingSample main
ATTENTION: Message d'avertissement
4 juin 2005 18:07:08 org.phpopensoft.java.tutoriels.commons.logging.CommonsLoggingSample main
GRAVE: Message d'erreur
4 juin 2005 18:07:08 org.phpopensoft.java.tutoriels.commons.logging.CommonsLoggingSample main
GRAVE: Message d'erreur fatale
```

On se contente d'appeler **getLog** en lui passant la classe en cours, le **Log** obtenu sera identifié par ce nom. Un **Log** n'est créé que s'il n'en n'existe pas avec le même identifiant. Il est également d'identifier le **Log** par une chaîne quelconque mais l'usage veut que l'on utilise le nom de la classe qui effectue l'appel (ou le nom de son package), qui plus est, cela facilite généralement la configuration du système concret.

Au niveau du code, l'utilisation de commons logging ne sera généralement pas plus compliquée que cela. Il faut noter que dans des cas complexes, une autre famille de méthodes peut être utilisée en conjonction avec celles de journalisation : la famille des **isxxxEnabled()**, où xxx est le nom d'un niveau (**isTraceEnabled()**, **isDebugEnabled()**,

etc.). Cette famille de méthodes renvoie true si le niveau considéré est activé, false sinon. Les instructions de journalisation ressemblent alors à cela :

```
if(log.isTraceEnabled()) {
    log.trace("Entrée dans la méthode xxxx, paramètres : " +
        uneInstanceComplicquéeAAfficher + ', ' + UneMethodeTresLongue());
}
```

Dans le cas où le niveau est journalisable, cela ne fait que dupliquer un test qui est déjà effectué pour savoir si le message est enregistré (le temps perdu est négligeable), mais dans le cas où le niveau ne l'est pas, cela évite tout le processus d'évaluation et de passage des paramètres ce qui permet d'éviter au maximum la perte de temps.

Le second aspect des commons logging, au moins aussi important que le code, est la configuration, que nous allons maintenant aborder.

III - Configuration

Il faut préciser que la configuration doit être faite par l'utilisateur de l'application et non pas par le programmeur. En effet, l'API commons logging sert de couche d'abstraction, si le programmeur la configure, c'est qu'il émet des suppositions sur l'environnement dans lequel va s'exécuter l'application et est donc en contradiction avec ce principe d'abstraction.

III.A - LogFactory

La **LogFactory** utilisée pour obtenir une instance de **Log** est en fait une classe abstraite, elle va rechercher quelle est l'implémentation à utiliser. **LogFactory** utilise le processus de recherche suivant :

- Recherche d'un attribut système nommé **org.apache.commons.logging.LogFactory** (cet attribut est défini en passant l'option `-Dorg.apache.commons.logging.LogFactory=classe d'implémentation à la JVM`) ;
- Utilisation du système de recherche de service du JDK 1.3 (<http://java.sun.com/j2se/1.3/docs/guide/jar/jar.html>) pour trouver un fichier appelé **META-INF/services/org.apache.commons.logging.LogFactory** dont la première ligne doit contenir le nom de la *LogFactory* ;
- Recherche du fichier de propriétés **commons-logging.properties** qui doit contenir l'attribut **org.apache.commons.logging.LogFactory** indiquant la Factory à utiliser ;
- Utilisation de la *Factory* par défaut (**org.apache.commons.logging.impl.LogFactoryImpl**).

La recherche s'effectue dans l'ordre présenté ci-dessus. Dès qu'une des conditions est remplie, la recherche s'arrête et c'est la *Factory* trouvée qui est utilisée.

Dans la plupart des cas, l'implémentation par défaut est suffisante. L'utilisation d'une factory différente peut être justifiée dans des cas particuliers comme l'exécution dans un environnement conteneur (serveur) ou des contraintes de taille (embarqué), il faut alors créer sa propre *Factory* et renseigner l'une des propriétés citées ci-dessus (utiliser la propriété système accélère le processus).

III.B - Système de journalisation

Nous nous concentrerons sur la configuration de la *Factory* par défaut. C'est elle qui va déterminer quelle est l'implémentation de **Log** à utiliser. Si un fichier de configuration **commons-logging.properties** existe, les propriétés qu'il définit sont passées à la *Factory*. La *Factory* par défaut utilise le mécanisme de recherche suivant pour le *Log*, il est assez similaire à celui de la recherche de *Factory* :

- Recherche d'une propriété **org.apache.commons.logging.Log** (et **org.apache.commons.logging.log** pour compatibilité) indiquant l'implémentation à utiliser ;
- Recherche des mêmes propriétés au niveau système ;
- Recherche du système de journalisation Log4J ;
- Recherche du système de journalisation du JDK 1.4 ;
- Recherche du système de journalisation du JDK 1.3 ;
- Utilisation de l'implémentation **org.apache.commons.logging.impl.SimpleLog** par défaut.

On voit donc que là encore le mécanisme de recherche a été conçu de façon à utiliser le meilleur système de journalisation disponible. La configuration de l'emploi d'un **Log** particulier n'est justifiée que si l'on utilise un système de journalisation qui n'est pas pris en compte dans le processus de recherche. Attention toutefois, si le système de journalisation n'est pas disponible à l'exécution le mécanisme de recherche ne reprend pas et une exception est levée.

Il est donc préférable de s'appuyer sur le mécanisme de détection proposé et de configurer Log4J et le SimpleLog, puisque la configuration de l'API du JDK se fait au niveau système. Ainsi, on est sûr d'avoir un système de journalisation fiable. **SimpleLog** est configurable *via* un fichier nommé **simplelog.properties**, les propriétés que l'on peut y renseigner sont les suivantes :

- **org.apache.commons.logging.simplelog.defaultlog** : indique le niveau de détail par défaut ("trace", "debug", "info", "warn", "error", ou "fatal", prend la valeur "info" par défaut) ;
- **org.apache.commons.logging.simplelog.log.nomlogger** : niveau de détail du logger indiqué ;
- **org.apache.commons.logging.simplelog.showlogname** : booléen indiquant si le nom du logger doit être affiché dans les messages (désactivé par défaut) ;
- **org.apache.commons.logging.simplelog.showShortLogname** : indique si un nom simple doit être affiché pour le logger (actif par défaut) ;
- **org.apache.commons.logging.simplelog.showdatetime** : indique si la date et l'heure doivent être affichées (masquées par défaut) ;
- **org.apache.commons.logging.simplelog.dateTimeFormat** : donne le format d'affichage de la date et de l'heure (au format SimpleDateFormat, format par défaut "yyyy/MM/dd HH:mm:ss:SSS zzz").

La configuration de Log4J fera l'objet d'un tutoriel séparé car elle est trop complexe pour être présentée ici.

Vous en savez maintenant suffisamment sur commons-logging pour l'utiliser de façon régulière. Essayez, vous verrez, il devient vite difficile de s'en passer une fois que l'on est habitué.

IV - Références

- [Le site des commons-logging](#) ;
- [Les services de journalisation d'Apache](#) (où vous trouverez notamment l'API Log4J).

V - Remerciements

Merci à lunatix pour sa relecture et à Bestiol pour ses corrections.

VI - Téléchargements

- L'article au format [HTML Zippé \(mirroir\)](#) ;
- L'article au format [PDF \(mirroir\)](#).