FRAMEWORK LOGGER

Projet – Architecture Logicielle

Résumé

Documentation du Framework de Log & choix techniques

Lucile Incardona & Renaud Lamarque

Incardona@et.esiea.fr & lamarque@et.esiea.fr

Table des matières

I. Détail technique

A. Version java

Nous avons choisi de développer ce Framework sous Java 1.8 afin de profiter de la dernière technologie disponible.

B. Maven

Nous avons choisi de réaliser ce Framework avec le module Maven afin de faciliter les builds et simplifier les imports. Il nous sert notamment pour l'utilisation de JUnit.

C. Juint

Afin de tester notre Framework, nous avons effectué de nombreux tests grâce à JUnit. Tests unitaires, tests d'intégrations…

D. Interface

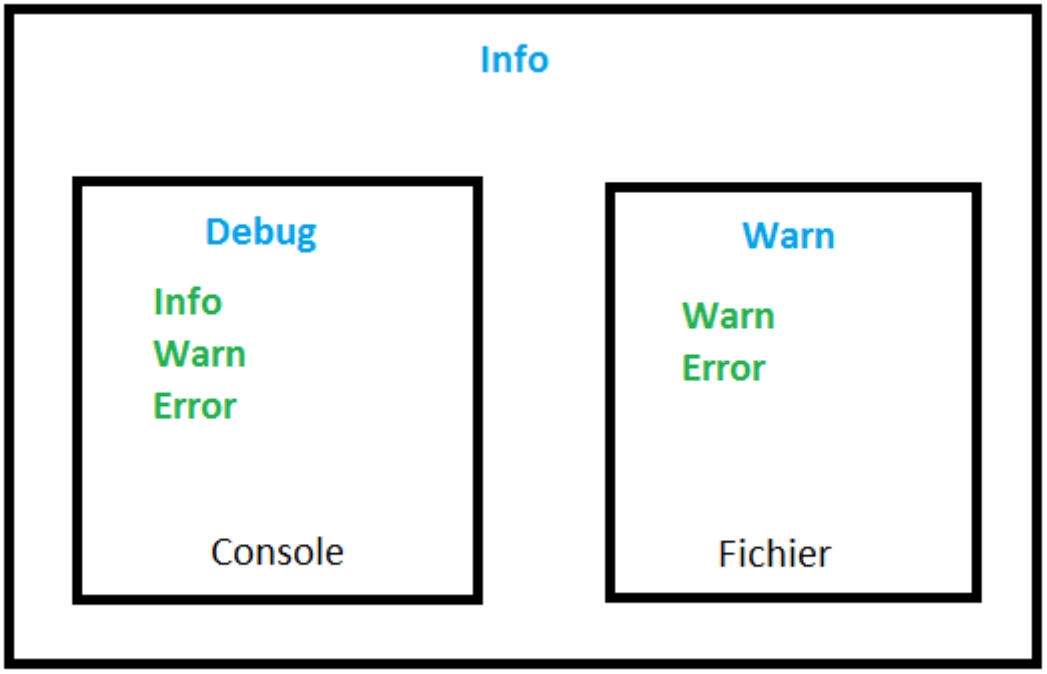
Lors de la conception de la cible du log, nous avons remarqué que nous avions besoin de fonction identiques quelques soit les cibles. De ce fait, nous avons voulu centraliser l'intelligence. Deux choix s'offraient à nous. Une classe abstraite ou une Interface. Nous avons opté pour l'interface car nous pouvons implémenter plusieurs interfaces dans une classe contrairement à la classe abstraite. Il ne peut y avoir qu'un seul héritage dans une classe fille. De plus, l'interface nous semble plus adaptée dans l'optique de personnalisation des cibles.

II. Fonctionnement du Framework

A. Présentation

Notre Framework présente les caractéristiques suivantes.

- Il possède 4 niveaux de sévérité
  - Debug
  - Info
  - Warn
  - Error
- Le format des logs est personnalisable
- Il existe 2 cibles pré-intégrées dans le Framework mais il est possible d'un configurer d'autres
- Un niveau de sévérité global est défini, mais il est possible de redéfinir un niveau de sévérité général et par type de sortie. Ce qui nous permet d'avoir cette utilisation possible.



B. Exemple

Voici un exemple d'utilisation du Framework. Il s'agit de son utilisation la plus simple qui repose sur les paramètres par défauts.

```java
import com.esiea.logging.LoggerFactory;


public class Main {
     public static final Logger LOGGER = LoggerFactory.getLogger(Main.class);

    public static void main(String[] args) {
        LOGGER.debug("message");
        LOGGER.info("message");
        LOGGER.warn("message");
        LOGGER.error("message");
    }

}
```

C. Paramètres par défaut

1. Format par défaut

Le programme précédent présentera ses résultats sous la forme suivante :

[SEVERITE] Date – Heure at classeConcerne – Message

Ce qui nous donne :

```
[INFO] 27/02/2015 - 03:15:54 at Main.java – Message
```

2. Sévérité par cible

Par défaut, toutes les sévérités maximums (globale, fichier et console) sont mise à Info.

Avec l'exemple précédent, nous obtenons donc les résultats suivants.

Dans un fichier :

```
[INFO] 27/02/2015 - 03:15:54 at Main.java – Message
[WARN] 27/02/2015 - 03:15:54 at Main.java – Message
[ERROR] 27/02/2015 - 03:15:54 at Main.java – Message
```

Dans la console:

```
[INFO] 27/02/2015 - 03:15:54 at Main.java – Message
[WARN] 27/02/2015 - 03:15:54 at Main.java – Message
[ERROR] 27/02/2015 - 03:15:54 at Main.java – Message
```

III. Personnalisation

A. Personnalisation du format de log

1. Formatteur par défaut

Grâce au Formatter, il est facile de changer le format du log.

```
Formatter formatter = new FormatterImpl(null);

Formatter formatter = new FormatterImpl("");
```

Ces deux déclarations prendront comme paramètres les paramètres par défaut. Mais il est possible de personnaliser le format des log. Il suffit de modifier la valeur des paramètres des méthodes. Attention : les « items » de log doivent être contenu entre #.

Exemple :

```
Formatter formatter = new FormatterImpl("[#SEVERITY#] #DATE# - #MESSAGE#");
Formatter formatter = new FormatterImpl("[#SEVERITY#] #DATE yyyy-MM-dd# - #MESSAGE#");
```

2. Implémentation de nouveaux formatteurs

Il est possible d'autres par d'implémenter de nouveaux formatteurs pour correspondre parfaitement aux besoins. Et ainsi redéfinir un nouveau format.

Exemple :

```
package com.esiea.logging.services.formatter;

public class NewFormatterImpl implements Formatter{

}
```

B. Personnalisation des cibles

1. Choix de la cible

2. Nouvelles cibles

De la même manière pour les cibles, il est possible d'implémenter de nouvelles cibles grâce à l'interface Target.

C. Personnalisation des niveaux de sévérités

D. Fichier Properties

Il est possible de configurer le logger grâce à un fichier properties.

Exemple :

```
com.esiea.loggin.severity=INFO

com.esiea.loggin.formatter.class=com.esiea.logging.services.formatter.FormatterImpl
com.esiea.loggin.formatter.pattern=[#SEVERITY#] #DATE# at #CLASS# - #MESSAGE#

com.esiea.loggin.target.1.class=com.esiea.logging.services.target.TargetConsole
com.esiea.loggin.target.1.severity=DEBUG

com.esiea.loggin.target.2.class=com.esiea.logging.services.target.TargetFile
com.esiea.loggin.target.2.file=testLogger.log
com.esiea.loggin.target.2.severity=WARN


testConfigClass.severity=DEBUG

testConfigClass.formatter.class=com.esiea.logging.services.formatter.FormatterImpl
testConfigClass.formatter.pattern=[#SEVERITY#] - #MESSAGE#

testConfigClass.target.1.class=com.esiea.logging.services.target.TargetFile
testConfigClass.target.1.file=testLogger2.log
testConfigClass.target.1.severity=INFO
```