CSC4509 — Journalisation avec LOG4j

Éric Lallet

Télécom SudParis

26 avril 2021

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 vous fournissent une classe préparée pour faire la journalisation des programmes : `common.Log`

- Trois *Loggers* pré-créé : *Log.GEN* («géneral»), *Log.COMM* («communication»), *Log.TEST* («test»).
- Affichage sur la console.
- *Loggers* préréglés sur le niveau *WARN*.
- Une méthode pour changer le niveau de journalisation : `configureALogger(final String loggerName, 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 common.Log.COMM;
import static common.Log.TEST;
import static common.Log.LOGGER_NAME_COMM;
import static common.Log.LOGGER_NAME_TEST;
import static common.Log.LOG_ON;
import org.apache.log4j.Level;

(...)

Log.configureALogger(LOGGER_NAME_COMM, Level.TRACE);

(...)
Log.configureALogger(LOGGER_NAME_TEST, Level.WARN);
```

Exemple d'usage (2)

Journalisation d'un message :

```
import static common.Log.COMM;
import static common.Log.TEST;
import static common.Log.LOGGER_NAME_COMM;
import static common.Log.LOGGER_NAME_TEST;
import static common.Log.LOG_ON;
import org.apache.log4j.Level;

(...)

if (LOG_ON && COMM.isInfoEnabled()) {
        COMM.info("Waiting for connection request...");
}

(...)
if (LOG_ON && TEST.isEnabledFor(Level.WARN)) {
        TEST.warn("starting the server...");
}
```


Note 1 : même sans le test «*if (LOG_ON && COMM.isInfoEnabled())*» le «*COMM.info("Waiting for connection request...");*» ne provoquera un affichage que si le niveau activé est inférieur ou égale à `INFO` (avec sous sans test, il n'y aura pas d'affichage avec le niveau `WARN` activé).
Le test sert à éviter l'évaluation des paramètres de la méthode. Il est possible de passer en paramètres des méthodes d'affichage des objets complexes dont la méthode *toString()* peut être coûteuse. Avec le test (très rapide), les niveaux de logs non-activés ne pénalisent pas les performances.

Note 2 : les niveaux `DEBUG` et `INFO` possèdent leur propre méthode pour tester s'ils sont activés (respectivement *isDebugEnabled()* et *isInfoEnabled()*), mais pas les autres niveaux. Pour eux il faut utiliser la méthode *isEnabledFor()* avec le bon niveau passé en paramètre. Voir l'exemple du niveau `WARN` sur la diapositive.