CSC4509 — Journalisation avec LOG4j

Éric Lallet

Télécom SudParis

30 avril 2025

La biobliothèque LOG4j

LOG4j est une bibliothèque JAVA qui permet la journalisation pendant l'exécution.
Trois composants principaux :

Les *Loggers* : permettent de créer un flux de journalisation. Possibilité de créer plusieurs *Loggers* différents pour une même application.

Les *Appenders* : permettent de sélectionner la ou les destinations du message : console, fichier, base de données, réseau...

Les *Layouts* : permettent de formater les messages.

Les niveaux de journalisation

Les *Loggers* permettent de discriminer le niveau d'importance du message.

	trace()	debug()	info()	warn()	error()	fatal()
ALL	OK	OK	OK	OK	OK	OK
TRACE	OK	OK	OK	OK	OK	OK
DEBUG		OK	OK	OK	OK	OK
INFO			OK	OK	OK	OK
WARN				OK	OK	OK
ERROR					OK	OK
FATAL						OK
OFF						

Donc par exemple, au niveau WARN, les méthodes `trace()` , `debug()` et `info()` ne font rien, et les méthodes `warn()` , `error()` et `fatal()` journalisent les messages.

La classe common.Log

Les archives de TP arrivent avec une classe préparée (`common.Log`) et un fichier de configuration (`src/main/resources/log4j2.xml`). Cela vous fournit :

- Trois *Loggers* pré-créé : *Log.GEN* («général»), *Log.COMM* («communication»), *Log.TEST* («test»).
- Affichage sur la console.
- *Loggers* préréglés sur le niveau *WARN*.
- Une méthode pour changer le niveau de journalisation : `setLevel(final Logger logger, final Level level)`

Format des messages

Messages affichés avec le format par défaut (5 champs) :
19 [main] INFO communication – position and capacity : 40 40

1. temps en millisecondes depuis le lancement du programme.
2. thread faisant l'appel de la méthode.
3. niveau du message.
4. nom du *Logger*.
5. le message à journaliser après le –

Exemple d'usage (1)

La préparation des *Loggers* :

```
import static tsp.csc4509.common.Log.COMM;
import static tsp.csc4509.common.Log.TEST;
import static tsp.csc4509.common.Log.GEN;
import org.apache.logging.log4j.Level;


(...)
Log.setLevel(COMM, Level.TRACE);


(...)
Log.setLevel(TEST, Level.WARN);
```

Exemple d'usage (2)

Journalisation d'un message :

```
import static tsp.csc4509.common.Log.COMM;
import static tsp.csc4509.common.Log.TEST;
import org.apache.logging.log4j.Level;

(...)
int lus;

for(int nb = 0; nb < 10; nb++) {
    (...)
    final int num = nb;
    COMM.info("{}", () -> "message " + num + " sent.");
}

lus = read(....);

final int val = lus;
TEST.trace("{}", () -> val + " bytes received.");
```


- Les patrons "{}" présents dans la chaîne en premièr paramètre des méthodes de log sont remplacés par les valeurs passées dans les autres paramètres de la méthode ;
- Nous utilisons pour la valeur du patron le résultat d'une lambda expression, ce qui permet son évaluation tardive : l'expression n'est pas évaluée lors de l'appel de la méthode mais lors de son usage. Si la méthode ne l'utilise pas (parce que le level du log ne demande pas d'affichage) , elle n'est pas évaluée.

Note : Une expression lambda peut utiliser une variable locale d'une méthode, mais **à condition que cette variable soit finale ou effectivement finale**. Ce n'est pas le cas des variables «nb» et «lus» que nous voulons loguer. Il faut donc les recopier dans une variable finale et utiliser cette copie dans l'expression lambda.