

Java Utils Logging

JUL

Olivier Cailloux

LAMSADE, Université Paris-Dauphine

Version du 2 novembre 2016

Utilité du logging

Logging : pour quoi faire ?

Utilité du logging

Logging : pour quoi faire ?

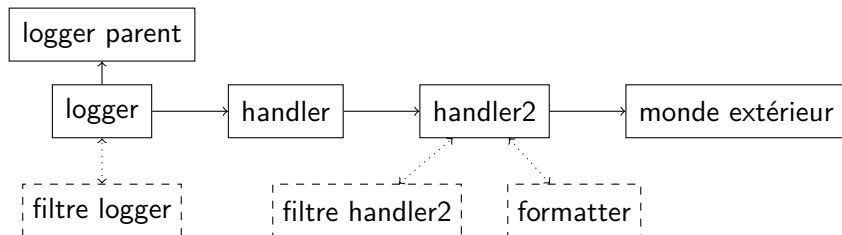
- Débug pour soi-même
- Écriture systématique de ce qui se passe : souhait de conserver les informations dans le code
- Tout en pouvant désactiver la sortie à la demande
- Gain de temps possible par rapport à points d'arrêt
- Débug chez le client
- Visualisation des opérations des bibliothèques utilisées
- Granularité fine : seulement tel type de message
- Exemple : voir les commandes SQL envoyées par fournisseur de persistance

Moteurs de logging

- Ici : utilisation de Java util logging (JUL)
- Partie de Java SE
- En Java SE comme en Java EE
- Autres moteurs populaires de logging en Java : SLF4J, Commons logging voir [annexe](#) pour brève justification du choix
- Interfaçage généralement presque transparent
- Exemple : Hibernate utilise JBoss Logging, mais fonctionne avec logging standard

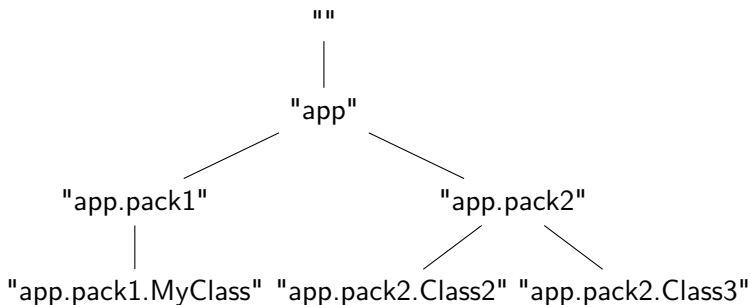
Vue d'ensemble

- Développeur utilise **Loggers** pour logger
- Relation parent – enfant d'après noms des loggers
- Loggers passent les logs sous forme de **LogRecords** à **Handler**
- Passés aussi à logger parent
- Handlers publient (avec **Formatters**)
- Ou Handler fait suivre à autre Handler
- Loggers et Handlers utilisent log levels et filtres



Hiérarchie

- Root logger nommé ""
- Hiérarchie suit typiquement le nom des packages
- Recommandation : nommer le logger d'une classe selon le nom de la classe



Obtention d'un Logger

- Recommandation : Logger pour une classe stocké dans champ `private static Logger logger`
- Obtenir logger d'après nom avec `Logger.getLogger(String)`
- Nom : `MyClass.class.getCanonicalName()`
- String renvoyé ?

Obtention d'un Logger

- Recommandation : Logger pour une classe stocké dans champ `private static Logger logger`
- Obtenir logger d'après nom avec `Logger.getLogger(String)`
- Nom : `MyClass.class.getCanonicalName()`
- String renvoyé ? `"app.pack1.MyClass"`

Obtention d'un Logger

- Recommandation : Logger pour une classe stocké dans champ `private static Logger logger`
- Obtenir logger d'après nom avec `Logger.getLogger(String)`
- Nom : `MyClass.class.getCanonicalName()`
- String renvoyé ? `"app.pack1.MyClass"`
- Avantage par rapport à `Logger.getLogger("app.pack1.MyClass")` ?

Obtention d'un Logger

- Recommandation : Logger pour une classe stocké dans champ `private static Logger logger`
- Obtenir logger d'après nom avec `Logger.getLogger(String)`
- Nom : `MyClass.class.getCanonicalName()`
- String renvoyé ? `"app.pack1.MyClass"`
- Avantage par rapport à `Logger.getLogger("app.pack1.MyClass")` ? Refactoring : nom lié explicitement à la classe

Obtention d'un Logger

- Recommandation : Logger pour une classe stocké dans champ private static Logger logger
- Obtenir logger d'après nom avec `Logger.getLogger(String)`
- Nom : `MyClass.class.getCanonicalName()`
- String renvoyé ? `"app.pack1.MyClass"`
- Avantage par rapport à `Logger.getLogger("app.pack1.MyClass")` ? Refactoring : nom lié explicitement à la classe

```
private static Logger logger = Logger.getLogger(  
    MyClass.class.getCanonicalName());
```

Loggons

- Recommandation : se concentrer sur 3 niveaux de log (parmi [sept](#))
- SEVERE (erreurs), INFO, FINE (debug fin)
- `logger.info(String)`
- `logger.log(Level, String, Throwable)`
- `logger.log(Level, String, Object[])`
- Ne pas logger une exception si elle est relancée (pourquoi ?)

Loggons

- Recommandation : se concentrer sur 3 niveaux de log (parmi [sept](#))
- SEVERE (erreurs), INFO, FINE (debug fin)
- `logger.info(String)`
- `logger.log(Level, String, Throwable)`
- `logger.log(Level, String, Object[])`
- Ne pas logger une exception si elle est relancée (pourquoi ? sinon, double log !)

Configuration

- `LogManager` chargé de la configuration
- `LogManager` est singleton
- Lit fichier de configuration au démarrage ou utilise classe spéciale
- D'après propriété système
`java.util.logging.config.file`
- Format standard fichier de propriétés java (`Properties`)
- Sinon, configuration par défaut fichier `lib/logging.properties` installé avec Java
- Aussi possible configurer via API de `LogManager`

Fichier de configuration

Fichier de configuration contient des paires `prop = valeur`.

Propriétés :

`monlogger.level` Niveau de log de monlogger et ses enfants (cf. [Level](#))

`monlogger.handlers` Liste de classes Handlers pour monlogger

`handlers` Liste de classes Handlers pour root logger

`monlogger.useParentHandlers` Bool, indique s'il faut faire suivre le message aux parents

`HandlerClass.level` Niveau de log de HandlerClass

`HandlerClass.prop` Autre propriété de HandlerClass

Handlers et formatters inclus

Handlers inclus :

`StreamHandler` Écrit dans un `OutputStream`

`ConsoleHandler` Écrit dans `System.err`

`FileHandler` Écrit dans fichiers

`SocketHandler` Écrit sur ports TCP

`MemoryHandler` Enregistre en mémoire

Formatters inclus :

`SimpleFormatter`

`XMLFormatter`

Configuration : recommandations

- Indiquer configuration dans un fichier `logging.properties`
- Livrer ce fichier avec l'application (dans classpath : requiert un chargement adapté)
- Permet à l'utilisateur de changer les options de log au besoin
- Indiquer à la JVM le fichier de configuration avec option `-Dprop=value`
- Changer le niveau de log de certains loggers en fonction intérêt du développeur

Option JVM

- Démarrer avec :
`"-Djava.util.logging.config.file=logging.properties"`
- Dans eclipse : Preferences / Java / Installed JREs / Edit / Default VM arguments

Exemple de configuration [ici](#)

Références

- [Guide Oracle](#)

SLF4J – JUL

- SLF4J souvent plébiscité sur le web comparé à JUL ([SO](#))
- Options toutes deux raisonnables, mais pour ce cours il fallait faire un choix : pourquoi JUL ?
- Favoriser les standards
- Fonctionne sans configuration dans environnement Java EE
- JUL *semble* aussi populaire d'après une estimation très hasardeuse
- On peut intégrer certains avantages de SLF4J **après coup** n'évite sans-doute pas une perte de performance, mais vraisemblablement sans importance

Popularité

Nombre de correspondances dans codes sur GitHub

SLF4J $2,0 \times 10^6$ Ou [org.slf4j.Logger](#) : $2,0 \times 10^6$

JUL $2,0 \times 10^6$ Ou [java.util.logging.Logger](#) : $1,7 \times 10^6$

Mais il vaudrait mieux comparer le nombre de (gros) projets (récents) qui utilisent chaque moteur...

Licence

Cette présentation, et le code LaTeX associé, sont sous [licence MIT](#). Vous êtes libres de réutiliser des éléments de cette présentation, sous réserve de citer l'auteur.
Le travail réutilisé est à attribuer à [Olivier Cailloux](#), Université Paris-Dauphine.