DÉPARTEMENT
Sciences Informatiques

Licence 2 I
2012–2013
Génie Logiciel
T.T. Dang Ngoc
dntt@u-cergy.fr

TD7 - Traces

Nous reprendrons dans ce TP, le projet PROJET-GENLOG du TP 1.

Log4j est un utilitaire de gestion des journaux permettant :
- d'afficher des renseignements telles que la date, thread, classe, numéro de ligne, etc...
- de spécifier lors de l'execution du programme et non de sa compilation du niveau des messages voulu (info, debug, etc ...), leurs sorties et leurs formats.

Il permet d'effectuer aussi bien du 'débuguage' qu'une journalisation des évenements.

Il est disponible sur :

`http://logging.apache.org/log4j/`

Il existe plusieurs types de configuration de la gestion de traces :
- *BasicConfigurator :* sans fichier de configuration
- *PropertyConfigurator :* avec un fichier de configuration de type *properties* nommé en général `log4j.prop`.
- *DOMConfigurator :* avec un fichier de configuration de type *XML* nommé en général `log4j.xml`.

Dans le fichier source de la classe `gl.test.TestZoo` :

1 Utilisation basique de `log4j`

1. Importez les paquetages `org.apache.log4j.Logger`, `org.apache.log4j.Level` et `org.apache.log4j.BasicConfigurator`
2. Déclarez la variable de classe suivante :
   `private static final Logger logger = Logger.getLogger(TestZoo.class) ;`
3. Initialisez avec :
   `PropertyConfigurator.configure();`

Le traçage se fait ensuite *via* la méthode `log` de la classe Logger (que vous avez instanciée avec la variable `logger`) de la manière suivante :
`logger.log (Level.`*`NIVEAU_DE_TRACE`*`, `*`message`*`) ;`
avec *NIVEAU_DE_TRACE* correspondant à :

Niveau de trace	Description
FATAL	erreur grave pouvant mener à l'arrêt prématuré de l'application
ERROR	erreur qui n'empêche cependant pas l'application de fonctionner
WARN	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	messages à caractère informatif (nom des fichiers, etc.)
DEBUG	messages pouvant être utiles au débogage.
TRACE	traces très fines pouvant être utiles à un débogage pas à pas

À noter qu'un ensemble de méthodes (info, fatal, error, debug, etc.) de la classe `Logger` ont été définies telles que : `logger.log (Level.INFO, "Message")` est équivalent à `logger.info ("Message")`

Question 1

Modifiez la classe TestZoo en remplaçant les sorties faites par `System.out.println` par un traçage de niveau INFO. Ajoutez également deux traces de type DEBUG lors de l'entrée et dans la fonction `main` et lors de sa sortie par exemple.

2 Utilisation de `log4j` avec properties

Nous travaillerons dans cette section avec la classe `org.apache.log4j.PropertyConfigurator` qui utilise un fichier de configuration externe que l'on appelle traditionnellement `log4j.prop`. Cela a pour énorme avantage de pouvoir changer la configuration une fois le programme compilé.

Dans le fichier source de la classe `gl.test.TestGeo` :

1. Importez les paquetages `org.apache.log4j.Logger`, `org.apache.log4j.Level` et `org.apache.log4j.PropertyConfigurator`
2. Déclarez la variable de classe suivante :
   `private static final Logger logger = Logger.getLogger(TestGeo.class) ;`
3. Initialisez avec :
   `PropertyConfigurator.configure("./log4j.prop");`

On y définira dans ce fichier `log4j.prop` les variables suivantes :

```
log4j.rootLogger=NIVEAU, NOM_APPENDER1, NOM_APPENDER2, ...

log4j.appender.NOM_APPENDER=CLASSE_APPENDER
log4j.appender.NOM_APPENDER.NOM_OPTION=VALEUR_OPTION
```

où :
- *log4j.rootLogger :* Définit le niveau (DEBUG, INFO, ERROR, etc.) à partir duquel tracer, ainsi qu'un ou des noms d'*appender*. Un *appender* est un élement permettant de décrire "où" sortira le message (sur la console à l'écran, dans un fichier, envoyé par mail, etc.)
- *log4j.appender.NOM_APPENDER=CLASSE_APPENDER :* on définit ici le type d'*appender* utilisé pour chaque nom d'*appender* déclaré dans `log4j.rootLogger`.
  - *org.apache.log4j.ConsoleAppender :* affiche sur la console
  - *org.apache.log4j.FileAppender :* stocke les messages dans un fichier
  - *org.apache.log4j.JDBCAppender :* enregistre les messages dans une base de données
  - *org.apache.log4j.SocketAppender :* envoie les messages par réseau via une socket
  - *org.apache.log4j.SMTPAppender :* envoie les messages par mail
  - *et encore :* `org.apache.log4j.JMSAppender`, `org.apache.log4j.AsyncAppender`, ...
- *log4j.appender.NOM_APPENDER.NOM_OPTION=VALEUR_OPTION :* Suivant l'*appender* choisit, les options permettant de paramétrer le comportement de cet *appender* pourront être initialisées par ce biais.

Question 2

1. En utilisant `PropertyConfigurator` et l'*appender* `FileAppender`, redirigez (grace à l'option `file` permettant de préciser le fichier de trace voulu) quelques traces de debug, info, erreur et log dans le fichier `geo.log`. Pour cela, on mettra dans le fichier `log4j.prop` les lignes suivantes :

```
log4j.rootLogger=DEBUG, A1
log4j.appender.A1=org.apache.log4j.FileAppender
log4j.appender.A1.file=/VOTRE_CHEMIN_DE_REPERTOIRE/fichier.log
```

2. Testez et regardez dans le fichier de trace que vous avez défini, les traces au fur et à mesure de l'exécution de votre programme (vous pouvez pour cela utiliser la commande UNIX : `tail -f` *`fichier_log`*
3. Modifiez ensuite le niveau de traçage (par exemple à ERROR) sans recompiler le programme en modifiant la variable `log4j.rootLogger`.
4. Testez à nouveau.

2.1 Layout

Le layout est utilisé pour formatter les messages. Il en existe différentes sortes :

- *SimpleLayout :* Simple affichage du message.
- *PatternLayout :* Utilisation de variables semblables à la fonction printf() du langage C
- *HTMLLayout :* Le 'HTMLLayout' affiche les messages au format html. Les informations sont rangés dans un tableau qui contient 5 colonnes : le temps, le thread, le niveau, la catégorie et le message.
- *XMLLayout :* Le 'XMLLayout' fournit les messages encapsulés dans des balises xml
- *TTCCLayout :* Affiche le contexte d'exécution du message.

Le PatternLayout est le Layout le plus souple et le plus configurable. Comme son nom l'indique, il utilise un motif pour mettre en forme les informations fournies par l'événement de journalisation. (il s'utilise un peu comme `printf` en C) :

%c	nom du logger
%m	message de journalisation
%n	caractère de nouvelle ligne spécifique à la plate-forme.
%p	niveau de gravité de l'événement.
...	...

Description de toutes les instructions de formatage sur :
`http://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/PatternLayout.html`

Question 3

Ajoutez dans le fichier `log4j.prop` les lignes suivantes :

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

Testez à nouveau.

3 Lectures des traces

Vous pouvez lire les traces sur la console ou par l'intermédiaire d'un fichier comme vu précédemment. Vous pouvez également utiliser un logiciel permettant de lire graphiquement les traces.

3.1 Chainsaw

`Chainsaw` est un utilitaire fourni avec `log4j`. Il vous permet de disposer d'une interface graphique pour lire vos messages transmis via un `XMLLayout` ou `SocketLayout`. Il est lancé par :

```
java -cp /usr/share/log4j/lib/log4j-1.2.8.jar org.apache.log4j.chainsaw.Main
```

Pour cela, vous modifierez votre fichier `log4j.prop` de la manière suivante :

```
log4j.rootLogger=DEBUG, A1
log4j.appender.A1=org.apache.log4j.FileAppender
log4j.appender.A1.layout=org.apache.log4j.xml.XMLLayout
log4j.appender.A1.append=true
log4j.appender.A1.file=/VOTRE_CHEMIN_DE_REPERTOIRE/log.xml
```

et lancerez ensuite `chainsaw`.

4 Références

http://logging.apache.org/log4j/1.2/apidocs/overview-summary.html
http://www.supinfo-projects.com/fr/2004/log4j%5Ffr/introduction/
http://beuss.developpez.com/tutoriels/java/jakarta/log4j/